UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

E INFORMÁTICOS

Tema:

PROTOTIPO DE UN CHATBOT PARA UN SISTEMA DE SERVICIOS VEHÍCULARES UTILIZANDO ADAPTIVE CARDS CON BOT FRAMEWORK

Trabajo de Titulación Modalidad: Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Sistemas Computacionales e Informáticos.

ÁREA: Software

LÍNEA DE INVESTIGACIÓN: Desarrollo de Software

AUTOR: Llerena Valencia, Cristian Homero

TUTOR: Ing. Clay Fernando Aldás Flores, Mg.

Ambato – Ecuador

Julio 2021

APROBACIÓN DEL TUTOR

En calidad de tutor del Trabajo de Titulación con el tema: PROTOTIPO DE UN CHATBOT PARA UN SISTEMA DE SERVICIOS VEHICULARES UTILIZANDO ADAPTIVE CARDS CON BOT FRAMEWORK, desarrollado bajo la modalidad Proyecto de Investigación por el señor Cristian Homero Llerena Valencia, estudiante de la Carrera de Ingeniería Sistemas Informáticos y Computacionales de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 15 del Reglamento para obtener el Título de Tercer Nivel, de Grado de la Universidad Técnica de Ambato, y el numeral 7.4 del respectivo instructivo.

Ambato, julio de 2021.

Firmado electrónicamente por:
CLAY FERNANDO ALDAS FLORES

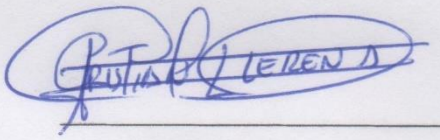
Ing. Clay Fernando Aldas Flores, Mg.

TUTOR

AUTORÍA

El presente Proyecto de Investigación titulado: PROTOTIPO DE UN CHATBOT PARA UN SISTEMA DE SERVICIOS VEHICULARES UTILIZANDO ADAPTIVE CARDS CON BOT FRAMEWORK, es absolutamente original, auténtico y personal. En tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, julio de 2021



Cristian Homero Llerena Valencia

C.C. 1803818432

Autor

APROBACIÓN TRIBUNAL DE GRADO

En calidad de par calificador del Informe Final del Trabajo de Titulación presentado por el señor Cristian Homero Llerena Valencia estudiante de la Carrera de Ingeniería en Sistemas Computacionales e Informáticos, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado PROTOTIPO DE UN CHATBOT PARA UN SISTEMA DE SERVICIOS VEHICULARES UTILIZANDO ADAPTIVE CARDS CON BOT FRAMEWORK, nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 17 del Reglamento para obtener el Título de Tercer Nivel, de Grado de la Universidad Técnica de Ambato, y al numeral 7.6 del respectivo instructivo. Para cuya constancia suscribimos, conjuntamente con la señora Presidenta del Tribunal.

Ambato, julio 2021.

Firmado electrónicamente por:
ELSA PILAR URRUTIA

Ing. Pilar Urrutia, Mg.

PRESIDENTA DEL TRIBUNAL

Firmado electrónicamente por:
HERNAN FABRICIO NARANJO AVALOS

Ing. Hernán Naranjo

PROFESOR CALIFICADOR

Firmado electrónicamente por:
JULIO ENRIQUE BALAREZO LOPEZ

Ing. Julio Balarezo

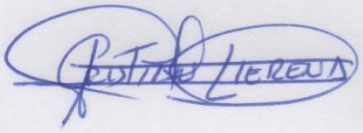
PROFESOR CALIFICADOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato, para que haga uso de este Trabajo de Titulación como un documento disponible para la lectura, consulta y procesos de investigación.

Cedo los derechos de mi Trabajo de Titulación en favor de la Universidad Técnica de Ambato o con fines de difusión pública. Además, autorizo su reproducción total o parcial dentro de las regulaciones de la Universidad.

Ambato, julio 2021



Cristian Homero Llerena Valencia

C.C. 1803818432

AUTOR

DEDICATORIA

A mi madre Mariana Valencia por el apoyo incondicional brindado a lo largo de mi vida. Por sus cuidados, cariños, educación y valores impartidos que han formado y marcado mi vida, ya que sin ella no hubiera logrado esta meta tan importante en mi vida profesional.

A mi padre Julio Valencia (q.e.p.d.) por el tiempo que estuvo conmigo compartiendo su amor, consejos y valores.

Gracias.

Cristian Homero Llerena Valencia

AGRADECIMIENTO

Quiero expresar mi gratitud a Dios, quién con su bendición llena siempre mi vida y a toda mi familia, por ser el apoyo en aquellos momentos de dificultad y debilidad.

Gracias a mi madre: Mariana Valencia por ser el pilar fundamental en mi vida, por sus consejos, valores y principios que se me ha inculcado.

Agradezco a los docentes de la Carrera en Ingeniería en Sistemas Computacionales e Informáticos y en especial a mi tutor el Ing. Clay Aldas, por haber compartidos sus conocimientos y haberme guiado en este largo proceso.

Cristian Homero Llerena Valencia

ÍNDICE GENERAL DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN EJECUTIVO

En la actualidad las tecnologías informáticas han evolucionado de manera sorprendente, brindando nuevas y mejores funciones en el campo del desarrollo de software. Esto ha permitido que los sistemas convencionales sufran cambios notorios en sus funciones primordiales gracias a la incorporación de la inteligencia artificial y lograr extender las funcionalidades en diversas tecnologías existentes.

Los avances en el área de inteligencia artificial, han favorecido de manera significativa al desarrollo de nuevas interfaces de usuarios volviéndolas más dinámicas y fáciles de utilizar a través de agentes conversacionales más conocidos como chatbots.

La presente investigación se la realizó con la finalidad de presentar los diversos contextos de aplicación para agentes conversacionales. Además, se **analizó las técnicas y herramientas más completas y sofisticadas de última generación** para el desarrollo de agentes conversacionales y de prototipos.

Para el desarrollo de chatbot se utilizó una plataforma potente basada en lenguaje de programación C# llamada Bot Framework. Para dotar de inteligencia artificial al agente conversacional se utilizó el procesamiento de lenguaje natural mediante el uso de la tecnología LUIS de Azure. Además, la demostración del agente será desde un entorno local.

Palabras clave: Agente conversacional, adaptive cards, bot framework.

ABSTRACT

Currently computer technologies have involved in an amazing way, providing new and better functions in the field of software development. This has allowed those conventional systems suffer a noticeable change in their primary functions because of the incorporation of artificial intelligence, which has made it possible to extend the functionalities to various existing technologies.

Advances in artificial intelligence area have significantly favored the development of new user interfaces, making them more dynamic and easier to use through conversational agents better known as chatbots.

This research was making in order to present the various application contexts for conversational agents. In addition, the most complete and sophisticated techniques and tools of today for the development of conversational agents are analyzed, for which a prototype has been developed.

For the development of the chatbot, a powerful platform based on the C # programming language called Bot Framework was used. To provide artificial intelligence to the conversational agent, natural language processing was used through the use of LUIS of Azure technology. In addition, for the demonstration of the agent it will be done in a local environment.

Keywords: Conversational agent, adaptive cards, bot framework.

INTRODUCCIÓN

Como estructura del presente proyecto de investigación denominado “PROTOTIPO DE UN CHATBOT PARA UN SISTEMA DE SERVICIOS VEHICULARES UTILIZANDO ADAPTIVE CARDS CON BOT FRAMEWORK”, consta de cuatro capítulos que se detallan a continuación.

CAPÍTULO I.-MARCO TEÓRICO, este capítulo representa la contextualización del problema en base a investigaciones similares. Además, del análisis crítico del estado actual del problema y los objetivos que se pretende alcanzar.

CAPÍTULO II.- METODOLOGÍA, sección que representa los materiales utilizados en la metodología. Además, se establece los tipos de investigación, recolección de información, procesamiento y análisis de datos.

CAPÍTULO III.-RESULTADOS Y DISCUSIÓN, sección donde se realizan el análisis y discusión en base a los datos obtenidos a través de la Metodología Ágil aplicada. Además, del desarrollo de la propuesta. Es decir, el desarrollo del Prototipo de Chatbot.

CAPÍTULO IV.-CONCLUSIONES Y RECOMENDACIONES, esta sección representa las conclusiones y recomendaciones que resultaron una vez terminado el desarrollo del proyecto de investigación.

CAPÍTULO I

MARCO TEÓRICO

1.1 Tema de Investigación

"PROTOTIPO DE UN CHATBOT PARA UN SISTEMA DE SERVICIOS VEHICULARES UTILIZANDO ADAPTIVE CARDS CON BOT FRAMEWORK".

1.2 Antecedentes Investigativos

En la investigación realizada por el estudiante Omar Zarabia de la Escuela Politécnica Nacional en el año 2018 denominado: "Implementación de un chatbot con Bot Framework: caso de estudio, servicios de clientes del área de seguros Equinoccial" describe el desarrollo de un agente transaccional enfocado al área de atención al cliente. Esta investigación propone una arquitectura que integra los servicios web propios del caso de estudio, además de una base de conocimiento, servicios cognitivos y Bot Framework de Microsoft, el agente utiliza inteligencia artificial por lo que tiene la capacidad de brindar respuestas efectivas a través de un dialogo personalizado y enfocado en el procesamiento de lenguaje natural y machine learning. **[1]**

Marcos Pérez de la Universidad de Málaga (España) en el año 2014 en su trabajo titulado: "Análisis y optimización de agentes conversacionales 3D para sistemas empotrados" plantea un modelo de chatbot para dispositivos móviles ya que en la actualidad la revolución tecnológica va creciendo sin precedentes. La investigación consta de dos puntos fundamentales, el primero se refiere en dotar al agente conversacional de una arquitectura distribuida mediante una aplicación que se ejecuta a modo de cliente en el terminal, el segundo punto trata acerca de la ejecución de forma remota. **[2]**

Erick Gamboa de la Universidad Técnica de Ambato en el año 2019 con su investigación de titulación denominada: "Prototipo de un chatbot para compras online utilizando Bot Framework" plantea un modelo de chatbot utilizando Bot Framework,

el cual está basado en lenguaje de programación C#. Además, el chatbot utiliza inteligencia artificial mediante el procesamiento de lenguaje natural a través de DialogFlow, por ser un tema de investigación pura se demuestra la implementación en un entorno local. **[3]**

Alexis Durán de la Universidad de las Fuerzas Armadas (ESPE) del año 2015 en su trabajo de titulación denominado:" Diseño e implantación de un asesor virtual con interfaz web basado en un sistema de gestión de conocimientos y autoaprendizaje" plantea un modelo de chatbot para un portal web de una empresa que brinda servicios de asesoría contable y tributaria, este agente dispone de un gestor de conocimientos, base de datos, motor de autoaprendizaje, motor de reportes y finalmente la interfaz principal. El chatbot realizará procesos interrelacionados de los módulos contables y tributarios que dispone el portal web. **[4]**

Javier Medina de la Revista Iberoamericana de Informática Educativa del año 2013, en su documento científico titulado como: "Asistentes virtuales en plataformas 3.0", plantea un chatbot con arquitectura basado en técnicas de procesamiento del lenguaje natural. Además, la opción de desplegar los agentes conversacionales en diversas plataformas. **[5]**

Los agentes conversacionales son excelentes trabajando en el área de relación con los clientes, especialmente en empresas que venden y compran productos por la web. Además, mejoran la atención que se brinda al cliente con el objetivo de optimizar recursos. **[6]**

Dependiendo de la utilidad que brinda el chatbots para resolver una tarea califica su nivel de credibilidad con el usuario. **[7]**

Los beneficios que prestan los agentes conversacionales son muy importantes, ya que prestan servicios las 24 horas del día generando más ganancias y reduciendo recursos a las empresas que los implementan en sus plataformas virtuales. **[7]**

Países desarrollados como Estados Unidos, China, España, Italia han implementado agentes conversacionales en sus plataformas virtuales con la finalidad de solventar las necesidades de usuarios que a diario navegan en diversos sitios web. **[8]**

Grandes empresas reconocida a nivel mundial como Siri (Apple), Cortana (Microsoft), Alexa (Amazon) y Asistente de Google (Google), han implementado chatbots de voz en sus plataformas virtuales con la finalidad de brindar servicios de búsqueda rápida mediante comandos de voz, a pesar de las funciones limitadas que tienen los chatbots al momento de mantener una interacción completa con respuestas razonables en situaciones diversas son muy útiles y recomendables de usarlos ya que estos agentes facilitan algunas tareas que al usuario le parecen muy engorrosas de realizar. **[7]**

En países de América Latina existe un desinterés mayoritario acerca de la utilización de dichos agentes especialmente en sistemas de servicios vehiculares ya sea por falta de conocimiento o de innovación en los módulos de servicios, la mayoría de bots se los encuentra en sistemas de compras online y de entidades financieras. **[3]**

En el contexto nacional existe algunas investigaciones de agentes conversacionales aplicados en varios campos de la web. En un caso puntual de la Universidad Tecnológica Equinoccial (UTE), han desarrollado agentes que interactúan con los socios de la Cooperativa Policía Nacional, este agente virtual ofrece una serie de servicios financieros a más de 70000 clientes de dicha Entidad Financiera, utiliza Inteligencia Artificial para poder reconocer en tiempo real el tipo de pregunta o comentario y así proveer las respuestas acertadas en base a un cuestionario de preguntas previamente programadas por los desarrolladores del sistema. **[8]**

Para solventar las necesidades de los usuarios mediante agentes conversacionales se comprenderá la experiencia del usuario dentro del contexto tecnológico además de tomar en cuenta el nivel de aceptación y de interacción de los usuarios con el sistema, es decir si le resulta fácil o difícil la utilización del sistema. **[8]**

El hecho de incorporar un agente conversacional a un sistema de servicios vehículares facilitará las tareas que debe realizar el usuario para adquirir los servicios que tiene el sistema, la experiencia entre el bot y el usuario resultara más natural y amigable.

Además, el motivo de plantear un prototipo es con fines de validar la funcionalidad del aplicativo. **[8]**

1.2.1 Fundamentación Teórica

1.2.2 Chatbots o Agentes Conversacionales

Un chatbot es un sistema informático que interactúa con los usuarios a través de una conversación, la cual está conformada por un conjunto de diálogos preprogramados, cuando un chatbot esta entrenado adquiere funciones extras que le permitirán cumplir las necesidades de los usuarios con más facilidad. **[3]**

Con la implementación de procesadores de lenguaje natural se puede entrenar al agente virtual con el objetivo de que pueda entender lo que el usuario trata decir o que función desea realizar, posteriormente el bot emitirá un dialogo coherente y relacionado a la tarea que se desea realizar con efectividad. **[3]**

Las características que tiene un chatbot lo asemeja a un sistema experto, basado en la cantidad de conocimiento que dispone, simula una conversación inteligente con el usuario. **[3]**

Un chatbot no está ligado solo a texto plano, si no a un conjunto de contenidos multimedia, permitiendo una mejor interacción con el usuario. **[6]**

Los agentes conversacionales basados en Inteligencia Artificial y herramientas de computación en la nube utilizan técnicas para procesar el lenguaje natural y poder desarrollar agentes con capacidad de razonamiento. **[3]**

1.2.2.1 Características de los Chatbots

Los chatbots tienen diferentes características que los diferencian, dichas características les permiten desempeñar múltiples funciones, entre las más importantes tenemos las siguientes:

- **Adaptabilidad:** está vinculada con la capacidad de aprendizaje que puede tener un chatbot, actuando de manera cambiante en su comportamiento en base a lo que va aprendiendo. **[1]**
- **Expresividad**: aprovecha los elementos que dispone las aplicaciones de mensajería o canales que utilizan multimedia (gráficos, videos y sonido) con la finalidad de que los diálogos sean más expresivo y amigables con el usuario. **[1]**
- **Racionalidad:** el chatbot analiza todas las respuestas y selecciona la más apropiada dependiendo de los datos que recepta. **[1]**
- **Proactividad**: el chatbot tiene la capacidad de exhibir un comportamiento hacia un objeto siempre y cuando tenga iniciativa al momento de entablar una conversación. **[1]**
- **Personalizable:** el programador decide las características que le agregara al chatbot. Estos agentes virtuales pueden mostrar emociones, interpretar sentimientos, intenciones. **[1]**
- **Sociabilidad:** es la capacidad que un chatbot tiene para comunicarse con otros agentes o entidades. **[3]**
- **Reactividad:** es la capacidad de poder emitir respuestas enriquecidas, es decir, el agente no se limita en dar respuestas genéricas. **[3]**
- **Veracidad:** es una cualidad que posee el chatbot para emitir información confiable. **[3]**

1.2.2.2 Tipos de Chatbots

No existe una estandarización general de chatbots, sin embargo, se los ha agrupado dependiendo de las características que posee. A continuación, se presenta en forma general los tipos de chatbots.

Chatbots según el servicio:

- **Operativos o Empresariales:** son bots que actúan como ayudantes o facilitadores de servicios de una organización. Además, estos chatbots agilitan procesos y mejoran los tiempos de respuesta. **[1]**
- **Informativos:** estos bots cumplen funciones específicas de poca complejidad, las cuales fueron programadas previamente por los desarrolladores. **[1]**
- **E-commerce:** más conocidos como asistentes virtuales comerciales, el objetivo primordial de este tipo de bot es ayudar a los usuarios con la compra de cualquier producto. **[1]**

Chatbots según el diseño de la interfaz:

- **Con interfaz solo de texto:** son bots que interactúan con los usuarios mediante textos de entradas y salidas. **[1]**
- **Con interfaz combinada entre texto, botones e imágenes:** son bots que interactúan con los usuarios a través de diálogos enriquecidos, estos diálogos pueden ser texto, imágenes, formularios y botones de acción. Una de las tecnologías que permite la incorporación de estas funciones son los Adaptive Cards. **[1]**

Chatbots según la tecnología usada:

- **Simple:** son bots cuyo funcionamiento se basa en la relación de patrones básicos con una respuesta. **[1]**
- **Complejos:** son bots más completos ya que utilizan técnicas de IA (Inteligencia Artificial) con un conjunto de estados conversacional y son integrados en servicios empresariales. **[1]**

Chatbots según el campo de aplicación:

- **De utilidad:** estos bots cumplen funciones específicas, se enfocan en un objetivo específico y su nivel de confiabilidad es medido dependiente el cumplimiento de dicho objetivo. **[3]**

- **Sociales:** son bots que tiene apariencia influyente hacia los usuarios, tratan de mantener un estado de conversación estable y confiable, por lo general encontramos estos agentes en servicios de atención al cliente y en redes sociales. **[3]**
- **Asistentes:** son bots que ayudan en la realización de cualquier tarea facilitando el trabajo al usuario, generalmente se los encuentra en sistemas operativos como Cortana o Google Assistant. Son similares a los bots sociales ya que posee un nivel de personalidad influyente. **[3]**

1.2.2.3 Arquitectura de un Chatbot

Un agente conversacional (chatbot) está conformado por tres componentes esenciales: interfaz de usuario, motor de inferencia y base de conocimiento, como se representa en la figura 1.1.

Interfaz de Usuario: es el puente de comunicación entre usuario y el chatbot. Además, permite que el usuario realice consultas o introduzca entradas de información y visualiza los resultados de las consultas realizadas. La interfaz de usuarios puede recibir y enviar información al motor de inferencia. [1, 18]

Motor de Inferencia: La información que ingresa, es analizada y procesada con la finalidad de obtener una respuesta de acuerdo a la base de conocimiento, esta respuesta es enviada de vuelta hacia la interfaz de usuario [1, 14]. En términos técnicos representa el procesamiento de lenguaje natural y como resultado de obtiene una respuesta acorde a la información correspondiente a la base de conocimiento. El motor de inferencia utiliza dos elementos importantes: los datos y el conocimiento. Los datos están conformados por los hechos y evidencias, el conocimiento por un conjunto de reglas almacenadas en la base de conocimiento. Estos dos componentes esenciales trabajan a la par con el objetivo de obtener nuevas conclusiones o hechos. [1, 8, 18]

Base de conocimiento: es un contenedor de toda la información almacenada por parte del experto humano, el ingreso de esta información se lo realiza a través de patrones, plantillas y reglas. [1, 18]

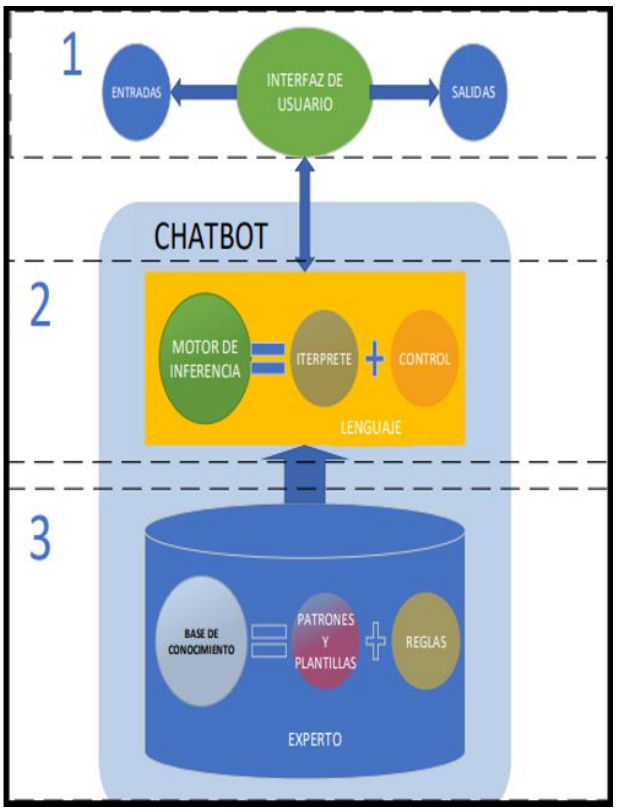


Figura 1.1: Arquitectura Genérica de un Chatbot.
Fuente: [1, 8, 18].

1.2.2.4 Diseño de Navegación de los Chatbots

La navegación de un bot es muy diferente a una aplicación tradicional, ya que no dispone de botones de retroceso y avance (atrás, adelante) ni rutas de navegación. Esto dificulta el desarrollo del mismo, ya que no se puede utilizar alguna técnica de navegación estandarizada dentro del agente conversacional. El diseño de navegación va de la mano con el manejo de interrupciones por lo que se debe considerar que se maneje correctamente dichas interrupciones con la finalidad de brindar un abuena experiencia con el usuario. **[9]**

Un bot puede tener elementos de navegación llamativos dependiendo de las características, funcionalidades que tenga. Para que un bot tenga una excelente

aceptación por parte de los usuarios se debe tomar en cuenta que las interfaces conversacionales estén bien diseñadas y no correr riesgos comunes. Los riegos comunes se pueden en cinco personalidades **[9]**:

- Agente Virtual Terco.
- Agente Virtual Despistado.
- Agente Virtual Misterioso.
- Agenete Virtual Capitan Obviedad.
- Agente Virtual que no olvida.

A continuacion, se detalla de forma mas profunda su analisis:

Agente Virtual Terco: esta personalidad insiste en mantener el estado actual de la conversación, incluso cuando el usuario quiere cambiar el rumbo de conversación. Este tipo de personalidad se presenta cuando el usuario cambia de parecer durante la conversación, por lo que los usuarios pierden la confianza en el agente y proceden a cancelan la conversa. Para evitar estos problemas se debe desvalorar agentes virtuales capaces de discernir la información, es decir, el bot debe tener en cuenta que el usuario puede cambiar el curso de la conversación, lo más recomendable es no ignorar la entrada del usuario y seguir repitiendo la misma pregunta en un buble infinito. **[9]**

Una manera de moderar esta personalidad es especificar un número máximo de reintentos para cada entrada, el bot talvez no pueda comprender al usuario y dar respuestas acertadas tratando de evitar que el bot entre en un buble infinito con la misma pregunta. **[9]** En la figura 1.2, se visualiza un ejemplo de un agente virtual terco:

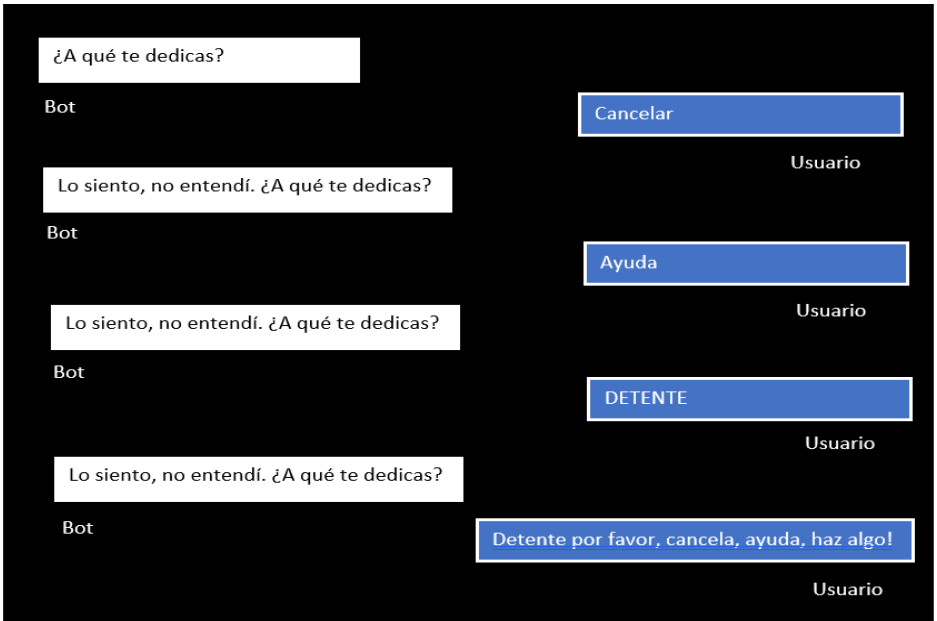


Figura 1.2: Ejemplo de un Bot Terco.
Fuente: Elaborado por el investigador y basado en ***[9]***.

Agente Virtual Despistado: esta personalidad causa que el bot responda de forma absurda, es decir no entiende la entrada de un usuario. A pesar de que el usuario busque alternativas para retomar la conversación mediante palabras clave, como "cancelar"," ayuda" el bot no es capaz de responder de forma correcta. Para evitar caer en este tipo de personalidad, se debe crear clases intermedias con el objetivo de poder procesar palabras claves, mencionadas anteriormente, también estas palabras claves se las puede pasar por alto si es necesario, no se puede diseñar diálogos exclusivos en el tratamiento de palabras claves. **[9]** En la figura 1.3, se visualiza un ejemplo de un agente virtual despistado:

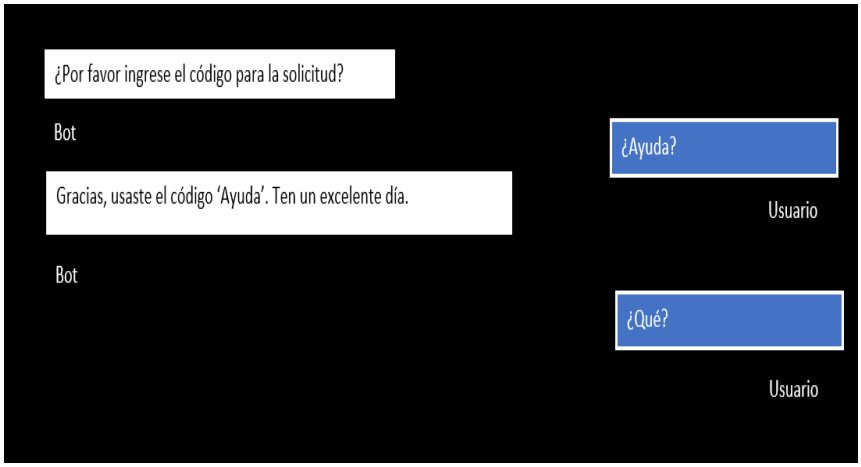


Figura 1.3: Ejemplo de un Bot Despistado.
Fuente: Elaborado por el investigador y basado en ***[9]***.

Agente Virtual Misterioso: en este tipo de personalidad el agente conversacional no tiene la capacidad de reconoces la entrada del usuario de ninguna manera posible. Esta situación podría indicar que algo sucede con el bot, ya sea una interrupción del servicio o simplemente no termina de procesar la entrada de información del usuario y por lo tanto no termina de compilar la respuesta, para evitar estos problemas se recomienda dotar al bot con la capacidad de poder reconocer la entrada del usuario lo más rápido posible, eliminando cualquier posibilidad de confusión al momento de analizar la entrada de información. **[9]**

Si la respuesta tardo demasiado en compilarse, se recomienda enviar un mensaje de "escritura" para notificar al usuario que el bot está trabajando o que algo sucedió durante el procesamiento de la respuesta. **[9]** En la figura 1.4, se visualiza un ejemplo de un agente virtual misterioso:

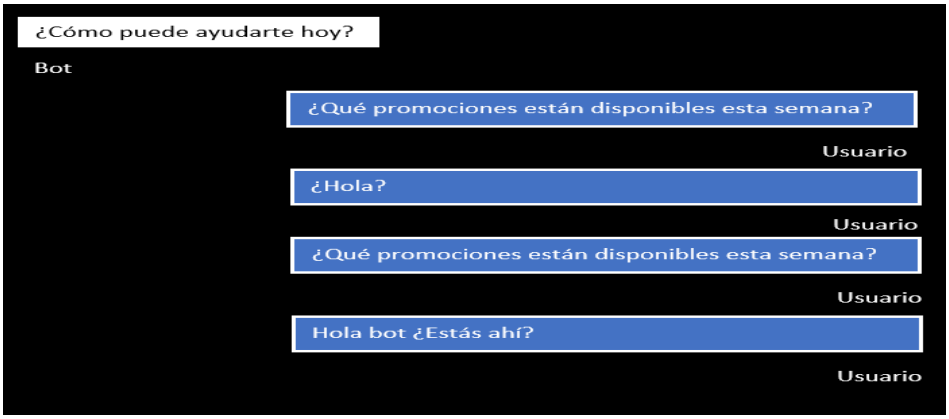


Figura 1.4: Ejemplo de un Bot Misterioso.
Fuente: Elaborado por el investigador y basado en ***[9]***.

Agente Virtual Capitán Obviedad: este tipo de personalidad proporciona información que en un inicio el usuario no solicito y que es totalmente obvia lo que resulta ser información inútil. Para evitar este problema se debe desarrollar un bot que disponga la cualidad de proporcionar información útil y confiable, esto aumentará la posibilidad de integración entre bot y usuario. **[9]**. En la figura 1.5, se visualiza un ejemplo de un agente virtual capitán obviedad:

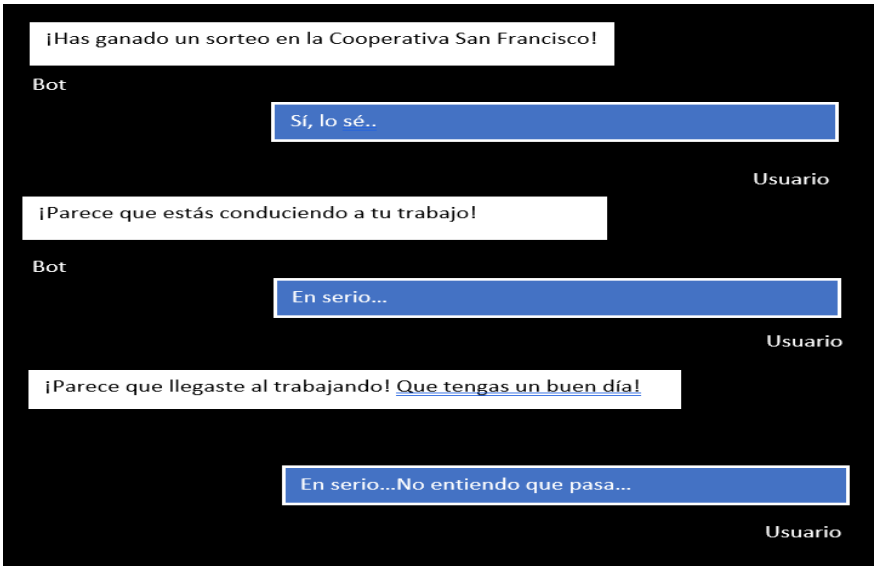


Figura 1.5: Ejemplo de un Bot Capitán Obviedad.
Fuente: Elaborado por el investigador y basado en **[9]**.

Agente Virtual que no olvidar: en este caso el bot integra de forma incorrecta información de conversaciones anteriores en la actual. Para evitar este tipo de personalidad se recomienda diseñar un bot que tenga la capacidad de mantener el estado de la conversación actual. Además, el bot debe retomar conversaciones anteriores cuando el usuario lo requiera. Mantener el estado actual de la conversación reduce considerablemente la posibilidad de confusión y desinterés de la misma. **[9]** En la figura 1.6, se visualiza un ejemplo de un agente virtual que no puede olvidar su estado actual:

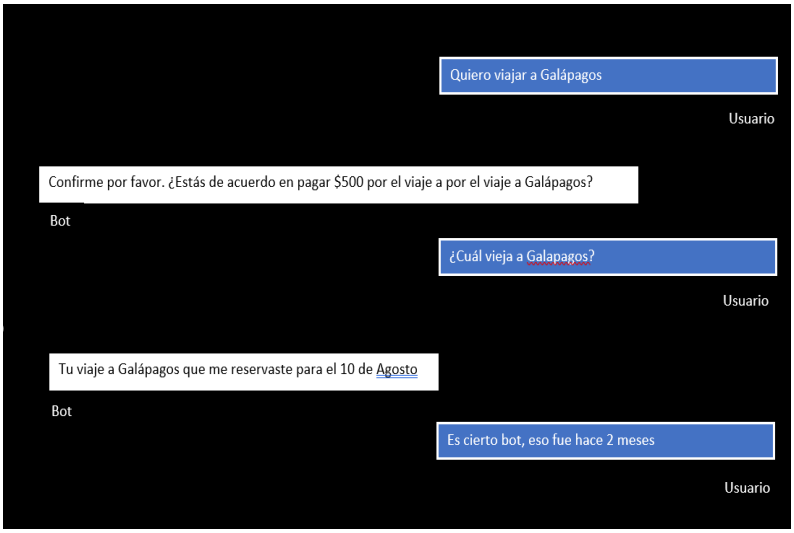


Figura 1.6: Ejemplo de un Bot que no Olvida.
Fuente: Elaborado por el investigador y basado en ***[9]***.

1.2.2.5 Aplicaciones de los Chatbots

Una de las ventajas más importantes de un chatbot es la de poder integrarse en distintos canales de mensajería. Además, de su alto nivel de disponibilidad y fácil usabilidad. En la figura 1.7 se visualiza los diversos campos donde son aplicados los chatbots.

Figura 1.7: Campos de Aplicación de un Chatbot.
Fuente: ***[1]***.

1.2.2.6 Diseño del Flujo de Conversación de un Chatbot

Al momento de desarrollar cualquier bot, un aspecto muy cuestionado es el diseño, ya que por lo general se necesita desarrollar un bot potente para que pueda resolver cualquier objetivo que se programe cumplir. **[10]**

Un concepto de programación muy importante al momento de desarrollar agentes virtuales es la migración ya que esto implica diseñar de forma más simple y objetiva todas las funciones de un bot. Se debe cumplir un orden al momento de estructurar los diálogos que servirán como guía para el usuario y poder cumplir correctamente las tareas asignadas al bot. **[10]**

Para un fácil desempeño durante el diseño del flujo de conversación, la empresa Microsoft recomienda el uso de tablas de comunicación, dichas tablas contienen información referente y concisa de una conversación, lo cual permitirá discernir de forma más sofisticada y sintética el desarrollo de los diálogos. **[10]**

Figura 1.8: Flujo de Comunicación de un Chatbot.
Fuente: ***[3]***.

Según los conceptos planteados por Microsoft, se debe incluir de forma obligatoria las tareas principales con sus respectivos objetivos al flujo de conversación.

Resulta relevante analizar cuál es la motivación a un usuario, que interactúa con el bot, Para realizar una determinada tarea. Otro punto muy importante es determinar los pasos necesarios en el objetivo de cumplir las tareas, estos pasos se los realiza mediante etapas de funcionalidad en diversas plataformas. **[3]**

En la aplicación de los pasos mencionados anteriormente, se debe tomar encueta algunos aspectos que son muy importantes en el diseño de un chatbot, tales como la navegación, interrupciones, validaciones de información, ya que estos factores afectan al flujo de conversación. **[3]**

El flujo de conversación es una herramienta que simplifica los proceso que se presentan en diversas funcionalidades que dispone el chatbot.

1.2.2.7 Descripción del Flujo de Trabajo de un Chatbot

Un agente conversacional no es lo mismo que una aplicación tradicional ya que el flujo de trabajo es complemente diferente. En las aplicaciones tradicionales su interfaz de usuario consta de un conjunto de pantallas para el intercambio de información con el usuario. Por lo general en la mayoría de las aplicaciones web o de escritorio disponen de una pantalla principal, dicha pantalla tiene algunas características de navegación con el objetivo de que los usuarios puedan realizar varias tareas. **[11]**

La interfaz de usuario de los chatbots está compuesta por un conjunto de diálogos. Estos diálogos cumplen la función de administrar el flujo de conversación y posteriormente ayudan a comprender las actividades que los usuarios realicen. **[11]**

Los diálogos son un aparte del desarrollo de chatbots muy esencial, ya que estos permiten la separación lógica de las funcionalidades de un agente y guiar el flujo de conversación. Además, los diálogos pueden tener interfaces gráficas como formularios, botones, imágenes, texto entre otros elementos más. Una función muy importante de los diálogos es la de invocar a otros diálogos o procesar la información ingresada por el usuario. **[11]**

En la siguiente figura 1.9, se visualiza la comparación entre el flujo de trabajo entre una aplicación a tradicional y un bot.

En las aplicaciones tradicionales, la interacción con el usuario empieza desde una interfaz inicial, esta interfaz posteriormente abre una nueva interfaz con la finalidad de realizar una nueva petición. Esta nueva interfaz permanece en actividad hasta que

se cierre o se invoque a otras interfaces. En caso de que se cierre la nueva interfaz, se redirecciona a la interfaz principal. [17, 3]

Por otro lado, en un bot, toda la interacción empieza con un dialogo raíz, posteriormente este dialogo raíz invoca a un nuevo dialogo solicitado. Si esta acción se presenta automáticamente el nuevo dialogo solicitado tomo el control de la conversación y permanecerá como dialogo principal hasta que se cierre la acción o se invoque otros diálogos. En el caso de que se cierre el diálogo nuevo solicitado, se redirecciona automáticamente al dialogo raíz. [17, 3]

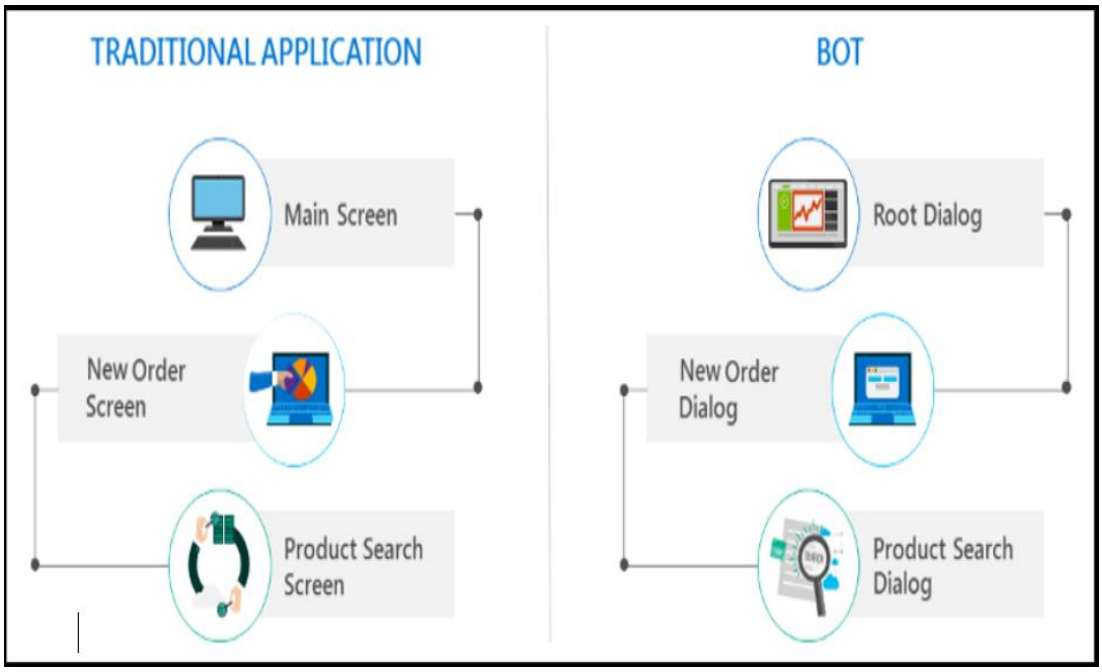


Figura 1.9: Comparación del Flujo de Trabajo entre una Aplicación Tradicional y un Bot.
Fuente: [17, 3].

1.2.2.8 Flujo de Conversación de un Chatbot

La interacción con un agente virtual más conocido como Chatbot, por lo general, se basa en una tarea específica que en futuro el agente intente cumplir. Para esto el chatbot recopila información que será procesada ordenadamente, esta acción se la denomina como flujo de procedimientos. Por otra parte, el flujo de trabajo del chatbot está definido por un flujo de conversación de procedimientos. [17, 3]

El Flujo de Conversación de procedimientos es un grupo de diálogos, cada dialogo perteneciente a este grupo tiene un propósito en común, esto significa que cada dialogo tiene una tarea específica para cumplir con el objetivo en común. En este flujo de procedimientos el desarrollador define concisamente un orden que debe cumplir los diálogos, después el bot se encarga de mantener una conversación respetando el orden que se les asignó a los diálogos. [17, 3]

En la siguiente figura 1.10, se visualiza un Flujo de Conversación de procedimientos, mediante un flujograma. El flujograma está estructurado con un conjunto de hilos de conversación que después se convierte en un dialogo.

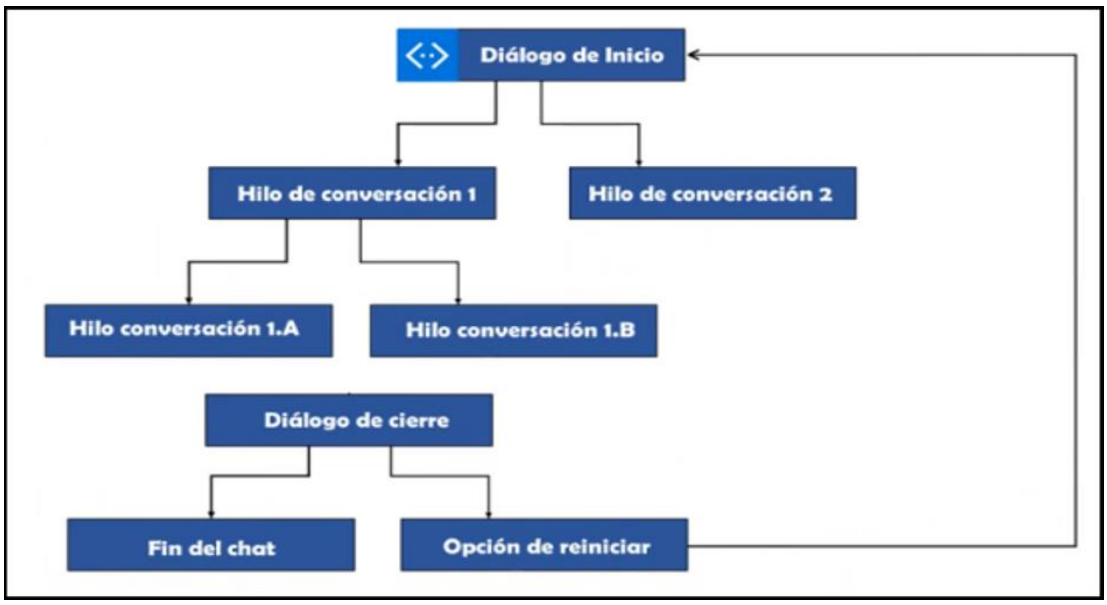


Figura 1.10: Flujograma Conversacional de Procedimientos.
Fuente: **[3]**.

En el análisis del Flujo de Conversación el dialogo invoca a otro. El bot dispone de un generador de diálogos, estos nuevos diálogos son colocados dentro de una pila. El ultimo dialogo de esta lista es el encargado del control de la conversación, también procesa cada mensaje que es interesado por el usuario, esta acción se la realiza varias veces hasta que se cierre o se agregue un nuevo diálogo. Los diálogos que no interactúan en ninguna acción automáticamente van eliminándose de la lista que contiene la pila, cuando sucede esta acción el dialogo anterior de la lista asume el control total de la conversación. El flujo de conversación del bot dispone de un diseño que se basa en comprender el uso de la pila de diálogos. [17, 3]

Para el diseño del flujo de procedimientos, se debe tener en cuenta la secuencia de pasos que un usuario debe seguir con la finalidad de cumplir a cabalidad una tarea. La secuencia de pasos no siempre será lineal u ordenada, ya que los usuarios tienden a cambiar de opinión. Para esto se debe llevar un control exhaustivo de las interrupciones que se pueden presentar durante la conversación. [17, 3]

- Para el control de interrupciones se debe considerar puntos muy importantes, los cuales deben basarse en la siguiente pregunta: ¿Como debe responder el bot en caso de presentarse una interrupción?
- Insistir hasta que el usuario responda corredtamente a la pregunta planteada por el bot.
- Omitir el flujo de conversacion que el bot a mantenido hasta el momento con el usuario y reiniciar la pila de dialogos, es decir, empezar de nuevo e intentar procesar la informacion ingresada por el usuario.
- Tratar de procesar la infiormacion que ingresa el usuario e intentar retomar el hilo de conversación con el objetivo de finalizarr la tarea del dialogo.

Estos puntos son importantes al momento de llevar un control de los diálogos, todo depende del escenario que se presente el bot. **[3]**

1.2.3 Procesamiento de Lenguaje Natural (PLN)

Es aquella habilidad que tiene una máquina para poder procesar la información recibida en letras o sonidos del lenguaje humano. El PLN utiliza el lenguaje natural para comunicarse entre los usuarios y las computadoras, este proceso debe comprender las oraciones que se le proporcionara. **[3]**

1.2.3.1 Arquitectura de un Sistema PLN

La arquitectura de un sistema de procesamiento de lenguaje natural se encuentra sustentada por niveles: **[1]**

- **Nivel Fonológico:** este nivel trata sobre el comportamiento de las palabras en relación con los sonidos que se presentan.
- **Nivel Morfológico:** trata acerca de la construcción a partir de unidades de significado.
- **Nivel Sintáctico:** este nivel habla acerca de cómo las palabras se fusionan para poder conformar oraciones y definiendo el orden de cada palabra en la oración.
- **Nivel Semántico:** aquí se analiza los significados de las palabras adquiridas. Además, del significado de cada palabra unida con el objetivo de dar coherencia a una oración.
- **Nivel Pragmático:** este último nivel trata acerca del uso de las oraciones en diversas situaciones y de cómo afecta el uso de cada oración en relación a oraciones anteriores.

En la presenta figura 1.11, se observa que la entrada de información es una expresión en lenguaje natural, esta entrada será sometida a un proceso de análisis e interpretación por el procesador de lenguaje natural. **[1]**

Figura 1.11: Arquitectura de un Sistema de Procesamiento de Lenguaje Natural. Fuente: ***[3]***

El sistema funciona de la siguiente manera. Una vez obtenido la información, se analiza el contenido de la información en forma morfológica y sintáctico, durante el análisis el sistema utiliza una Técnica Lexicográfica conocido como scanner sintáctico denominado "parser". El scanner realiza un barrido de componentes léxicos definidos a priori y el parser verifica el orden gramatical entre los elementos escaneados. **[3]**

A continuacion, el sistema realiza un alanalisis semantico de las oraciones con la finalidad de obtener el significado de cada oracion, despues se le asigna a cada oracion una expresion logica que debe ser falsa o verdadera (true, false). **[3]**

Finalmente, el sistema realiza un analisis pragmatico de dicha información adquirida, el analisis consiste en agruprar las oraciones que ya fueran verificadas y va comparando cada oracion según su situacion. Como resultado de este analisis se obtiene una expresion resultante, la cuyal dseria remitida al usuario. **[3]**

1.2.3.2 Aplicaciones del Procesador de Lenguaje Natural

Un sistema de procesamiento de lenguaje natural se puede aplicar en diversos campos gracias a su función de combinar entre otras aplicaciones con inteligencia artificial para construir aplicaciones potentes a nivel cognitivo, entre las importantes son **[3]**:

- Asistentes inteligentes.
- Agentes Conversacionales (chatbots).

- Respuestas automáticas.
- Análisis de sentimientos.
- Reconocimiento por comandos de voz.
- Traducción automática.
- Extracción de información.

1.2.4 Prototipo

Un prototipo es una representación de cualquier sistema convencional que posee algunas características del sistema final. Además, se puede definir como una representación limitada de un producto permitiendo ser puesto a prueba en situaciones reales o explicativas. **[12]**

En el área del desarrollo de software un prototipo es un sistema funcional a pequeña escala es decir no entra en proceso de producción.

Entre sus ventajas primordiales están: el rápido desarrollo y bajo costo económico del mismo. Además, un prototipo se lo desarrolla de forma iterativa hasta llegar a la versión definitiva y deseada. **[3]**

Por lo general un prototipo no consta con seguridad y captura de errores de programación ya que eso se lo realiza posteriormente, esta versión de prototipo se centra en la parte funcional que es la que afecta al usuario. **[3]**

La utilidad de un prototipo es el de poder evaluar un producto desarrollado, ya que permite clarificar los requisitos de los usuarios. Además, brinda la posibilidad de poder discutir y cimentar ideas entre desarrolladores, diseñadores y partes responsables del desarrollo del producto. **[3]**

1.2.4.1 Tipos de Prototipos

En la actualidad existen varios tipos de prototipo dependiendo el uso que se dé, entre los principales encontramos **[12]**:

- **Modelo rápido**: utiliza una metodología de diseño mediante la cual se puede desarrollar nuevos diseños, una vez finalizado el nuevo diseño, este será sometido a evaluación.
- **Modelo reutilizable:** conocido como evolutivo, las partes usadas durante la construcción del prototipo son reutilizados para la construcción de la versión final del producto.
- **Modelo modular:** también denominado como prototipo incremental, este modelo permite añadir elementos a medida que el diseño está en curso de desarrollo.
- **Modelo de baja fidelidad:** este modelo se lo implementa usando un bosquejo hecho en papel y lápiz, la función principal es la de simular el producto final sin necesidad de mostrar el aspecto real del producto final.
- **Modelo de alta fidelidad:** este modelo se diseña como una aproximación real de la versión final del producto, es decir se basa en el diseño y la interacción con el usuario tomando en cuenta el tiempo de desarrollo.

1.2.5 Inteligencia Artificial

Más conocida como Inteligencia Computacional, es aquella exhibida por maquinas, una maquina "inteligente" es un agente racional que percibe su entorno y tiene como funciones primordiales maximizar las posibilidades de éxito de cualquier objetivo o tarea establecida. **[13]**

1.2.6 Adaptive Cards (Tarjetas Adaptables)

Son fragmentos de interfaz de usuario independientes creados mediante JSON (JavaScript Object Notation), que las aplicaciones y servicios pueden intercambiar abiertamente. **[14]**

Una vez que se entrega el archivo JSON (JavaScript Object Notation) a la aplicación se transforma automáticamente en una interfaz de usuario nativa que se adapta automáticamente a su entorno. **[14]**

Los Adaptive Cards permiten incrustar formularios, botones, imágenes dentro de la estructura interna de un chatbot asiéndolo más estético y dinámico de usar. Además, se lo puede implementar en diversas plataformas ya que es multiplataforma. **[14]**

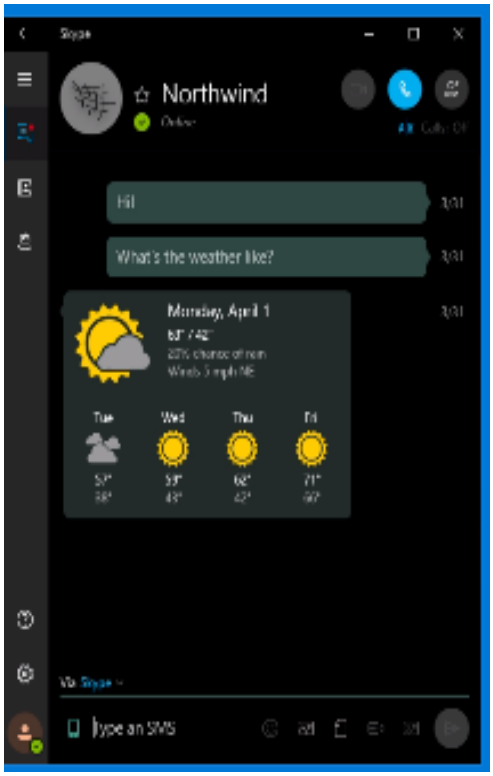


Figura 1.12: Diseño de un Adaptive Cards.
Fuente: **[14]**

1.2.6.1 Uso de Tarjetas Adaptables con Microsoft Bot Framework

Al utilizar las Tarjetas Adaptables obtenemos interfaces de usuarios autónomas a un usuario dentro de una interfaz más grande, como un cliente de chat. Esta incorpora la mayoría de funciones de las tarjetas enriquecidas de Bot Framework. Además, son compatibles con Web Chat, Cortana y Microsoft Teams. **[15]**

Una función muy importante de las tarjetas adaptables es la capacidad de renderizar sus propias aplicaciones. **[15]**

Están diseñadas para adaptarse a cualquier entorno que se utilice ya que la aplicación de host tiene un control máximo sobre la presentación. **[15]**

Los esquemas son acumulativos esto quiere decir de cada esquema se puede utilizar para cada tarjeta de una versión inferior ya que los esquemas son explícitos sobre la versión en la que se introdujeron. **[15]**

Al momento de especificar un esquema para una tarjeta adaptable también se especifica una ruta especifica de la versión. **[15]**

Figura 1.13: Ruta Especifica de un Adaptive Cards.
Fuente: **[15]**.

1.2.6.2 Paquetes

Las Tarjetas Adaptables se presentan como cadenas JSON y envían el contenido mediante el protocolo HTTP (Protocolo de Transferencia de Hipertexto) a la API REST (Transferencia de Estado Representacional) de Bot Framework. Estos paquetes facilitan el desarrollo de bots ya que proporcionan clases que ayudan a construir y manipular las tarjetas adaptables. **[15]**

Como funciona una Adaptive Cards internamente: Al contar con la propiedad de intercambiar tarjetas abiertas permiten los desarrolladores intercambiar contenido desde la interfaz de usuario de forma más habitual y coherente.

Los desarrolladores son encargados de crear el contenido dentro de un objeto JSON, el cual se representa dentro de una aplicación cliente, Además, con Adaptive Cards se puede diseñar la estructura de las vistas en tiempo real. **[15]**

1.2.6.3 Características generales de los Adaptive Cards

- Son multiplataforma.
- Las bibliotecas y el esquema son de código abierto y uso compartido.
- Están dirigidas a grandes cantidades de contenido que los desarrolladores producen.
- No necesitan permiso de edición de ningún código. **[15]**

1.2.7 Bot Framework

Es aquella tecnología que permite construir, conectar, probar y distribuir chatbots potentes e inteligentes. Además, esta tecnología permite construir bots que soporten diferentes tipos de interacciones con los usuarios. **[16]**

1.2.7.1 La Seguridad con Bot Framework

Para asegurar la conectividad de Bot Connector y el bot mediante canales o servicios, se lo realiza mediante una configuración avanzada en https o a través de algún canal seguro que utilice autenticación. Dicha configuración se la realiza en el momento de registrar el bot en la plataforma Bot Framework. Bot Connector determina un token, este permite acceder a la cabecera de cada solicitud. El acceso mediante el token asegura la comunicación con el bot. **[1]**

Para establecer la conexión entre el bot y Bot Connector se utilizan las tecnologías de autenticación que se observa en la tabla 1.1.

TECNOLOGÍA	DESCRIPCIÓN
SSL/TLS	SSL/TLS se usa para todas las conexiones de servicio a servicio. Los certificados se usan para establecer la identidad de todos los servicios HTTPS.
Oauth 2.0	El inicio de sesión de Oauth 2.0 en el servicio de inicio de sesión de cuenta de Microsoft (MSA)/AAD v2 se utiliza para generar un token seguro que un bot puede usar para enviar mensajes.
Token web JSON (JWT)	Los tokens web JSON se utilizan para codificar tokens que se envían desde y hacia el bot.
Metadatos OpenID	El servicio Bot Conector publica un alista de tokens validos que utiliza para firmar sus propios tokens JWT a los metadatos OpenID en un punto final estático bien conocido.

Tabla 1.1: Tecnologías para la Autenticación que utiliza Bot Framework.
Fuente: **[1]**.

1.2.7.2 Funcionamiento de Bot Framework

En la figura 1.14, se visualiza la forma de comunicación de Bot Framework de forma general.

Los elementos que conforman el diagrama de la comunicación de Bot Framework, son los siguientes:

Usuario: es la persona que ocupa el rol de cliente, que va interactuar con el bot mediante una conversación (texto, grafico, voz).

Canal: es la plataforma o aplicación que sirve como intermediaria entre el cliente y el bot, aquí en bot se desplegará.

Bot: es aquella aplicación que contendrá un conjunto de funciones y servicios programados, los cuales se pondrán a disposición de los clientes a través de una conversación.

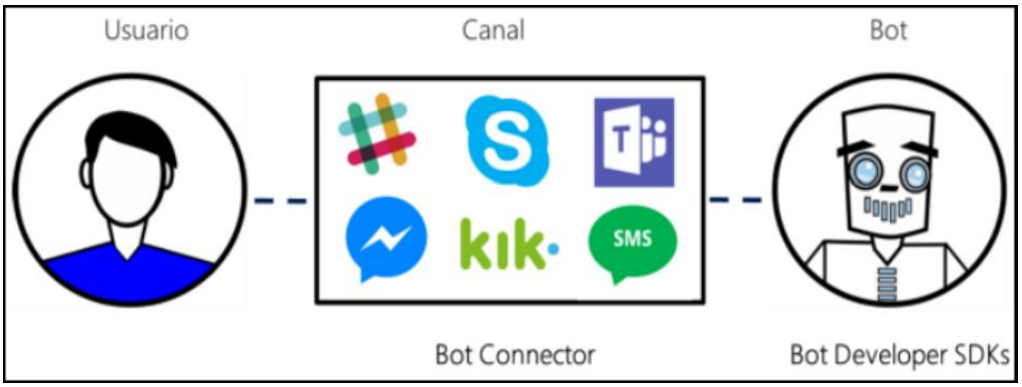


Figura 1.14: Forma General de la Comunicación de Bot Framework.
Fuente: ***[3]***.

El enlace de comunicación entre el bot y los canales de información es un servicio online denominado Bot Connector. **[1]**

1.2.7.3 Arquitectura de Bot Framework

La Arquitectura de Bot Framework está conformada por dos componentes fundamentales el Bot Builder SDK y el Bot Connector, los canales de información no son tomados en cuenta como parte interna de la arquitectura, ya que son independientes de Bot Framework.

En la siguiente figura 1.15, se visualiza la Arquitectura de Bot Framework:

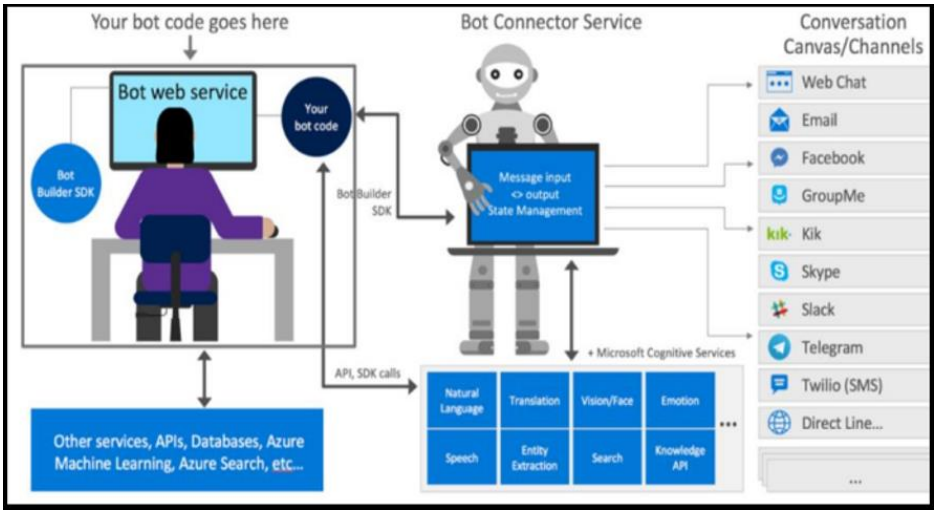


Figura 1.15: Arquitectura de Bot Framework.
Fuente: **[1]**

La figura 1.15, está conformada por tres bloques. El primer bloque contiene un conjunto de canales de conversación, los cuales intervienen las entradas y salidas de dialogo entre el bot y el cliente. **[1]**

El segundo bloque contiene el Bot Connector, el cual se encarga del envío y recepción de mensajes entre el bot y cliente, Además puede conectarse con servicios cognitivos de Inteligencia Artificial y recopilar información sobre el estado del servicio. **[1]**

La comunicación entre Bot Web Service, Bot Connector y los canales de conversación que utiliza el cliente se da mediante un servicio REST, esto significa que toda la información intercambiada entre el cliente y el bot será mediante un objeto JSON (acrónimo de JavaScript Object Notation). Este funcionamiento permite que la comunicación fluya sin importar el canal que se esté desplegando el bot. **[1]**

En el tercer bloque se visualiza el código del bot, básicamente la funcionalidad y lógica que será programada al bot. Todo el código que se encuentra en este bloque se conecta con el Bot Builder SDK (librerías para gestionar las conversaciones). El código del bot permite conectarse con diversos servicios, ya sean propios de una empresa o pertenecientes de Azure. **[1]**

1.2.7.4 Descripción de Bot Framework para el Desarrollo de Chatbots

La tecnología Bot Framework es software libre, desarrollada para la construcción de bots, ofrece servicios cognitivos de Microsoft mediante la creación canales de información. Estos canales sirven como puente entre tecnologías que se utilice en el diseño de agentes virtuales, por ejemplo, al momento de dotar de inteligencia artificial a un bot a través de servicios de Microsoft Azure o simplemente mejorar la apariencia de las interfaces del bot con el uso de Adaptive Cards. Una ventaja muy importante de los canales de información, es brindar soporte oportuno a los usuarios que interactúen con el bot, gracias a la facilidad de integración con otras aplicaciones (web o móviles) de mensajería (Facebook, Messenger y Skype). **[17]**

Bot Framework dispone de ventajas importantes para el desarrollo de bots inteligentes, ya que proporciona los elementos necesarios tanto para la construcción y conexión de bots con diversas plataformas. Además, una ventaja primordial de Bot Framework es permitir incorporar la lógica de negocio dependiendo del escenario que se presente. **[17]**

Este Framework también proporciona un entorno integrado, permitiendo construir, probar, implementar y administrar de manera óptima agentes conversacionales potentes, Además cuenta con soporte para .Net y Node.js a través de su SDK (Software Development Kit) de Bot Builder. **[1]**

1.2.8 Bot Builder SKD (Software Development Kit)

Es el ambiente de desarrollo donde se construyen bots inteligentes desde cero. Además, suministra bibliotecas y herramientas para la construcción de bots inteligentes. También permite la incorporación de diálogos integrados, los cuales son esenciales en el manejo de interacciones con el cliente. **[16]**

Es aquel que proporciona un SDK (Software Development Kit), completo para las plataformas .NET y Node.js **[16]**

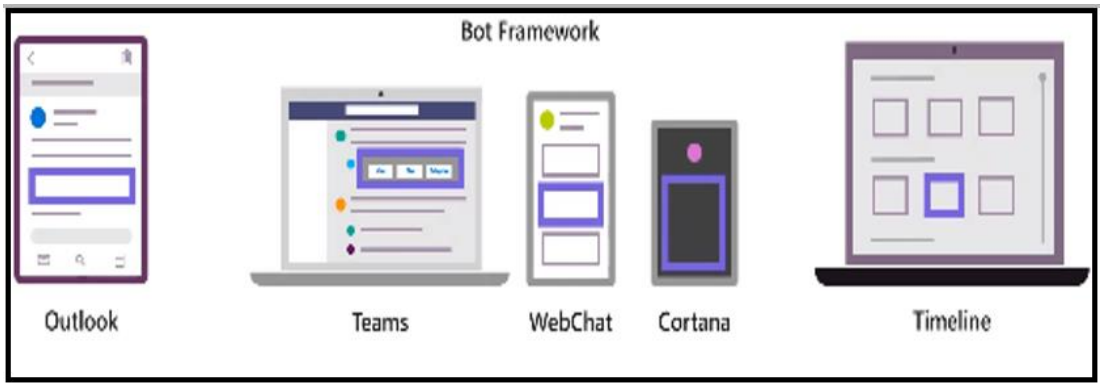


Figura 1.16: Aplicaciones Desarrolladas con Bot Framework.
Fuente: **[16]**

1.2.9 Bot Connector

Es el puente de enlace entre un bot y uno o más canales de información, mediante una API REST (Representational State Transfer), provocando que el esquema de mensajería sea el mismo sin importar el canal que se esté inutilizando. [1, 3]. Además, el Bot Connector puede conectarse con servicios de Inteligencia Artificial con el fin de almacenar el estado de una conversación. Una función primordial que tiene el Bot Connector es la de enlazar clientes que no tengan el mismo idioma y recopilar información del funcionamiento del bot. **[3]**

Este servicio se comunica mediante comunicación REST, el intercambio de información se lo realiza mediante un formato JSON. Esta funcionalidad permite que la comunicación sea independiente del canal o plataforma donde se está desplegando el bot. **[3]**

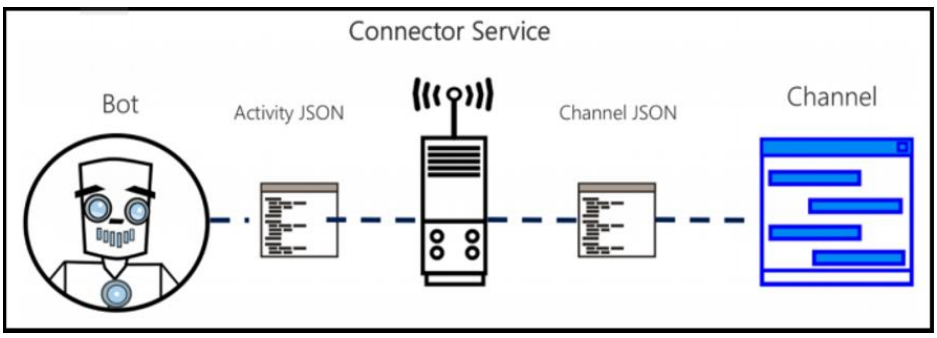


Figura 1.17: Funcionamiento de Bot Connector.
Fuente: **[3]**.

Es importante tener en cuenta que el despliegue de un bot en diversas plataformas, se lo puede realizar en Azure Bot Service, el cual contiene el Bot Connector. Esto no es obligatorio ya que también se lo puede realizar en cualquier otro servicio o de forma local. **[3]**

2.2.10 Metodologías de Desarrollo de Software

Denominadas Metodologías pesadas ya que centran su atención en llevar un registro de la documentación exclusiva acerca dl desarrollo del proyecto con la finalidad de cumplir un plan de proyecto. **[7]**

Estas Metodologías se basan en una disciplina de trabajo para poder sustentar los tiempos establecidos durante el desarrollo del proyecto y también garantizar un software eficiente. **[7]**

Estas metodologías deben cumplir cuatro actividades obligatoriamente que son esenciales durante la construcción del software. **[3]**

- **Especificación de Software:** aquí se define la funcionalidad del software como las restricciones de operación.
- **Diseño e Implementación del Software:** en esta actividad ya se debe desarrollar el software y poder cumplir con las especificaciones establecidas.
- **Validación de Software:** Se debe validar el software con el objetivo de verificar si cumple los requerimientos del cliente.
- **Evolución de Software:** de forma obligatoria el software debe evolucionar y poder satisfacer las necesidades o cambios que solicito el cliente.

1.2.11 Componentes de una Metodología de Desarrollo de Software

En la presente figura 1.18, se visualiza los elementos básicos que conforman la estructura interna de una metodología.

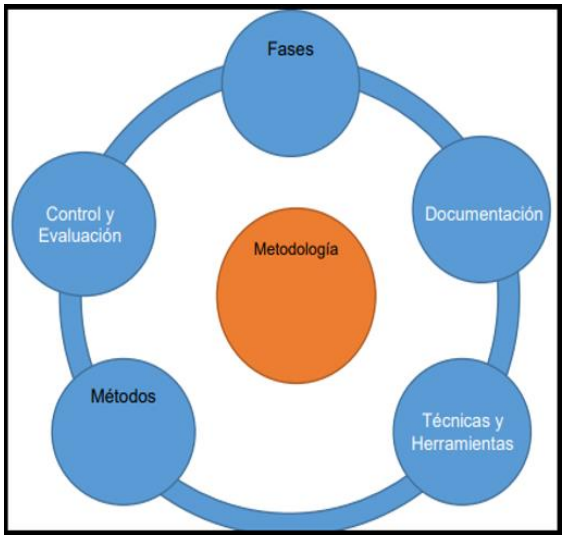


Figura 1.18: Elementos de una Metodología de Desarrollo de Software.
Fuente: **[3]**

1. **Las Fases:** son un conjunto de actividades que se relacionan con un objeto durante el desarrollo del proyecto. **[7]**
2. **Los Métodos:** aquí se definen los procesos y la manera de como se debe seguir con el desarrollo con la finalidad de finalizar el producto. Además, este componente permite la descomposición de los procesos en un conjunto de tareas mas pequeñas y de la definicon de entregables y tecnicas generados para cada fase. **[7]**
3. **Técnicas y Herramientas:** este componente representa la forma de resolver cada actividad y que herramienta se podría implementar. Entre las técnicas más importantes tenemos. **[18]**
    - Recolección de datos: entrevistas, formularios, etc.
    - Técnicas graficas: organigramas, matrices, diagramas, etc.
    - Técnicas de modelado: Estructuración del desarrollo y orientación de objetos.
4. **Documentación:** aquí se define toda la documentación que corresponde a cada fase, dicha documentación se la realiza de forma completa y concreta, siempre

tomando en cuenta los valores que se generaron, esto sirve para poder agrupar los resultados, a partir de estos poder tomar decisiones. **[7]**

5. **Control y Evaluación**: estas actividades se las realiza a lo largo de cada fase con el objetivo de poder identificar errores y corregirlos a tiempo. Es decir, realizar un seguimiento del avance en base al cronograma de trabajo establecido desde un inicio. **[7]**

1.2.12 Clasificación de las Metodologías de Desarrollo de Software

Primeramente, es importantes conocer la definición de cada metodología y desarrollo posterior a esto podemos clasificarlas dependiendo el enfoque que se presenta cada una de ellas, una de las clasificaciones más aceptadas es por el tipo de filosofía de desarrollo.

Según este tipo de metodología se subdividen en dos grupos:

1.2.13 Metodologías Tradicionales

Estas metodologías son conocidas como modelo de proceso prescriptivo y son utilizadas especialmente para establecer un orden cuando se presenta algún caos de desarrollo de software. Además, este tipo de metodologías brindan una estructura útil al trabajo de IS (ingeniería de software) y contribuyen facilitando un mapa eficaz para equipos de software, estas son [3, 7, 9] :

- RUP (Relacional Unified Process).
- MSF (Microsoft Solución Framework).
- Win-Win Spiral Model.
- Iconix.

1.2.14 Metodologías Ágiles

Este tipo de metodología aparece como una respuesta a los problemas que puedan generar las Metodologías Tradicionales. Además, adaptables a los procesos del desarrollo de software. **[19]**

El desarrollo Ágil referencia a métodos de IS (Ingeniería de Software) basado en el desarrollo iterativo e incremental, la implementación de estas metodologías es perfectas para un mundo donde nos exponemos a cambios diarios. Además, debemos acotar que estas metodologías son tendencia en el campo del desarrollo de software.

En la actualidad, las empresas operan en un entorno variante cada día, por lo que afrontan nuevas oportunidades, cambios económicos, mercado, surgimiento de productos y servicios nuevos. A causa de estos factores es necesario emplear computadoras y dispositivos computacionales, por lo que el software es participe de todas las operaciones empresariales, de tal modo que es obligatorio desarrollar de manera ágil para poder solventar con calidad todo lo que se requiera. **[19]**

1.2.14.1 Clasificación de las Metodologías Ágiles

Entre las principales Metodologías Ágiles encontramos:

- XP (Programación extrema), una de las más exitosas y emergentes.
- Mobile-D (ágil y perfecta para móviles).
- Scrum.
- Cristal.
- EVO (Evolutionary Project Management).
- FDD (Feature Driven Development).
- ASD (Adaptive Software Development.
- Lean Development.

1.2.15 Diferencias entre Metodologías Ágiles y Tradicionales

En la tabla 1.2, se evidencia las principales diferencias entre ambas metodologías. Estas diferencias son muy importantes ya que afectan directamente al proceso como a la organización de los equipos responsables del desarrollo. **[3]**

Metodologías Ágiles	**Metodologías Tradicionales**
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo).	Impuestas externamente.
Proceso menos controlado, con pocos principios.	Proceso muchos más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Mas roles-
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

Tabla 1.2: Diferencias entre Metodologías Ágiles y Tradicionales.
Fuente: Elaborado por el investigador y basado en **[3]**.

1.2.16 Arquitectura de Software

La Arquitectura de Software representa una vista general del sistema, está conformada por componentes. Además de la relación entre ellos, el ambiente y principios que orientan su diseño y evolución. **[3]**

No existe una definición estandarizada de Arquitectura de software, por lo que se define como aquella que abarca lo relativo de la estructura de alto nivel de un sistema. **[20]**

La Arquitectura de Software tiene relación con el diseño general del sistema, por lo que se manifiesta en fases iniciales del proceso del desarrollo de cualquier sistema. **[20]**

Esta se encarga de los componentes mientras que el diseño de los procedimientos. A medida que la arquitectura de nivel alto se optimiza, sus puntos de conexión pierden de forma substancial el grado de abstracción, distribuyendo de sus elementos arquitectónicos de nivel bajo, esto ocasiona la transformación de la arquitectura en diseño. **[20]**

La Arquitectura de Software representa los aspectos del Software con la finalidad de describir todos los aspectos que utilizan los modelos y vistas. **[3]**

Las vistas formulan uno o varios lenguajes, por ejemplo, al utilizar: lenguaje natural, diagramas de flujo, diagramas de estado, entre otros. **[3]**

Durante el diseño de la arquitectura se usan modelos estructurados, los cuales representa una colección ordenada de un conjunto de componentes de programa. Dichos modelos incrementan su nivel de abstracción lo cual permite identificar diseños arquitectónicos repetidos, estos diseños replicados son denominados patrones que se encuentran en aplicaciones de similares características. **[3]**

1.2.16.1 Patrones

Conocidos como bloques de construcción mental útiles en el diseño limitado. Son específicos durante el desarrollo del software, están basados en la reutilización de promover exentes prácticas de diseño. **[3]**

Son considerados como una disciplina para resolver problemas en la ingeniería de software, los patrones han tenido un mayor impacto en la programación orientada a objetos. Además, pueden ser utilizados en cualquier ámbito de la informática. **[3]**

Como se visualiza en la figura 1.19, trata de un Patrón con un esquema conformado por sus respectivas partes. **[3]**

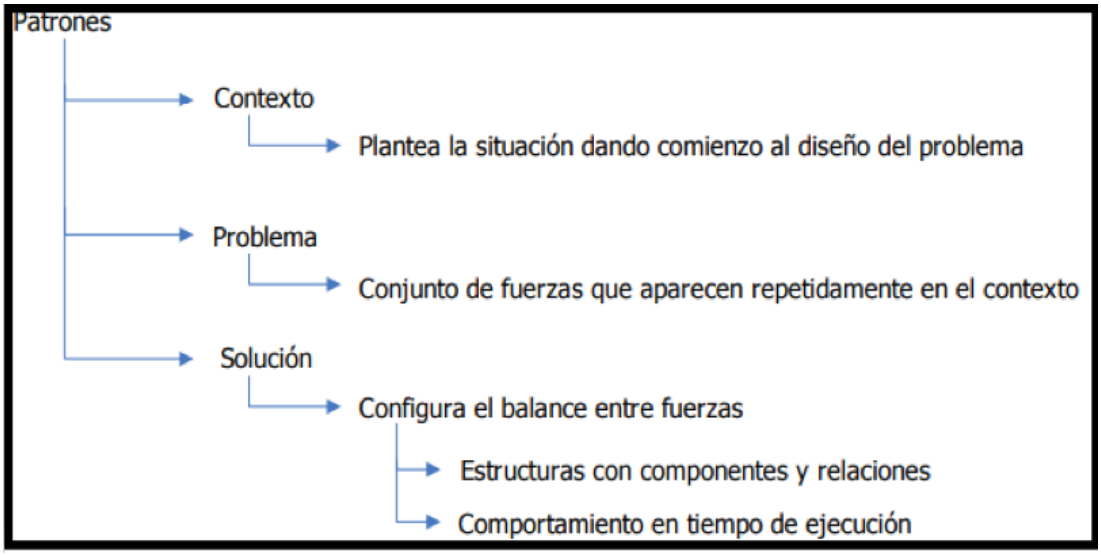


Figura 1.19: Esquema del Patrón.
Fuente: **[3]**

El esquema del Patrón representa una regla que establece una relación entre un contexto inicial, un cierto problema que tiene lugar en ese contexto actual y una solución apropiada al problema propuesto. **[20]**

- **Contexto:** denota la situación actual en la que ocurre el problema. Es difícil especificar el contexto de un patrón, esto implica que resulta casi imposible determinar las situaciones en las que deben aplicar. **[20]**
- **Problema:** es la situación problemática que se presenta en el contexto actual. Genera una especificación con la finalidad de establecer el problema y la posible solución. **[20]**

- **Solución:** establece la forma de cómo se resuelve un problema recurrente o como balancear las fuerzas que interactúan con el problema dado. **[20]**

Categoría de Patrones

Estos se agrupan dependiendo de la función de un rango de abstracción durante las tres categorías: patrones arquitectónicos, patrones de diseño e idiomas. **[3]**

Patrones Arquitectónicos

Este tipo de Patrones representan el nivel más alto del Sistema de Patrones. Además, expresan la estructura fundamental de la organización para sistemas de software. Expresan un conjunto de subsistemas predefinidos. **[3]**

Este tipo de Patrón se clasifica en las siguientes categorías:

- **Del Fango a la Estructura:** evitan el exceso de componentes u objetos. Simplificando las tareas del sistema global en subtareas más pequeñas. **[3]**
  Dentro de esta categoría encontramos los patrones: Filters, Pipes, Layers y Blackboard. **[3]**
- **Sistemas Distribuidos:** definen aquellos patrones que se utilizaran en el desarrollo de sistemas distribuidos. En esta categoría encontramos: Pipes, Broker, Filters y Microkernel. **[3]**
- **Sistemas Interactivos:** asignan los patrones que ayudan a la estructura del sistema de software, ofreciendo una interacción entre usuario y computador. Dentro de esta categoría se encuentran los siguientes patrones: PAC (Presentation Abstraction Control), MVC (Model View Controller). **[3]**
- **Sistemas Adaptables:** son aquellos patrones que definen la estabilidad de los sistemas, son adaptables a modificaciones en los requerimientos y permite la implementación de nuevas tecnologías. En esta categoría se encuentran los siguientes patrones: Microkernel y Reflection. **[3]**

Patrones de Diseño

Asigna un esquema para depurar los componentes de un sistema de Software. Además, controla y define la forma como se relacionan los componentes dentro de un contexto particular. Su nivel de abstracción es bajo en comparación con los arquitectónicos. Son independientes de lenguaje o paradigma de programación. **[20]**

Patrón MVC (Model View Controller)

Es aquel patrón que divide una aplicación iterativa en tres componentes primordiales: modelo, vista y el controlador. **[20]**

- **Modelo:** contiene la funcionalidad principal y los datos, la parte lógica que conserva el estado de una aplicación.
- **La vista:** despliegan la información al usuario.
- **El controlador:** manejan la entrada del usuario.

En la siguiente figura 1.20, se visualiza la estructura interna de un Patrón MVC:

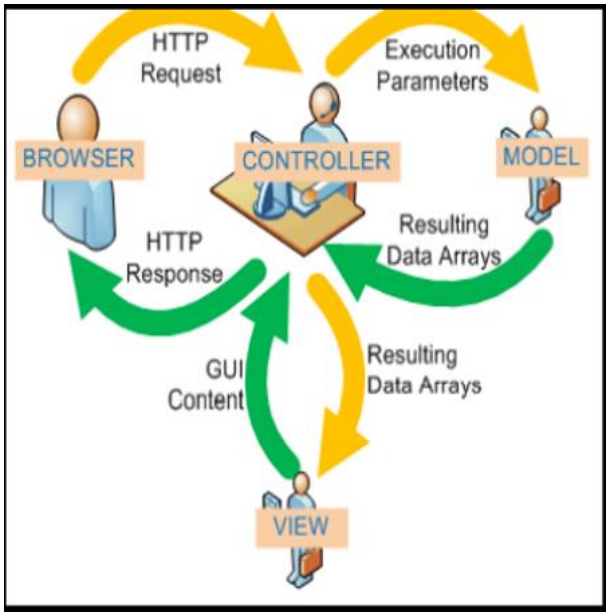


Figura 1.20: Estructura Interna del Funcionamiento del Patrón MVC.
Fuente: **[3]**.

La vista y el controlar conforman la interfaz del usuario, la consistencia de la interfaz del usuario y el modelo depende en su totalidad del mecanismo o de las tecnologías que se utilice para el desarrollo del bot. [11, 12]

El Patrón MVC en sus inicios fueron utilizados con la finalidad de construir interfaces de usuarios en Smalltalk80 mediante los laboratorios de investigación Xerox y su estilo fue consolidado por primera vez en el año 1979 por Trygve Reenskaug. **[21]**

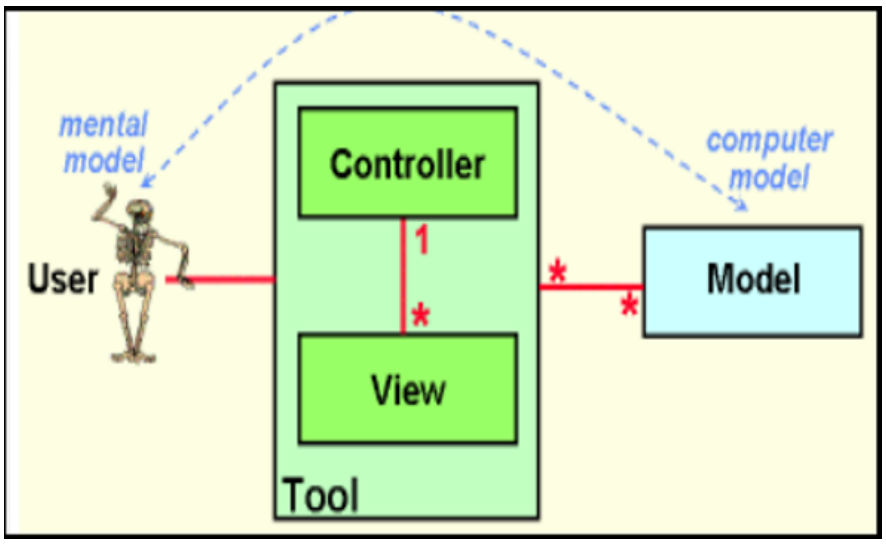


Figura 1.21: Primer Diseño del Patrón MVC.
Fuente: **[21]**.

El Patrón MVC es utilizado por la empresa Microsoft en sus módulos para la creación de aplicaciones web y API (Application Programming Interfaces). **[3]**

Bot Framework es resultado de una API de mensajería específicamente para el desarrollo de bots, por lo que es obligatorio investigar el funcionamiento más a fondo de este patrón. **[3]**

Una ventaja muy importante de este tipo de Patrón es la de separar el Software en función del tipo de trabajo que se realizara, es decir, separa la parte lógica de negocios, de la parte lógica de la interfaz y de la infraestructura. **[3]**

1.2.17 Framework

Es un diseño reutilizable del sistema completo o talvez de alguna de las partes, se expresa mediante un conjunto de clases abstractas y aquella forma de interactuar con las instancias. **[20]**

Un Framework contiene un conjunto de patrones de diseño, lo que significa que los patrones son más pequeños que el mismo framework. Además, dispone de un nivel menor de instrucción. **[3]**

1.3 Objetivos

1.3.1 Objetivo General

- Desarrollar un prototipo de chatbot para un sistema de servicios vehiculares utilizando Adaptive Cards con Bot Framework.

1.3.2 Objetivo Especifico

- Describir las técnicas actuales e idóneas para el desarrollo de un chatbot (agente conversacional).
- Establecer el flujo de trabajo de un chatbot acorde al contexto de la aplicación a desarrollar.
- Investigar el uso de las Adaptive Cards para el desarrollo de chatbots.
- Plantear la arquitectura del prototipo propuesto mediante el uso de una metodología ágil de desarrollo de software.
- Demostrar la incorporación de Adaptive Cards a los flujos de diálogo de un agente convencional mediante el desarrollo de un prototipo de chatbot para un sistema de servicios vehiculares.

CAPÍTULO II

METODOLOGÍA

2.1 Materiales

Para el presente proyecto de Investigación se utilizarán fuentes fidedignas de información, tales como: repositorios académicos, revistas científicas, sitios web, trabajos de titulación en el área de informática. Además, se pondrá en práctica el conocimiento adquirido durante el tiempo de estudio en la carrera.

Para la recolección de información se tomó como base un sistema genérico de Servicios Vehiculares, el cual servirá como eje fundamental para plantear los requerimientos para el prototipo. Además, es necesario la recepción de información referente a la arquitectura y estado actual de agentes virtuales, proveniente de repositorios virtuales internacionales que garanticen la veracidad de la información.

2.2. Métodos

2.2.1 Modalidad de Investigación

Modalidad Bibliográfica

La investigación será de tipo bibliográfica ya que se utilizará fuentes tales como libros, documentos de titulación y artículos de tipo científico, entre otros. Para la elaboración del marco teórico y posteriormente para la contextualización del prototipo propuesto.

Modalidad Aplicada

La modalidad será aplicada para este caso, ya que para la realización del prototipo de chatbot utilizando tarjetas adaptables para un sistema de servicios vehiculares será necesario aplicar los conocimientos adquiridos durante la carrera.

Modalidad Exploratoria

La investigación es de tipo exploratoria porque al tratarse de una tecnología nueva dentro del ámbito de los agentes conversacionales, se realizará una investigación de este tipo y poder incorporar los Adaptive Cards en las interfaces de usuario. Además, se analizará el comportamiento de un chatbot y la interacción con el usuario al extender y enriquecer los diálogos.

Modalidad Descriptiva

La investigación es descriptiva ya que se realizará una síntesis completa con respecto a tecnologías, ámbitos de aplicación y técnicas para el desarrollo de chatbots, por lo tanto, el desarrollo del prototipo propuesto será más óptimo. Además, se explicará el funcionamiento de un chatbot y de los Adaptive Cards.

Modalidad Explicativa

La investigación es explicativa porque mediante el uso de metodologías tanto para la construcción de agentes conversacionales como para el desarrollo de software, se planteará y describirá una arquitectura de un agente conversacional adaptado a un sistema de Servicios Vehiculares. Además, se analizará y expondrá las tecnologías y técnicas de ultima generación que permiten la construcción y despliegue bots inteligentes.

2.2.2 Recolección de Información

La información que se requiere para la realización del proyecto se receptará de bibliotecas y repositorios virtuales pertenecientes a la Universidad Técnica de Ambato. Es necesario receptar la información más relevante, acerca de la arquitectura y estado actual de los Chatbots, estos datos serán analizados y filtrados logrando discernir lo más esencial e importante para la investigación, el uso de repositorios virtuales de instituciones internacionales será de gran ayuda, ya que estos garantizan la veracidad de la información. También se utilizó como apoyo los datos pertenecientes al cuestionario de la tabla 2.3:

Preguntas Básicas	Explicación
¿Para qué?	Para desarrollo de la investigación
¿De qué personas u objetos?	Usuarios que interactúen con el agente conversacional
¿Sobre qué aspectos?	Prototipo de un sistema de servicios vehiculares
¿Quién, Quienes?	Investigador: Cristian Homero Llerena Valencia
¿Cuándo?	Periodo académico octubre 2020 – febrero 2021
¿Dónde?	En un prototipo
¿Cuántas veces?	Una
¿Qué técnicas de recolección?	Análisis de Información
¿Con qué?	Artículos investigativos, documentos de titulación, documentos en la web
¿En qué situación?	En la experiencia conversacional con los usuarios

Tabla 2.3: Recolección de Información.
Fuente: Elaborado por el investigador.

2.2.3 Procesamiento y Análisis de Datos

Procesamiento de la Información

1. Verificación critica de la información adquirida, es decir; filtrado de la información errónea o inservible para la investigación.
2. Revisión profunda de la información recolectada, es decir, determinar los modelos necesarios y correctos que ayudaran al modelo planteado.
3. Tabulación de características de cada modelo.
4. Estudio de clasificadores que apoyen el incremento de rendimiento del modelo planteado.
5. Análisis y estudio estadístico de datos para la demostración de resultados.

Análisis del Resultado

1. Análisis de resultados estadísticos, valorando el nivel de aprendizaje obtenido.
2. Interpretación de los resultados obtenidos, en base al marco teórico.
3. Planteamiento de conclusiones y recomendaciones.

CAPÍTULO III

RESULTADOS Y DISCUCIÓN

3.1 Desarrollo de la Propuesta

3.1.1 Metodología de Desarrollo

3.1.1.1 Comparativa de Metodologías de Desarrollo de Software

Previo al desarrollo del Chatbot se necesita el uso de una Metodología acorde a los objetivos planteados al inicio de la investigación. Una vez realizado el análisis correspondiente de las diferencias entre metodologías para el desarrollo de software, las mismas que se visualizan en la tabla 3.4, se decide utilizar una Metodología Ágil, ya que dicha metodología se ajuste a la construcción del prototipo.

	Crystal	**DSDM**	**EDD**	**Scrum**	**XP**
Sistema como algo cambiante	4	3	3	5	5
Colaboración	5	4	4	5	5
Características de la metodología (CM)					
-Resultados	5	4	4	5	5
-Simplicidad	4	3	5	5	5
-Adaptabilidad	5	3	3	4	3
-Excelencia técnica	3	4	4	3	4
-Prácticas de colaboración	5	4	3	4	5
Media CM	4.4	3.6	3.8	4.2	4.4
Media Total	4.5	3.6	3.6	4.7	4.8

Tabla 3.4: Comparativa de metodologías ágiles.
Fuente: Elaborado por el investigador y basado en **[3]**.

Antes de tomar una decisión definitiva acerca de la metodología que se utilizará previo al desarrollo del prototipo, se aplicará un análisis a fondo de las características más importantes de las metodologías ágiles actuales, con el objetivo de seleccionar la más acorde a la naturaleza del proyecto.

Entre las metodologías ágiles más importantes encontramos:

Metodología XP (Extreme Programming)

Es una metodología ágil que se basa específicamente en un conjunto de reglas y principios que se aplican durante el desarrollo de Software, con la finalidad de crear un proceso ágil, que se centre en la asignación de tareas con su respectivo valor y simplifique los procedimientos que ocasionan burocracia en el mismo. **[22]** En la figura 3.22, se visualiza el marco de trabajo de la Metodología XP.

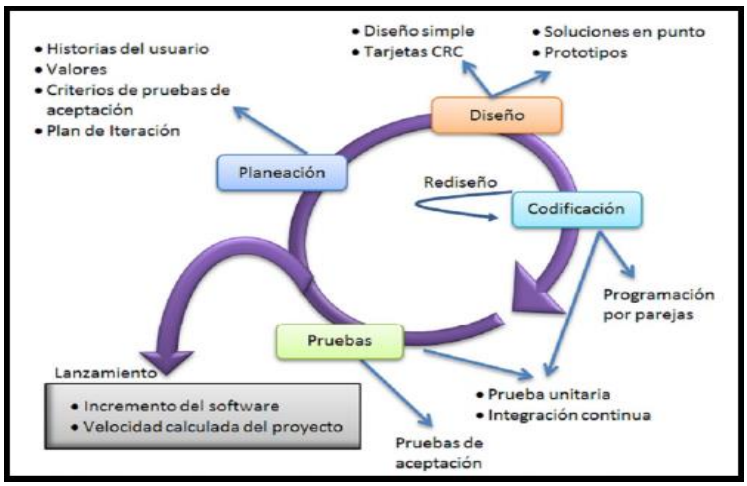


Figura 3.22: Marco de Trabajo de la Metodología XP.
Fuente: [22].

La programación extrema se divide en cuatro categorías primordiales, los cuales son **[22]**:

- **Retroalimentación a Escala Fina:** esta fase cuenta con diversos principios, como realizar pruebas, proceso de planificación, el cliente en el sitio y la programación en parejas.
- **Proceso continuo en lugar de por lotes:** permite una la integración continua, refactorización, esto permite evaluar el diseño del sistema a lo largo del desarrollo del proyecto y realizar codificación si lo requiere. Además, requiere que se entreguen avances pequeños.
- **Entendimiento compartido:** en esta fase se establece criterios, como construir diseños fáciles, el uso de tarjetas CRC (Clase, Responsabilidad y

Colaboración). Hace hincapié en la utilización de la metáfora del sistema o historia completa.

- **Bienestar del programador:** analiza el estado de ánimo del programador, en base al lema un programador cansado o exhausto crea código de baja calidad. Por lo tanto, esta fase recomienda que los programadores no trabajen más de 40 horas por semana y pocas horas extras.

La programación XP se basa en proceso de comunicación continuos entre el cliente y el equipo de trabajo, esto provoca que se simplifiquen las tareas asignadas a los miembros del grupo de trabajo. **Además**, se puede realizar cambios previos de una forma muy simple. Esta metodología es implementada específicamente en proyectos con requisitos imprecisos es decir que pueden sufrir cambios. **[3]**

A continuación, se explica varios conceptos importantes que utiliza la metodología XP:

Historias de Usuario

Las Historias de Usuario manejan un tratamiento muy dinámico y flexible, estas pueden destruirse o remplazarse por historias más específicas. Además, cada Historia de Usuario debe entenderse fácilmente, con el objetivo de ser implementada en un periodo corto de tiempo. **[3]**

Las Historias de Usuario por lo general son independientes de otras, en el caso de que exista alguna dependencia entre las historias ocasionan dificultades engorrosas durante la planificación y ejecución de las mismas. Una recomendación importante para evitar este problema es combinar las historias o plantear nuevas historias de forma diferente. **[3]**

Cuando una Historia de Usuario está bien desarrollada no excede en el nivel de esfuerzo de 2 o 3 personas por semana de trabajo en el proyecto. **[3]**

Planeación del Lanzamiento

Al utilizar Historias de Usuario, el cliente establece sus requerimientos y enfatiza en cada uno de ellos, a continuación, los desarrolladores analizan estos aspectos y estiman el tiempo y esfuerzo que se necesitara para desarrollar cada de forma óptima cada Historia de Usuario. **[3]**

Iteraciones

Son denominados ciclos de desarrollo, estos permiten visualizar los procesos automatizados al cliente y posteriormente recibir una retroalimentación con el fin de optimizar el nivel y la calidad del software desarrollado. **[3]**

Velocidad del Proyecto

Es una medida que se utiliza para analizar la capacidad que dispone el equipo de trabajo para terminar las historias de usuario durante una iteración. Dicha medida se calcula contabilizando el número de Historias de Usuario que se han finalizado durante una iteración. **[3]**

Programación en Parejas

La Metodología inculca el trabajo en parejas, es decir, el código será mixto usando el mismo orden establecido. Para la Metodología XP la programación en parejas optimiza la calidad del código sin ocasionar ningún problema en la fecha de entrega. **[3]**

Reuniones Diarias

El objetivo primordial de las reuniones diarias es la mantener una comunicación constante con el equipo de desarrollo, y así poder exponer los problemas y soluciones que se presenten. En dichas reuniones los participantes escuchan sugerencias que ayuden a solventar cualquier inquietud, esto ayuda para no perder más tiempo del estimado por el equipo. **Además**, estas reuniones plantean posibilidades de que sean en circulo y de pie. **[3]**

Simplicidad

Se debe tener en cuenta que el desarrollo de un diseño simple es más rápido de implementarlo que uno complejo. A causa de esto, la metodología XP propone implementar diseños simples, pero óptimos. Una sugerencia importante es la de no implementar funcionalidades que no correspondan a la iteración en la que fue desarrollada. También, el código debe ser testeable, claro, entendible y explicable. **[3]**

Metáforas

Las metáforas son términos que todos conocen y entienden, por lo que la metodología XP recomienda utilizar con el objetivo de dar a entender cuál es el propósito del proyecto. **[3]**

Solución "Spike"

Consiste principalmente el dar soluciones sencillas a problemas que se presenten, esto permite encapsular el problema aislándolo de otro tipo de problemas. **[3]**

Refactorización

Mas conocida como recodificación consiste a rescribir parte del código sin afectar la funcionalidad del mismo, la refactorización es recomendable utilizarla ya que optima el código. **[23]**

Glosario de Términos

Es un conjunto de términos técnicos, si se utiliza de forma correcta ayuda a comprender el diseño, ya que se lleva un patrón de orden en los nombres de las clases y métodos. **Además,** permite la escalabilidad de la aplicación y reutilización de código. **[24]**

Riesgos

La metodología plantea el trabajo en parejas, para reducir la frecuencia con que se presentan los problemas potenciales durante el desarrollo. **[3]**

Funcionalidad Extra

No es recomendable añadir funcionalidades extras al software, ya que implica consumo de más recursos vitales como el tiempo. **[3]**

Codificación

Una vez establecido definitivamente las historias de usuarios, posteriormente se procede a transformar estas historias en código funcional. En esta fase la participación del cliente es esencial, ya que él debe especificar enfáticamente cada historia de usuario y verificar la funcionalidad de cada una de ellas. **[3]**

Pruebas

Una ventaja importante de la Metodología XP es el uso de pruebas con el objetivo de comprobar el correcto funcionamiento del código, la metodología define puntos importantes para la realización de pruebas de funcionalidad. **[23]**

Metodología Scrum

Se basa en la teoría del control de procesos empíricos o empirismo, esta teoría se asegura que el conocimiento depende de la experiencia obtenida y las decisiones que se tomen deben ser en base a lo que se conoce. Está metodología utiliza un enfoque iterativo e incremental con el objetivo de optimizar la predictibilidad y el control del riesgo. La metodología exige la entrega de avances del desarrollo por parte del equipo de trabajo. **[22]**

Actualmente, esta metodología cuenta con tres pilares fundamentales que gestionan el control de procesos empíricos, los cuales son **[22]**:

- Transparencia.
- Inspección.
- Adaptación.

La metodología Scrum describe cuatro eventos primordiales que conforman los entregables **[22]**:

- Reunión de planificación (Sprint Planing Meeting).
- Scrum Diario (Daily Scrum).
- Revision del Sprint (Sprint Review).
- Retrospectiva del Sprint (Sprint Retrospectiva).

La división de trabajo de Scrum se centra en el término Product Backlog, significa que el tiempo de trabajo se lo realiza en bloques que pueden ser aproximados en periodos cortos entre 1 a 4 semanas, las cuales son conocidas como Sprint. **[22]**

En la siguiente figura 3.23, se visualiza el equipo de trabajo de la Metodología Scrum:

El equipo de trabajo cuenta con un grupo de profesionales desarrolladores con diversidad de competencias, con el fin de proporcionar un producto terminado (Sprint). **[22]**

Los tipos de roles que establece esta metodología son:

- **Scrum Master:** este rol lo ocupa el líder del equipo de trabajo, este proporciona soporte al equipo y a los clientes externos. Además, él es el encarado de garantizar que el equipo de trabajo aplique las teorías, prácticas y reglas que corresponde a la metodología Scrum. **[22]**
- **Scrum Owner:** es la persona responsable de trasmitir la visión del producto que se desea desarrollar, apoyado en la perspectiva de negocio. **[22]**
- **Equipo de desarrollo:** son los encargados de desarrollar el producto, mantienen un orden de trabajo horizontal con tareas asignadas en periodos de tiempo, donde cada miembro se auto-gestiona y actúa libremente en la definición y ejecución de los distintos sprints. **[22]**

A continuación, en la figura 3.23, se visualizan los roles específicos de la metodología scrum:

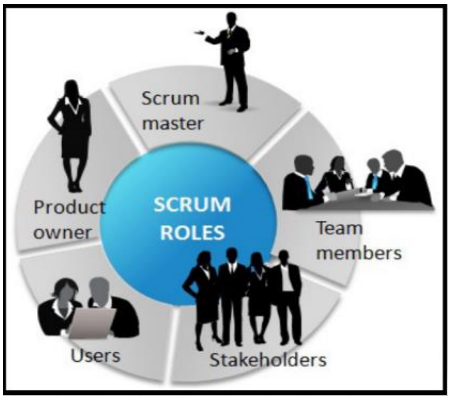


Figura 3.23: Equipo de Trabajo de la Metodología Scrum.
Fuente: **[22]**.

Crystal Methodologies

Es aquella Metodología en la cual se establece códigos de colores diferentes como parte esencial de la definición de la complejidad de la misma. La tonalidad del color se basa en la complejidad del trabajo, mientras más oscuro es la tonalidad del color, más pesado y critico es el software. Esta metodología sugiere asignar un color diferente para cada proyecto en función al nivel de complejidad y tamaño de desarrollo de los mismos. Por lo tanto, no existe una metodología Cristal en general sino una metodología Cristal asignada a cada tipo de proyecto. **[22]**

Este tipo de metodología define el desarrollo de software como un juego cooperativo de comunicación a través de los recursos que se utilicen. **[3]**

El equipo de trabajo es el elemento principal en esta metodología, ya que se invierte el esfuerzo en mejorar habilidades y destrezas. Se establece un conjunto de políticas de trabajo, las cuales dependen del tamaño del equipo de trabajo. Según el tamaño del equipo de trabajo se le asigna los siguientes términos **[3]**:

- Crystal Clear (3 a 8 participantes).
- Crystal Yellow (10 a 20 participantes).
- Crystal Orange (25 a 50 participantes).
- Crystal Red (50 a 100 participantes).

En la siguiente figura 3.24, se visualiza la representación de colores según su nivel de complejidad y tamaño de los proyectos:

Figura 3.24: Complejidad de la Metodología Crystal.
Fuente: **[22]**.

DSDM (Dynamic Systems Development Method)

Es una Metodología Ágil que tienen enfoque iterativo e incremental, que se centra principalmente en la rapidez para remitir los entregables, a diferencia de las otras metodologías de desarrollo de software esta involucra al usuario en todo el proyecto. Dicha metodología hace hincapié dos enfoques fundamentales los cuales son: el enfoque orientado y el de diseño funcional, los cuales se pueden utilizar para sistemas que no requieran la elaboración de avances. **[25]**

Esta metodología define un marco para el desarrollo de software, nace por la necesidad de crear una metodología unificada para logara un desarrollo más rápido y fluido de aplicaciones. **[3]**

La característica que la diferencia de otras metodologías de desarrollo de software es **[3]**:

- Dispone de procesos iterativos e incrementales.
- Trabajan mancomunadamente el equipo de desarrollo con el usuario interesado del producto.
- Está conformado de cinco fases:
    1. Estudio de viabilidad.
    2. Estudio del negocio.

3. Modelo funcional.
4. Diseño y construcción.
5. Implementación.

Las fases 3, 4 y 5 son iterativas, todas las fases disponen de un proceso de retroalimentación. **[3]**

La Metodología DSDM cuenta con nueve principios subyacentes, cuatro fundamentados y cinco puntos de partida. Estos principios son pilares fundamentales para el desarrollo de software, estos son **[26]**:

- **Involucrar al cliente:** esto significa que le cliente estará presente durante todo el proceso de desarrollo, es pieza clave para obtener resultados óptimos del producto final.
- **El equipo de proyecto debe tener el poder para tomar decisiones:** en este principio las decisiones son importantes para el progreso del proyecto sin la necesidad de esperar la aprobación de niveles superiores.
- **La metodología DSDM se centra en la entrega frecuente de productos:** este principio establece que mientras más temprano sea la entrega algo del producto es siempre mejor que entregar todo al final, esto ayuda en la verificación y revisión temprana con el objetivo de llevar un registro de las revisiones realizadas, esta documentación servirá como evidencia para la siguiente fase o iteración.
- **El principal criterio de aceptación de entregables en DSDM:** consiste en entregar un producto que satisfaga las necesidades actuales de negocio. Se basa principalmente en entregar un producto no muy sofisticado pero que resuelva la mayoría de las necesidades del negocio.
- **El desarrollo es iterativo e incremental:** este principio este orientado por la retroalimentación de los usuarios con el objetivo de converger en una solución de negocio exacta.
- **Cambios durante el desarrollo**: todos los cambios que se necesiten realizar durante el desarrollo son reversibles.

- **Alcance de alto nivel y requerimientos:** estos dos factores deben ser base-lined (evaluados formalmente antes de comenzar el desarrollo del proyecto).
- **Realización de pruebas:** Deben aplicarse durante todo el ciclo de vida del proyecto, se recomienda realizar este principio con el fin de evitar gastos extras de recursos en arreglos y mantenimiento del producto final luego de su entrega.
- **La comunicación y cooperación:** es un prerrequisito importante entre todas las partes involucradas en el desarrollo del proyecto, con el objetivo de obtener un producto efectivo y eficiente.

Para una aplicación correcta de esta metodología es necesario la aplicación de ciertos requisitos previos:

- Interactividad entre los usuarios y jefes de desarrollo.
- Motivación y participación entre todas las partes que integran el equipo de trabajo.
- Intercambio de criterios e ideas necesarias.

Los siguientes aspectos no son aplicables en la metodología DSDM:

- No existe aceptación por parte de la dirección hacia sus empleados.
- Falta de motivación y participación por los involucrados en el desarrollo del proyecto.
- Nivel deficiente de habilidades por parte de los integrantes del equipo de trabajo.
- Ausencia de apoyo por parte del cliente al proveedor.

En siguiente figura 3.25, se visualiza una comparativa de las principales Metodologías Ágiles para el desarrollo de Software, se ha tomado en cuenta tres aspectos importantes, La evaluación comprobara si cumplen las distintas fases del ciclo de vida de desarrollo de software, los cuales son **[27]**:

- Inicio del proyecto.
- Especificación de requisitos.
- Diseño.
- Codificación.
- Pruebas de unidad.
- Pruebas de integración.
- Pruebas de sistema.
- Pruebas de aceptación.
- Sistema en servicio.

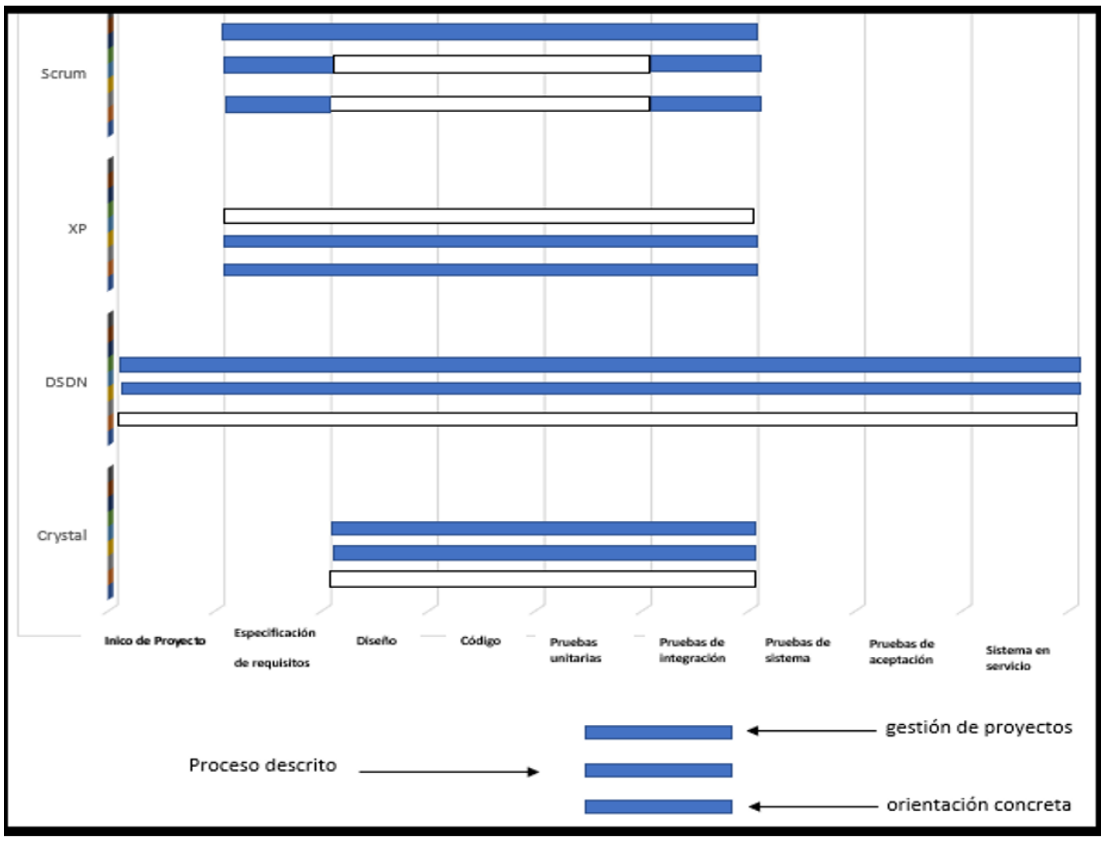


Figura 3.25: Comparativa entre Metodologías Ágiles.
Fuente: Elaborado por el investigador y basado en **[25]**.

Los aspectos que serán tomados en cuenta para la evaluación son:

1. **Gestión de Proyecto:** es importante analizar si la metodología ofrece directrices a las actividades de gestión del proyecto, ya que los métodos deberían ser eficientes por lo que se requiere de forma obligatoria la existencia de actividades de gestión de proyectos, estas actividades permitirán la ejecución correcta de las tareas de desarrollo de software. **[27]**
2. **Proceso Descrito:** este aspecto es muy importante evaluarlo, ya que se necesita saber que etapas del ciclo de vida del software cubren cada metodología ágil. **[27]**
3. **Orientación Concreta:** primeramente, es importante entender en que principios abstractos se basa cada metodología y si proporciona orientación concreta. La orientación concreta trata sobre las practicas, actividades y productos de trabajo. Este conjunto de elementos proporciona una guía sobre cómo se puede ejecutar correctamente una tarea específica. **[27]**

Esta comparativa se la analizo de la siguiente manera, la primera barra corresponde al primer aspecto que es la **"gestión de proyecto",** la segunda es **"proceso discreto"** y la tercera **"orientación concreta".** Si las barras se presentan en color azul significa que el aspecto está presente en esa fase, caso contrario la barra estará en blanco.

Como primera premisa de análisis de los resultados se observa que solo la Metodología DSDM cubre todas las fases del ciclo de vida del software, y también ofrece directrices para la gestión del proyecto, pero por otra parte no proporcionan orientaciones concretas, que sirven para la ejecución de tareas específicas, esta carencia permite deducir que no es una metodología completa.

La Metodología Scrum presenta una deficiencia ya que no cubre algunas fases del ciclo de desarrollo. Pero se resalta que la Metodología está presente en las fases más importantes.

Por lo general la Metodología Scrum y Crystal presentan deficiencias, la ausencia de orientación concreta (aspecto 3) para desarrollar las actividades de desarrollo. Sin embargo, la metodología XP si cubre esta necesidad importante.

Una vez analizado a fondo las respectivas Metodologías Ágiles, se llega a la conclusión de utilizar la Metodología XP (Extreme Programming) para el desarrollo del Chatbot, esta se ajusta perfectamente a la naturaleza del proyecto. En base al cuadro comparativo de la figura 3.25, mencionada anteriormente, se pudo constatar que la Metodología XP está presente en las fases más importantes de la vida de desarrollo de software y cubre dos aspectos fundamentales los cuales son: proceso descrito y orientación concreta. Además, esta permite obtener productos tangibles en tiempos cortos.

3.1.1.2 Metodología Escogida

La Metodología XP (Extreme Programming) se aplicará al desarrollo de proyecto de investigación, ya que se centra en las fases más esenciales del ciclo de vida del desarrollo de un Prototipo de Chatbot, los cuales son: análisis, diseño, desarrollo y pruebas de forma más sencilla y rápida. Además, del uso de iteraciones para la creación de productos incrementales. Dicha Metodología consta de las siguientes fases, que serán aplicados en el desarrollo del prototipo. **[3]**

- Planificación del Proyecto.
- Diseño.
- Codificación.
- Pruebas.

3.1.2 Análisis de Tecnologías para el Desarrollo de Chatbots

En la actualidad, los agentes conversacionales son indispensables en grandes empresas gracias a la forma de interactuar con los usuarios: estos Chatbots ofrecen soluciones con infraestructura que agilita y facilita el desarrollo de aplicaciones basadas en Inteligencia Artificial. **[28]**

Incursionando en el desarrollo de bots existen un sin número de herramientas. La mayoría de estas herramientas son de pago, entre las más connotadas en el área del desarrollo de Chatbots encontramos:

- **Microsoft Bot Framework:** Ofrece una plataforma para construir desde cero agentes conversacionales que tiene la capacidad de aprender y emitir respuestas con un nivel de precisión confiable sin la necesidad de disponer de un humano presente. **[16]** Microsoft Bot Framework simula una interfaz de usuario que permite conectar la parte lógica de la capa de negocio con el usuario desde cualquier parte donde se encuentre. **Además,** se puede implementar recursos de Inteligencia Artificial, Machine Learning y sistemas de procesamiento de imágenes mediante datos REST (Representational State Transfer). **[3]**
- **IBM Watson Conversation**: es una API (Application Programming Interfaces) que sirve para desarrollar aplicaciones que contengan entrada de lenguaje natural y mediante aprendizaje automático responde a los usuarios, simulando una conversación entre seres humanos. Está tecnología permite el despliegue de agentes conversacionales en diversos canales (móvil, web, mensajería, robots), esta API dispone de un kit de desarrollo basado en servicios cognitivos de Watson e IBM (International Business Machines) Cloud. **[1]**
- **Wit.ai:** es un Servicio Web de Facebook, para crear bots para diferentes plataformas (slack, messenger, telegram, entre otros). Este servicio web utiliza historias y flujos de diálogo para el desarrollo de agentes conversacionales. Un punto primordial de esta tecnología es su interpretación del lenguaje natural. **[1]**
- **Amazon Lex:** es un Servicio de AWS (Amazon Web Services) para construir interfaces de conversación dedicadas especialmente para las aplicaciones que utilizan comandos de voz y texto, el motor de conversación que utiliza Amazon lex es sofisticado y está a disposición de cualquier desarrollador, gracias a esta ventaja se puede construir chatbots potentes con lenguaje natural. **[29]**. Esta tecnología actual ocupa flujos de conversación básicos para la creación de bot. Otra de las ventajas importantes de Amazon Lex es el despliegue del agente conversacional en dispositivos móviles, aplicaciones web y plataformas de chat. **[29]**

- **LUIS (Language Understanding Intelligent Service):** es un Servicio Cognitivo e Inteligente que utiliza técnicas de aprendizaje automático con la finalidad de facilitar la extracción el significado de los mensajes de entrada en lenguaje natural. Para el uso de LUIS en un chatbot, se lo realiza mediante una publicación, la aplicación de parte del cliente envía expresiones de texto a la API donde se encuentra el punto de conexión de procesamiento de lenguaje natural de LUIS y recibe como respuesta un objeto JSON. **[30]** Este servicio se expone mediante una API REST simple. En definitiva, mediante una API REST se puede administrar complemente una aplicación y en el futuro irla mejorando su desempeño. **[30]**
- **DialogFlow:** es una plataforma de comprensión de lenguaje natural que permite la construcción de interfaces conversacionales. Además, cuenta con un entorno grafico didáctico. Al igual que otros servicios permite el despliegue de Chatbots en páginas web, aplicaciones móviles, Google Assistant, Amazon, Alexa, Facebook Messenger, entre otras. **[3]**

3.1.2.1 Comparativa de Tecnologías para el Desarrollo de Chatbots

En la tabla 3.5, se visualiza una comparativa con los aspectos más relevantes de las tecnologías de desarrollo de Chatbots.

	Microsoft Bot Framework	IBM Watson Conversation	Wit.ai	Amazon Lex	Bot Libre	LUIS	DialogFlow
Plataformas de desarrollo	Escritorio / Cloud	Cloud	Cloud	Cloud	Cloud	Cloud	Cloud
Tipos de herramientas	Plataforma	API	Servicio Web	Servicio Web	Plataforma	Servicio Cognitivo	Servicio Web
Lenguaje de programación	C#	-	-	-	Java	-	JavaScript
Procesamiento de lenguaje natural	No	Si	Si	Si	Si	Si	Si
Presenta SDK	Si	Si	Si	Si	Si	Si	Si
Compatibilidad con apps externas	Si	No	No	Si, sólo servicios de AWS	No	Si	Si
Despliegue chatbots	Local / Multicanal mediante Azure	Multicanal	Redes Sociales	Multiplataforma	Multicanal	Local / Multicanal mediante Azure	Multicanal / Multiplataforma
Licenciamiento	Gratis / Azure es pagado	Gratis / Pagado	Gratis / Pagado	Gratis / Pagado	Gratis / Pagado	Gratis / Azure es pagado	Gratis / Pagado
Soporta varios idiomas	Si	Si	Si	Si	Si	Si	Si

Tabla 3.5: Comparativa de Tecnologías para el Desarrollo de Chatbots.
Fuente: Elaborado por el investigador y basado en [1, 3, 8, 11, 12, 13, 14, 15, 16, 17, 18, 19].

3.1.2.2 Tecnologías Escogidas

Una vez realizada las investigaciones pertinentes y necesarias acerca de las tecnologías dedicadas al desarrollo de agentes conversacionales y en base a la tabla 3.5, antes mencionada, se decide utilizar Bot Framework como herramienta para el desarrollo del prototipo, ya que es una de las más completas y sencillas de utilizar. Dicha herramienta permite la construcción y despliegue de chatbots de forma local o web. **Además,** permite la integración de diversas tecnologías.

Bot Framework no dispone de un procesador de lenguaje natural por lo que decide utilizar LUIS de Azure para cubrir esta necesidad. Esta tecnología dispone de varios servicios cognitivos para el entrenamiento del bot. Una característica muy importante de Azure es su lógica de "intenciones", la cual será fundamental en el entrenamiento del bot.

3.2 Desarrollo de la Metodología

3.2.1 Requerimientos del Chatbot

El agente conversacional permitirá realizar los siguientes procesos:

- Ingresar en el sistema de servicios vehiculares.
- Listar los vehículos existentes.
- Agregar vehículos nuevos al listado de existentes.
- Listar los servicios vehiculares.
- Guardar la orden de los servicios vehiculares.
- Visualizar la orden de compra de los servicios vehiculares.

Todos estos procesos se convertirán en flujo de diálogos.

3.2.2 Fase 1.-Planificación del Proyecto

Para esta fase denominada Planificación del Proyecto se utilizará Historias de Usuario, ya que estas permiten definir los requerimientos necesarios para el desarrollo del Chatbot. **Además,** tener en cuenta que se trata del desarrollo de un prototipo, por lo tanto, las historias de usuario se plantearon desde el punto de vista de un usuario y basados en la experiencia del investigador.

Rol	Descripción
Desarrollador	Autor intelectual del presente trabajo de investigación.
Usuario final	Son aquellas personas que tendrán la experiencia de interactuar con el Chatbot y realizarán pruebas de funcionalidad.

Tabla: 3.6: Descripción de Roles del Chatbot.
Fuente: Elaborado por el investigador.

Una vez establecido las funcionalidades del Chatbot, se procede a elaborar las respectivas Historias de Usuario acorde a los requerimientos planteados. Los cuales se basaron utilizando como apoyo los prediseños de las interfaces de usuario, que se encuentran en el Anexo B, con el objetivo de entender que es lo que se va a entregar como producto final.

3.2.2.1 Historias de Usuario

Historia de Usuario	
Código: H1	Usuario: Usuario final
Nombre historia: Diálogo de inicio	
Prioridad en negocio: Media	Riesgo en desarrollo: Medio
Puntos estimados: 1	Iteración asignada: 1
Programador responsable: Cristian Llerena	
Descripción: Una vez se despliegue el chatbot, iniciara la conversación empezando con un saludo.	
Observaciones: El saludo se deberá indicar que se trata de un asistente virtual.	

Tabla 3.7: Historia de Usuario - Diálogo de Inicio.

Fuente: Elaborado por el investigador.

Historia de Usuario	
Código: H2	Usuario: Usuario final
Nombre historia: Diálogo de consulta de órdenes de compra	
Prioridad en negocio: Media	Riesgo en desarrollo: Medio
Puntos estimados: 1	Iteración asignada: 1
Programador responsable: Cristian Llerena	
Descripción: El chatbot permitirá visualizar las compras realizadas mediante el ingreso del número de orden de compra por parte del usuario	
Observaciones: Se validará que el usuario ingrese caracteres de números que representará el código de la orden de compra.	

Tabla 3.8: Historia de Usuario - Diálogo de Consulta de Órdenes de Compra.
Fuente: Elaborado por el investigador.

Historia de Usuario	
Código: H3	Usuario: Usuario final
Nombre historia: Diálogo de ingreso al sistema	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Puntos estimados: 3	Iteración asignada: 2
Programador responsable: Cristian Llerena	
Descripción: El chatbot debe presentar la posibilidad de ingresar al sistema, autenticando el usuario y contraseña del usuario correspondiente.	
Observaciones: Solo los usuarios registrados en el sistema tendrán acceso a las funcionalidades que brinda el chatbot.	

Tabla 3.9: Historia de Usuario - Diálogo de Ingreso al Sistema.
Fuente: Elaborado por el investigador.

Historia de Usuario	
Código: H4	Usuario: Usuario final
Nombre historia: Diálogo para consultar los vehículos	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Puntos estimados: 3	Iteración asignada: 3
Programador responsable: Cristian Llerena	
Descripción: El chatbot debe presentar los vehículos que corresponden a ese usuario.	
Observaciones: Dependiendo de la cantidad de vehículos que tenga cada usuario se visualizaran mediante tarjetas adaptables (Adaptive Cards).	

Tabla 3.10: Historia de Usuario - Diálogo para Consultar los Vehículos.
Fuente: Elaborado por el investigador.

Historia de Usuario	
Código: H5	Usuario: Usuario final
Nombre historia: Diálogo para realizar una orden de compra	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Puntos estimados: 3	Iteración asignada: 4
Programador responsable: Cristian Llerena	
Descripción: El chatbot mostrar los diferentes vehículos que corresponden al usuario, el usuario deberá seleccionar uno de ellos y posteriormente se visualizará los respectivos servicios vehiculares a comprar, finalmente el chatbot permitirá guardar la respectiva orden de compra.	
Observaciones: El usuario tiene la posibilidad de cancelar el proceso de compra en cualquier momento que el desee. Además, el usuario podrá añadir uno o más servicios a la orden de compra.	

Tabla 3.11: Historia de Usuario - Diálogo para Realizar una Orden de Compra.
Fuente: Elaborado por el investigador.

Historia de Usuario	
Código: H6	Usuario: Usuario final
Nombre historia: Diálogo para guardar un nuevo vehículo	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Puntos estimados: 3	Iteración asignada: 5
Programador responsable: Cristian Llerena	
Descripción: El chatbot presentará una tarjeta Adaptable que contendrá un formulario para el ingreso de un vehículo, el usuario deberá ingresar los datos del nuevo vehículo.	
Observaciones: Para realizar esta función el usuario debe estar obligatoriamente autenticado al sistema caso contrario el chatbot no le permitirá realizar esta función. Además, se el usuario deberá confirmar el ingreso del nuevo vehículo.	

Tabla 3.12: Historia de Usuario - Diálogo para Guardar un Nuevo Vehículo.
Fuente: Elaborado por el investigador.

Historia de Usuario	
Código: H7	Usuario: Usuario final
Nombre historia: Diálogo para visualizar la orden de compra	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Puntos estimados: 3	Iteración asignada: 6
Programador responsable: Cristian Llerena	
Descripción: El chatbot debe permitir visualizar la orden de compra realizada por el usuario una vez se haya finalizado la compra mediante el uso de una tarjeta adaptiva.	
Observaciones: Para visualizar la orden de compra el chatbot debe preguntar si desea finalizar la compra al usuario	

Tabla 3.13: Historia de Usuario - Diálogo para Visualizar la Orden de Compra.
Fuente: Elaborado por el investigador.

Historia de Usuario	
Código: H8	Usuario: Usuario final
Nombre historia: Diálogo de fin	
Prioridad en negocio: Media	Riesgo en desarrollo: Medio
Puntos estimados: 2	Iteración asignada: 7
Programador responsable: Cristian Llerena	
Descripción: Una vez finalice la conversación el chatbot presentara un mensaje de despedida.	
Observaciones: El mensaje de despedida estará definida por defecto	

Tabla 3.14: Historia de Usuario - Diálogo de Fin.
Fuente: Elaborado por el investigador.

Historia de Usuario	
Código: H9	Usuario: Desarrollador
Nombre historia: Entrenamiento del procesador de lenguaje natural	
Prioridad en negocio: Muy alta	Riesgo en desarrollo: Alto
Puntos estimados: 4	Iteración asignada: 8
Programador responsable: Cristian Llerena	
Descripción: El procesador de lenguaje natural obligatoriamente debe ser entrenado con posibles frases que los usuarios talvez utilicen para comunicarse con el chatbot.	
Observaciones: El procesador de lenguaje natural será entrenado con un grupo de frases de prueba.	

Tabla 3.15: Historia de Usuario - Entrenamiento del Procesador de Lenguaje Natural.
Fuente: Elaborado por el investigador.

Una vez establecido las respectivas historias de usuario, se generaron las siguientes actividades, las cuales son:

3.2.2.2 Actividades de las Historias de Usuario

Historia: Diálogo de Inicio

Tarea	
Código: T1	Código de historia: H1
Nombre tarea: Diseñar el diálogo de inicio	
Tipo de tarea: Desarrollo	Puntos estimados: 0.3
Tiempo: 1 día	
Programador responsable: Cristian Llerena	

Descripción: Definir correctamente el diálogo que iniciara la conversación entre el chatbot y el usuario.

Tabla 3.16: Actividad 1- Historia 1 - Diseñar el Diálogo de Inicio.
Fuente: Elaborado por el investigador.

Tarea	
Código: T2	Código de historia: H1
Nombre tarea: Diseñar el diálogo de sin servicio.	
Tipo de tarea: Desarrollo	Puntos estimados: 0.3
Tiempo: 1 día	
Programador responsable: Cristian Llerena	
Descripción: Definir correctamente el diálogo que presentara el chatbot para notificar que no se encuentra en servicio.	

Tabla 3.17: Actividad 2 - Historia 1 - Diseñar el Diálogo de Sin Servicio.
Fuente: Elaborado por el investigador.

Tarea	
Código: T3	Código de historia: H1
Nombre tarea: Implementar diálogo de inicio	
Tipo de tarea: Desarrollo	Puntos estimados: 0.1
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Implementar el diálogo de inicio tanto en el chatbot como en el PLN.	

Tabla 3.18: Actividad 3 - Historia 1 - Implementar Diálogo de Inicio.
Fuente: Elaborado por el investigador.

Tarea	
Código: T4	Código de historia: H1
Nombre tarea: Implementar diálogo de fuera de servicio	
Tipo de tarea: Desarrollo	Puntos estimados: 1.2
Tiempo: 1 día	
Programador responsable: Cristian Llerena	
Descripción: Implementar el diálogo defuera de servicio.	

Tabla 3.19: Actividad 4 - Historia 1 - Implementar Diálogo de Fuera de Servicio.
Fuente: Elaborado por el investigador.

Historia: Diálogo de Consulta de Órdenes de Compra

Tarea	
Código: T1	Código de historia: H2
Nombre tarea: Diseñar el diálogo para consultar al servicio	
Tipo de tarea: Desarrollo	Puntos estimados: 0.3
Tiempo: 1 día	
Programador responsable: Cristian Llerena	
Descripción: Definir el diálogo que el chatbot presentara cuando el usuario desee consultar las ordenes de compras generadas.	

Tabla 3.20: Actividad 1 - Historia 2 - Diseñar el Diálogo para Consultar al Servicio.
Fuente: Elaborado por el investigador.

Tarea	
Código: T2	Código de historia: H2
Nombre tarea: Implementar la consulta al Web Service de Órdenes	
Tipo de tarea: Desarrollo	Puntos estimados: 1.6
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Definir el diálogo que el chatbot presentara el menú de funcionalidades del sistema.	

Tabla 3.21: Actividad 2 - Historia 2 - Implementar la Consulta al Web Service de Ordenes.
Fuente: Elaborado por el investigador.

Tarea	
Código: T3	Código de historia: H2
Nombre tarea: Diseñar el diálogo para mostrar las órdenes de compra	
Tipo de tarea: Desarrollo	Puntos estimados: 0.3
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Definir el diálogo que el chatbot presentara el menú de funcionalidades del sistema.	

Tabla 3.22: Actividad 2 - Historia 2 - Diseñar el Diálogo para Mostrar las Órdenes de Compra.
Fuente: Elaborado por el investigador.

Historia: Diálogo de Ingreso al Sistema

Tarea	
Código: T1	Código de historia: H3
Nombre tarea: Diseñar el diálogo de inicio al sistema	
Tipo de tarea: Desarrollo	Puntos estimados: 1.8
Tiempo: 3 días	
Programador responsable: Cristian Llerena	
Descripción: Definir el diálogo que permita el ingresar el usuario y contraseña del usuario.	

Tabla 3.23: Actividad 1 - Historia 3 - Diseñar el Diálogo de Inicio del Sistema.
Fuente: Elaborado por el investigador.

Tarea	
Código: T2	Código de historia: H3
Nombre tarea: Validar el ingreso al sistema	
Tipo de tarea: Desarrollo	Puntos estimados: 1.8
Tiempo: 3 días	
Programador responsable: Cristian Llerena	
Descripción: Establecer la conexión con el sistema de servicios vehiculares con el objetivo de validar el usuario.	

Tabla 3.24: Actividad 2 - Historia 3 - Validar el Ingreso al Sistema.
Fuente: Elaborado por el investigador.

Tarea	
Código: T3	Código de historia: H3
Nombre tarea: Implementar el diálogo de ingreso al sistema	
Tipo de tarea: Desarrollo	Puntos estimados: 2.0
Tiempo: 4 días	
Programador responsable: Cristian Llerena	
Descripción: Implementar el diálogo que permita la autenticación del usuario.	

Tabla 3.25: Actividad 3 - Historia 3 - Implementar el Diálogo de Ingreso al Sistema.
Fuente: Elaborado por el investigador.

Historia: Diálogo para Consultar los Vehículos

Tarea	
Código: T1	Código de historia: H4
Nombre tarea: Diseñar el diálogo para presentar los vehículos	
Tipo de tarea: Desarrollo	Puntos estimados: 0.3
Tiempo: 1 día	
Programador responsable: Cristian Llerena	
Descripción: Definir el diálogo que permita mostrar los vehículos correspondientes al usuario.	

Tabla 3.26: Actividad 1 - Historia 4 - Diseñar el Diálogo para Presentar los Vehículos.
Fuente: Elaborado por el investigador.

Tarea	
Código: T2	Código de historia: H4
Nombre tarea: Consultar los vehículos	
Tipo de tarea: Desarrollo	Puntos estimados: 1.6
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Se debe establecer la conexión con el sistema de servicios vehiculares para poder consultar los vehículos existentes.	

Tabla 3.27: Actividad 2 - Historia 4 - Consultar los Vehículos.
Fuente: Elaborado por el investigador.

Tarea	
Código: T3	Código de historia: H4
Nombre tarea: Implementar diálogo para consultar los vehículos	
Tipo de tarea: Desarrollo	Puntos estimados: 1.6
Tiempo: 2 día	
Programador responsable: Cristian Llerena	
Descripción: Implementar el diálogo que permita consultar los vehículos registrados en el sistema de servicios vehiculares.	

Tabla 3.28: Actividad 3 - Historia 4 - Implementar Diálogo para Consultar los Vehículos.
Fuente: Elaborado por el investigador.

Historia: Diálogo para Realizar una Orden de Compra

Tarea	
Código: T1	Código de historia: H5
Nombre tarea: Diseñar el diálogo para realizar una orden de compra	
Tipo de tarea: Desarrollo	Puntos estimados: 1.6
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Diseñar el diálogo que permita realizar una orden de compra.	

Tabla 3.29: Actividad 1 - Historia 5 – Diseñar el Diálogo para Realizar una Orden de Compra.
Fuente: Elaborado por el investigador.

Tarea	
Código: T2	Código de historia: H5
Nombre tarea: Diseñar el dialogo para seleccionar un vehículo	
Tipo de tarea: Desarrollo	Puntos estimados: 1.6
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Implementar el diálogo que permita seleccionar un vehículo	

Tabla 3.30: Actividad 2 - Historia 5 – Diseñar el Diálogo para Seleccionar un Vehículo.
Fuente: Elaborado por el investigador.

Tarea	
Código: T3	Código de historia: H5
Nombre tarea: Diseñar el dialogo para seleccionar servicios vehiculares	
Tipo de tarea: Desarrollo	Puntos estimados: 1.8
Tiempo: 3 días	
Programador responsable: Cristian Llerena	
Descripción: Implementar el diálogo que permita seleccionar uno o más servicios vehiculares	

Tabla 3.31: Actividad 3 - Historia 5 – Diseñar el Diálogo para Seleccionar Servicios Vehiculares.
Fuente: Elaborado por el investigador.

Tarea	
Código: T4	Código de historia: H5
Nombre tarea: Implementar el diálogo para para realizar una orden de compra	
Tipo de tarea: Desarrollo	Puntos estimados: 1.8
Tiempo: 3 días	
Programador responsable: Cristian Llerena	
Descripción: Implementar el diálogo que permita realizar una compra de los servicios vehiculares	

Tabla 3.32: Actividad 4 - Historia 5 – Diseñar el Diálogo para Seleccionar un Vehículo.
Fuente: Elaborado por el investigador.

Historia: Diálogo para Guardar un Nuevo Vehículo

Tarea	
Código: T1	Código de historia: H6
Nombre tarea: Diseñar el diálogo para guardar un nuevo vehículo	
Tipo de tarea: Desarrollo	Puntos estimados: 0.8
Tiempo: 1 día	
Programador responsable: Cristian Llerena	
Descripción: Definir el diálogo que permita guardar el nuevo vehículo en el sistema.	

Tabla 3.33: Actividad 1 - Historia 6 - Diseñar el Diálogo para Guardar un Nuevo Vehículo.
Fuente: Elaborado por el investigador.

Tarea	
Código: T2	Código de historia: H6
Nombre tarea: Implementar el diálogo para guardar un nuevo vehículo	
Tipo de tarea: Desarrollo	Puntos estimados: 1.0
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Implementar el diálogo que permita guardar el nuevo vehículo en el sistema.	

Tabla 3.34: Actividad 2 - Historia 6 - Implementar el Diálogo para Guardar un Nuevo Vehículo.
Fuente: Elaborado por el investigador.

Historia: Diálogo para Visualizar la Orden de Compra

Tarea	
Código: T1	Código de historia: H7
Nombre tarea: Diseñar el diálogo para visualizar la orden de compra	
Tipo de tarea: Desarrollo	Puntos estimados: 0.3
Tiempo: 1 día	
Programador responsable: Cristian Llerena	
Descripción: Definir el diálogo que permita visualizar el detalle de la orden de compra.	

Tabla 3.35: Actividad 1 - Historia 7 - Diseñar el Diálogo para Visualizar la Orden de Compra.
Fuente: Elaborado por el investigador.

Tarea	
Código: T2	Código de historia: H7
Nombre tarea: Consultar orden de compra	
Tipo de tarea: Desarrollo	Puntos estimados: 1.6
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Se necesita establecer la conexión con el sistema de servicios vehiculares para consultar la orden de compra mediante un numero de orden.	

Tabla 3.36: Actividad 2 - Historia 7 - Consultar Órdenes de Compra.
Fuente: Elaborado por el investigador.

Tarea	
Código: T3	Código de historia: H7
Nombre tarea: Implementar el diálogo para visualizar la orden de compra	
Tipo de tarea: Desarrollo	Puntos estimados: 1.6
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Implementar el diálogo que permita visualizar el detalle de la orden de compra	

Tabla 3.37: Actividad 3 - Historia 7 - Implementar el Diálogo para Visualizar las Órdenes de Compra.
Fuente: Elaborado por el investigador.

Historia: Diálogo de Fin

Tarea	
Código: T1	Código de historia: H8
Nombre tarea: Diseñar el diálogo de fin	
Tipo de tarea: Desarrollo	Puntos estimados: 1.6
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Definir el diálogo que dará fin a la conversación con el chatbot.	

Tabla 3.38: Actividad 1 - Historia 8 - Definir el Diálogo de Fin.
Fuente: Elaborado por el investigador.

Tarea	
Código: T2	Código de historia: H8
Nombre tarea: Implementar el diálogo de finalización	
Tipo de tarea: Desarrollo	Puntos estimados: 2.0
Tiempo: 3 días	
Programador responsable: Cristian Llerena	
Descripción: Implementar el diálogo que dará por finalizado tanto con el chatbot como en el PLN.	

Tabla 3.39: Actividad 2 - Historia 9 - Implementar el Diálogo de Fin.
Fuente: Elaborado por el investigador.

Historia: Entrenamiento del Procesador de Lenguaje Natural (PLN)

Tarea	
Código: T1	Código de historia: H9
Nombre tarea: Definir las frases de entrenamiento del procesador del lenguaje natural	
Tipo de tarea: Desarrollo	Puntos estimados: 1.4
Tiempo: 2 días	
Programador responsable: Cristian Llerena	
Descripción: Definir el conjunto de frases que los usuarios utilicen para comunicarse con el chatbot.	

Tabla 3.40: Actividad 1 - Historia 9 - Definir las Frases de Entrenamiento del Procesador de Lenguaje Natural.
Fuente: Elaborado por el investigador.

Tarea	
Código: T2	Código de historia: H9
Nombre tarea: Entrenar el procesador de lenguaje natural	
Tipo de tarea: Desarrollo	Puntos estimados: 2.6
Tiempo: 3 días	
Programador responsable: Cristian Llerena	
Descripción: Ingresar y entrenar el procesador de lenguaje natural con el conjunto de frases definidas anteriormente.	

Tabla 3.41: Actividad 2 - Historia 9 - Entrenar el Procesador de Lenguaje Natural.
Fuente: Elaborado por el investigador.

3.2.2.3 Estimación de Tiempo

Una vez finalizado las tareas se puede realizar una valoración en base a cada historia de usuario y estimar el tiempo que se necesitara para desarrollar cada una de ellas. Cada historia de usuario está definida en semanas de 5 días.

Para el desarrollo del prototipo propuesta, se aplicó una estimación del esfuerzo mediante la técnica puntos de función, la cual estima el tamaño y esfuerzo que requiere el desarrollo del proyecto, al tratarse de un prototipo se puede estimar en base a las funcionalidades que recibirá el usuario y no en la manera como se desarrollaron estas funciones. Por lo tanto, se aplicó la estimación de esfuerzo a cada historia de usuario, las cuales están agrupados en tres módulos:

N.º	**Historia de Usuario**	**Tiempo estimado**		
		Semanas	**Días**	**Horas**
1	Diálogo de inicio	1	5	30
2	Diálogo de fin	1	5	30
TIEMPO ESTIMNADO TOTAL		2	10	60

Tabla 3.42: Estimación del Módulo de Presentación.
Fuente: Elaborado por el investigador a partir de las historias de usuario y tareas.

Módulo de Acceso

N.º	Historia de Usuario	Tiempo estimado		
		Semanas	Días	Horas
3	Diálogo de ingreso al sistema	2	10	60
TIEMPO ESTIMNADO TOTAL		2	10	60

Tabla 3.43: Estimación del Módulo de Acceso.
Fuente: Elaborado por el investigador a partir de las historias de usuario y tareas.

Módulo de Integración

N.º	Historia de Usuario	Tiempo estimado		
		Semanas	Días	Horas
4	Diálogo de consulta de órdenes de compra	2	10	60
5	Diálogo para consultar los vehículos	1	5	30
6	Diálogo para realizar una orden de compra	2	10	60
7	Diálogo para guardar un nuevo vehículo	1	5	30
8	Diálogo para visualizar la orden de compra	1	5	30
9	Entrenamiento del procesador de lenguaje natural	1	5	30
TIEMPO ESTIMNADO TOTAL		8	40	240

Tabla 3.44: Estimación del Módulo de Integración.
Fuente: Elaborado por el investigador a partir de las historias de usuario y tareas.

Una vez realizado los cálculos respecticos del tiempo estimado de los respectivos modulo en base a las historias de usuario y tareas se establece la siguiente planificación de entregables, la cual esta descrita de la siguiente manera:

Modulo	Historia de Usuario	Tiempo estimado		
		Semanas	Días	Horas
Presentación	Diálogo de inicio	1	5	30
	Diálogo de fin	1	5	30
Acceso	Diálogo de ingreso al sistema	2	10	60
Integración	Diálogo de consulta de órdenes de compra	2	10	60
	Diálogo para consultar los vehículos	1	5	30

	Diálogo para realizar una orden de compra	2	10	60
	Diálogo para guardar un nuevo vehículo	1	5	30
	Diálogo para visualizar la orden de compra	1	5	30
	Entrenamiento del procesador de lenguaje natural	1	5	30
TIEMPO ESTIMNADO TOTAL		12	60	360

Tabla 3.45: Resumen Estimado de Módulos.
Fuente: Elaborado por el investigador a partir de la estimación de cada módulo.

3.2.3 Fase 2.-Diseño

Para la realización del diseño correspondiente del Software, la Metodología XP recomienda el uso de Tarjetas CRC (Clase-Responsabilidad-Colaboración), estas tarjetas permiten realizar un diseño orientado a objetos, por ser una representación clara de cada funcionalidad.

3.2.3.1 Tarjetas CRC (Clase-Responsabilidad-Colaboración)

Se debe tener en cuenta que para cada Historia de Usuario se necesita realizar una Tarjeta CRC, las cuales deben contener: una clase que puede ser de tipo persona, concepto, evento, reporte o pantalla.

A continuación, en la tabla 3.46, se observa la tabla correspondiente al cronograma del plan de entrega de iteraciones de proyecto.

Modulo	**Historia de usuario**	**Iteración Asignada**								**Entrega Asignada**							
		1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Presentación	Diálogo de inicio	X								X							
	Diálogo de fin	X								X							
Acceso	Diálogo de ingreso al sistema		X								X						
Integración	Diálogo de consulta de órdenes de compra			X								X					
	Diálogo para consultar los vehículos				X								X				
	Diálogo para realizar una orden de compra					X								X			
	Diálogo para guardar un nuevo vehículo						X								X		
	Diálogo para visualizar la orden de compra							X								X	
	Entrenamiento del procesador de lenguaje natural								X								X

Tabla 3.46: Plan de Entrega de Iteraciones.
Fuente: Elaborado por el investigador a partir de la estimación de cada módulo y basado en **[3]** .

Es importante mencionar que se necesita el uso de responsabilidades que son las acciones que realizan los métodos y atributos, también se necesita de colaboraciones que vienen siendo otras clases que ayudan en la aplicación de las responsabilidades.

Diálogo de Inicio

Diálogo de inicio	
Responsabilidad	**Colaboradores**
Presentar el mensaje de texto	
Observaciones: ninguna	

Tabla 3.47: Tarjeta CRC – Diálogo de Inicio.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo de Ingreso al Sistema

Diálogo de ingreso al sistema	
Responsabilidad	**Colaboradores**
Validar usuario y contraseña	Método de integración con el sistema de servicios vehiculares.
Observaciones: ninguna	

Tabla 3.48: Tarjeta CRC – Diálogo de Ingreso al Sistema.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo de Consulta de Órdenes de Compra

Diálogo para consulta de órdenes de compra	
Responsabilidad	**Colaboradores**
Obtener parámetros Mostar órdenes de compra	Método de integración con el sistema de servicios vehiculares.
Observaciones: ninguna	

Tabla 3.49: Tarjeta CRC – Diálogo de Consulta de Órdenes de Compra.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo para Consultar los Vehículos

Diálogo para consultar los vehículos	
Responsabilidad	**Colaboradores**
Obtener parámetros Mostrar vehículos	Método de integración con el sistema de servicios vehiculares.
Observaciones: ninguna	

Tabla 3.50: Tarjeta CRC – Diálogo para Consultar los Vehículos.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo para Realizar una Orden de Compra

Diálogo para realizar una orden de compra	
Responsabilidad	**Colaboradores**
Obtener parámetros Seleccionar un vehículo Seleccionar uno o varios servicios vehiculares Guardar orden de compra	Método de integración con el sistema de servicios vehiculares.
Observaciones: ninguna	

Tabla 3.51: Tarjeta CRC – Diálogo para Realizar una Orden de Compra.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo para Guardar un Nuevo Vehículo

Diálogo para guardar un nuevo vehículo	
Responsabilidad	**Colaboradores**
Obtener parámetros Guardar nuevo vehículo	Método de integración con el sistema de servicios vehiculares.
Observaciones: ninguna	

Tabla 3.52: Tarjeta CRC – Diálogo Guardar un Nuevo Vehículo.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo para Visualizar la Orden de Compra

Diálogo para visualizar la orden de compra	
Responsabilidad	**Colaboradores**
Obtener parámetros Mostrar orden de compra	Método de integración con el sistema de servicios vehiculares.
Observaciones: ninguna	

Tabla 3.53: Tarjeta CRC – Diálogo para Visualizar la Orden de Compra.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo de Fin

Diálogo de Fin	
Responsabilidad	**Colaboradores**
Presentar mensaje de texto	
Observaciones: ninguna	

Tabla 3.54: Tarjeta CRC – Diálogo de Fin.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Entrenamiento del Procesador de Lenguaje Natural

Procesador de Lenguaje Natural (PLN)	
Responsabilidad	**Colaboradores**
Obtener texto en lenguaje natural. Retornar las intenciones encontradas durante el procesamiento del lenguaje natural.	Procesador de lenguaje natural (PLN).
Observaciones: ninguna	

Tabla 3.55: Tarjeta CRC – Procesador de Lenguaje Natural.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

3.2.3.2 Flujos de Comunicación

En la fase de díselo es necesario generar Flujos de Comunicación, los cuales se definen en base a las historias de usuario. Solo las historias de usuario necesarias se generarán sus respectivos Flujos de Comunicación.

Dialogo de Inicio

Flujo de Comunicación	
Tarea	Esperar que el usuario ingrese cualquier texto y posteriormente presentar el dialogo de ingreso al sistema
Objetivos	Informar al usuario que inicio una conversación mediante le presentación del formulario de login
Motivación del usuario	Iniciar una conversación con el chatbot
Pasos	-Iniciar la conversación -Llenar el formulario de login
Previsiones	El asistente virtual puede estar fuera de línea

Tabla 3.56: Flujo de Conversación – Diálogo de Inicio.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo de Ingreso al Sistema

Flujo de Comunicación	
Tarea	Ingreso al sistema.
Objetivos	Permitir la autenticación del usuario.
Motivación del usuario	Poder utilizar el sistema.
Pasos	-Procesar el formulario de ingreso -Ingresar información de usuario -Validar la información -Presentar mensaje dependiendo la circunstancia, si se pudo o no autenticar el usuario
Previsiones	El usuario puede utilizar alguna frase fuera del ámbito de autenticación.

Tabla 3.57: Flujo de Conversación – Diálogo de Ingreso al Sistema.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo para Consultar las Órdenes de Compra

Flujo de Comunicación	
Tarea	Consultar las órdenes de compra registradas en sistema filtrando mediante el número de orden.
Objetivos	Mostar la orden de compra correspondiente al número de orden que se ingresó.
Motivación del usuario	Visualizar las órdenes de compra existentes en el sistema de servicios vehiculares.
Pasos	-Autenticarse exitosamente en el sistema -Ingresar a la opción Visualizar Compra o escribir la palabra “ver” -Solicitar el número de orden -Ingresar el número de orden -Verificar si la orden existe -Visualizar los servicios vehiculares en la orden de compra
Previsiones	-El usuario puede utilizar alguna frase fuera del ámbito de consulta -El usuario puede ingresar un numero de orden que no existe dentro del registro de órdenes del sistema de servicios vehículares

Tabla 3.58: Flujo de Conversación – Diálogo para Consultar las Órdenes de Compra.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo para Consultar los Vehículos

Flujo de Comunicación	
Tarea	Consultar vehículos.
Objetivos	Mostrar los vehículos existentes en el sistema de servicios vehiculares.
Motivación del usuario	Ver los vehículos existentes dentro de tarjetas adaptables.
Pasos	-Autenticarse exitosamente en el sistema -Ingresar a la opción Listar Vehículos o escribir la palabra “vehículos” -Mostrar vehículos
Previsiones	-El usuario puede utilizar alguna frase fuera del ámbito de Listar Vehículos -No se logue exitosamente el usuario -El usuario no dispone de ningún vehículo registrado

Tabla 3.59: Flujo de Conversación – Diálogo para Consultar los Vehículos.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo para Guardar un Nuevo Vehículo

Flujo de Comunicación	
Tarea	Guardar un nuevo vehículo.
Objetivos	Registrar un nuevo vehículo en el sistema de servicios vehiculares.
Motivación del usuario	Ver la lista de vehículos registrados dentro de tarjetas adaptables.
Pasos	-Autenticarse exitosamente en el sistema -Ingresar en la opción Ingresar Vehículos o escribir la palabra "ingresar" -Ingresar la placa, chasis, modelo y color del vehículo -Confirmar los datos del nuevo vehículo -Indicar si se añadió o no el nuevo vehículo al sistema de servicios vehiculares
Previsiones	-El usuario puede utilizar alguna frase fuera del ámbito de Ingresar vehículos -No se pudo ingresar el nuevo vehículo

Tabla 3.60: Flujo de Conversación – Diálogo para Guardar un Nuevo Vehículo.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo para Realizar una Orden de Compra

Flujo de Comunicación	
Tarea	Realizar una compra de servicios vehiculares.
Objetivos	Guardar la orden de compra realizada por el usuario.
Motivación del usuario	Terminar con el proceso de compra.
Pasos	-Loguearse exitosamente en el sistema -Ingresar a la sección Comprar o escribir la palabra "comprar" -Seleccionar un vehículo de la lista -Seleccionar uno o más servicios vehiculares -Guardar el detalle de la orden de compra -Indicar el estado del proceso de guardado de la orden de compra -Visualizar la orden de compra realizada
Previsiones	-El usuario puede utilizar alguna frase fuera del ámbito de compra -El usuario puede cancelar la compra

Tabla 3.61: Flujo de Conversación – Diálogo para Realizar una Orden de Compra.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo para Visualizar la Orden de Compra

Flujo de Comunicación	
Tarea	Presentar mensaje de fin.
Objetivos	Informar al usuario que ha culminado la conversación.
Motivación del usuario	Finalizar la conversación con el chatbot.
Pasos	-Preguntar si el chatbot puede ayudar en alguna tarea mas -Terminar la conversación -Presentar un mensaje
Previsiones	- El servicio del agente virtual puede estar fuera de línea -El usuario intencionalmente salga del contexto que el chatbot le pregunta

Tabla 3.62: Flujo de Conversación – Diálogo para Visualizar la Orden de Compra.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

Diálogo de Fin

Flujo de Comunicación	
Tarea	Mostrar los servicios vehiculares de una orden de compra.
Objetivos	Visualizar un resumen de una compra.
Motivación del usuario	Ver los servicios vehiculares que se compró.
Pasos	-Confirmar la finalización de la orden de compra -Visualizar la orden de compra que se realizo
Previsiones	-El usuario puede utilizar alguna frase fuera del ámbito de visualización -El usuario cancela la confirmación de la compra

Tabla 3.63: Flujo de Conversación – Diálogo de Fin.
Fuente: Elaborado por el investigador a partir de las historias de usuario.

3.2.4 Fase 3.- Codificación

Para un mejor entendimiento se desarrolló los diagramas de flujo correspondientes para cada dialogo, los cuales están representados en el Anexo A, posteriormente como

parte del proceso de codificación se realizaron prototipos de las interfaces de cada dialogo, las cueles están representadas en el Anexo B.

En el Anexo C se representará un conjunto de imágenes correspondientes al Backend y Frontend.

Finalmente, en el Anexo D se muestra imágenes del sistema ejecutándose de forma local.

3.2.4.1 Desarrollo del Prototipo

El sistema genérico de Servicios Vehiculares esta desarrollado con la tecnología ASP.NET, para el desarrollo del prototipo del bot se lo realizo mediante el lenguaje de programación C# y bajo el uso de varias tecnologías actuales para la construcción de agentes conversacionales. Además, se utilizó el procesador de lenguaje natural "LUIS de Azure" con un servicio cognitivo que dispone de una lógica de intenciones, efectivo al momento de entrenar al bot.

Para la construcción del servidor se lo realizo mediante ASP.NET Core 3.1.1, creando un proyecto tipo ASP.NET CORE MVC, donde se suprimió la vista del modelo ya que no es necesario para el tipo de aplicación que se va desarrollar.

3.2.4.2 Modelado de la Base de Datos

La Base de Datos del sistema de Servicios Vehiculares se desarrolló en SQL Server 2019, la cual almacenara todos los datos ingresados por el usuario a través del Chatbot. En la figura 3.26, se expone el modelo entidad relación correspondiente a la base de datos.

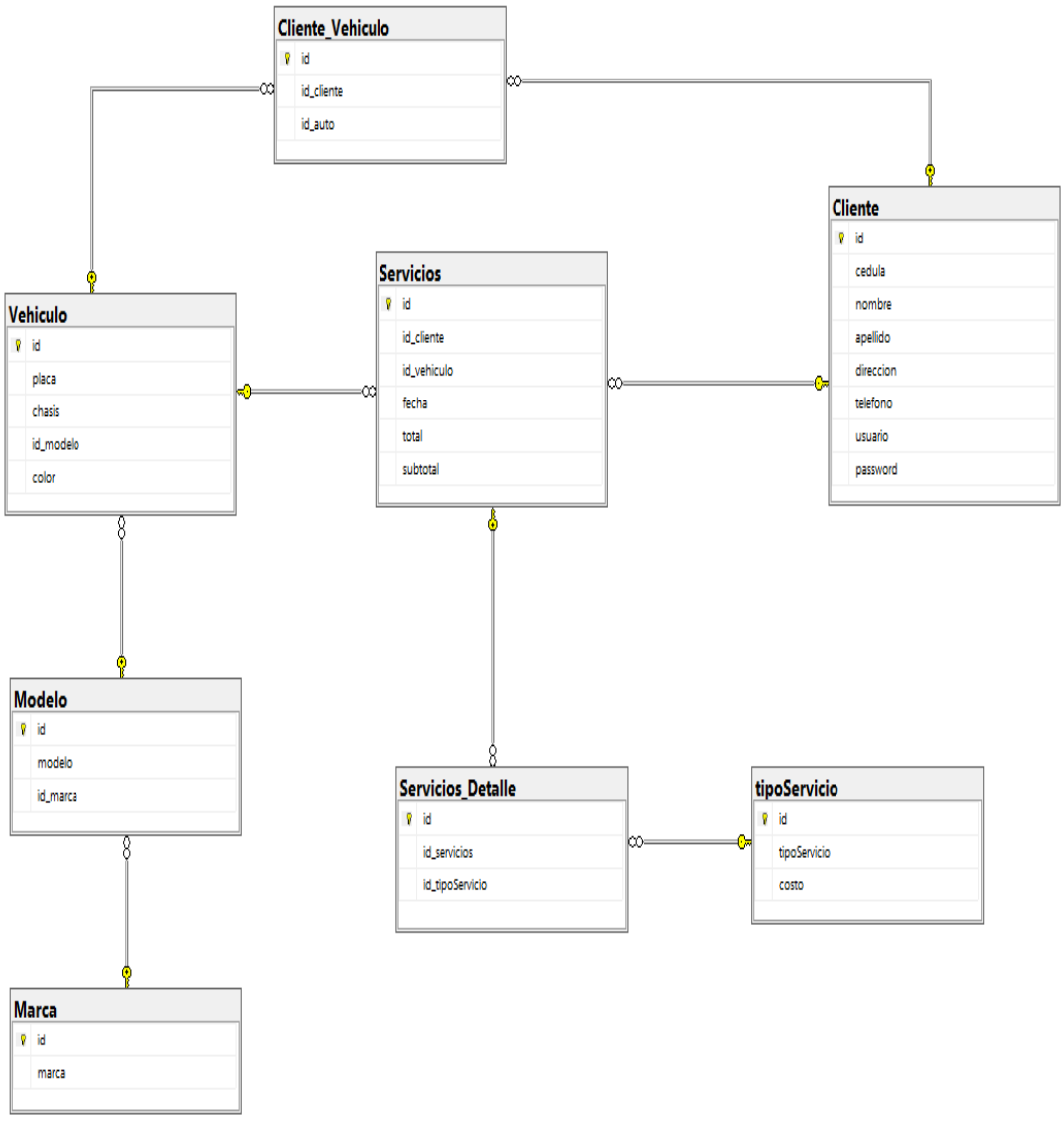


Figura 3.26: Modelo Entidad Relación de la Base de Datos.
Fuente: Elaborado por el investigador.

3.2.4.3 Arquitectura para el Desarrollo del Chatbot

Se utilizará Arquitectura MVC (Model View Controller) para el desarrollo del bot.

En la figura 3.27, se visualiza la Arquitectura a nivel general que tendrá el prototipo propuesto en la investigación.

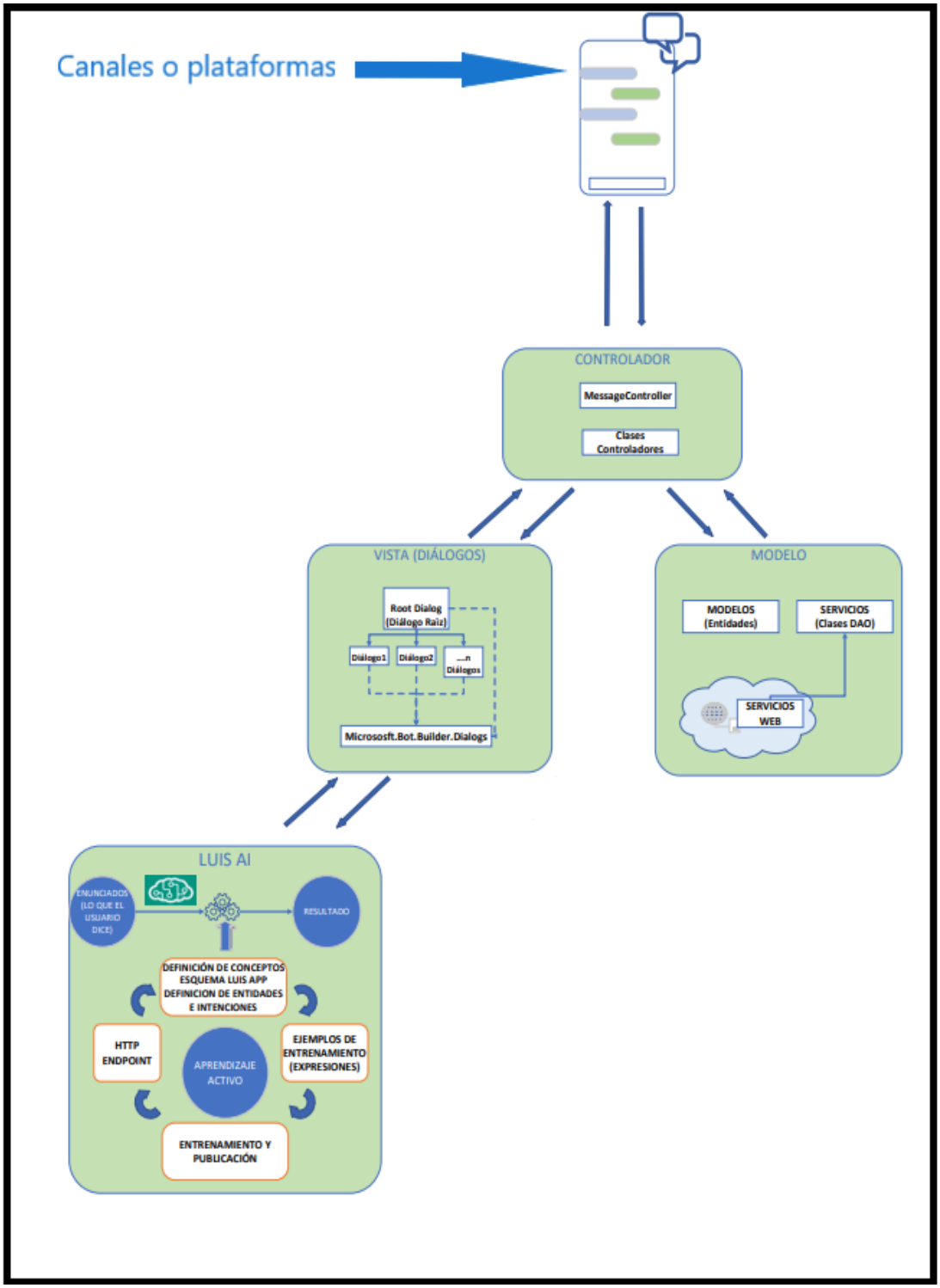


Figura 3.27: Arquitectura MVC para el Desarrollo del Chatbot.
Fuente: Elaborado por el investigador y basado en **[1]**.

3.2.4.4 Procesador de Lenguaje Natural utilizado en el Chatbot

Basado en el análisis que se realizó de las tecnologías para desarrollo de agentes conversacionales y apoyado en la comparativa de estudio de las tecnologías actuales que se encargan del Procesamiento de Lenguaje Natural se decidió utilizar la tecnología LUIS (Language Understanding Intelligent Service).

Las características más importantes por lo que se decidió escoger esta tecnología son:

- Licenciamiento gratuito durante un año, posterior a esto se debe contratar un paquete de mil transacciones por el precio de $5.50 dólares americanos.
- Facilidad para consumir su API.
- Facilidad para entrenar el agente conversacional.
- Trabaja con una lógica de intenciones lo que facilita la interpretación de diálogos.

Se escogió esta tecnología para el Procesamiento de Lenguaje Natural ya que permite la creación de intenciones, cada acción que este registrado en el servicio de LUIS contendrá un grupo de intenciones, estas serán entrenadas para que el bot pueda reconocer que acción debe realizar.

3.2.4.5 Backend

Desarrollo del Servidor

Para la creación de la API REST se utilizó el Framework ASP.NET Core versión 3.1, posteriormente una vez creado el Servicio Web se generó las siguientes carpetas:

Modelo

Es aquel que contiene los datos y la lógica de negocio, para la representación de los datos se lo realizo mediante Entity Framework Core 3.1.1.

Entity Framework Core

Se genero el acceso a los datos a partir de la base de datos existente, esta acción se la realiza mediante un método de creación de datos denominado Database First.

Para la construcción del modelo de acceso de datos se utilizó el comando Scaffold-DbContext que se visualiza a continuación:

```
PM> Scaffold-DBContext "Server=localhost;Database=Servicio_Vehiculo;User ID=sa;Password=[redacted]"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Figura 3.28: Construcción del Modelo a través de Entity Framework Core.
Fuente: Elaborado por el investigador.

Entity Framework Core genera automáticamente algunos archivos, los cuales en su mayoría son clases correspondientes a cada objeto, estas representan las entidades de la base de datos y una clase especial denominada “Servicio_VehiculoContext” que contiene los diversos “dbset” pertenecientes a las entidades que fueron mapeadas, para este caso en particular tenemos las siguientes clases, las cuales se visualizan en la figura 3.29.

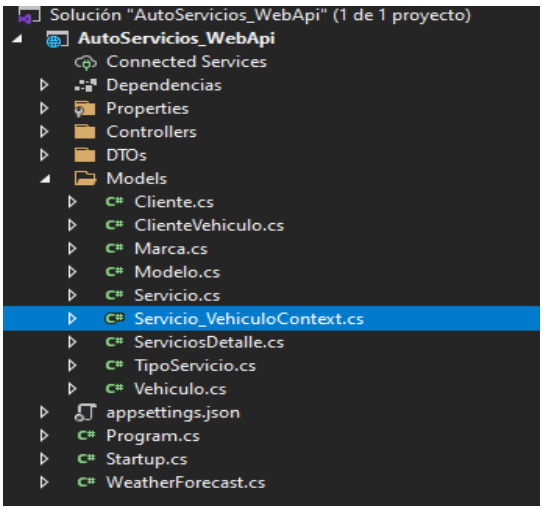
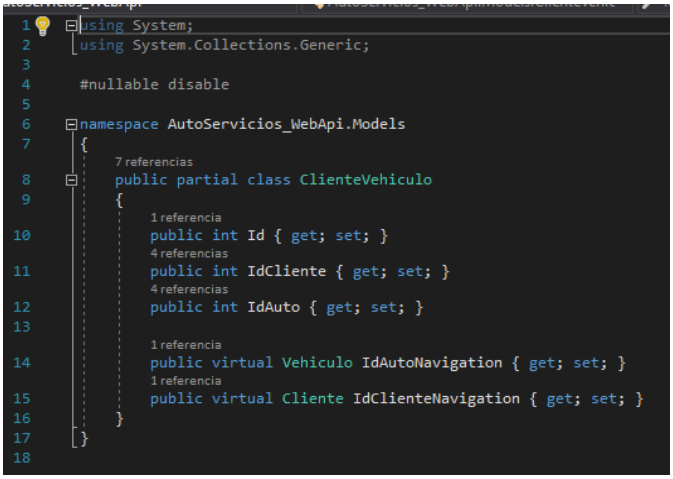


Figura 3.29: Clases Generadas por Entity Framework Core.
Fuente: Elaborado por el investigador.

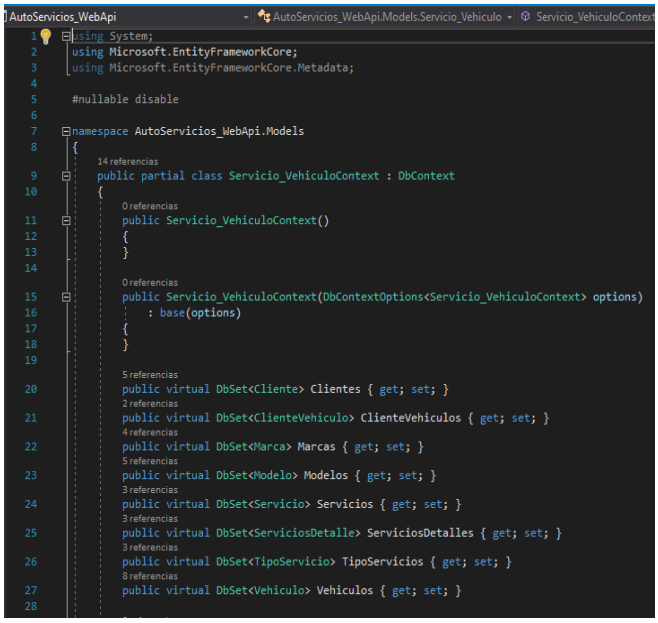
A continuación, en la figura 3.30, se visualiza el código generado automáticamente por Entity Framework Core con respecto a la clase ClienteVehiculo.cs



```
using System;
using System.Collections.Generic;

#nullable disable

namespace AutoServicios_WebApi.Models
{
    7 referencias
    public partial class ClienteVehiculo
    {
        1 referencia
        public int Id { get; set; }
        4 referencias
        public int IdCliente { get; set; }
        4 referencias
        public int IdAuto { get; set; }

        1 referencia
        public virtual Vehiculo IdAutoNavigation { get; set; }
        1 referencia
        public virtual Cliente IdClienteNavigation { get; set; }
    }
}
```

Figura 3.30: Clase ClienteVehiculo.cs Generada por Entity Framework Core.
Fuente: Elaborado por el investigador.

Posteriormente en la figura 3.31, se muestra un fragmento de código generado por Entity Framework Core con respecto a la clase especial Servicio_VehiculoContext.cs, la cual contiene un conjunto de DBSet de las entidades mapeadas.



```
AutoServicios_WebApi
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;

#nullable disable

namespace AutoServicios_WebApi.Models
{
    14 referencias
    public partial class Servicio_VehiculoContext : DbContext
    {
        0 referencias
        public Servicio_VehiculoContext()
        {
        }

        0 referencias
        public Servicio_VehiculoContext(DbContextOptions<Servicio_VehiculoContext> options)
            : base(options)
        {
        }

        5 referencias
        public virtual DbSet<Cliente> Clientes { get; set; }
        2 referencias
        public virtual DbSet<ClienteVehiculo> ClienteVehiculos { get; set; }
        4 referencias
        public virtual DbSet<Marca> Marcas { get; set; }
        5 referencias
        public virtual DbSet<Modelo> Modelos { get; set; }
        3 referencias
        public virtual DbSet<Servicio> Servicios { get; set; }
        3 referencias
        public virtual DbSet<ServiciosDetalle> ServiciosDetalles { get; set; }
        3 referencias
        public virtual DbSet<TipoServicio> TipoServicios { get; set; }
        8 referencias
        public virtual DbSet<Vehiculo> Vehiculos { get; set; }
```

Figura 3.31: Fragmento del Código de la Clase Servicio_VehiculoContext.cs.
Fuente: Elaborado por el investigador.

En la figura 3.32, se observa el método HttpPost del Controlador VehiculoController, recibe información otorgada por el usuario desde modelo ServicioVehiculo_EchoBot y posteriormente aplica una validación a la información ingresada caso contrario retorna un valor falso, posteriormente se crea los siguientes objetos: vehículo y clienteVehículo, luego se aplica una condición, si el ID del vehículo no existe corresponde a un nuevo vehículo caso contario será una edición y automáticamente se llena los datos ya sea de tipo nuevo o edición y consecutivamente asignarle si pertenece a la operación de agregar o actualizar el vehículo. Finalmente, si no se presentan errores durante el proceso se procede a utilizar la operación "transaction.Commit()" para garantizar el ingreso del nuevo registro y retornar el nuevo vehículo.

```
[HttpPost]
0 referencias
public async Task<IActionResult> PostHorario([FromBody] VehiculoIE vehiculoIE)
{
    if (!ModelState.IsValid)
        return Ok(false);
        //throw new NotFoundCustomException("Error en los datos enviados", $"Por
    var vehiculo = new Vehiculo();
    var clienteVehiculo = new ClienteVehiculo();

    //bool esAgregar = false;
    using (var transaction = _context.Database.BeginTransaction())
    {
        try
        {
            if (vehiculoIE != null)
            {
                if (!VehiculoExists(vehiculoIE.Id))
                {
                    //esAgregar = true;
                    vehiculo.Placa = vehiculoIE.Placa;
                    vehiculo.Chasis = vehiculoIE.Chasis;
                    vehiculo.Color = vehiculoIE.Color;
                    vehiculo.IdModelo = vehiculoIE.IdModelo;
                    _context.Vehiculos.Add(vehiculo);
                    await _context.SaveChangesAsync();

                    clienteVehiculo.IdCliente = vehiculoIE.IdCliente;
                    clienteVehiculo.IdAuto = vehiculo.Id;
                    _context.ClienteVehiculos.Add(clienteVehiculo);
                }
                else
                {
                    vehiculo = await _context.Vehiculos.FindAsync(vehiculoIE.Id);
                    vehiculo.Placa = vehiculoIE.Placa;
                    vehiculo.Chasis = vehiculoIE.Chasis;
                    vehiculo.Color = vehiculoIE.Color;
                    vehiculo.IdModelo = vehiculoIE.IdModelo;
                    _context.Vehiculos.Update(vehiculo);
                }
            }
            await _context.SaveChangesAsync();
            var vehiculoIES = new VehiculoIES();
            vehiculoIES = this.GetVehiculoPorId(vehiculo.Id);
            transaction.Commit();

            return Ok(vehiculoIES);


        }
        catch (Exception ex)
        {
            transaction.Rollback();

            return Ok(false);

        }
    }
}
```

Figura 3.32: Método HttpPost correspondiente al Controlador VehiculoController.
Fuente: Elaborado por el investigador.

DTOs

Mas conocido como mensajería, cuya función primordial es enviar y recibir información desde el controlador hacia el cliente, la finalidad de esta acción es evitar referencias circulares. Además, se encarga de ocultar datos extras en referencia a la información remitida por el modelo.

El archivo denominado “ServicioDetalleDTO.cs” presenta dos clases importantes, el “ServicioDetalleDTO” y “ServicioDetalleIS” que muestran la información de entrada y salida correspondiente al controlador “ServicioDetalleController.cs”. La figura 3.33 muestra el código referente al archivo denominado ServicioDetalleDTO.cs.

```
1 referencia
public class ServicioDetalleDTO
{
    0 referencias
    public int Id { get; set; }
    0 referencias
    public int IdServicios { get; set; }
    1 referencia
    public int IdTipoServicio { get; set; }
}
3 referencias
public class ServicioDetalleIS
{
    2 referencias
    public int Id { get; set; }
    2 referencias
    public int IdServicios { get; set; }
    2 referencias
    public int IdTipoServicio { get; set; }
    2 referencias
    public string TipoServicio { get; set; }
    2 referencias
    public double Costo { get; set; }


}
```

Figura 3.33: Fragmento de Código correspondiente a la Clase ServicioDetalleDTO.cs. Fuente: Elaborado por el investigador.

3.2.4.6 Frontend

Desarrollo del Cliente

El Cliente se lo desarrollo utilizando técnicas y tecnologías actuales, se lo estructuro de la siguiente forma:

Módulo de Desarrollo	**Herramienta o Tecnología**
Diseño de diálogos de comunicación	Waterfall dialogues (Diálogos en cascada)
Presentación de las interfaces de usuario	Adaptive Cards (Tarjetas Adaptables)
Construcción del bot	Bot Framework
Entrenamiento del bot	LUIS Language Understanding de Azure (Procesador de Lenguaje Natural)
Desarrollo del servidor API Rest	ASP.NET Core 3.1

Tabla 3.64: Descripción de Técnicas y Herramientas para el Desarrollo del Cliente.
Fuente: Elaborado por el investigador.

Bot Builder V4

Es un SDK (Software Development Kit) que proporciona al entorno de desarrollo Visual Studio 2019 las plantillas correspondientes para el desarrollo de Chatbots, por lo que se descargó el paquete con la versión más actual. **[16]**

Figura 3.34: SDK Templates para Visual Studio.
Fuente: Elaborado por el investigador.

Una vez instalado los templates se procede a instalar el emulador donde se desplegará el prototipo del bot.

Bot Framework Emulator-4.7.0 para Windows

Es una herramienta propia de Microsoft Azure que permite emular, desplegar y construir bots de alta calidad a través de instrucciones precisas.

El uso de este emulador es esencial, ya que se podrán realizar las pruebas necesarias antes de obtener el producto final. **[31]**

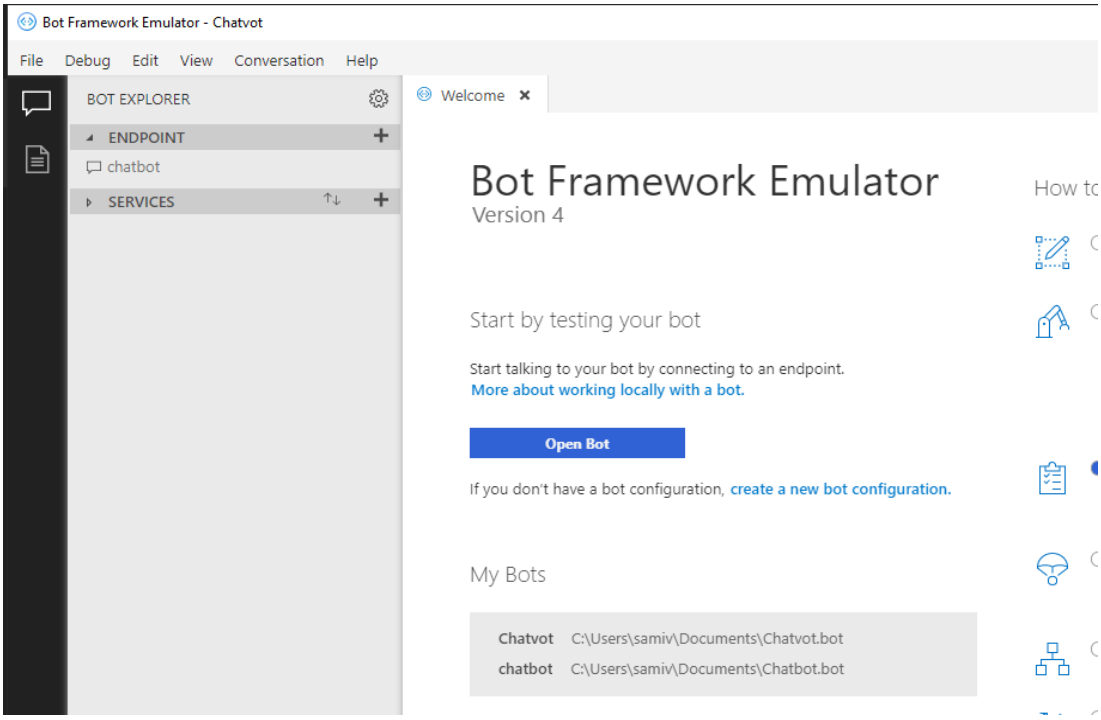


Figura 3.35: Interfaz principal de Bot Framework Emulator.
Fuente: Elaborado por el investigador.

Plantilla Echo Bot (Bot Framework-.NET Core 3.1)

Para la realización del proyecto referente al Bot se escogió la plantilla Echo Bot (Bot Framework-.NET Core 3.1), la cual generara un ejemplo práctico de un bot representado en la figura 3.36, este bot responde exactamente lo que ingresa el usuario. Además, mediante este ejemplo se podrán realizar modificaciones necesarias con el objetivo de construir el prototipo planteado.

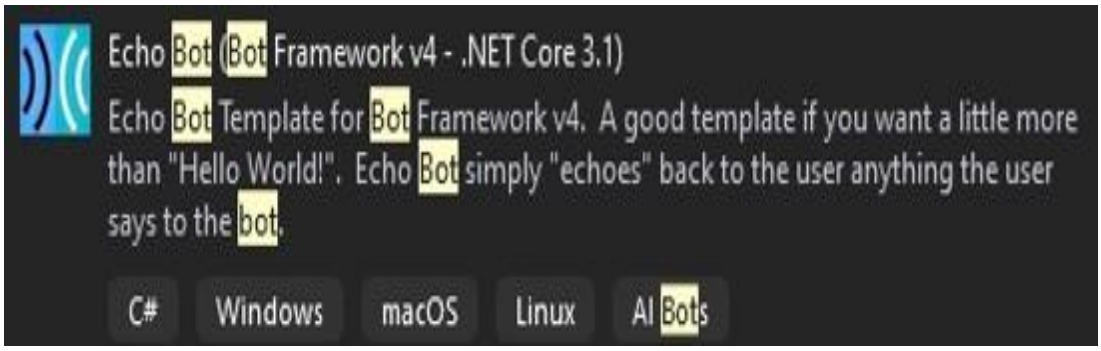


Figura 3.36: Plantilla Echo Bot.
Fuente: Elaborado por el investigador.

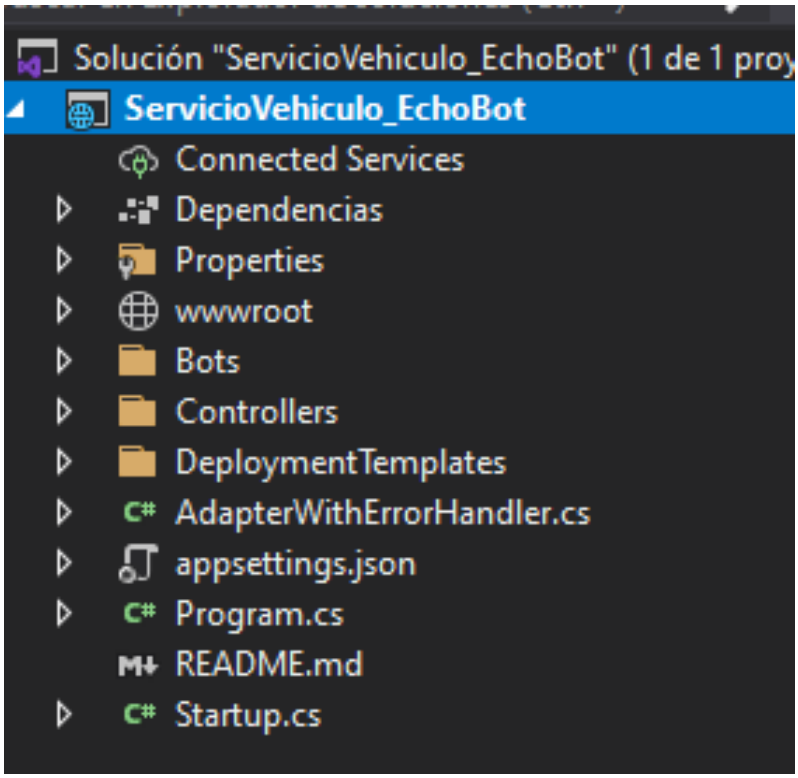


Figura 3.37: Proyecto Generado con el Template Echo Bot.
Fuente: Elaborado por el investigador.

Como complemento se agregaron paquetes que servirán para el desarrollo del bot a través del administrador de paquetes NuGet, como se visualiza en la figura 3.38:

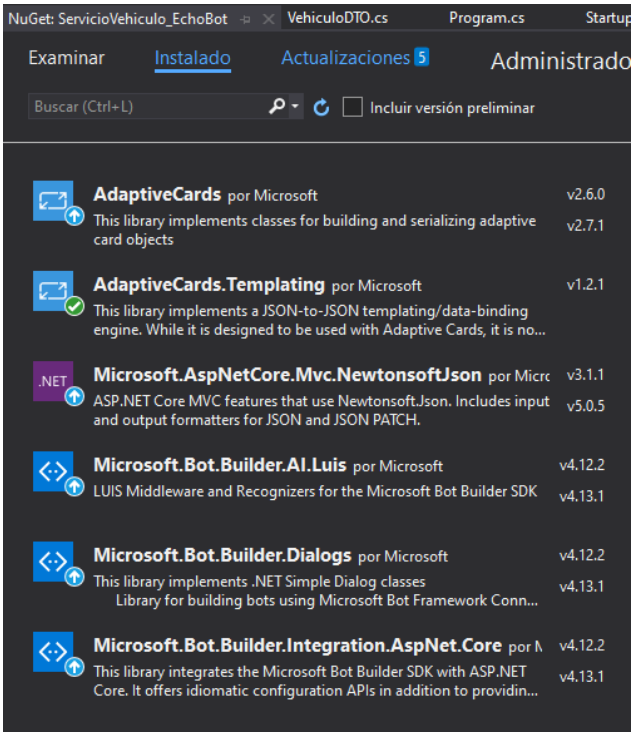


Figura 3.38: Paquetes Agregados desde NuGet.
Fuente: Elaborado por el investigador.

Diseño de Interfaces con Adaptive Cards

Las Adaptive Cards son un formato de intercambio de código abierto, su objetivo principal es intercambiar contenido de las interfaces de usuario de forma común y coherente. Además, las tarjetas adaptables describen su contenido como un objeto JSON, posteriormente ese contenido se representa de forma nativa de una aplicación de host permitiendo adaptarse a la apariencia del mismo. **[14]**

Esta tecnología permite diseñar de forma eficiente y sofisticada las respectivas interfaces de usuario que servirán para la comunicación entre usuario y bot, los diseños se los realizo a través del diseñador de tarjetas adaptables propio de Windows, el cual se lo puede encontrar en la siguiente página: https://vnext.adaptivecards.io/designer/, una ventaja muy importante del diseñador es la de poder visualizar los diseños mediante una vista previa y posteriormente ir generando automáticamente el código

JSON (JavaScript Object Notation) que servirá como plantilla de estilo que luego serán consumidas por la capa de presentación del proyecto.

Diseñador de la Interfaz de Ingreso al Sistema

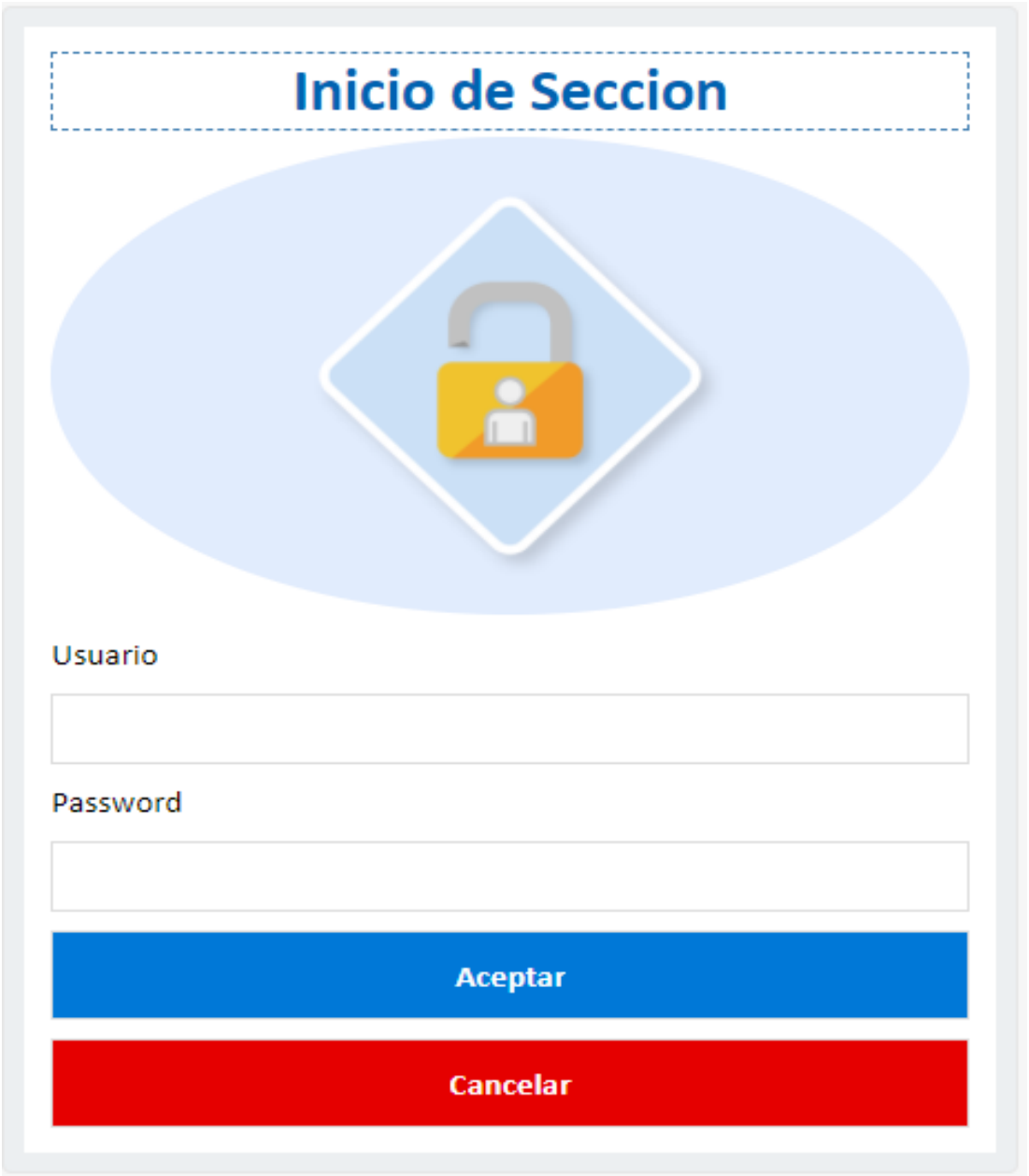


Figura 3.39: Diseño de la Interfaz de Ingreso al Sistema.
Fuente: Elaborado por el investigador.

Diseño de la Interfaz de Ingreso de un Vehículo

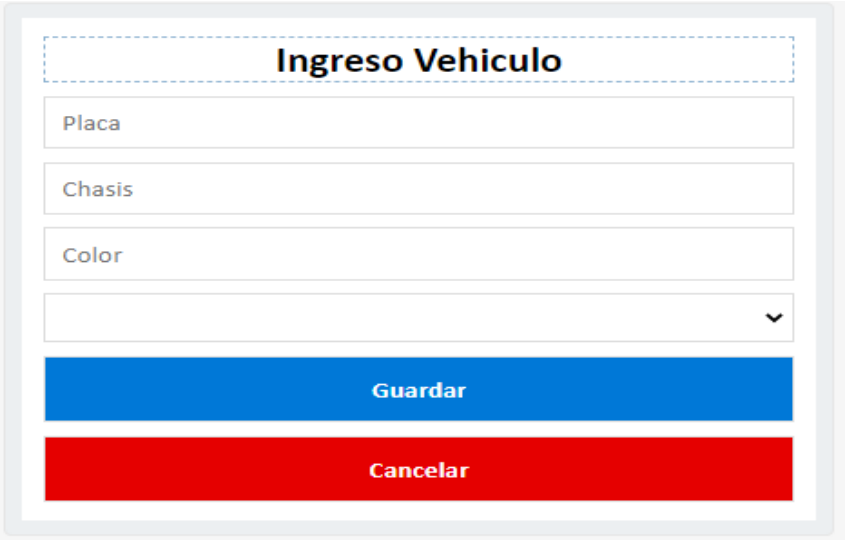


Figura 3.40: Diseño de la Interfaz para Ingresar un Vehículo.
Fuente: Elaborado por el investigador.

Diseño de la Interfaz de Bienvenida

Figura 3.41: Diseño de la Interfaz de Bienvenida.
Fuente: Elaborado por el investigador.

Diseño de la Interfaz para Listar los Servicios Vehículares

Figura 3.42: Diseño de la Interfaz para Listar los Servicios Vehículares.
Fuente: Elaborado por el investigador.

Diseño de la Interfaz para Listar los Vehículos

Figura 3.43: Diseño de la Interfaz para Listar los Vehículos.
Fuente: Elaborado por el investigador.

Diseño de la Interfaz para Seleccionar un Vehículo

Figura 3.44: Diseño de la Interfaz para Seleccionar un Vehículo.
Fuente: Elaborado por el investigador.

Diseño de la Interfaz de Opciones

Figura 3.45: Diseño de la Interfaz de Opciones.
Fuente: Elaborado por el investigador.

Diseño de la Interfaz de la Orden de Compra

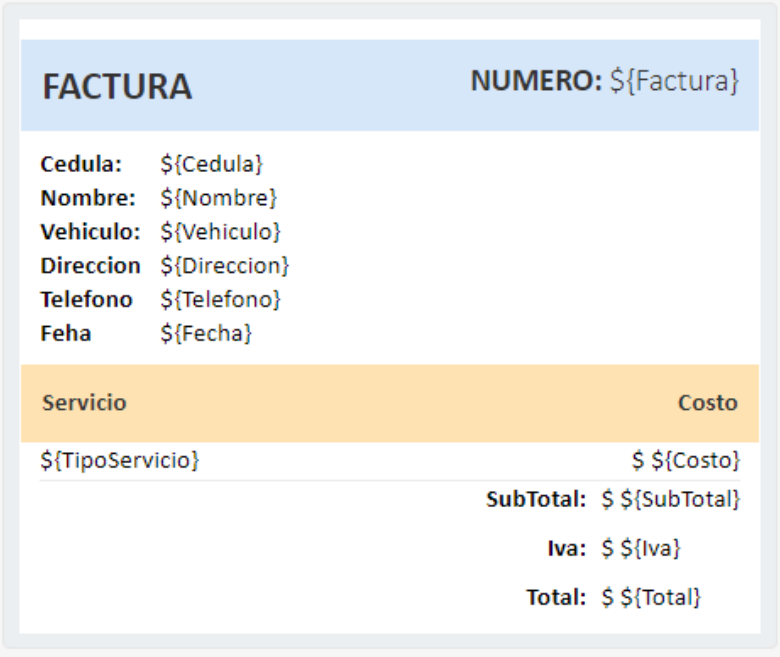


Figura 3.46: Diseño de la Interfaz de la Orden de Compra.
Fuente: Elaborado por el investigador.

Diseño de la Interfaz de la Vista Previa de la Orden de Compra

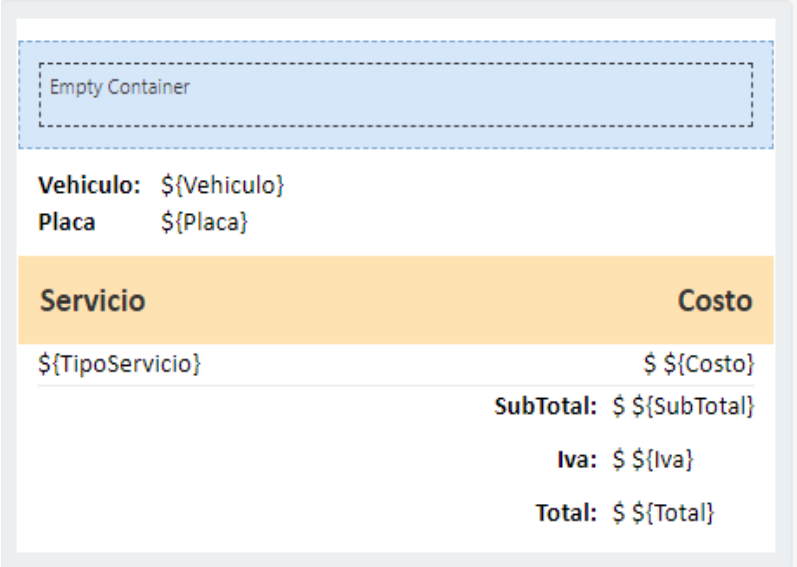


Figura 3.47: Diseño de la Interfaz de la Vista Previa de la Orden de Compra.
Fuente: Elaborado por el investigador.

Diseño de la Interfaz de Despedida

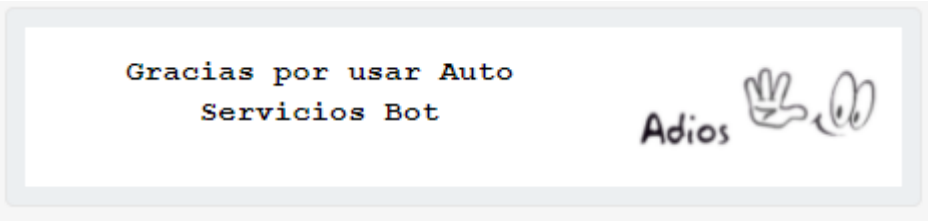


Figura 3.48: Diseño de la Interfaz de Despedida.
Fuente: Elaborado por el investigador.

Microsoft Bot Builder Dialogs

Es una biblioteca del SDK (Software Development Kit), que admite la administración de una conversación duradera con el usuario, un dialogo puede realizar varias tareas las cuales pueden representar parte de un hilo de conversación. Un cuadro de dialogo representa una tarea conversacional que puede ejecutarse durante todo el proceso y a su vez puede devolver información recopilada. **Además,** un dialogo representa una unidad básica de flujo de control. El flujo de control está conformado de la siguiente manera **[32]**:

- Comenzar.
- Continuar y terminar.
- Pausar y reanudar.
- Cancelar.

La utilización de esta biblioteca ayudo en la elaboración de los diálogos del bot, ya que permitió definir el flujo de conversación que genera un orden de ejecución en las actividades, es decir establece cuando debe iniciar y terminar. **Además,** realiza la validación de datos a través de la clase “TextPromt”, esta solicita la entrada de texto al usuario, dispone de otra función esencial, la de permitir la visualización de mensajes de texto o de Adaptive Cards **[33]**.

WallerfallDialogs (Diálogos en cascada)

Para el desarrollo del flujo de diálogos se implementó la técnica WallerfallDialogs, esta técnica establece un cuadro de dialogo que permite generar una serie de preguntas las cuales deberán ser respondidas por el usuario. Los diálogos en cascada establecen un conjunto de pasos al flujo de conversación, inicia estableciendo un conjunto de pasos, los cuales se asigna las diversas actividades que debe cumplir obligatoriamente y posteriormente pasar a la siguiente actividad establecida, mientras no se cumpla la actividad actual el dialogo no podrá seguir al siguiente paso.

En la clase principal de los diálogos perteneciente al proyecto y denominada "MainDialog.cs" la cual hereda de "ComponentDialog", esta permite acceso a las clases "BotStateService", "LuisService" y se define las variables definidas correspondiste para cada una. También en el main se podrá visualizar la creación de varios métodos que inicializaran los diálogos en cascada, entre ellos tenemos:

InitWaterfallDialogs(): este método de tipo private presenta un array de "waterfallSteps" , el cual contiene el conjunto de pasos del dialogo los cuales a su vez se convierten en submétodo, los cuales son:

- **InitialStepAsync (paso inicial):** obtiene la información del usuario (UserProfile), posteriormente inicializa el dialogo de sesión, el cual recibe como parámetro el ID del dialogo el cual está conformado por el dialogo de la clase "MainDialog" y el nombre que se le asigno a la clase "InicioSeccion". Además, recibe la clase tipo "BotStateService" que alberga los estados de la conversación.
- **OptionStepAsync (paso de opciones):** Funciona de la mimas forma del anterior método, adicionalmente este método va asignando la información ingresa por el cliente con los atributos del objeto cliente.
- **SelectOptionStepAsync (paso de selección):** es aquel que selecciona la opción y redirecciona a las diferentes opciones, es decir aquí se controla la selección de las opciones.

- **FinalStepAsync (paso final):** verifica si se cumplió todos los pasos correspondientes del dialogo.

Implementación de Adaptive Cards

En la carpeta Dialogs se creó la clase denominada “InicionSeccionDialog.cs”, la cual contiene los métodos necesarios para el consumo de las tarjetas adaptables en la interfaz de login, el cual se lo describe a continuación: Se creo un método llamado “LogginAdaptiveCardt()”, al cual se le agrego un “Attachment”, este contendrá todas las activadas que se van a mostrar en Bot Framework, adicionalmente se agregó una variable “reply” la cual almacenara los “Attachment” existentes , posteriormente se agrupa las actividades correspondientes al “stepContext” con el objetivo de no perder el estado de conversación, luego con la propiedad “Add” se añade las respuestas que viene desde el método “LogginAdaptiveCardt()” perteneciente a la clase “Metodos.cs” de la carpeta Servicios, dicho método se encarga de cargar las Adaptive Cards, ya que esta clase contiene todos los métodos necesarios para realizar esta acción.

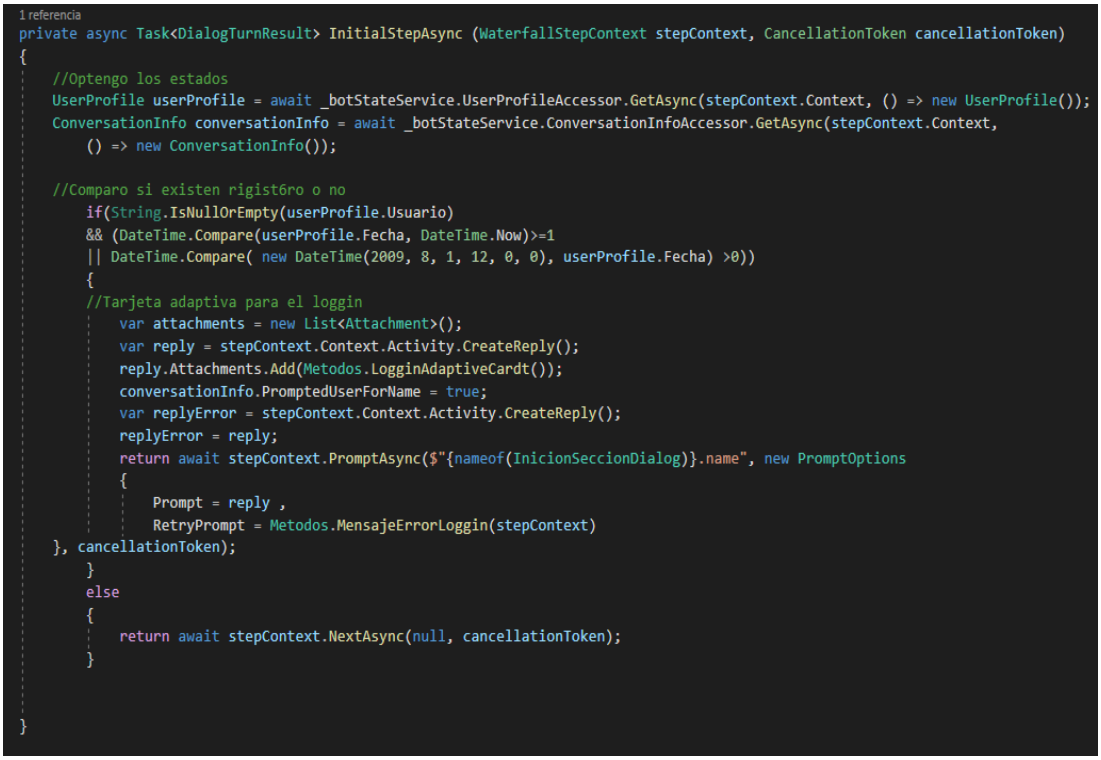
El método “LogginAdaptiveCardt()” trabaja de la siguiente manera: carga la información que se encuentra almacenada en el “paths”, es decir el directorio que contiene el archivo denominado “BienvenidaTarjeta.json”, ya que este archivo contiene el diseño de la tarjeta adaptable que se desea aplicar a la interfaz pero con formato JSON, en el “paths” se cargara el directorio de los archivos antes mencionados , luego se creara un archivo tipo “ReadAllText” el cual servirá para cargar el contenido del archivo JSON y consecuentemente guardar dentro de la variable “adaptiveCardJson”, luego procede a deserializar el archivo JSON convirtiéndolo en una clase tipo “AdaptiveCardTemplate” perteneciente al paquete previamente instalado desde el NuGet, después se crea una variable llamada “myData” esta permitirá mostrar los datos en el JSON y poder enviar información al Adaptive Cards, esto se lo realiza de la siguiente manera: se crea una variable tipo String denominada “cardJson”, la cual contendrá la carga final del Adaptive Cards por lo que expande la “myData” a la variable “template”, dentro de la variable “myData” que contiene la interpolación llamada “${nombre}” posteriormente se carga un nuevo “Attachmen”

que será de tipo “adaptiveCardAttachment”, el cual contendrá el archivo desrealizado correspondiente al “cardJson”.

Realizado todo este proceso lo siguiente que se realiza es enviar la lista de Attachmen al “reply”, este debería retornar un “PromAsync”, la propiedad que permite mostrar los respectivos mensajes de error es “RetryProm”, el método responsable de realizarme la validación es el “validarlogin”, verifica si el usuario ingresa una palabra diferente a la que se estableció, este automáticamente ejecutara lo que se defina ya sea un mensaje de error o algún submétodo de validación. Si fuese el caso de que retorne el mensaje de error automáticamente llama al método “MensajeErrorLoggin” que se encuentra ubicado en la clase “Metodos.cs”, la cual contiene los respectivos mensajes de error, este método de validación será definido en “InicionSeccionDialog”, previamente el método tendrá una configuración establecida para la verificación del login, este método compara la información de los campos pertenecientes a la tarjeta adaptiva como: id, usuario y password, luego envía a la clase “ClienteServicio.cs” la palabra login que contiene toda esta información, este se comunicara con el servicio (API REST), si el login llega vacío retorna “false”, esto significa que hubo un error en la validación y automáticamente mostrara el respectivo mensaje de error, caso contrario si el login es exitoso se obtiene toda la información que posteriormente será almacenada en la clase “BotStateService”, esta clase contiene el promptContext, Context y UserProfile pertenecientes al usuario y estado de la conversa, finalmente si se retorna “true” significa que se realizó correctamente la validación.

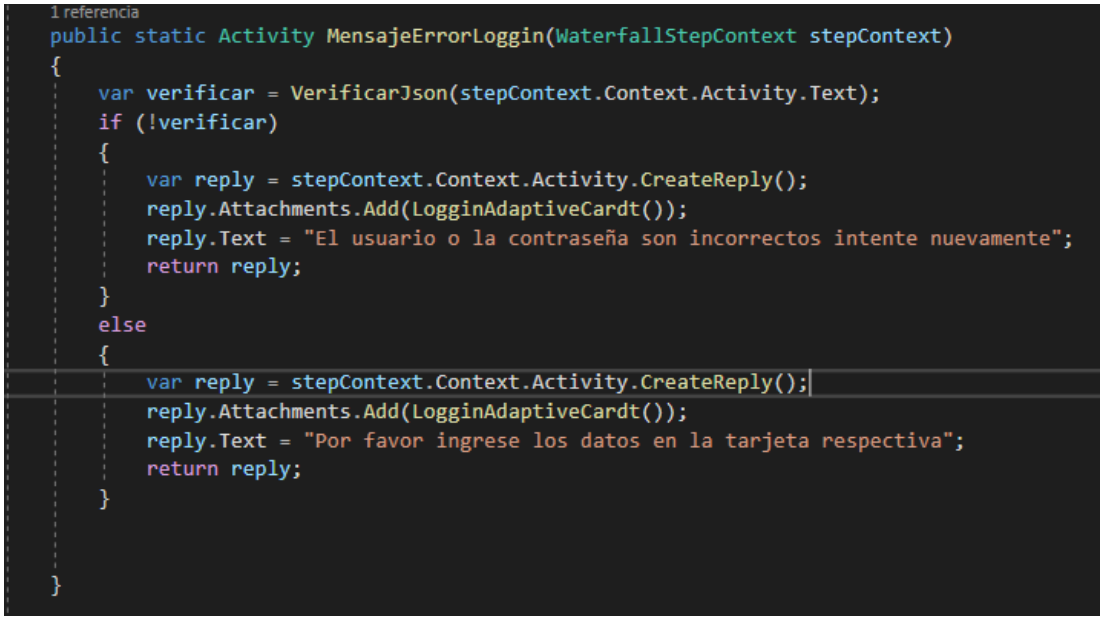
Si el proceso llega a este punto se considera aprobado el primer paso y posteriormente pasa al segundo paso llamado OptionStepAsync, el proceso se repite sucesivamente hasta llevar al paso final.

La misma lógica se aplica en todas las tarjetas adaptivas que se utilicen. En la siguiente figura 3.49, se observar la implementación del Adaptive Cards en la interfaz de Ingreso al Sistema.



```
1 referencia
private async Task<DialogTurnResult> InitialStepAsync (WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    //Optengo los estados
    UserProfile userProfile = await _botStateService.UserProfileAccessor.GetAsync(stepContext.Context, () => new UserProfile());
    ConversationInfo conversationInfo = await _botStateService.ConversationInfoAccessor.GetAsync(stepContext.Context,
        () => new ConversationInfo());

    //Comparo si existen rigist6ro o no
        if(String.IsNullOrEmpty(userProfile.Usuario)
        && (DateTime.Compare(userProfile.Fecha, DateTime.Now)>=1
        || DateTime.Compare( new DateTime(2009, 8, 1, 12, 0, 0), userProfile.Fecha) >0))
        {
        //Tarjeta adaptiva para el loggin
            var attachments = new List<Attachment>();
            var reply = stepContext.Context.Activity.CreateReply();
            reply.Attachments.Add(Metodos.LogginAdaptiveCardt());
            conversationInfo.PromptedUserForName = true;
            var replyError = stepContext.Context.Activity.CreateReply();
            replyError = reply;
            return await stepContext.PromptAsync($"{nameof(InicionSeccionDialog)}.name", new PromptOptions
            {
                Prompt = reply ,
                RetryPrompt = Metodos.MensajeErrorLoggin(stepContext)
    }, cancellationToken);
        }
        else
        {
            return await stepContext.NextAsync(null, cancellationToken);
        }


}
```

Figura 3.49 Implementación del Adaptive Cards en la Interfaz de Ingreso al Sistema.
Fuente: Elaborado por el investigador.



```
1 referencia
public static Activity MensajeErrorLoggin(WaterfallStepContext stepContext)
{
    var verificar = VerificarJson(stepContext.Context.Activity.Text);
    if (!verificar)
    {
        var reply = stepContext.Context.Activity.CreateReply();
        reply.Attachments.Add(LogginAdaptiveCardt());
        reply.Text = "El usuario o la contraseña son incorrectos intente nuevamente";
        return reply;
    }
    else
    {
        var reply = stepContext.Context.Activity.CreateReply();
        reply.Attachments.Add(LogginAdaptiveCardt());
        reply.Text = "Por favor ingrese los datos en la tarjeta respectiva";
        return reply;
    }


}
```

Figura 3.50: Método de Validación con Mensajes de Error.
Fuente: Elaborado por el investigador.

```
private async Task<bool> validarlogin(PromptValidatorContext<string> promptContext, CancellationToken cancellationToken)
{
    var verificar = Metodos.VerificarJson(promptContext.Context.Activity.Text);
    if (verificar)
    {
        JToken commandToken = JToken.Parse(promptContext.Context.Activity.Text);
        string command = commandToken["id"].Value<string>();
        string usuario = commandToken["usuario"].Value<string>();
        string password = commandToken["password"].Value<string>();
        if(!String.IsNullOrEmpty(usuario) && !String.IsNullOrEmpty(password))
        {
            var login = await ClienteServicio.Loggin(usuario, password);
            UserProfile userProfile = new UserProfile();
            ConversationInfo conversationInfo = new ConversationInfo();
            if (login.Usuario != null)
            {
                userProfile.Id = login.Usuario.Id;
                userProfile.Cedula = login.Usuario.Cedula;
                userProfile.Nombre = login.Usuario.Nombre;
                userProfile.Apellido = login.Usuario.Apellido;
                userProfile.Direccion = login.Usuario.Direccion;
                userProfile.Telefono = login.Usuario.Telefono;
                userProfile.Usuario = login.Usuario.Usuario;
                userProfile.Token = login.Token;
                userProfile.Fecha = login.Expiration;
                conversationInfo.PromptedUserForName = true;
                await _botStateService.UserProfileAccessor.SetAsync(promptContext.Context, userProfile);
                await _botStateService.ConversationInfoAccessor.SetAsync(promptContext.Context, conversationInfo);
                await _botStateService.UserState.SaveChangesAsync(promptContext.Context);
                await _botStateService.ConversationState.SaveChangesAsync(promptContext.Context);
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            return false;
```

Figura 3.51: Método de Validación de la Información del Login.
Fuente: Elaborado por el investigador.

```
"type": "Container",
"items": [
  {
    "type": "TextBlock",
    "text": "Bienvenido/a, soy el bot de auto servicios ${nombre}  Agradecemos que te hayas puesto
    "wrap": true,
    "spacing": "Medium",
    "height": "stretch",
    "color": "Dark",
    "weight": "Bolder"
  }
]
```

Figura 3.52: Interpolación en BienvenidaTarjeta.json.
Fuente: Elaborado por el investigador.

Implementación de Diálogos en Cascada

La lógica de programación para estructurar los diálogos en cascada se aplicará en todos los pasos establecidos de cada dialogo, los cuales son: inicial, intermedios y final,

como paso inicial se estableció el dialogo llamado “InitialStepAsync”, esta lógica se explicará a continuación:

Una vez que el usuario se haya logueado exitosamente, el hilo del dialogo procede a ejecutar el método denominado “OptionStepAsync”, este paso corresponde a la lista de opciones que mostrará el dialogo después haber ingresado al sistema satisfactoriamente, este paso funciona de la siguiente forma: primero se obtiene la información del cliente mediante el “UserProfile”, el cual contiene el id, cédula; nombre, apellido, dirección y teléfono del usuario, luego se agrega un “attachments” y un “reply”, los cuales vienen desde el método del cliente “OpcionesBot”, este contiene la carga de la tarjeta adaptiva, la cual contiene el directorio del paths que alberga el archivo JSON del diseño de la interfaz de usuario, este archivo deberá leerse y posteriormente enviar el “myData” que contiene le nombre del cliente y la información al Adaptive Cards y poder retornar un “adaptiveCardAttachment”, el cual deberá agregarse a un “reply”, el cual dispondrá un “PromptAsync” y finalmente enviar a visualizar la carga de la tarjeta adaptiva, en este paso no existe ningún método de validación.

Luego se procese a pasar al siguiente paso denominado “SelectOptionStepAsync”, el cual muestra las opciones y determina cuál de ellas selecciono el usuario, el proceso está compuesto por dos regiones de código: el cliente y las opciones. La región cliente funciona de la misma forma que el paso anterior, obtiene la información del cliente mediante el “UserProfile”, conformado por el id, cédula, nombre, apellido, dirección, teléfono y password del usuario, en la región opciones se debe tener claro la forma de interactuar del usuario, por ejemplo si el usuario selecciona alguna opción que se visualiza dentro de la tarjeta adaptiva a través de algún botón y otra diferente cuando el usuario requiere de algún de mensaje de texto , teniendo claro este panorama se analiza la posibilidad de utilizar Adaptive Cards o LUIS. La región opciones dispone de un método denominado “VerificarJson”, este retorna un “Activity.Text” dentro de un archivo JSON siempre que el usuario seleccione a través de la tarjeta adaptiva cualquier opción, posteriormente se aplica la siguiente condición; si se trata de un archivo JSON se obtiene el Id y nombre del botón responsable de la acción ya que en

el archivo JSON, luego se obtiene el valor del “Activity.Text“ para posteriormente transformarlo en un JSON, el cual pueda capturar el parámetro que contiene el Id, posterior a esta acción se utilizará el comando “command” el cual obtendrá el valor de aquel botón que se seleccionó, estos son:

- LISTARVEHICULOS.
- INGRESARVEHICULO.
- COMPRAR.
- VISUALIZARCOMPRA.
- SALIR.

Dependiendo de esta selección se procede a llamar a los diferentes diálogos, si el usuario selecciono “SALIR” automáticamente se elimina el “UserProfile” y el estado de la sesión, posteriormente se mostrará el dialogo de despedida dentro de un Adaptive Cards, luego se ejecutará el método “FinalStepAsync” el cual mostrará un mensaje sugiriendo realizar otra función adicional caso contario dar por finalizado el dialogo. Este proceso se lo aplica en todas las opciones antes mencionadas. Una vez cumplido todo los pasos anteriores salta al último paso llamado “FinalStepAsync”, este paso obtiene los valores correspondiente de los diversos servicios que se encuentra en la variable “ServicioDetalleLista”, esto se lo realiza mediante una condición: si el usuario ya no desea realizar más funciones se debe finalizar el dialogo, para realizar esta actividad se requiere como parámetro el listado de los servicios y adicional el token de cancelación, caso contrario se mostrara de nuevo el dialogo de Opciones.

A continuación, se muestra un grupo de imágenes correspondientes al código generado durante la programación que corroboran lo antes mencionado:

Figura 3.53: Método del Paso InitialStepAsync.
Fuente: Elaborado por el investigador.

```
1 referencia
private async  Task<DialogTurnResult> OptionStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    UserProfile userProfile = await _botStateService.UserProfileAccessor.GetAsync(stepContext.Context, () => new UserProfile());
    ConversationInfo conversationInfo = await _botStateService.ConversationInfoAccessor.GetAsync(stepContext.Context, () => new ConversationInfo());
    ClienteIES cliente = new ClienteIES();
    cliente.Id = userProfile.Id;
    cliente.Cedula = userProfile.Cedula;
    cliente.Nombre = userProfile.Nombre;
    cliente.Apellido = userProfile.Apellido;
    cliente.Direccion = userProfile.Direccion;
    cliente.Telefono = userProfile.Telefono;
    cliente.Usuario = userProfile.Usuario;
    var attachments = new List<Attachment>();
    var reply = MessageFactory.Attachment(attachments);
    reply.Attachments.Add(Metodos.OpcionesBot(cliente));
    return await stepContext.PromptAsync($"{nameof(MainDialog)}.opciones", new PromptOptions
    {
        Prompt = (Activity)reply
    }, cancellationToken);
}
1 referencia
```

Figura 3.54: Método del Paso OptionStepAsync.
Fuente: Elaborado por el investigador.

```
1 referencia
private async Task<DialogTurnResult> SelectOptionStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    #region Cliente
    UserProfile userProfile = await _botStateService.UserProfileAccessor.GetAsync(stepContext.Context, () => new UserProfile());
    ConversationInfo conversationInfo = await _botStateService.ConversationInfoAccessor.GetAsync(stepContext.Context, () => new ConversationInfo());
    ClienteIES cliente = new ClienteIES();
    cliente.Id = userProfile.Id;
    cliente.Cedula = userProfile.Cedula;
    cliente.Nombre = userProfile.Nombre;
    cliente.Apellido = userProfile.Apellido;
    cliente.Direccion = userProfile.Direccion;
    cliente.Telefono = userProfile.Telefono;
    cliente.Usuario = userProfile.Usuario;
    cliente.Usuario = userProfile.Usuario;
    cliente.Password = "";

    #endregion

    #region Opciones
    var verificar = Metodos.VerificarJson(stepContext.Context.Activity.Text);
    if (verificar)
    {
        JToken commandToken = JToken.Parse(stepContext.Context.Activity.Value.ToString());
        string command = commandToken["id"].Value<string>();
        var attachments = new List<Attachment>();
        var reply = MessageFactory.Attachment(attachments);

        switch (command)
        {
            case "LISTARVEHICULOS":
                return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.ListarVehiculos", null, cancellationToken);
            case "INGRESOVEHICULO":
                return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.IngresarVehiculo", null, cancellationToken);
            case "COMPRAR":
                return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.Factura", null, cancellationToken);
            case "VISUALIZARCOMPRA":
                return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.ListarFactura", null, cancellationToken);
            case "SALIR":

                await _botStateService.UserProfileAccessor.DeleteAsync(stepContext.Context);
                await _botStateService.ConversationInfoAccessor.DeleteAsync(stepContext.Context);
                await _botStateService.UserState.DeleteAsync(stepContext.Context);
                await _botStateService.ConversationState.DeleteAsync(stepContext.Context);
                //ad salir
                 reply = MessageFactory.Attachment(Metodos.Salir());
                await stepContext.Context.SendActivityAsync(reply, cancellationToken);
                return await stepContext.NextAsync(null, cancellationToken);
            default:
                return await stepContext.NextAsync(null, cancellationToken);
        }
```

Figura 3.55: Método SelectOptionStepAsync Parte 1.
Fuente: Elaborado por el investigador.

```
        else
        {
            #region Luis
            RecognizerResult recognizerResult = await luisService.LuisRecognizer.RecognizeAsync(
                stepContext.Context, cancellationToken
                );
            (string intent, double score) topIntent = recognizerResult.GetTopScoringIntent();
            if (!String.IsNullOrEmpty(topIntent.intent))
            {
                switch (topIntent.intent.ToUpper())
                {
                    case "LISTARVEHICULOS":
                        return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.ListarVehiculos", null, cancellationToken);
                    case "INGRESOVEHICULOS":
                        return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.IngresarVehiculo", null, cancellationToken);
                    case "COMPRAR":
                        return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.Factura", null, cancellationToken);
                    case "VISUALIZARCOMPRA":
                        return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.ListarFactura", null, cancellationToken);
                    case "CANCELAR":
                        await _botStateService.UserProfileAccessor.DeleteAsync(stepContext.Context);
                        await _botStateService.ConversationInfoAccessor.DeleteAsync(stepContext.Context);
                        await _botStateService.UserState.DeleteAsync(stepContext.Context);
                        await _botStateService.ConversationState.DeleteAsync(stepContext.Context);
                        var reply = MessageFactory.Attachment(Metodos.Salir());
                        await stepContext.Context.SendActivityAsync(reply, cancellationToken);

                        return await stepContext.NextAsync(null, cancellationToken);
                    default:
                        return await stepContext.NextAsync(null, cancellationToken);
                }
            }
            else
            {
                return await stepContext.NextAsync(null, cancellationToken);

            }
            #endregion

        }
        #endregion
    }
```

Figura 3.56: Método SelectOptionStepAsync Parte 2.
Fuente: Elaborado por el investigador.

```
1 referencia
private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
  await stepContext.Context.SendActivityAsync(MessageFactory.Text("Si deseas realizar otra opcion digite cualquier cosa para iniciar nuevamente el proceso. "), cancellationToken);
   return await stepContext.EndDialogAsync(null, cancellationToken);
}
```

Figura 3.57: Método del Paso FinalStepAsync.
Fuente: Elaborado por el investigador.

```
//Inicializa el dialogo
1 referencia
private void InitWaterfallDialogs()
{
    WaterfallStep[] waterfallSteps = new WaterfallStep[]
    {
        InitialStepAsync,
        OptionStepAsync,
        SelectOptionStepAsync,
        FinalStepAsync
    };
    AddDialog(new InicionSeccionDialog($"{nameof(MainDialog)}.InicioSeccion", _botStateService));
    AddDialog(new ListarVehiculosDialog($"{nameof(MainDialog)}.ListarVehiculos", _botStateService,luisService));
    AddDialog(new AgregarVehiculoDialog($"{nameof(MainDialog)}.IngresarVehiculo", _botStateService,luisService));
    AddDialog(new FacturaDialog($"{nameof(MainDialog)}.Factura", _botStateService,luisService));
    AddDialog(new ListarFacturaDialog($"{nameof(MainDialog)}.ListarFactura", _botStateService, luisService));
    AddDialog(new AgregarServicioDialog($"{nameof(MainDialog)}.AgregarServico", _botStateService,luisService));


    AddDialog(new TextPrompt($"{nameof(MainDialog)}.opciones"));


    AddDialog(new WaterfallDialog($"{nameof(MainDialog)}.MainFlow", waterfallSteps));
    InitialDialogId = $"{nameof(MainDialog)}.MainFlow";
}
```

Figura 3.58: Constructor donde se Inicializan los Diálogos.
Fuente: Elaborado por el investigador.

```
http://adaptivecards.io/schemas/adaptive-card.json
{
  "type": "AdaptiveCard",
  "body": [
    {
      "type": "Container",
      "items": [
        {
          "type": "TextBlock",
          "text": "Hola! ${nombre} , soy ${nombreBot} y estoy aqui par ayudarte, selecciona una de las siguientes opciones o dime en que puedo ayudarte.",
          "wrap": true,
          "height": "stretch",
          "color": "Dark",
          "weight": "Bolder"
        }
      ],
      "style": "warning"
    }
  ],
  "actions": [
    {
      "type": "Action.Submit",
      "title": "Listar Vehiculos",
      "data": {
        "id": "LISTARVEHICULOS"
      },
      "iconUrl": "https://cdn2.iconfinder.com/data/icons/xomo-basics/128/document-03-512.png"
    },
    {
      "type": "Action.Submit",
      "title": "Ingresar Vehiculos",
      "data": {
        "id": "INGRESOVEHICULO"
```

Figura 3.59: Archivo JSON del Dialogo de Opciones.
Fuente: Elaborado por el investigador.

Implementación de LUIS de Azure

Es un servicio de Microsoft basado en aprendizaje automático, su objetivo principal es desarrollar la comprensión del lenguaje natural en bots, al ser un procesador de lenguaje natural potente utiliza técnicas que ayudan en la comprensión del lenguaje, una de ellas es el uso de intenciones. Las intenciones son tareas que el usuario desea realizar, el uso de estas tareas es fundamental al momento de desarrollar flujos de conversación. **[34]**

LUIS define un conjunto de intenciones, cada intención pertenece a una acción específica, para el entrenamiento del bot se estableció de la siguiente manera las actividades con sus respectivas intenciones:

Acciones	**Intenciones**
Cancelar	nada, abortar, ya no quiero comprar, cancelar, salir.
Comprar	comprar, comprar un servicio.
Confirmar	positivo, deacuerdo, ok, sí.
IngresoVehiculo	ingresar vehículos, ingresar vehículo
ListarVehiculos	quiero ver los vehículos, muéstrame los vehículos.
Negar	no quiero, negativo, no.
None	nada, ninguno.
VisualizarCompra	quiero ver mis compras, ver factura, factura.

Tabla 3.65: Actividades con sus respectivas Intenciones.
Fuente: Elaborado por el investigador.

Estas intenciones son sometidas a entrenamiento con la intención de aumentar la capacidad de procesamiento del bot. En la figura 3.60, se visualiza la lista de intenciones creadas desde LUIS de Azure.

A continuación, en la figura 3.61, se muestra la acción Compra con sus respetivas intenciones.

Se aplico entrenamiento al bot, creando una nueva intención denominada “factura” para la acción “VisualizarCompra”, como se muestra en las figuras 3.62 y 3.63.

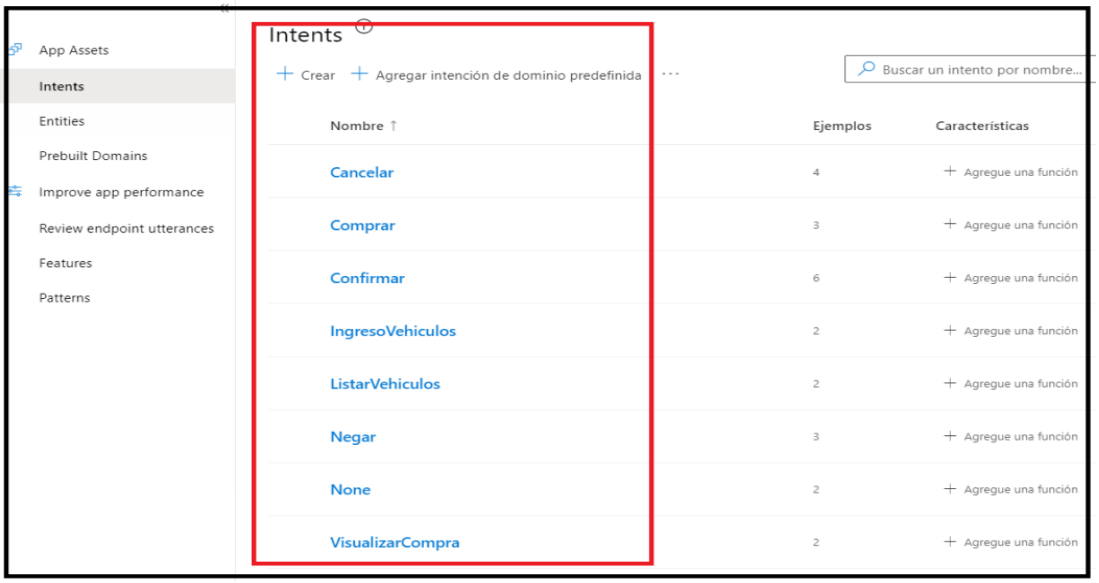


Figura 3.60: Lista de Intenciones en LUIS.
Fuente: Elaborado por el investigador.

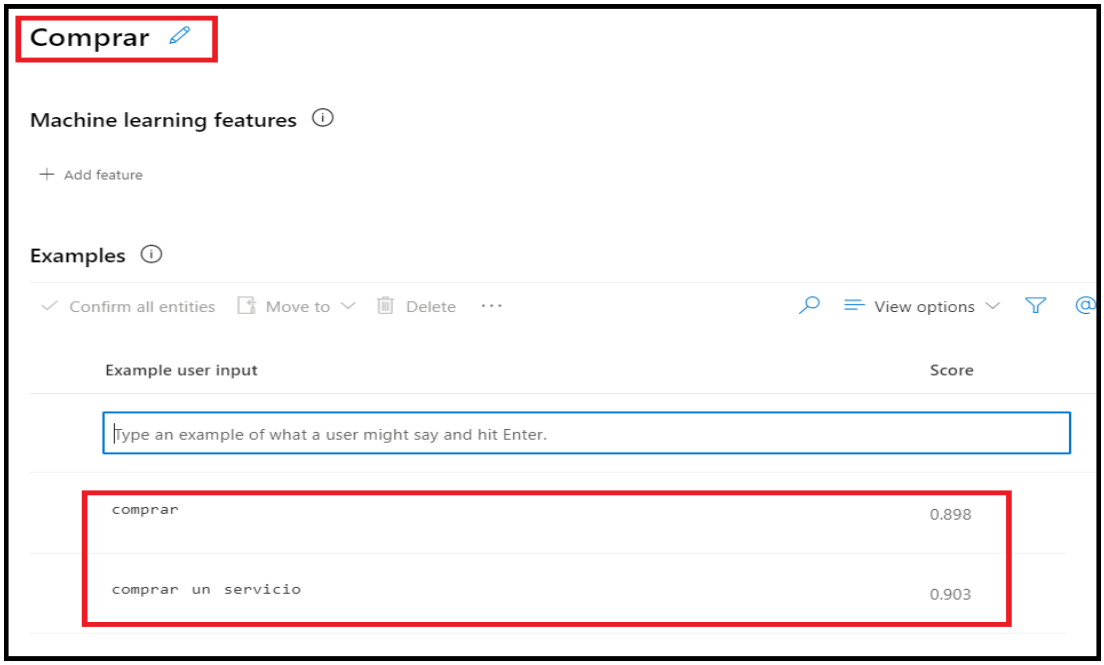


Figura 3.61: Acción Comprar con sus respectivas Intenciones.
Fuente: Elaborado por el investigador.

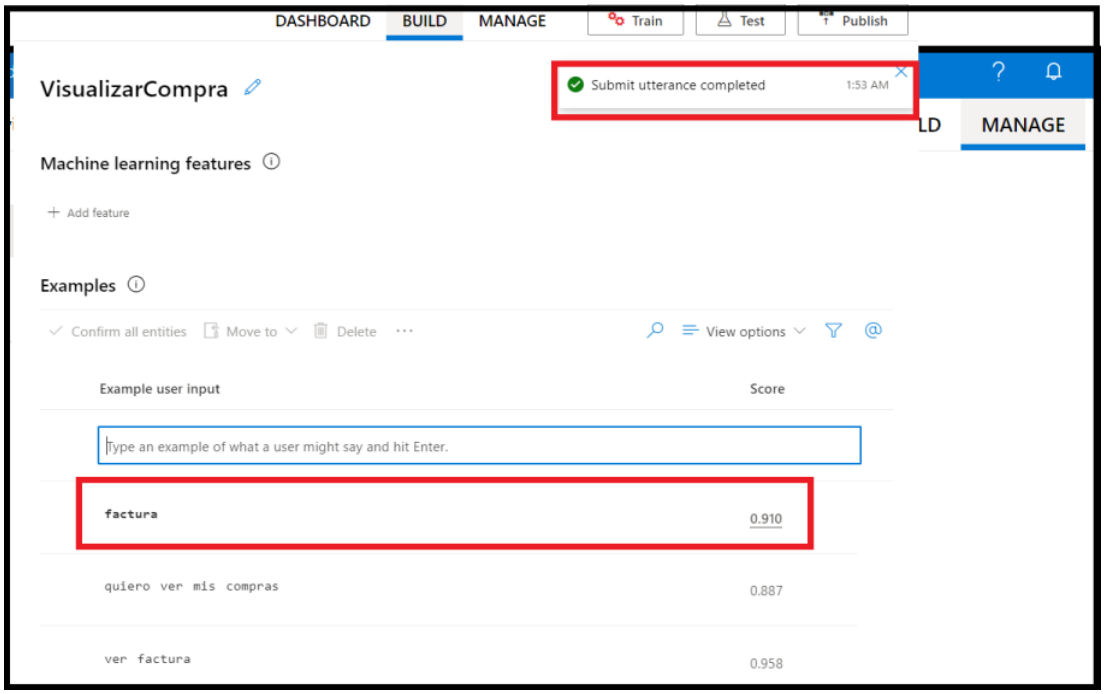


Figura 3.62: Entrenamiento del Bot Parte 1.
Fuente: Elaborado por el investigador.

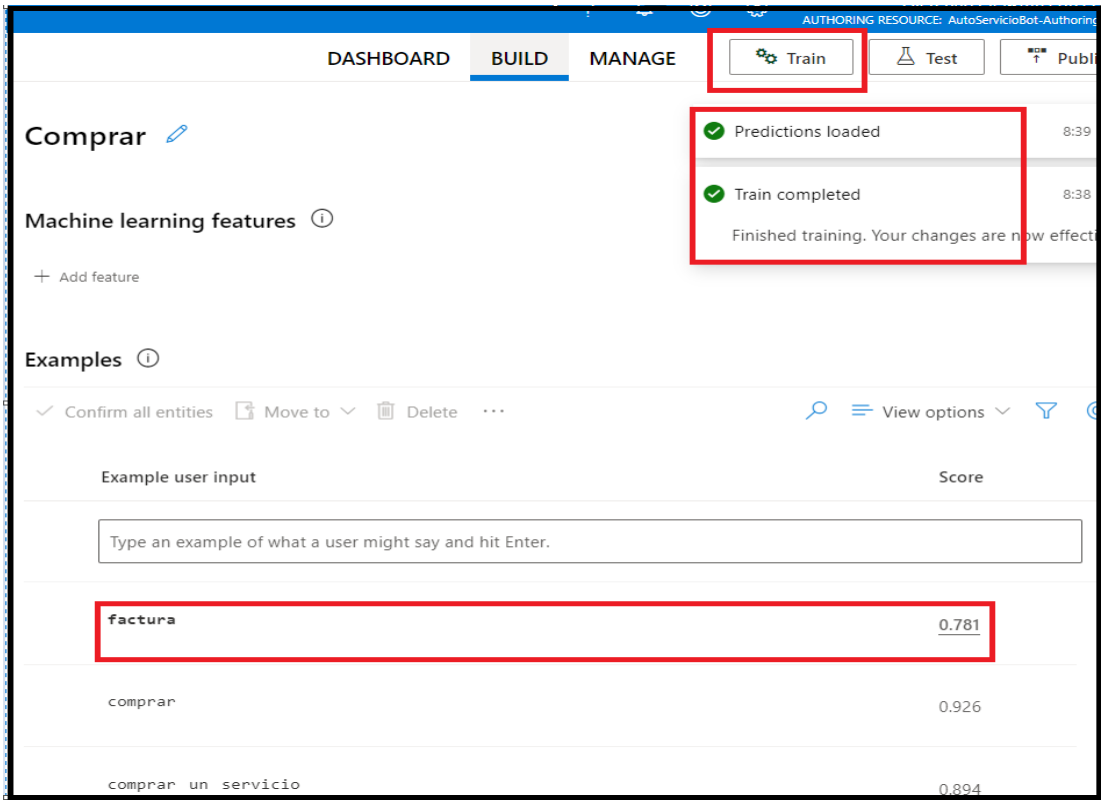


Figura 3.63: Entrenamiento del Bot Parte 2.
Fuente: Elaborado por el investigador.

Consumo de LUIS en el Cliente

Una vez establecido las actividades y sus respectivas intenciones y posteriormente entrenado el bot , lo siguiente será instalar el paquete de LUIS desde el Administrador NuGet llamado “Microsoft.Bot.Builder.AI.Luis”, ya instalado todas las dependencias correspondientes se procede agregar una nueva clase llamada “LuisService.cs” dentro de la carpeta “Servicios”, la cual servirá para agregar el servicio , dentro de la clase se crea una instancia de tipo “LuisRecognizer” que es propia del paquete antes instalado, esta instancia tendrá el mismo nombre y dentro del constructor del servicio se utilizara inyección de dependencias para tener acceso a las configuraciones que están establecidas en el servidor de LUIS, se utiliza la propiedad “LuisRecognizer” al cual deberá contener opciones, posterior a esto se agrega las respectivas configuración en el archivo “appsettings.json” basándose de la estructura de ASP:NET Core, esta información corresponde al ID de LUIS que se encuentra en la página de LUIS.io en el apartado manage, como se muestra a continuación en la figura 3.64:

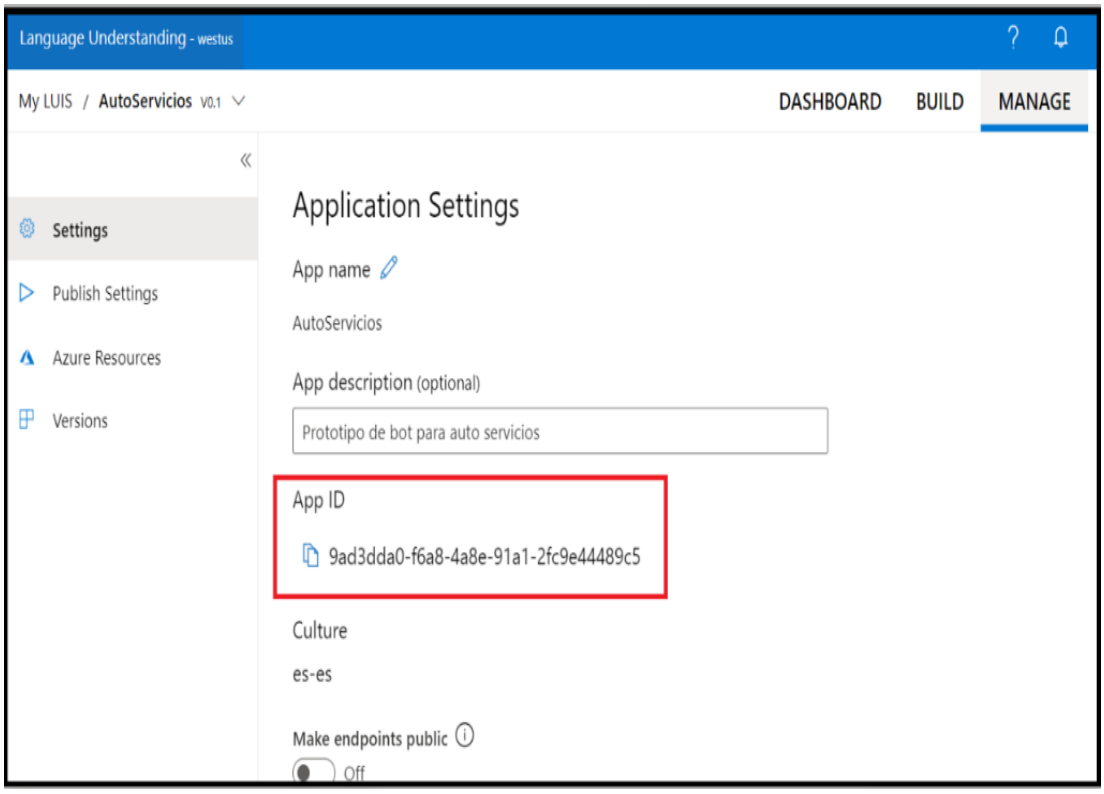


Figura 3.64: Código ID de LUIS.
Fuente: Elaborado por el investigador.

Una vez copiado este código se procede a pegarlo dentro del archivo "appsettings.json", de la misma forma se debe ingresar el Primary Key y Endpoint que se encuentra en la opción "Azure Recurses" de la misma página:

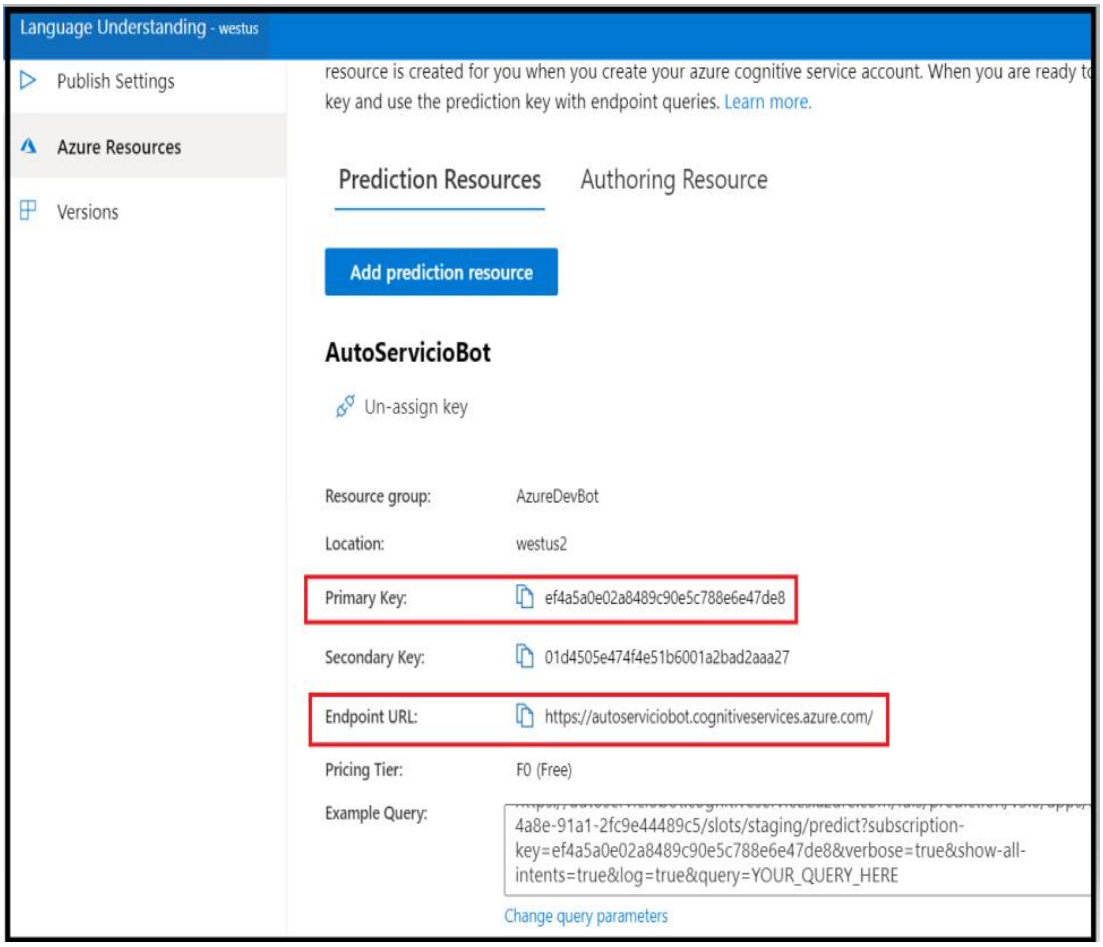


Figura 3.65: Primary Key y Endpoint de LUIS.
Fuente: Elaborado por el investigador.

El último dato que se requiere es la URL (Uniform Resource Locator), esta se debe construir manualmente dentro del constructor de "LuisService.cs", primero se debe pasar la siguiente información: App ID, Primar Key y Endpoint, este último dato se lo debe pasar a través de una interpolación de cadenas, como se muestra en la siguiente figura 3.66.

Finalmente el archivo "appsettings.json", quedaría configurado como se puede visualizar en la figura 3.67:

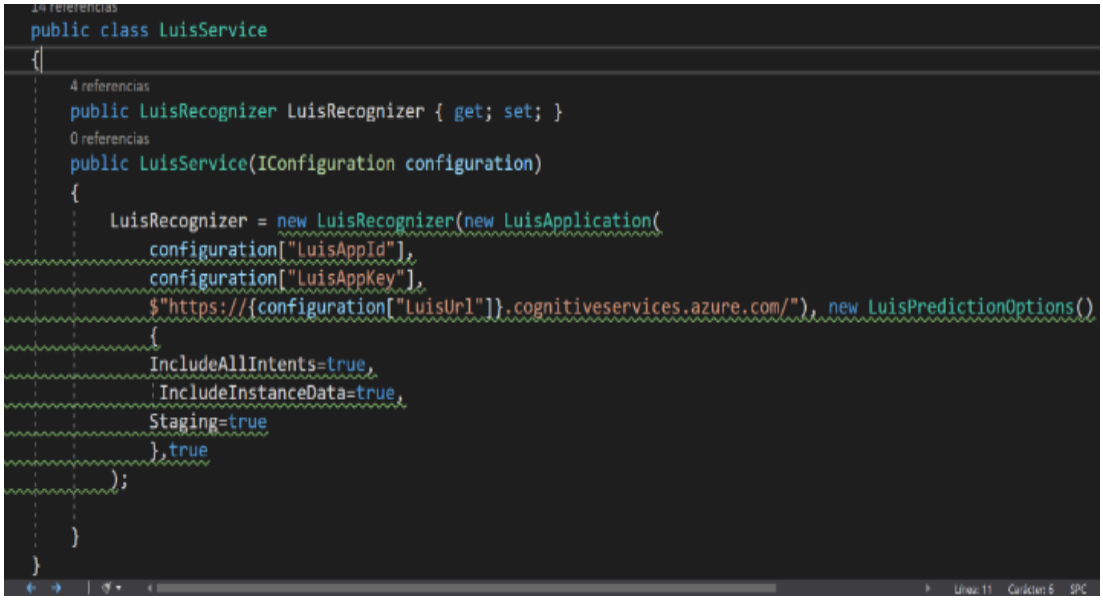


Figura 3.66: Constructor de “LuisService.cs”.
Fuente: Elaborado por el investigador.

Figura 3.67: Archivo Modificado “appsettings.json”.
Fuente: Elaborado por el investigador.

Posterior a esto se necesita registrar la clase “LuisService.cs” dentro del archivo “Startup.cs”, como se visualiza a continuación en la figura 3.68.

Para que el “MainDialog.cs” soporte estas nuevas actualizaciones se debe modificar como se muestra en la figura 3.69.

Además, se necesita modificar el método “VerificarJson” ya que anteriormente se le asigno dos regiones de código, uno de ellos corresponde a LUIS, dentro de esta región se aplicarán las siguientes modificaciones:

Para el reconocimiento de “intenciones” previamente entrenadas en el servicio de LUIS, se llama al servicio de LUIS a través de la propiedad “recognizerResult”, este debe contener la intención y puntaje para posteriormente almacenar estos datos dentro de la variable “topIntent”, luego se toma la intención y se la aplica una conversión del texto a mayúsculas con la propiedad “ToUpper”, posterior a esta conversión se necesita de la creación de un “case” para que controle la selección de las opciones, por ejemplo si se selecciona la primera opción la cual es “LISTARVEHICULOS” automáticamente debe enviar la intención “listarvehiculos”, como ya se tiene entrenado las frases en el servicio reconocerá sin problema la función que necesita realizar el usuario. A continuación, se visualiza la región de código correspondiente a LUIS en la figura 3.70:

```
public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to th
0 referencias
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddNewtonsoftJson();


    // Create the Bot Framework Adapter with error handling enabled.
    services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();
    services.AddSingleton<IStorage,MemoryStorage>();
    services.AddSingleton<UserState>();
    services.AddSingleton<ConversationState>();
    services.AddSingleton<DialogState>();
    services.AddSingleton<BotStateService>();
    services.AddSingleton<MainDialog>();
    services.AddSingleton<LuisService>();
```

Figura 3.68: Registro de la Clase “LuisService.cs” en “Startup.cs”.
Fuente: Elaborado por el investigador.

```
0 referencias
public MainDialog(BotStateService botStateService, LuisService luisService)
{
    _botStateService = botStateService ?? throw new ArgumentException(nameof(botStateService));
    this.luisService = luisService ?? throw new ArgumentException(nameof(luisService));
    InitWaterfallDialogs();
}
```

Figura 3.69: Clase “MainDialog.cs”.
Fuente: Elaborado por el investigador.

```
#region Luis
RecognizerResult recognizerResult = await luisService.LuisRecognizer.RecognizeAsync(
    stepContext.Context, cancellationToken
    );
(string intent, double score) topIntent = recognizerResult.GetTopScoringIntent();
if (!String.IsNullOrEmpty(topIntent.intent))
{
    switch (topIntent.intent.ToUpper())
    {
        case "LISTARVEHICULOS":
            return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.ListarVehiculos", null, cancellationToken);
        case "INGRESOVEHICULOS":
            return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.IngresarVehiculo", null, cancellationToken);
        case "COMPRAR":
            return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.Factura", null, cancellationToken);
        case "VISUALIZARCOMPRA":
            return await stepContext.BeginDialogAsync($"{nameof(MainDialog)}.ListarFactura", null, cancellationToken);
        case "CANCELAR":
            await _botStateService.UserProfileAccessor.DeleteAsync(stepContext.Context);
            await _botStateService.ConversationInfoAccessor.DeleteAsync(stepContext.Context);
            await _botStateService.UserState.DeleteAsync(stepContext.Context);
            await _botStateService.ConversationState.DeleteAsync(stepContext.Context);
            var reply = MessageFactory.Attachment(Metodos.Salir());
            await stepContext.Context.SendActivityAsync(reply, cancellationToken);

            return await stepContext.NextAsync(null, cancellationToken);
        default:
            return await stepContext.NextAsync(null, cancellationToken);
    }
}
else
{
    return await stepContext.NextAsync(null, cancellationToken);

}
#endregion
```

Figura 3.70: Región de Código de LUIS.
Fuente: Elaborado por el investigador.

3.2.4.7 Implementación en Entorno Local (Internet Information Services)

Con respecto al despliegue del bot se lo realizara mediante un entorno local, es decir desde localhost. Una vez realizado el despliegue se aplicarán las pruebas necesarias de las funcionalidades.

Para realizar la implementación local se debe tener activado el IIS (Internet Information Services) si se encuentra deshabilitada esta función. Se debe ingresar al panel de control, en dicha pantalla encontramos una serie de opciones, se debe ingresar a la opción "Activar o desactivar características de Windows" como se visualiza en la presenta figura 3.71, al dar clic en esta opción se desplegará una interfaz como se muestra en la figura 3.72:

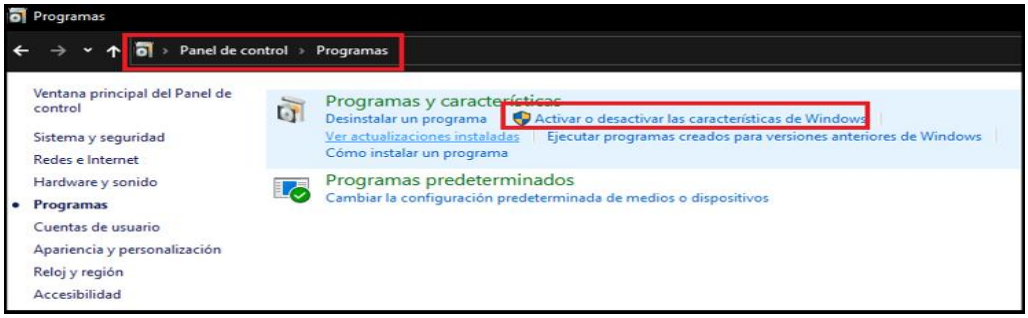


Figura 3.71: Panel de Control.
Fuente: Elaborado por el investigador.

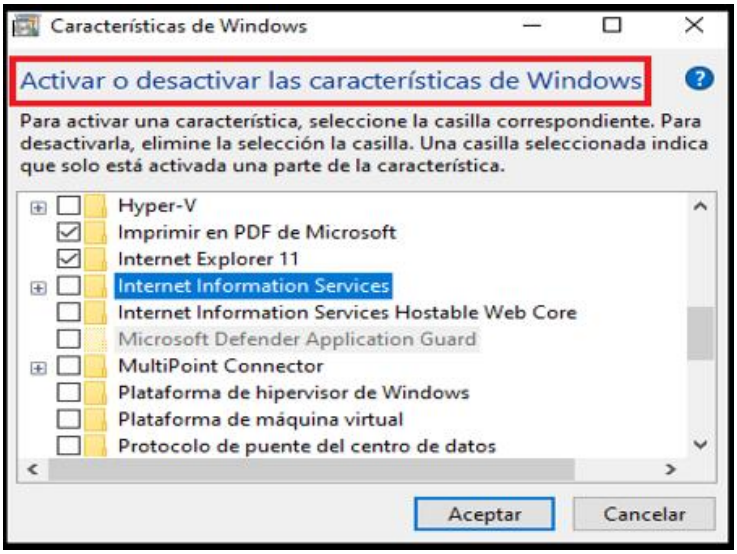


Figura 3.72: Activar o Desactivar Características de Windows. Fuente: Elaborado por el investigador.

Una vez realizada esta acción debemos activar las opciones por defecto del ISS, por lo cual se debe activar la opción “Internet Information Services”, como se visualiza en la siguiente figura 3.73, posteriormente se debe aceptar y esperar que se realicen las configuraciones pertinentes.

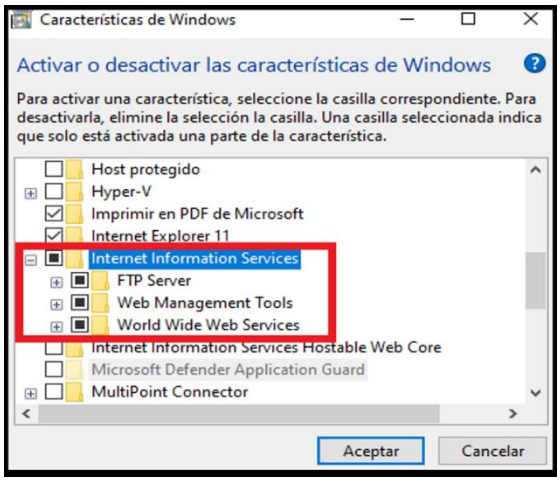


Figura 3.73: Habilitar Internet Information Services. (ISS).
Fuente: Elaborado por el investigador.

Ya activado el ISS, se debe realizar la publicación de la aplicación en un directorio local, la forma directa para realizar esta accione es dar clic derecho sobre la aplicación y escoger la opción que dice “Publicar”. Posteriormente, se debe crear un perfil el cual servirá para que se realice la publicación como se visualiza en la siguiente figura 3.74 y finalmente dar clic en publicar.

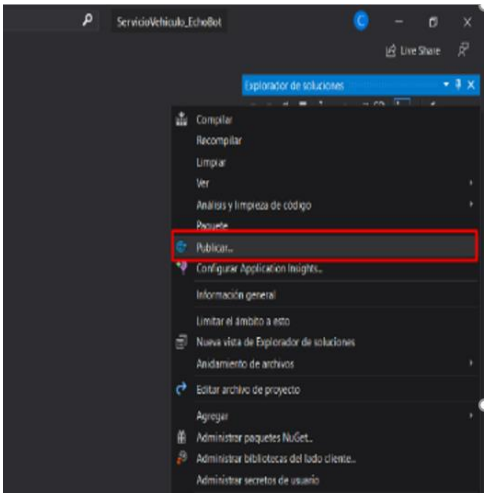


Figura 3.74: Publicación de la Aplicación.
Fuente, Elaborado por el investigador.

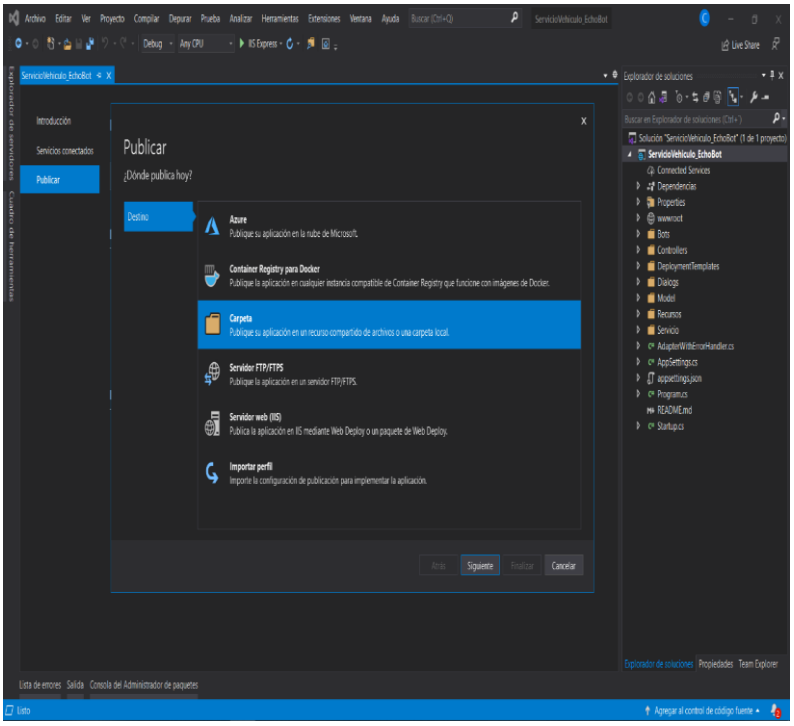


Figura 3.75: Asignación del Directorio Local donde se va Realizar la Publicación.
Elaborado por el investigador.

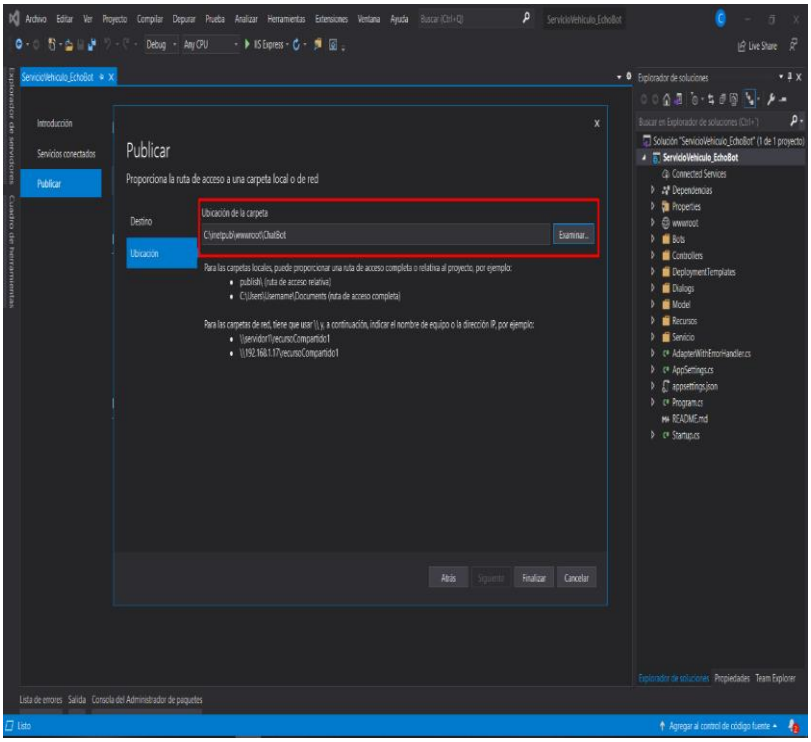


Figura 3.76: Publicación de la Aplicación en Directorio Local.
Elaborado por el investigador.

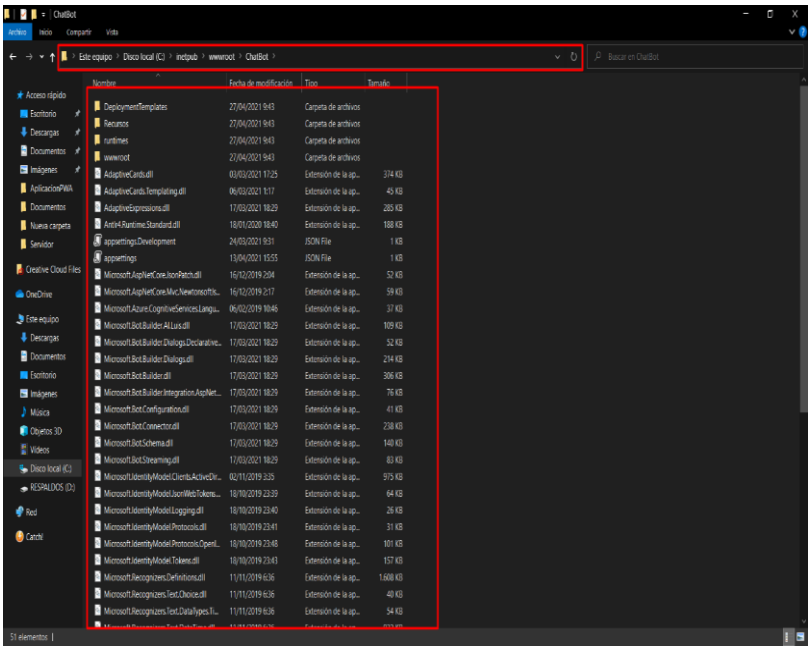


Figura 3.77: Visualización del Directorio Local.
Fuente: Elaborado por el investigador.

Una vez realizada la publicación correspondiente, se procede a configurar esta publicación dentro del ISS, para realizar esta acción se debe ingresar al Administrador del Internet Information Services. A continuación, se debe dar clic derecho sobre "Agregar sitio web" y posteriormente se visualizar una interfaz donde se debe configurar el respectivo sitio web que contendrá la aplicación como se visualiza en la siguiente figura 3.78.

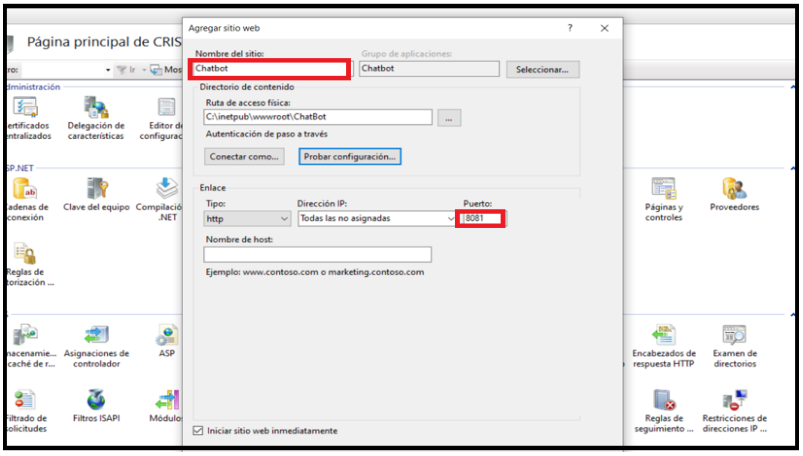


Figura 3.78: Configuración del Nuevo Sitio Web en IIS.
Fuente: Elaborado por el investigador.

Por defecto reconocerá automáticamente todas las carpetas que se encuentran dentro del directorio que se estableció desde un inicio para el sitio web. A continuación, se debe dar clic derecho en la opción “Convertir en aplicación”, como se visualiza en la siguiente figura 3.79.

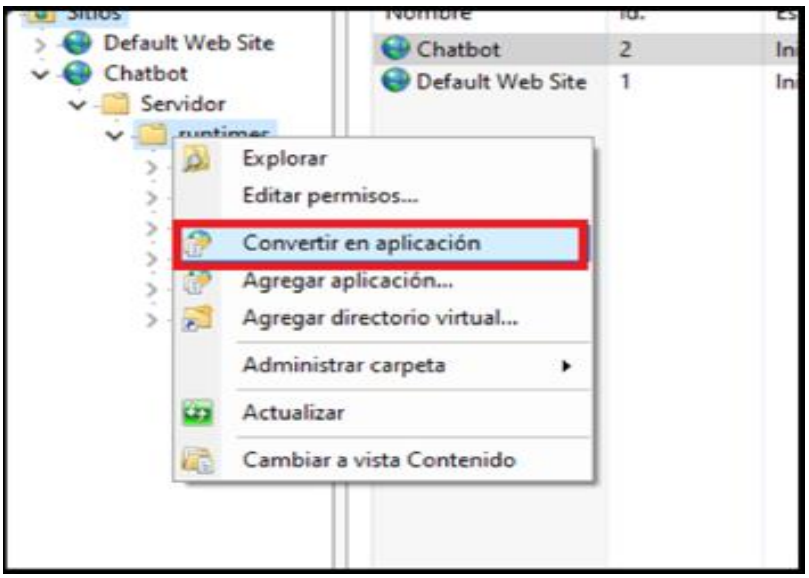


Figura 3.79: Convertir en Aplicación.
Fuente: Elaborado por el investigador.

Una vez finalizado el proceso de publicación, la aplicación estará lista para realizar las pruebas necesarias desde el emulador y obligatoriamente se deberá escribir la dirección URL perteneciente a la ubicación donde se encuentra publicado el chatbot, posteriormente se debe dar clic en “Save and connect”, como se muestra en la siguiente figura 3.80. Finalmente se podrá visualizar el chatbot desplegado localmente, como se muestra en la siguiente figura 3.81:

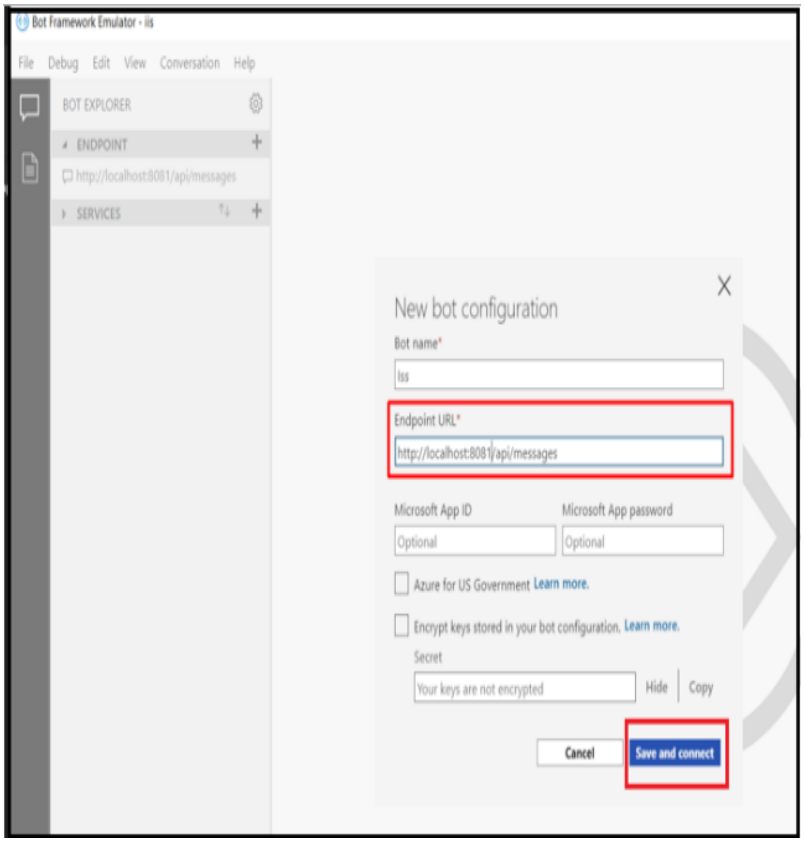


Figura 3.80: Emulador de Bot Framework.
Fuente: Elaborado por el investigador.

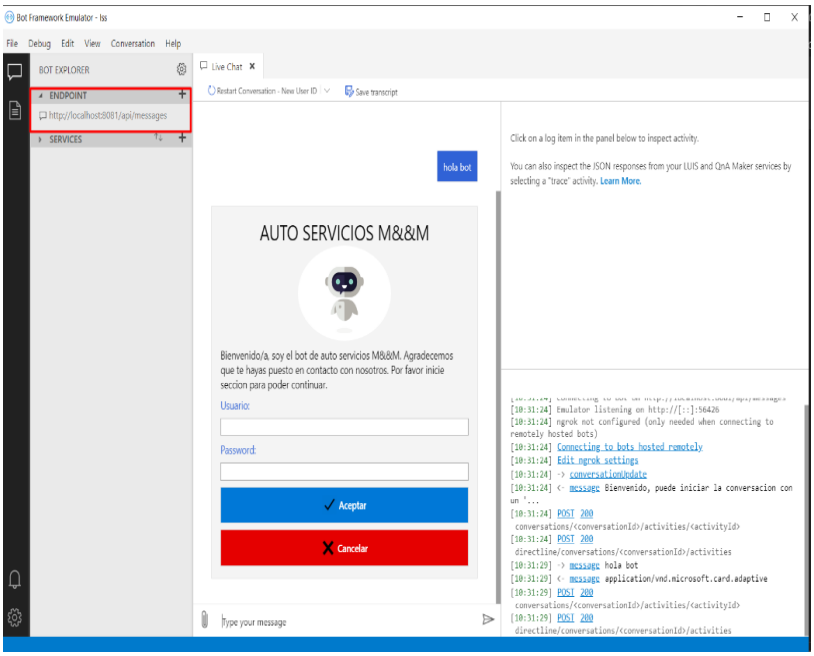


Figura 3.81: Chatbot Desplegado en Entorno Local.
Fuente: Elaborado por el investigador.

3.2.5 Fase 4.-Pruebas de Funcionalidad

La Fase de Pruebas es una parte esencial de la Metodología XP ya que permite comprobar el estado de cada funcionalidad del sistema. **Además**, comprueba si se obtuvo el producto final deseado. Para comprobar estos procesos se realizaron pruebas a cada historia de usuario.

Prueba de Aceptación	
Código: P1	Código de historia: H1, Diálogo de inicio
Descripción: El chatbot iniciará el dialogo una vez detecte el ingreso de algún texto por parte del usuario, si el servicio no está disponible el agente virtual presentará un texto notificando al usuario que el servicio no está disponible por el momento, caso contario el chatbot mostrará un formulario de logueo en un Adaptive Cards (tarjetas adaptables) el cual deberá autenticarse exitosamente para poder realizar cualquier funcionalidad.	
Condición de ejecución: El usuario deberá inicializar al chatbot.	
Entrada: -Escribir cualquier palabra en el chat	
Resultados: Tarjeta adaptable a manera de formulario de login.	
Evaluación de prueba: La prueba se realizó de forma satisfactoria.	

Tabla 3.66: Prueba 1 – Diálogo de Inicio.
Fuente: Elaborado por el investigador**.**

Prueba de Aceptación	
Código: P2	Código de historia: H3, Diálogo de ingreso al sistema
Descripción: El chatbot permitirá la autenticación del usuario para lo cual mostrara un Adaptive Cards (tarjeta adaptable) con un formulario de ingreso los cuales contendrán los parámetros: Usuario y Contraseña.	
Condición de ejecución: El usuario obligatoriamente debe estar registrado en el sistema.	
Entrada: -Ingresar el usuario y contraseña -Dar clic dentro del botón Aceptar	
Resultados: Tarjeta adaptable a manera de mensaje notificando al usuario que se autentico exitosamente o no en el sistema. Si el usuario se autentico exitosamente mostrara automáticamente el listado de funcionalidades.	
Evaluación de prueba: La prueba se realizó de forma satisfactoria.	

Tabla 3.67: Prueba 2 – Diálogo de Ingreso al Sistema.
Fuente: Elaborado por el investigador.

Prueba de Aceptación	
Código: P3	Código de historia: H4, Diálogo de consulta de órdenes de compra
Descripción: El chatbot permitirá consultar las órdenes de compra que se encuentran guardadas en el sistema de servicios vehiculares.	
Condición de ejecución: El usuario deberé autenticarse exitosamente y posteriormente ingresar el número de orden de compra.	
Entrada: -Ingresar el número de orden de compra.	
Resultados: Dialogo mostrando el detalle de la orden de compra.	
Evaluación de prueba: La prueba se realizó de forma satisfactoria.	

Tabla 3.68: Prueba 3 – Diálogo para Consultar Vehículos.
Fuente: Elaborado por el investigador.

Prueba de Aceptación	
Código: P4	Código de historia: H5, Diálogo para consultar los vehículos.
Descripción: El chatbot permitirá consultar los vehículos registrados en el sistema de servicios vehiculares.	
Condición de ejecución: El usuario deberá estar autenticado exitosamente y tener por lo menos un vehículo registrado.	
Entrada: -Clic sobre el botón Listar Vehículos -Escribir la palabra Listar Vehículos	
Resultados: Lista de vehículos dentro de tarjetas adaptables.	
Evaluación de prueba: La prueba se realizó de forma satisfactoria.	

Tabla 3.69: Prueba 4 – Diálogo para Seleccionar un Vehículo.
Fuente: Elaborado por el investigador.

Prueba de Aceptación	
Código: P5	Código de historia: H6, Diálogo para realizar una orden de compra
Descripción: El chatbot permitirá al usuario realizar una orden de compra de los servicios vehiculares.	
Condición de ejecución: El usuario deberá seleccionar un vehículo por lo menos, y uno o más servicios vehiculares para posteriormente confirmar la finalización de la compra.	
Entrada: -Clic sobre el botón Comprar -Escribir la palabra Comprar	
Resultados: El detalle de la orden de compra dentro de una tarjeta adaptable.	
Evaluación de prueba: La prueba se realizó de forma satisfactoria.	

Tabla 3.70: Prueba 5 – Diálogo para Realizar una Orden de Compra.
Fuente: Elaborado por el investigador.

Prueba de Aceptación	
Código: P6	Código de historia: H7, Diálogo para guardar un nuevo vehículo
Descripción: El chatbot permitirá al usuario ingresar un nuevo vehículo al sistema de servicios vehiculares.	
Condición de ejecución: El usuario deberá estar registrado en el sistema.	
Entrada: -Clic sobre el botón Ingresar Vehículos -Escribir la palabra ingresar Vehículos	
Resultados: Texto confirmando que se ingresó correctamente el vehículo.	
Evaluación de prueba: La prueba se realizó de forma satisfactoria.	

Tabla 3.71: Prueba 6 – Diálogo para Guardar un Nuevo Vehículo.
Fuente: Elaborado por el investigador.

Prueba de Aceptación	
Código: P7	Código de historia: H8, Diálogo para visualizar la orden de compra
Descripción: Una vez finalizado la compra el chatbot mostrar la orden de compra que se realizó.	
Condición de ejecución: Preguntar al usuario si desea agregas más servicios vehiculares.	
Entrada: -Escribir la palabra No	
Resultados: Detalle de la orden de compra dentro de una tarjeta adaptable.	
Evaluación de prueba: La prueba se realizó de forma satisfactoria.	

Tabla 3.72: Prueba 7 – Diálogo para Visualizar la Orden de Compra.
Fuente: Elaborado por el investigador.

Prueba de Aceptación	
Código: P8	Código de historia: H2, Diálogo de fin
Descripción: El chatbot presentara un dialogo final en forma de texto como despedida.	
Condición de ejecución: El usuario deberá haber culminado la conversación con el chatbot.	
Entrada: -Mensaje indicando que ya no desea realizar algo más	
Resultados: Diálogo final notificando que finalizo la conversación.	
Evaluación de prueba: La prueba se realizó de forma satisfactoria.	

Tabla 3.73: Prueba 8 – Diálogo de Fin.
Fuente: Elaborado por el investigador.

Prueba de aceptación	
Código: P9	Código de historia: H9, Entrenamiento del procesador de lenguaje natural
Descripción: El procesador de lenguaje natural (PLN) será entrenado con frases de prueba.	
Condición de ejecución: El programador responsable del desarrollo del prototipo será en el encargado de definir el conjunto de frases que se adapten a las funcionalidades del sistema.	
Entrada: -Frases comunes definidas	
Resultados: Correcto desempeño del chatbot.	
Evaluación de prueba: La prueba se realizó de forma satisfactoria.	

Tabla 3.74: Prueba 9 – Entrenamiento del Procesador de Lenguaje Natural.
Fuente: Elaborado por el investigador.

3.2.5.1 Pruebas de Funcionamiento desde Entorno Local

Con la finalidad de garantizar el funcionamiento óptimo del Chatbot se realizaron pruebas de funcionamiento desde el entorno local donde se desplego el bot. El objetivo de aplicar estas pruebas es de evidenciar errores, las evaluaciones fueron aplicadas a cada dialogo de comunicación, verificando que se cumplan todas las funcionalidades asignadas. Los resultados obtenidos de las evaluaciones se pueden visualizar en la tabla 3.75.

N.º	**Flujo de comunicación**	**Evaluación de la prueba**
1	Diálogo de inicio	Prueba exitosa
2	Diálogo de fin	Prueba exitosa
3	Diálogo de ingreso al sistema	Prueba exitosa
4	Diálogo de consulta de órdenes de compra	Prueba exitosa
5	Diálogo para consultar los vehículos	Prueba exitosa
6	Diálogo para realizar una orden de compra	Prueba exitosa
7	Diálogo para guardar un nuevo vehículo	Prueba exitosa
8	Diálogo para visualizar la orden de compra	Prueba exitosa

Tabla 3.75: Resultados de las Pruebas de Funcionamiento.
Fuente: Elaborado por el investigador.

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

- Previo al análisis aplicado de algunas tecnologías idóneas para el desarrollo y despliegue de bots, se tomó la decisión de utilizar Bot Framework, para la construcción del chatbot, porque permite combinar varias tecnologías como: Adaptive Cards para el diseño de las interfaces de usuario y Luis de Azure para el Procesamiento de Lenguaje Natural.

- El Flujo de Trabajo del prototipo se basa en una técnica de diálogos en cascada, ya que, al tratarse de un Chatbot, su interfaz de usuario tiene limitaciones y gracias a la aplicación de dicha técnica se pudo llevar un control adecuado de los diálogos.

- Gracias a la aplicación de una Metodología Ágil, se pudo llevar un control optimo del desarrollo del prototipo, ya q esta se ajusta perfectamente a la naturaleza del proyecto.

- El uso de las Adaptive Cards ayudo en el diseño de las interfaces que requiere los diálogos del Chatbot, convirtiéndolas en interfaces más dinámicas, mejorando la experiencia de interacción con el usuario.

4.2. Recomendaciones

- Se sugiere el uso de la tecnología Bot Framework, porque permite la incorporación de varias tecnologías en un mismo proyecto, logrando una independencia al momento de construir un bot.

- Se recomienda utilizar la técnica Diálogos en Cascada, porque me permite estructurar los diálogos de forma ordenada y sencilla.

- Se aconseja utilizar la tecnología Adaptive Cards en el diseño de las interfaces de usuario, porque las convierte en interfases más dinámicas, ya que estas permiten añadir componentes extras, tales como: imágenes, botones, tablas, formularios y multimedia.

- Antes de replicar las funcionalidades de un sistema genérico, a través de un chatbot se debe analizar a fondo los flujos de trabajo, ya que los bots tienen limitaciones.

BIBLIOGRAFÍA

[1] O. H. Zarabia Zuñiga, Artist, Implementación de un chatbot con botframework: caso de estudio, servicios a clientes del área de fianzas de seguros Equinoccial.. [Art]. Escuela Politécnica Nacional, 2018.

[2] M. S. Perez, Artist, Análisis y Optimización de Agentes Conversacionales 3D para Sistemas Empotrados.. [Art]. Escuela Tecnica Superior de Ingeniería de Telecomunicacion, 2014.

[3] E. D. G. Teneda, Artist, PROTOTIPO DE UN CHATBOT PARA COMPRAS ONLINE UTILIZANDO BOT FRAMEWORK.. [Art]. UNIVERSIDAD TÉCNICA DE AMBATO, 2019.

[4] A. D. D. Pincheira, Artist, Diseño e implementación de un asesor virtual con interfaz web basado en un sistema de gestión de conocimientos y autoaprendizaje. [Art]. Universidad de las Fuerzas Armadas ESPE, 2015.

[5] E. M. E. ,. L. C. Javier Medina, «Asistentes virtuales en plataformas 3.0,» Revista Iberoamericana de Informática Educativa, nº 18, pp. 41-49, 2013.

[6] J. C. C. Torres, Artist, INTEGRACIÓN DE UN CHATBOT COMO HABILIDAD DE UN ROBOT SOCIAL CON GESTOR DE DIÁLOGOS. [Art]. UNIVERSIDAD CARLOS III DE MADRID, 2013.

[7] M. A. Cevallos Tóala, Artist, Propuesta tecnológica de una página web con la implementación de Bots para la gestión de relaciones con el cliente en la empresa Vipcell Electrónics. [Art]. Universidad de Guayaquil, 2017.

[8] M. G. F. D. Agnese Augello, «Una descripción general de las habilidades sociales de los chatbots de código abierto,» 22-24 11 2017. [En línea]. Available: https://www.researchgate.net/publication/321331001_An_overview_of_open-source_chatbots_social_skills. [Último acceso: 19 04 2021].

[9] A. Bozzon, «Computación colectiva empresarial para chatbots asistidos por humanos,» de 2018 IEEE / ACM 1st International Workshop on Software Engineering for Cognitive Services (SE4COG), Gotemburgo, Suecia, 2018.

[10] U. N. A. d. Mexico, «Inteligencia Artificial-,» 2015. [En línea]. Available: http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.100/219/A7.pdf?sequence=7. [Último acceso: 19 03 2021].

[11] C. P. C. Barreno, Artist, PROTOTIPO ENFOCADO AL APRENDIZAJE DE LÓGICA DE PROGRAMACIÓN EN NIÑOS DE EDADES COMPRENDIDAS ENTRE 4 A 10 AÑOS.. [Art]. UNIVERSIDAD TÉCNICA DE AMBATO, 2018.

[12] C. I. C. V. P. G. J. F. y. H. L. RIVAS, «Metodologías actuales de desarrollo de software,» Tecnología e Innovación, vol. 2, nº 5, p. 982, 2015.

[13] V. P. C. Adriana Sandra Almeida, Artist, Arquitectura de Software: Estilos y Patrones. [Art]. Universidad Nacional de la Patagonia San Juan Bosco, 2007.

[14] C. N. T. Z. Gelen Esneydi Guzman Lote, Artist, Implemenectación del patrón MVC para el proceso de selección de personal. [Art]. UNIVERSIDAD

LIBRE-PROGRAMA DE INGENIERIA DE SISTEMAS DE BOGOTA, 2013.

[15] Futurizable, «Estado del arte en el desarrollo de chatbots a nivel mundial.,» Online, 22 09 2017. [En línea]. Available: https://futurizable.com/chatbot/. [Último acceso: 22 03 2021].

[16] Microsoft, «Bot Framework SDK,» Online, 15 11 2019. [En línea]. Available: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-4.0. [Último acceso: 22 03 2021].

[17] aws, «Amazon lex Guía de desarrolladores,» 2021. [En línea]. Available: https://docs.aws.amazon.com/es_es/lex/latest/dg/lex-dg.pdf. [Último acceso: 22 03 2021].

[18] S. Malik, «CODE MAGAZINE,» Online, 05 02 2021. [En línea]. Available: https://www.codemag.com/Article/1809021/Natural-Language-Understanding-with-LUIS. [Último acceso: 22 03 2021].

[19] M. Azure, «La guía de Azure para desarrolladores,» 05 2019. [En línea]. Available: file:///C:/Users/samiv/Downloads/Azure_Developer_Guide_eBook_es-ES.pdf. [Último acceso: 22 03 2021].

[20] Microsoft, «Diseño de navegación de bots,» Online, 13 12 2017. [En línea]. Available: https://docs.microsoft.com/es-es/azure/bot-service/bot-service-design-navigation?view=azure-bot-service-4.0. [Último acceso: 22 03 2021].

[21] Microsoft, «Diseño y control del flujo de conversación,» Online, 19 11 2019. [En línea]. Available: https://docs.microsoft.com/es-es/azure/bot-service/bot-service-design-conversation-flow?view=azure-bot-service-4.0. [Último acceso: 23 03 2021].

[22] Y. Y. L. B. F. J. S. M. Jonathan Stalin Delgado Guerrero, «Desarrollo de chatbot usando bot framework de Microsoft,» Espirales revista multidisciplinaria de investigación, vol. 1, nº 11, pp. 1-8, 2017.

[23] E. E. e. I. ITCA-FEPADE, «Metodologías Ágiles de Desarrollo de Software Aplicadas a la Gestión de Proyectos Empresariales,» Revista Tecnologica, nº 8, pp. 8-9-10, 2015.

[24] A. d. M. Gonzáles, Artist, Sistema de transferencia de datos de origen múltiple a Oracle para la empresa Solinfo.. [Art]. Universidad Técnica de Ambato, 2019.

[25] J. D. F. M. J. M. V. Andrés Navarro Cadavid, «Revisión de metodologías ágiles para el desarrollo de software,» Fundation Dialnet, vol. 11, nº 2, pp. 32-33-34-35, 20 09 2013.

[26] I. Z. M. A. Aiman Khan Nazir, «The Impact of Agile Methodology (DSDM) on Software Project Management,» 06 03 2018. [En línea]. Available: file:///C:/Users/samiv/Downloads/ccs-2018-TheImpactofAgileMethodologyDSDMonSoftwareProjectManagement.pdf. [Último acceso: 02 04 2021].

[27] J. T. María Oya, Artist, MÉTODO DE DESARROLLO DE SISTEMAS DINÁMICOS (DSDM). [Art]. UNIVERSIDAD ALEJANDRO DE HUMBOLDT , 2016.

[28] M. J. G. Rodriguez, Artist, Estudio comparativo entre las metodologías tradicionales para la gestión de proyectos de software. [Art]. UNIVERSIDAD DE OVIEDO, 2015.

[29] P. G. Guerrero, «Bot Framework Emulator v4 — Review,» 07 07 2019. [En línea]. Available: https://planetachatbot.com/bot-framework-emulator-v4-review-e9333f323b38. [Último acceso: 09 05 2021].

[30] «Microsoft- Adaptive Cards,» Microsoft, 2021. [En línea]. Available: https://vnext.adaptivecards.io/designer/. [Último acceso: 07 07 2021].

[31] Microsoft, «dialogs library,» Microsoft, 2021. [En línea]. Available: https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-concept-dialog?view=azure-bot-service-4.0. [Último acceso: 07 07 2021].

[32] Microsoft, «Waterfalldialog class,» Microsoft, 2021. [En línea]. Available: https://docs.microsoft.com/en-us/javascript/api/botbuilder-dialogs/waterfalldialog?view=botbuilder-ts-latest. [Último acceso: 07 07 2021].

[33] Microsoft, «Intenciones en la aplicación de LUIS,» Microsoft, 2021. [En línea]. Available: https://docs.microsoft.com/es-es/azure/cognitive-services/luis/luis-concept-intent. [Último acceso: 12 05 2021].

ANEXOS

Anexo A.- Diagramas de Flujos de los Diálogos del Chatbot

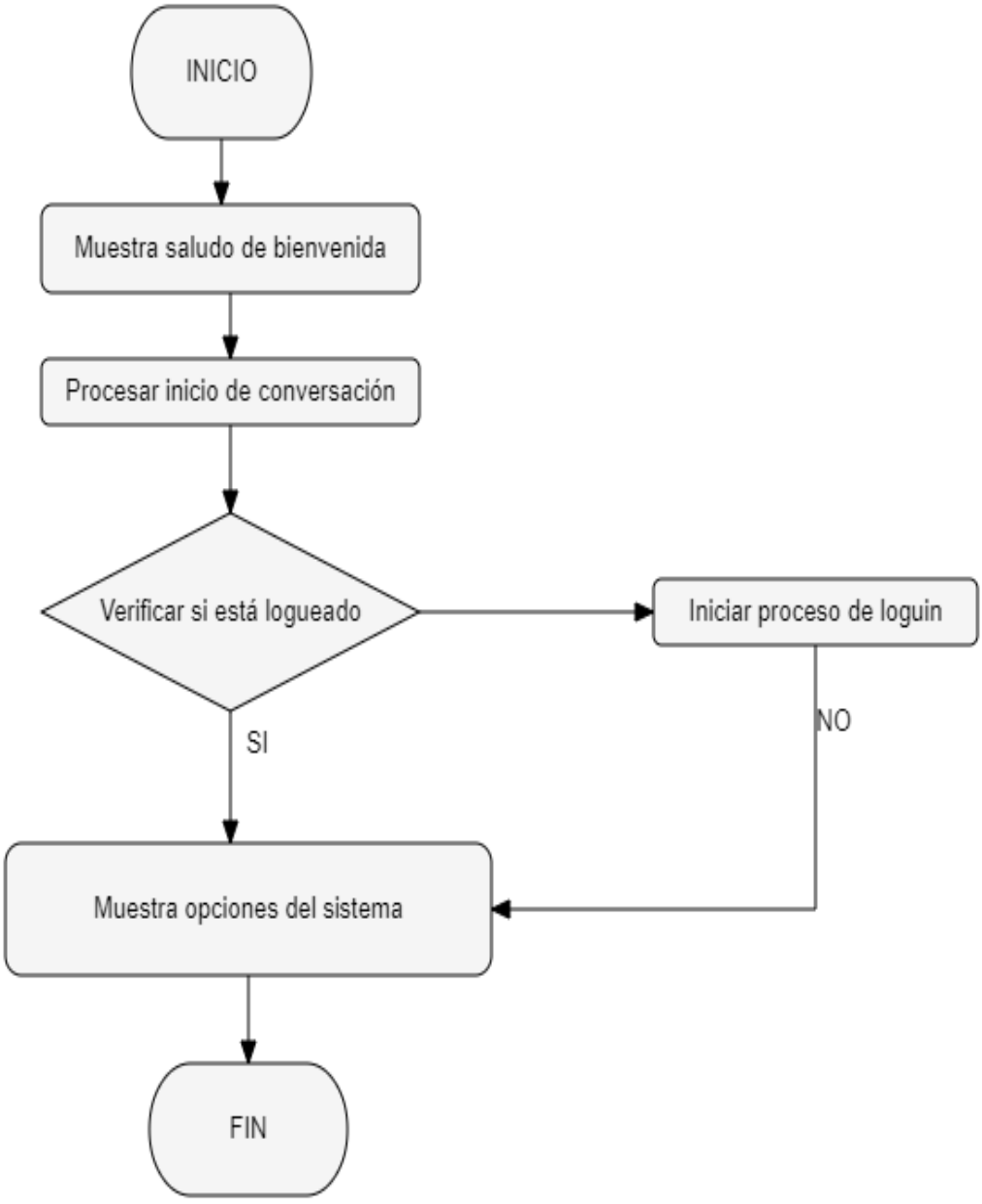


Figura A.1: Diagrama de Flujo del Diálogo de Inicio
Fuente: Elaborado por el investigador.

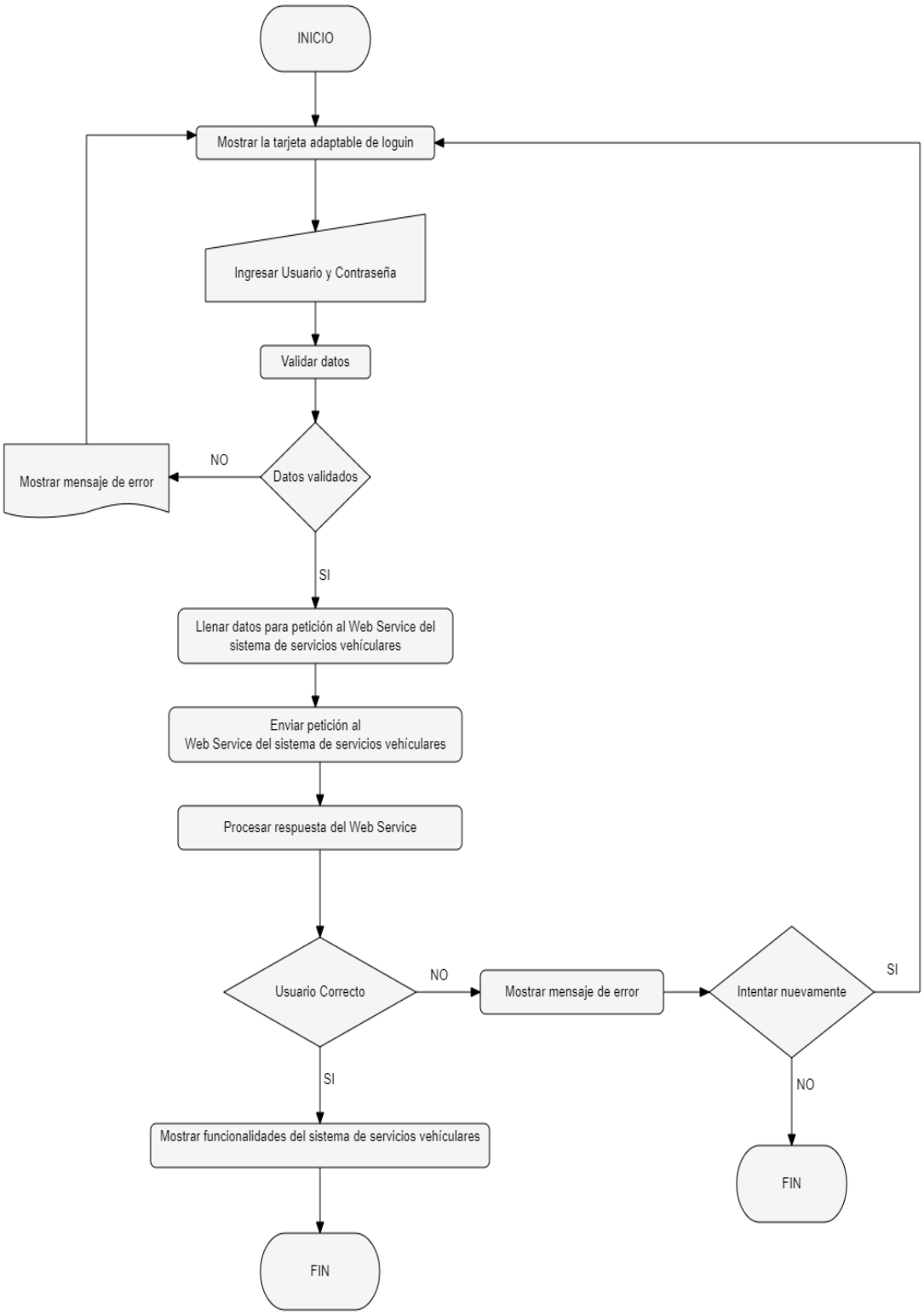


Figura A.2: Diagrama de Flujo del Diálogo de Ingreso al Sistema.
Fuente: Elaborado por el investigador.

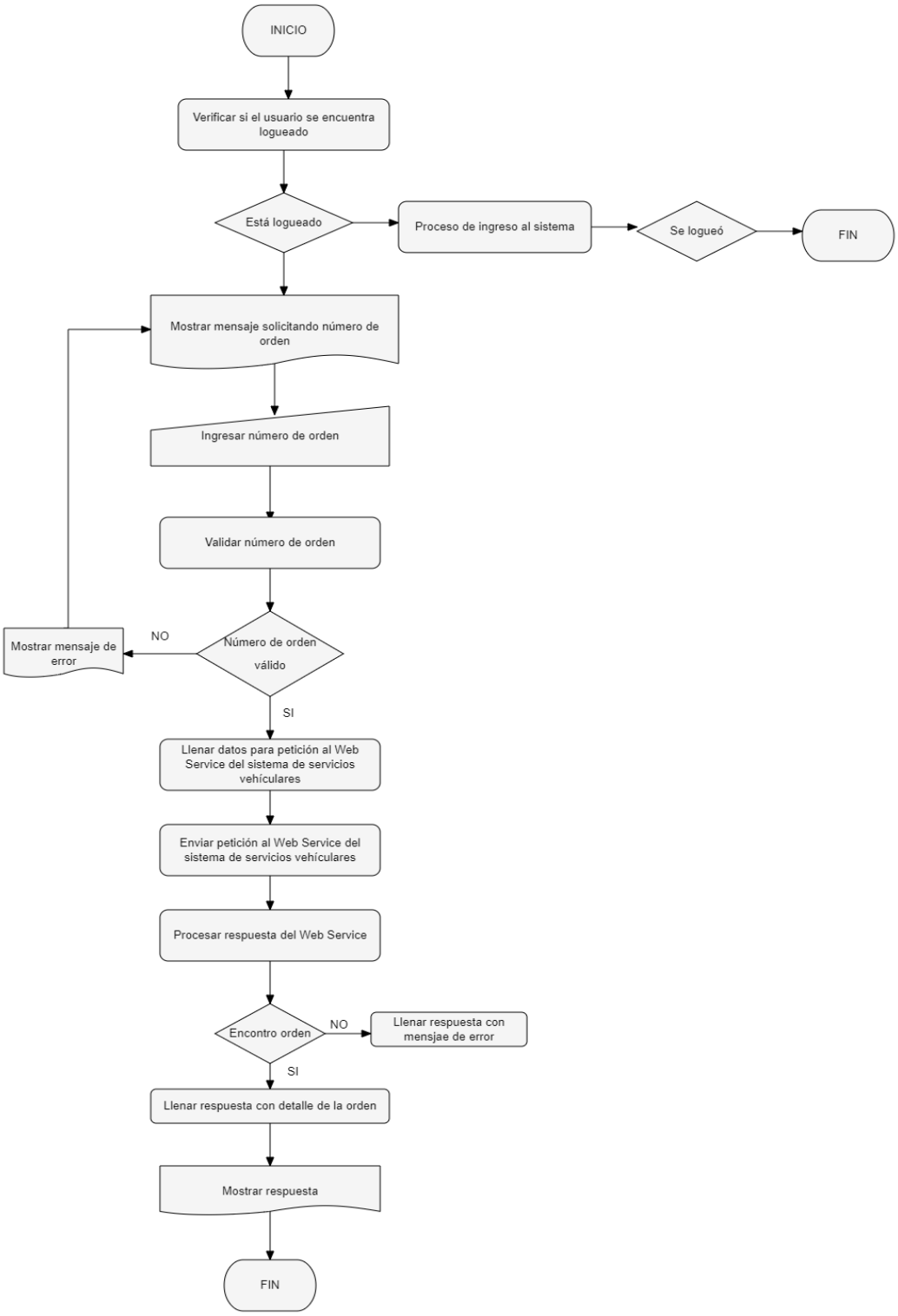


Figura A.3: Diagrama de Flujo del Diálogo para Consultar las Órdenes de Compra.
Fuente: Elaborado por el investigador.

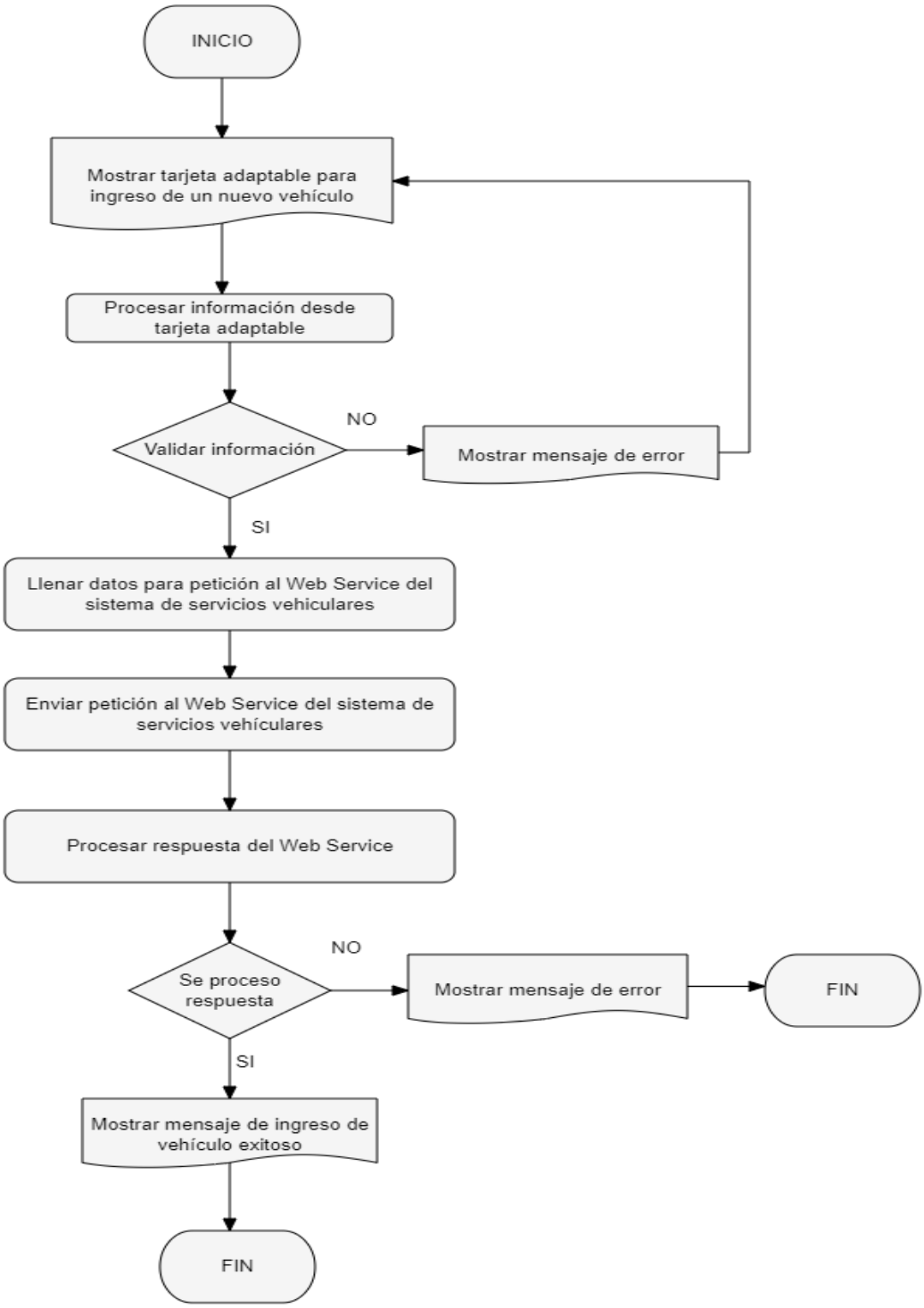


Figura A.4: Diagrama de Flujo del Diálogo para Ingresar un Vehículo.
Fuente: Elaborado por el investigador.

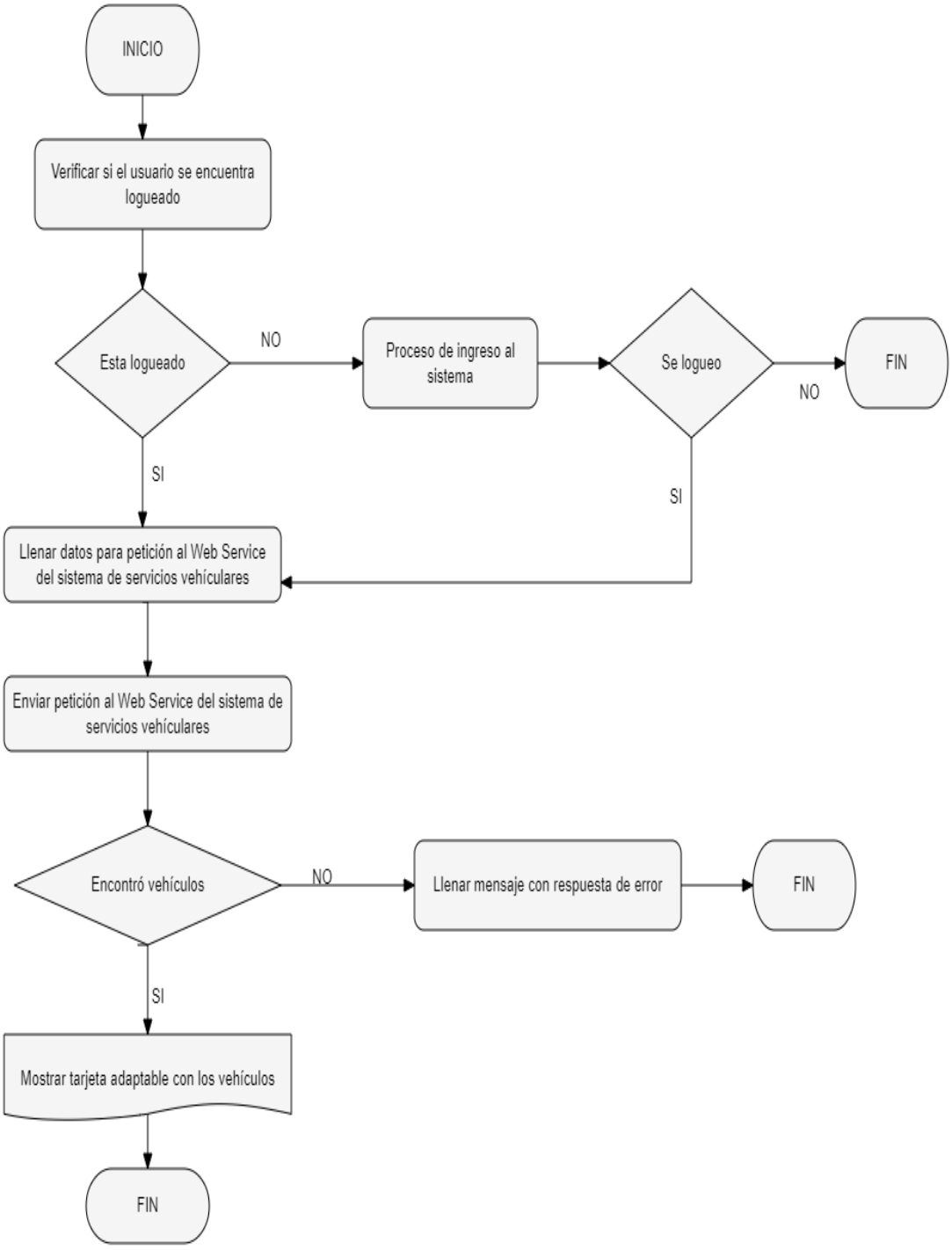


Figura A.5: Diagrama de Flujo del Diálogo para Consultar los Vehículos.
Fuente: Elaborado por el investigador.

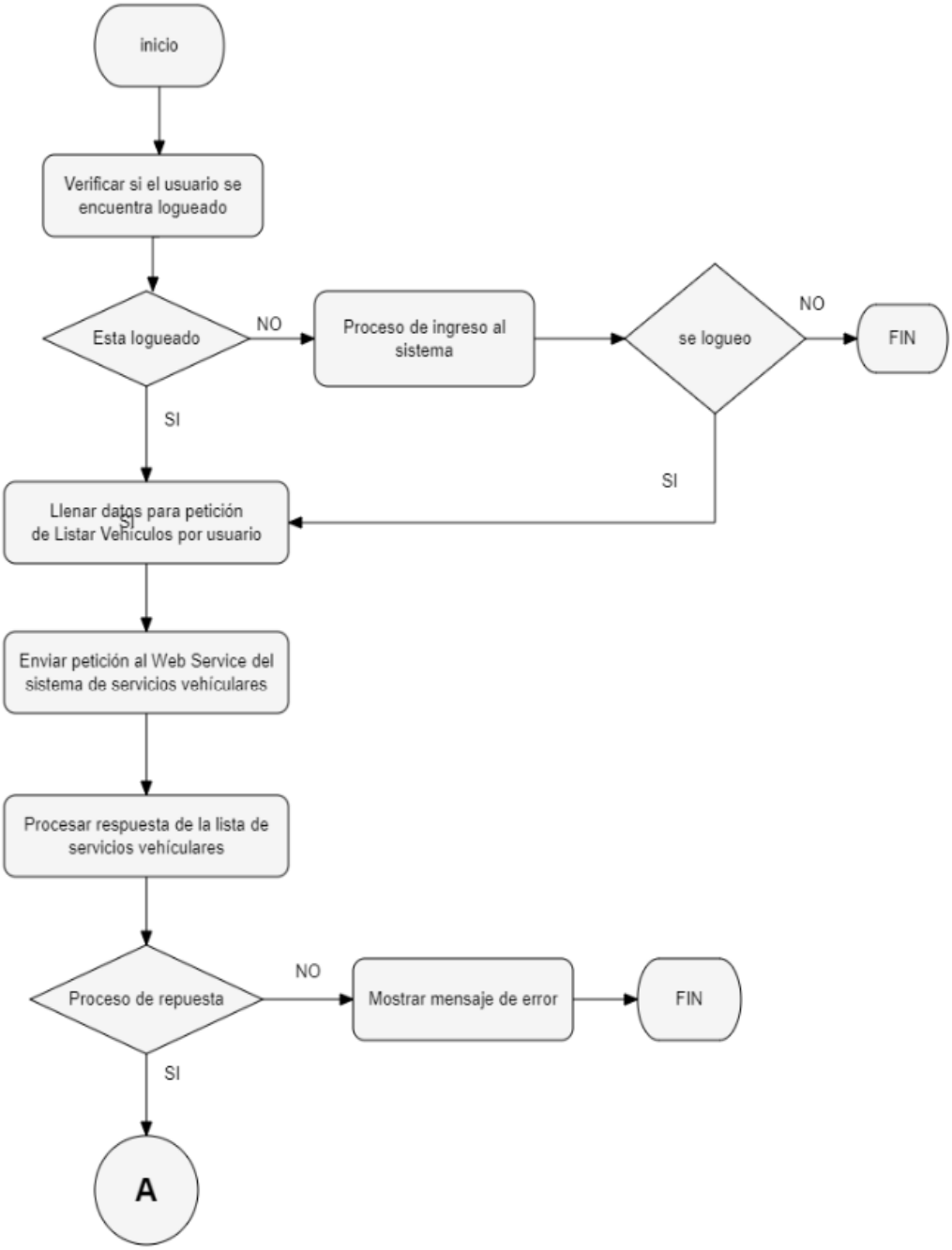


Figura A.6: Diagrama de Flujo del Diálogo para Realizar una Orden de Compra.
Fuente: Elaborado por el investigador.

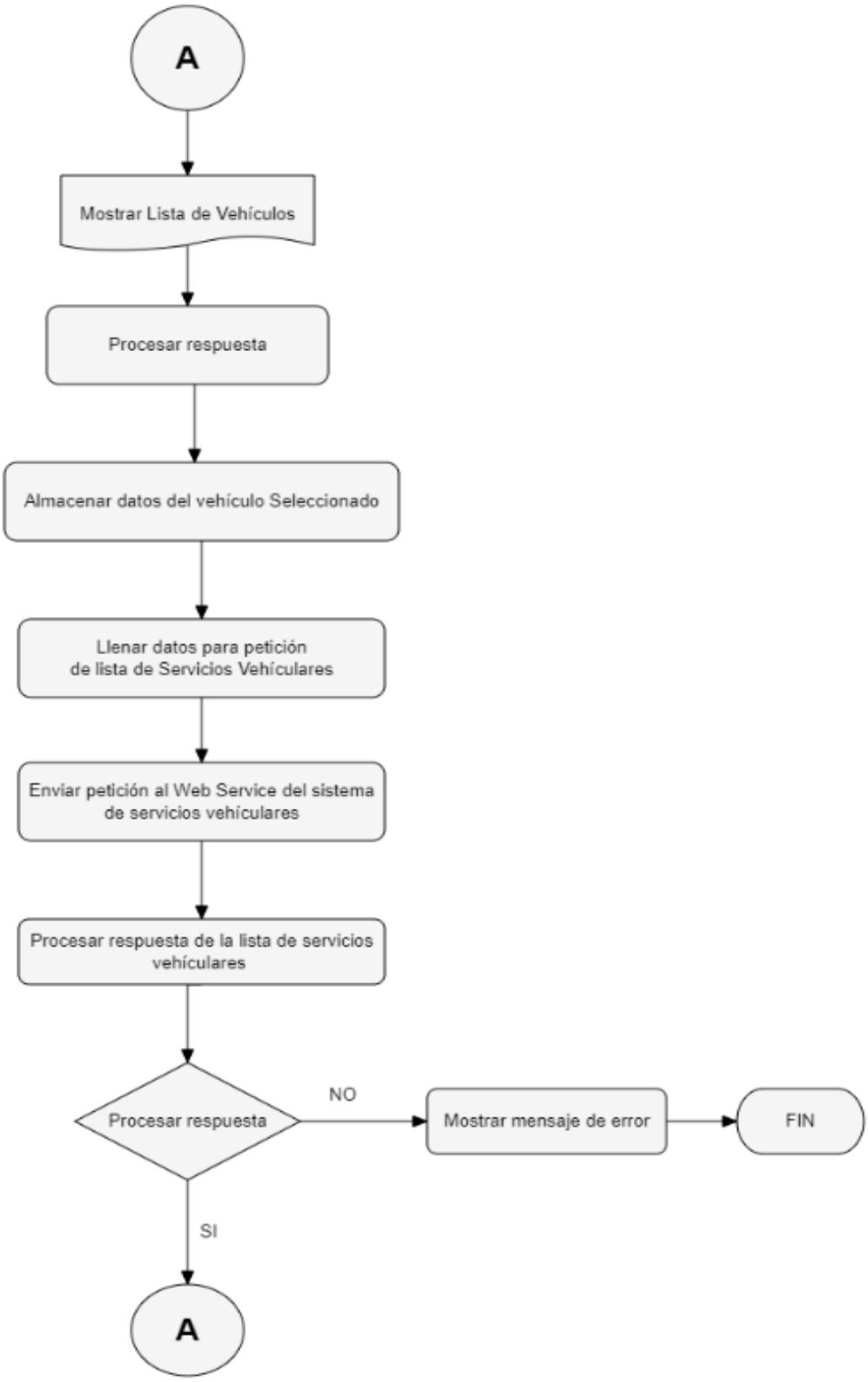


Figura A.7: Continuación Diagrama de Flujo de la figura A.6.
Fuente: Elaborado por el investigador.

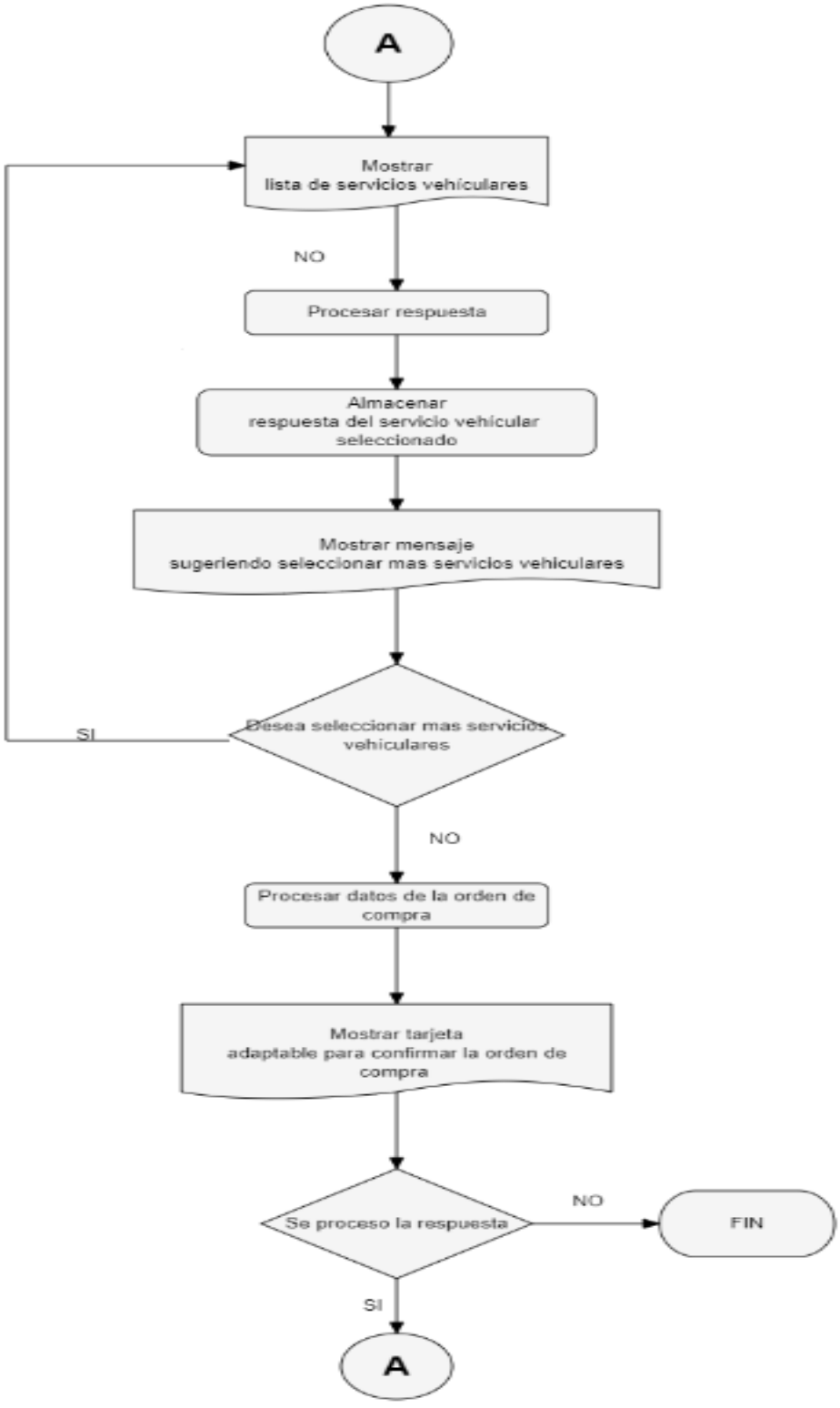


Figura A.8: Continuación Diagrama de Flujo de la figura A.7.
Fuente: Elaborado por el investigador.

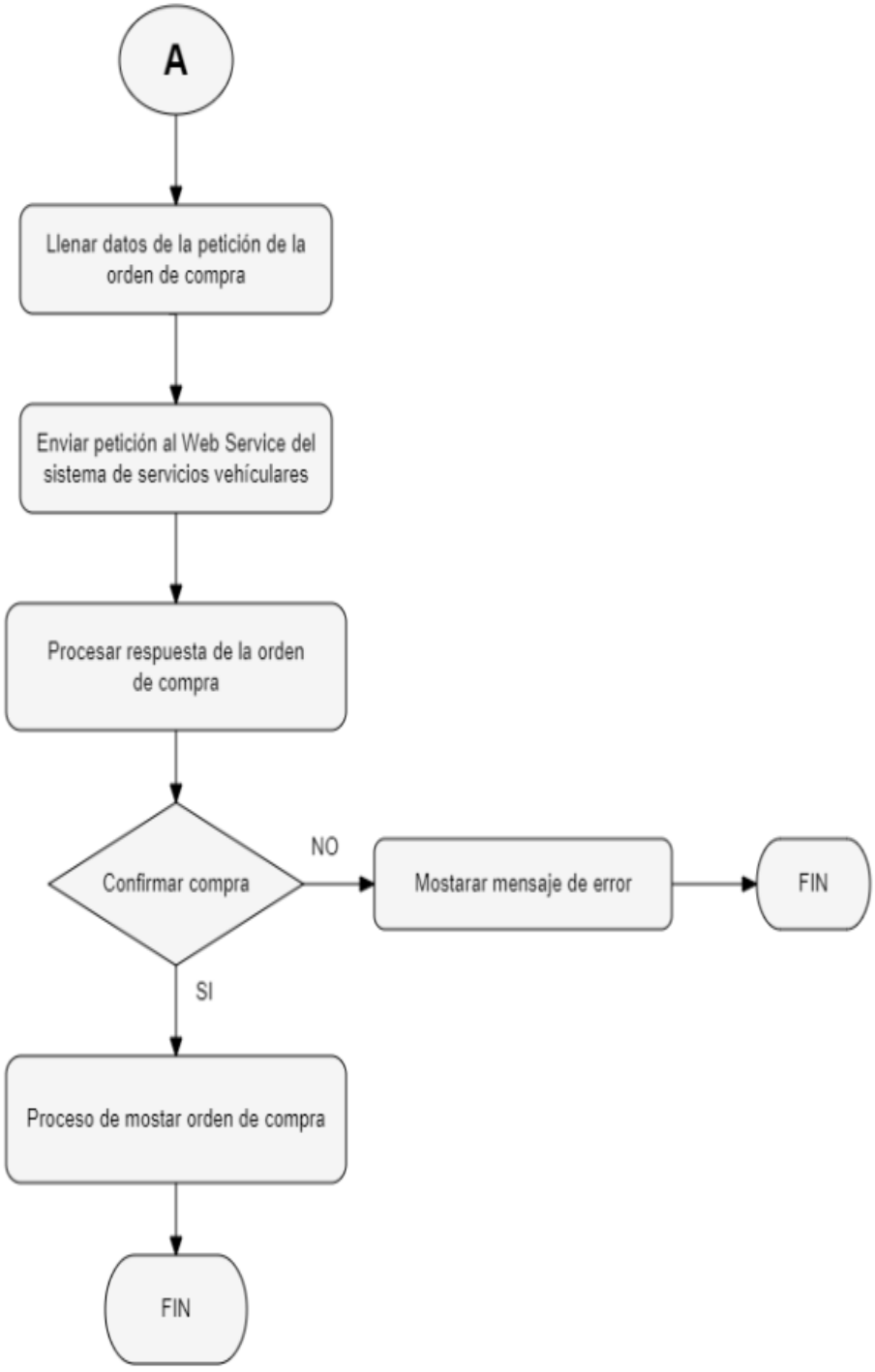


Figura A.9: Continuación Diagrama de Flujo de la figura A.8.
Fuente: Elaborado por el investigador.

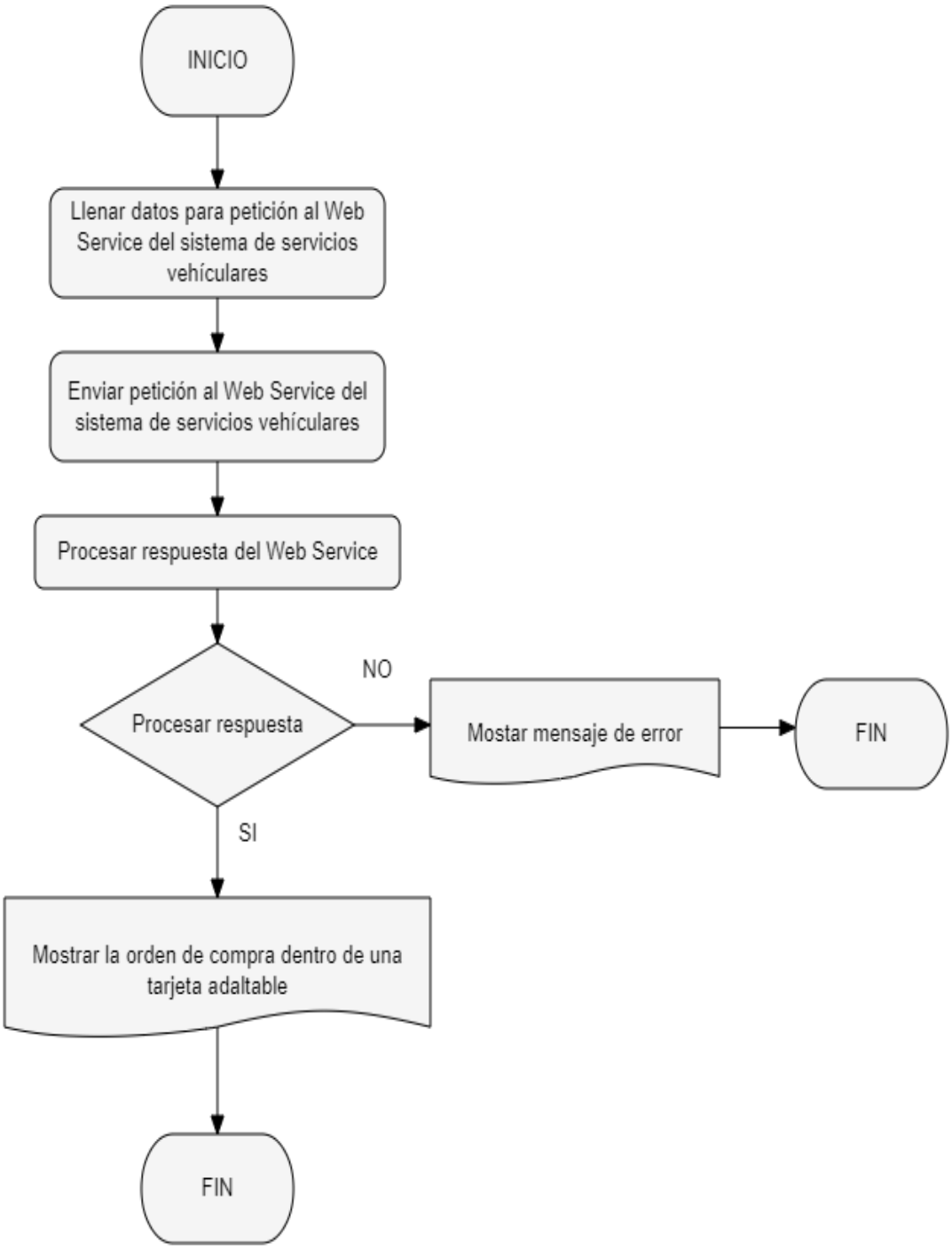


Figura A.10: Diagrama de Flujo del Diálogo para Visualizar la Orden de Compra.
Fuente: Elaborado por el investigador.

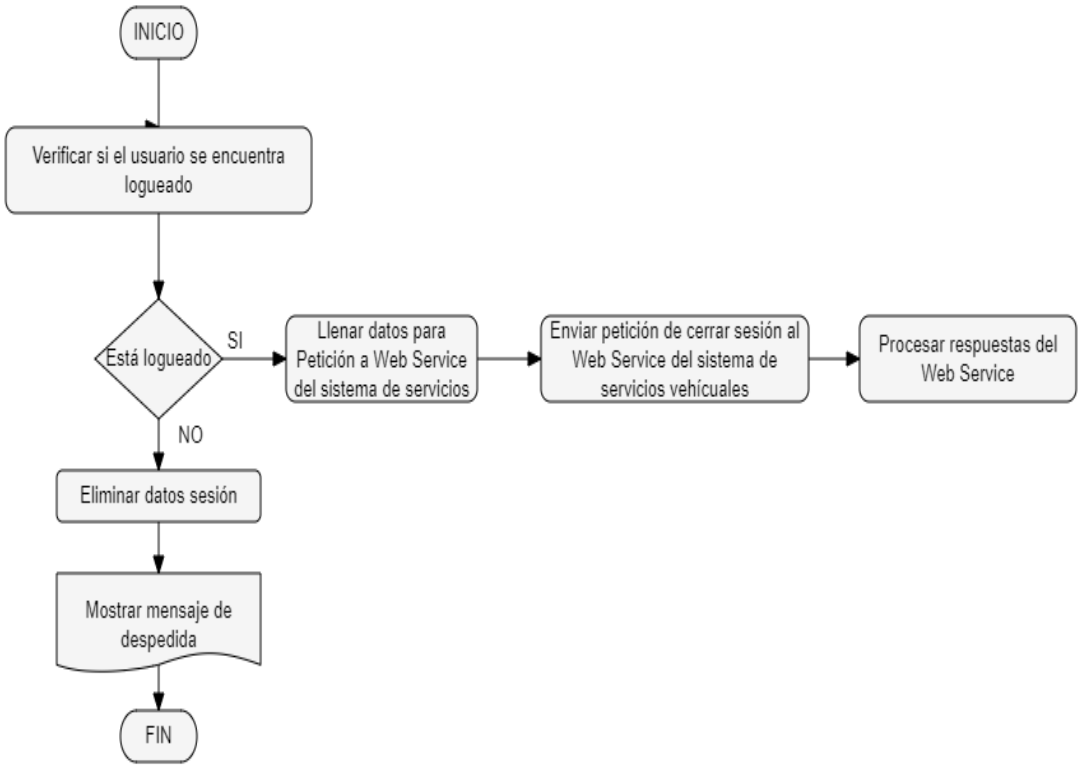


Figura A.11: Diagrama de Flujo del Diálogo de Fin.
Fuente: Elaborado por el investigador.

Anexo B.- Prototipos de las Interfaces de Usuario

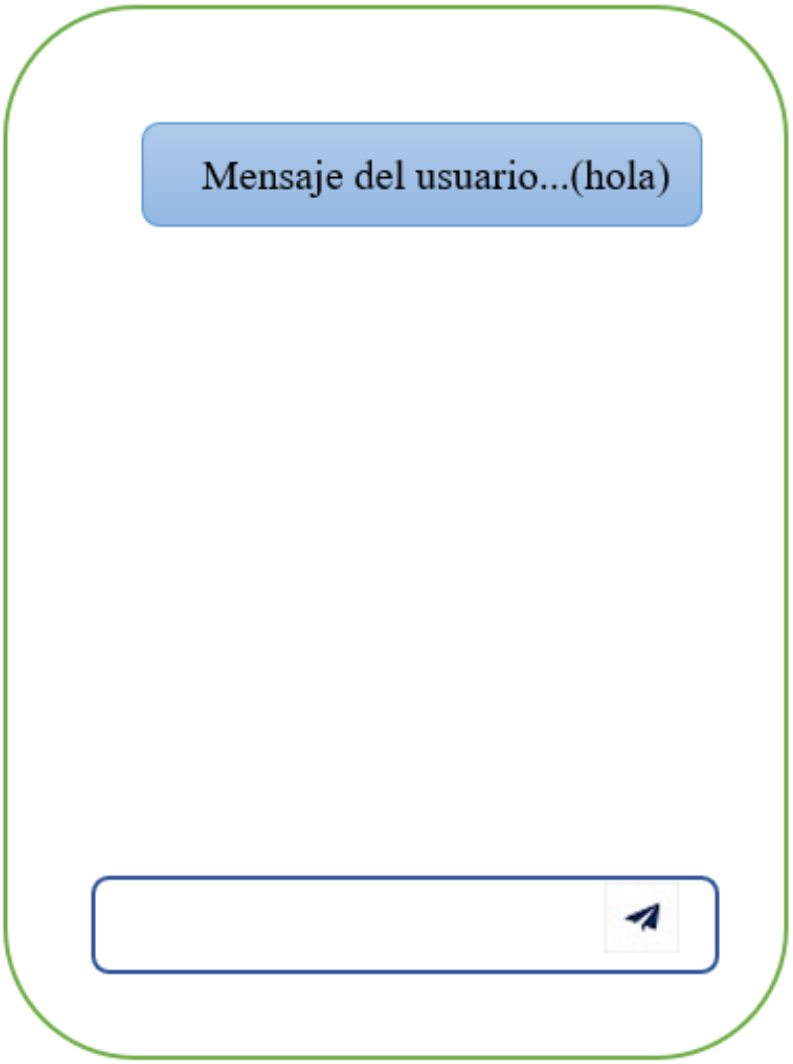


Figura B.1: Prototipo del Diálogo de Inicio.
Fuente: Elaborado por el investigador.

Figura B.2: Prototipo del Diálogo de Ingreso al Sistema.
Fuente: Elaborado por el investigador.

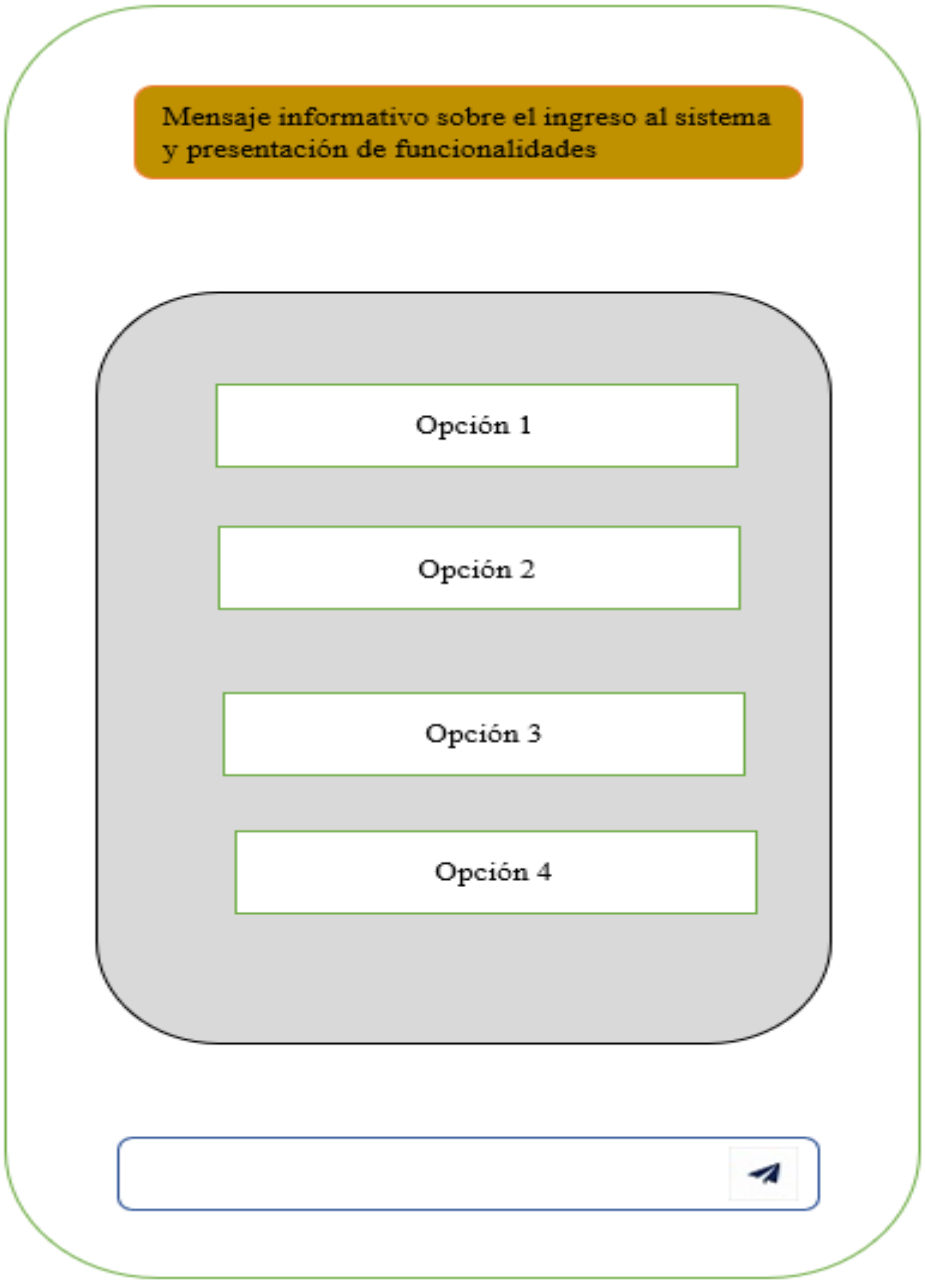


Figura B.3: Prototipo del Diálogo de Opciones.
Fuente: Elaborado por el investigador.

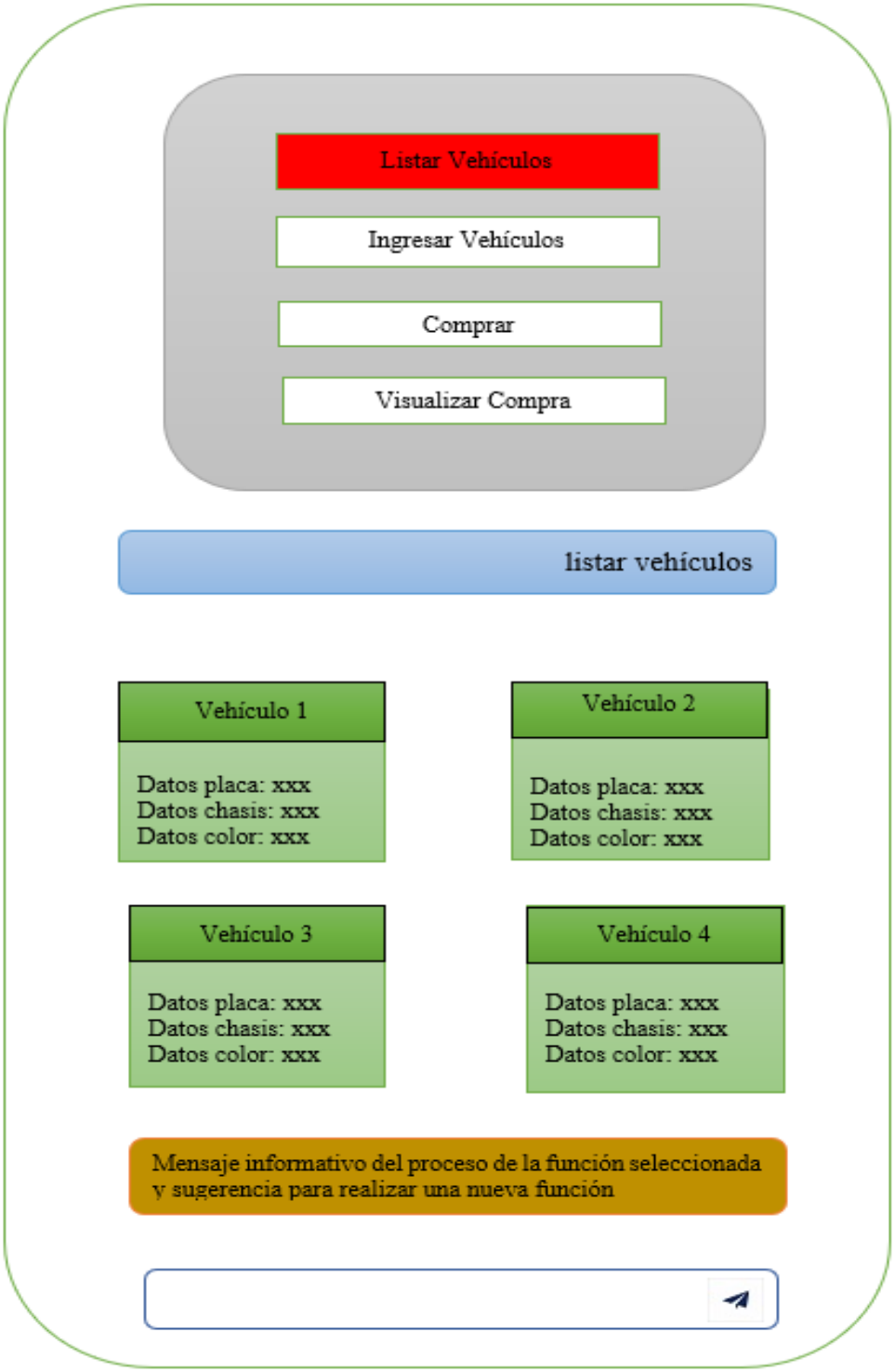


Figura B.4: Prototipo del Diálogo para Listar Vehículos.
Fuente: Elaborado por el investigador.

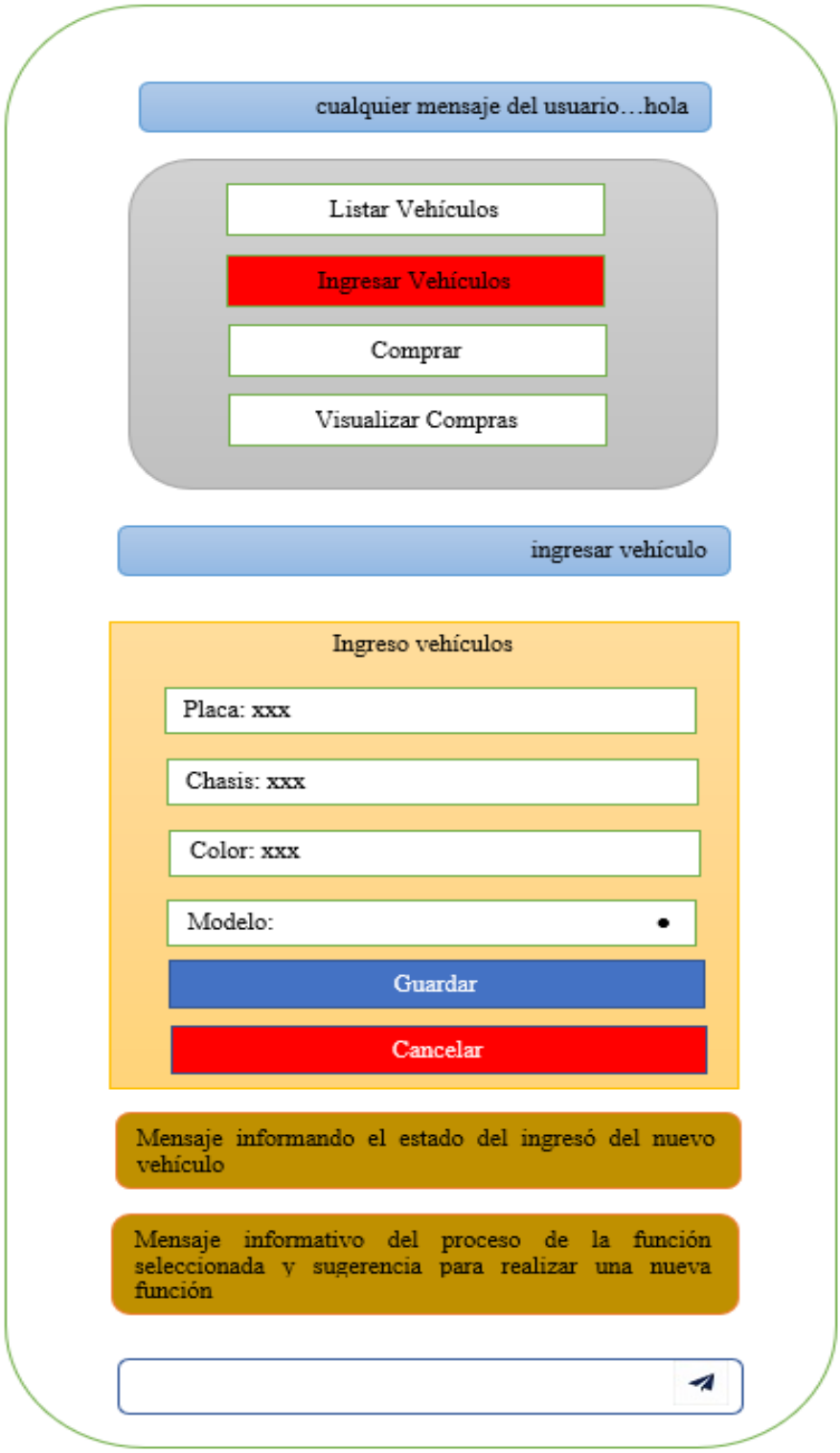


Figura B.5: Prototipo del Diálogo para Ingresar un Nuevo Vehículo.
Fuente: Elaborado por el investigador.

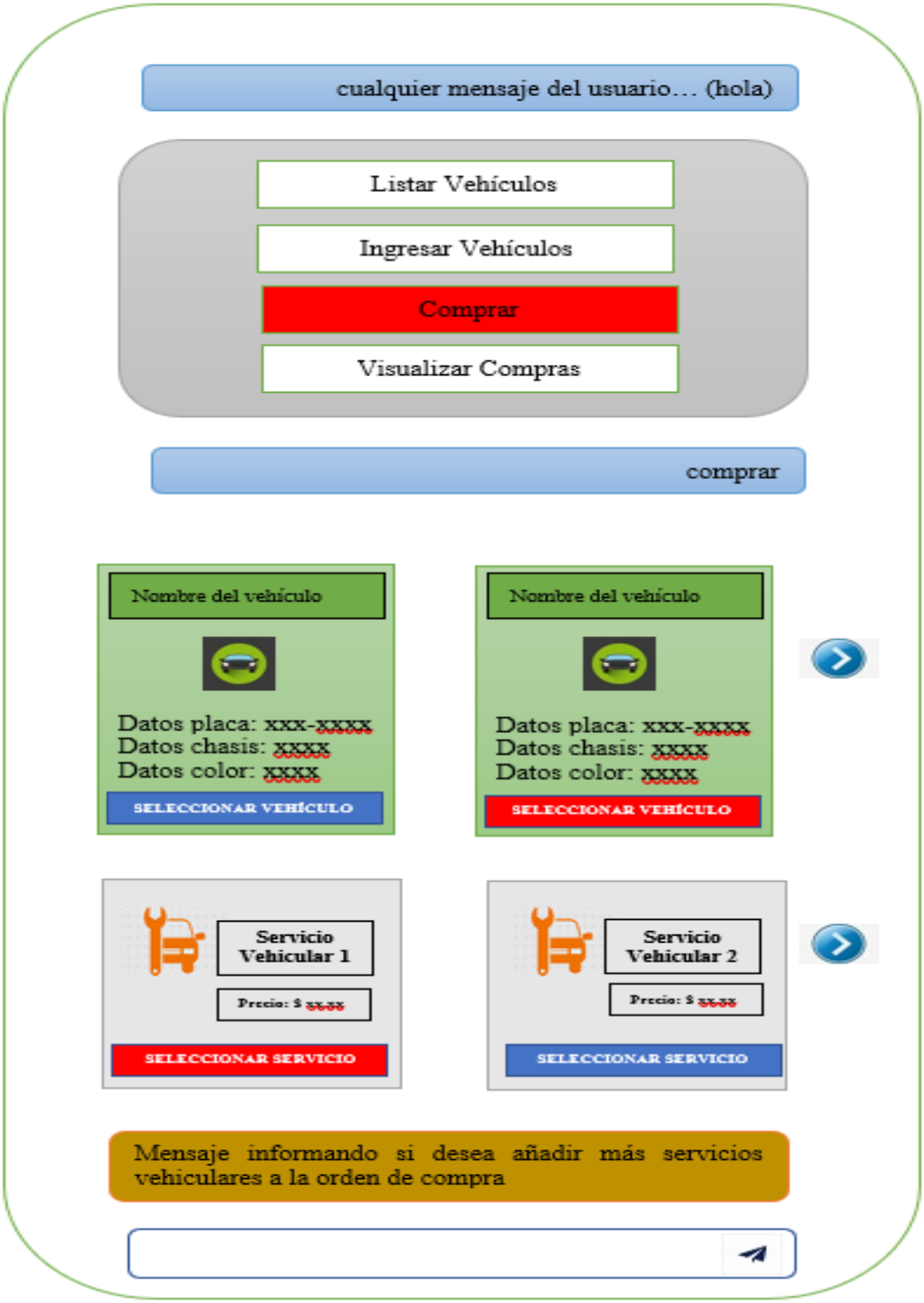


Figura B.6: Prototipo del Diálogo para Realizar una Orden de Compra parte 1.
Fuente: Elaborado por el investigador.

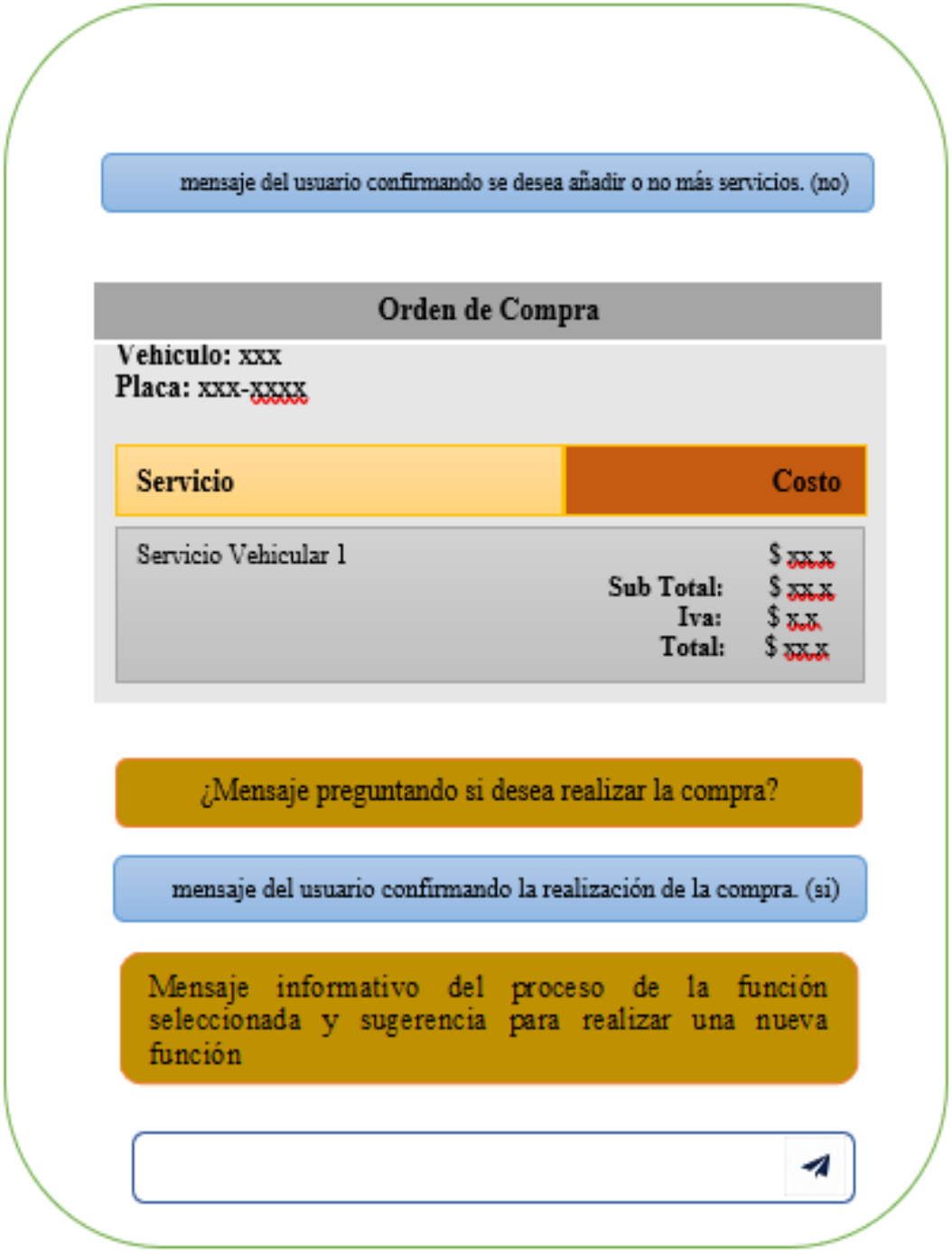


Figura B.7: Prototipo del Diálogo para Realizar una Orden de Compra parte 2.
Fuente: Elaborado por el investigador.

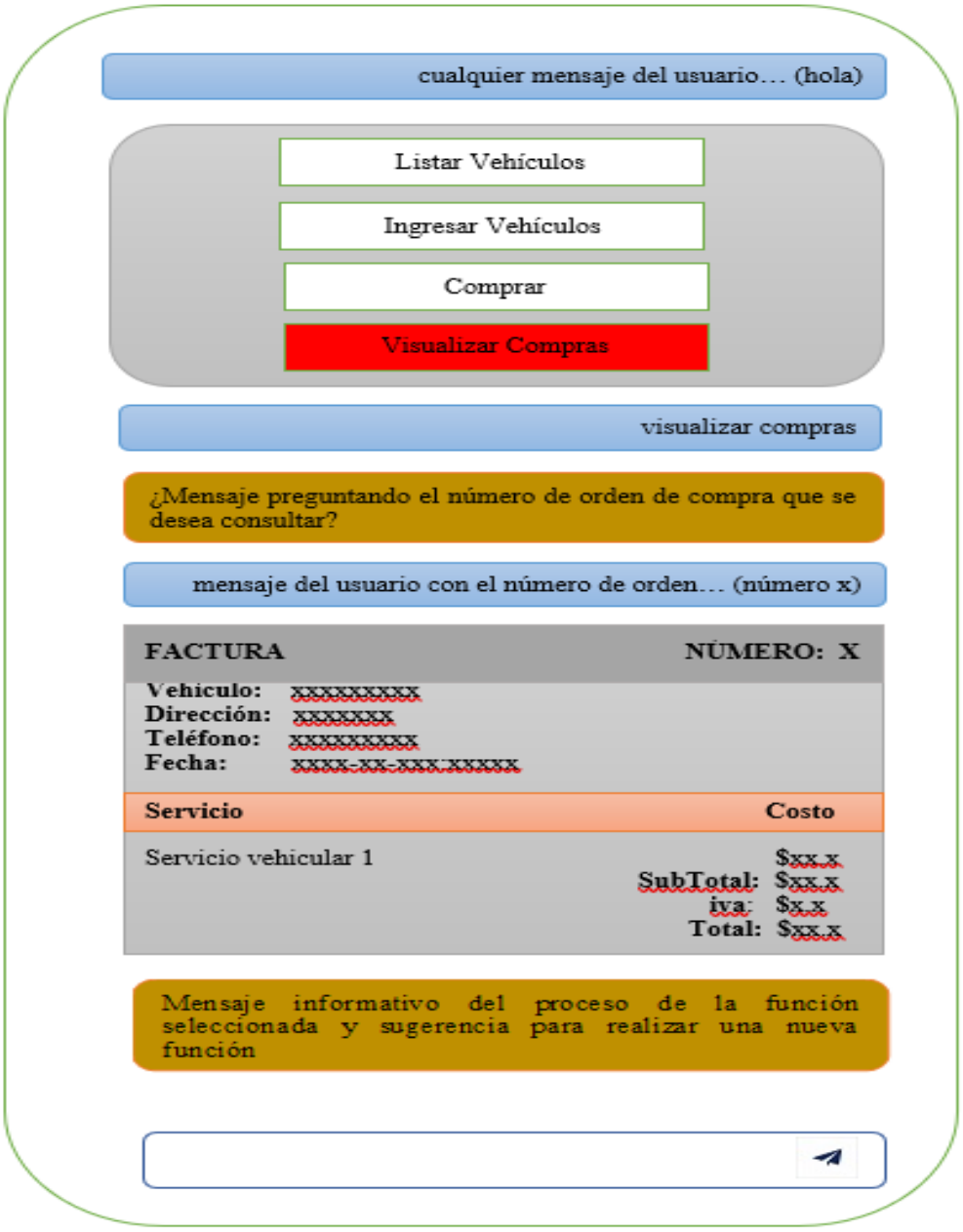


Figura B.8: Prototipo del Diálogo para Visualizar una Orden de Compra.
Fuente: Elaborado por el investigador.

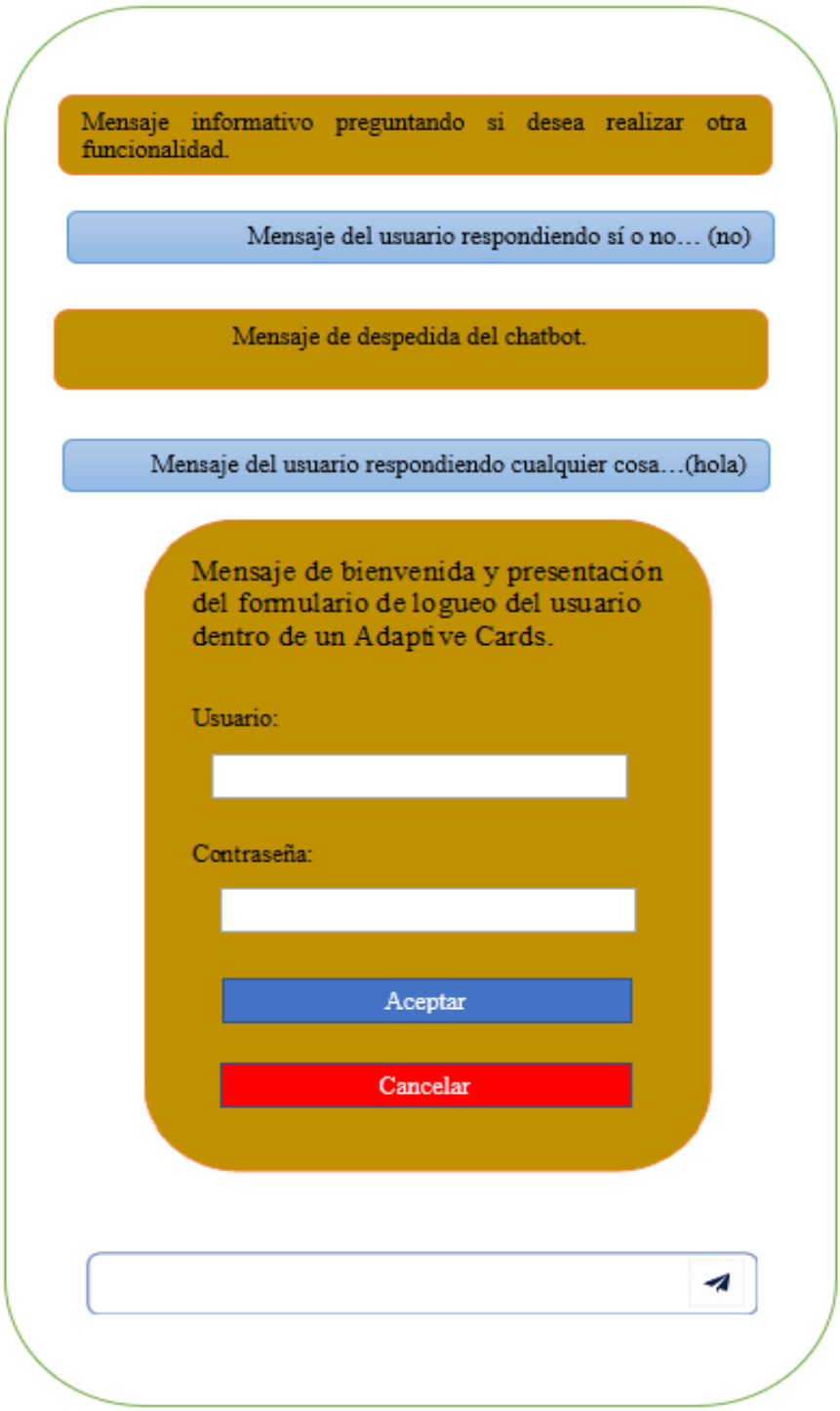


Figura B.9: Prototipo del Diálogo de Fin.
Fuente: Elaborado por el investigador.

Anexo C.-Imágenes del Código del Backend y Frontend

- **Backend**

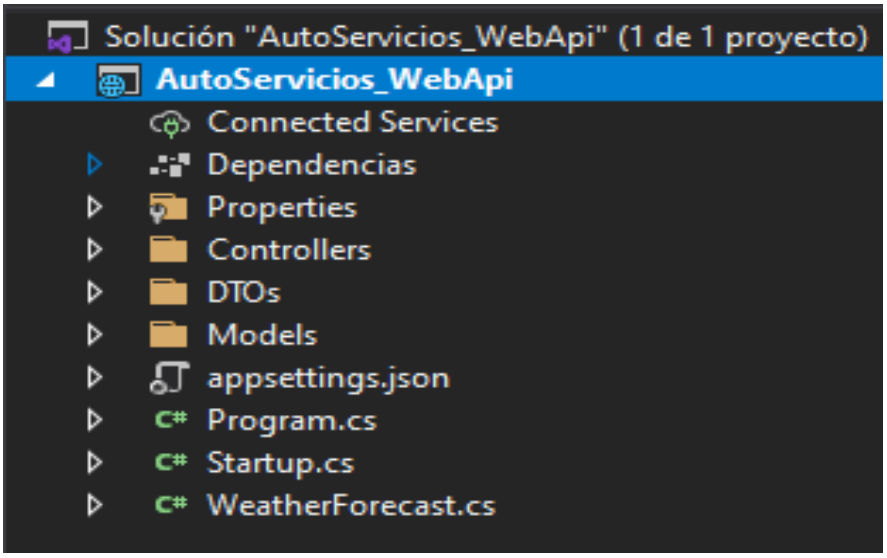


Figura C.1: Estructura General del Servidor.
Fuente: Elaborado por el investigador.

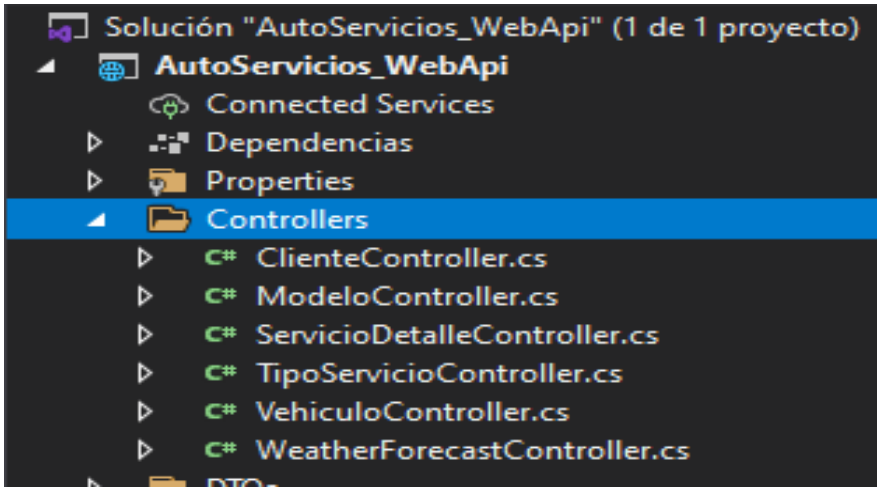


Figura C.2: Controladores Utilizados en el Servidor.
Fuente: Elaborado por el investigador.

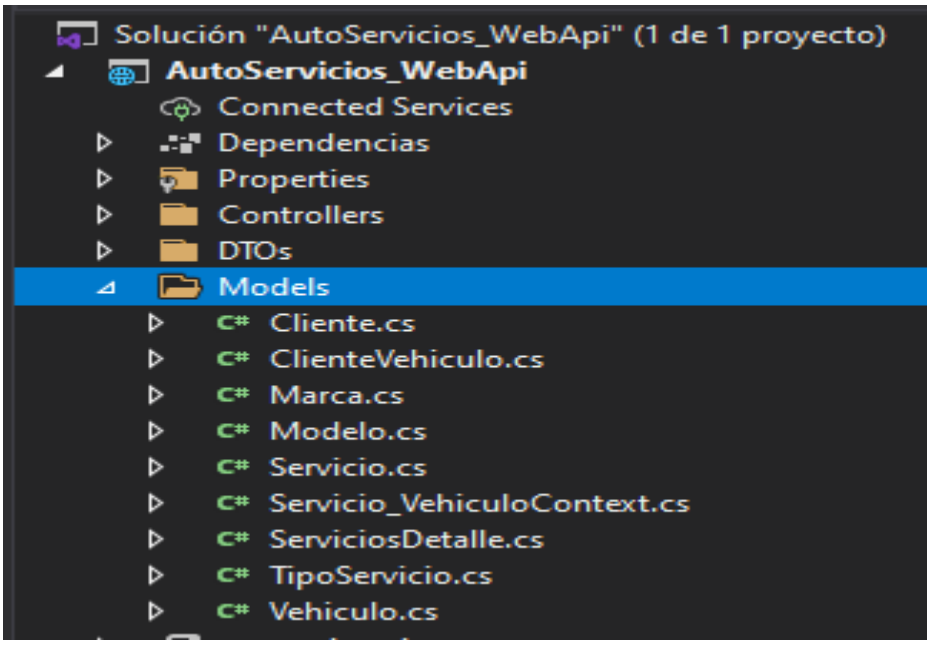


Figura C.4: Modelos Utilizados en el Servidor.
Fuente: Elaborado por el investigador.

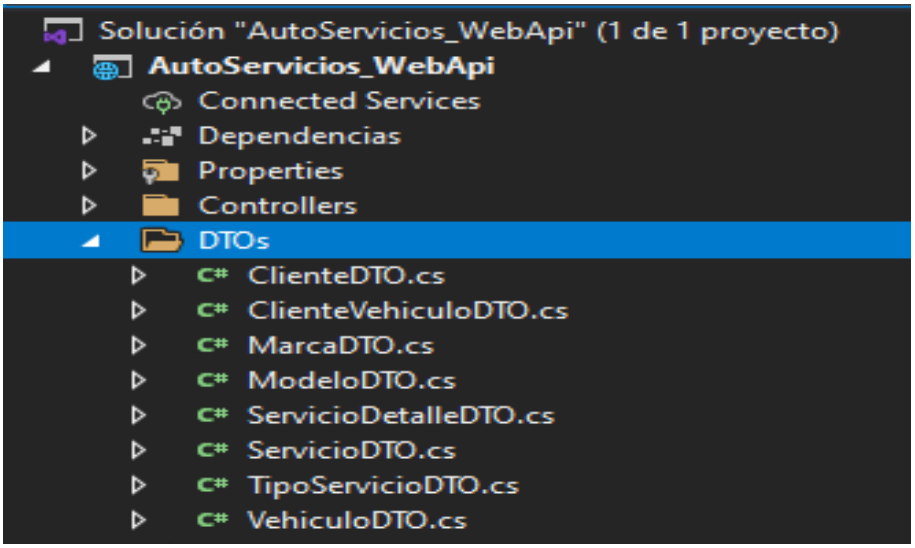


Figura C.4: DTOs Utilizados en el Servidor.
Fuente: Elaborado por el investigador.

- **Frontend**

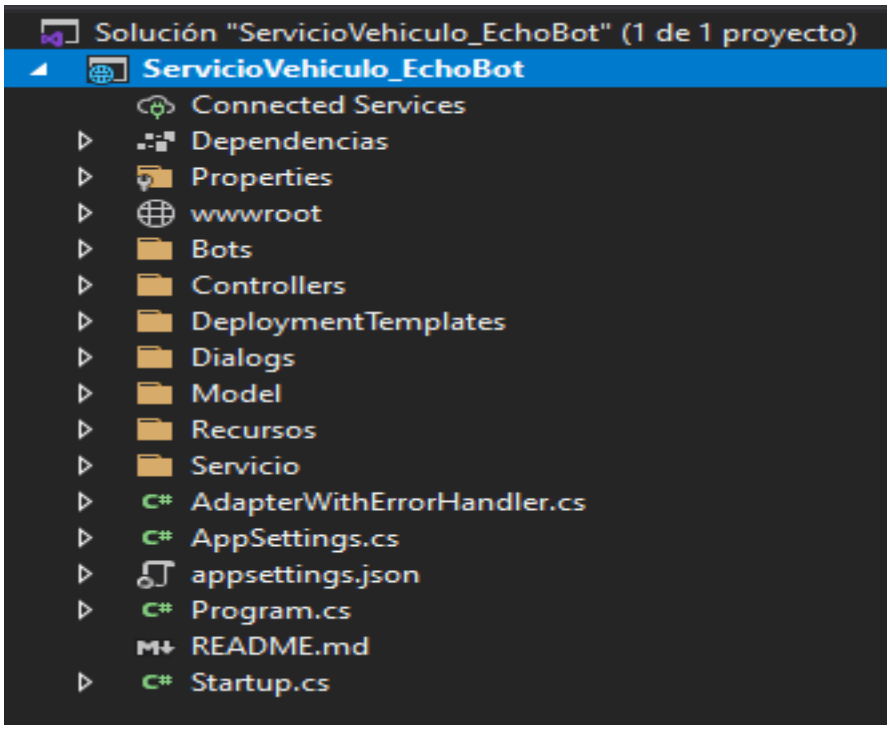


Figura C.5: Estructura General del Cliente.
Fuente: Elaborado por el investigador.

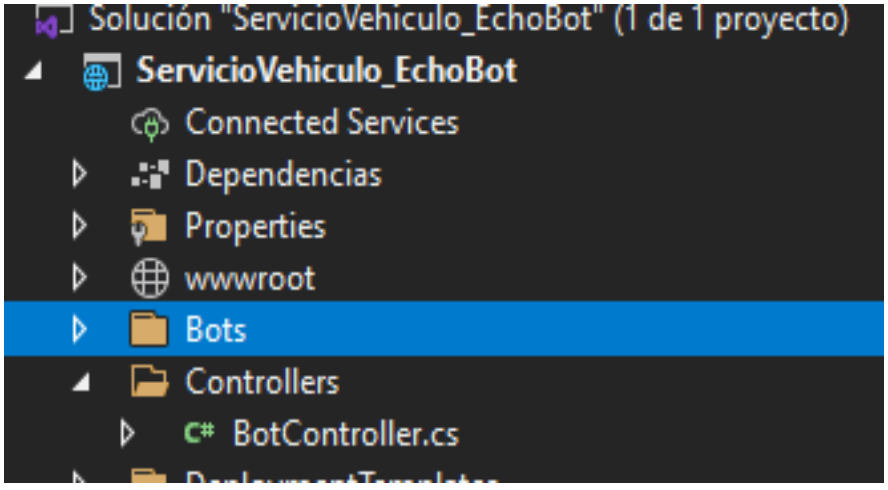


Figura C.6: Controlador Utilizado en el Cliente.
Fuente: Elaborado por el investigador.

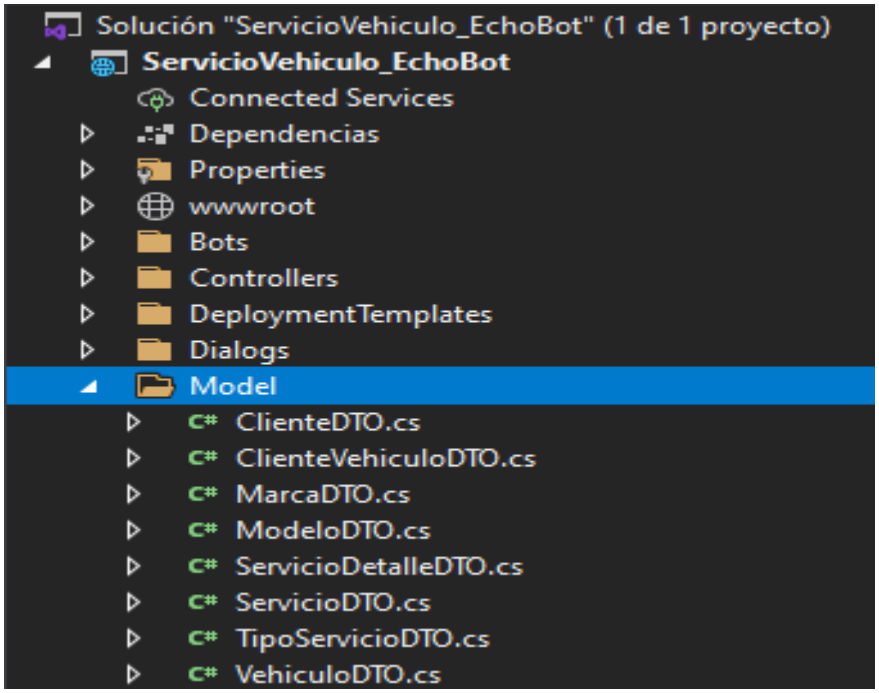


Figura C.7: Modelos Utilizados en el Cliente.
Fuente: Elaborado por el investigador.

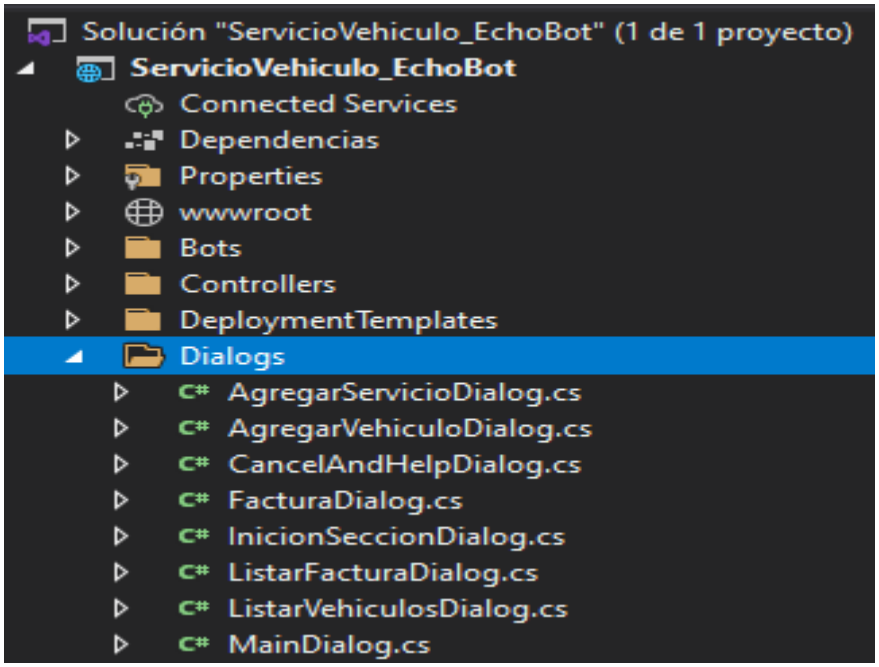


Figura C.8: Diálogos Utilizados en el Cliente.
Fuente: Elaborado por el investigador.

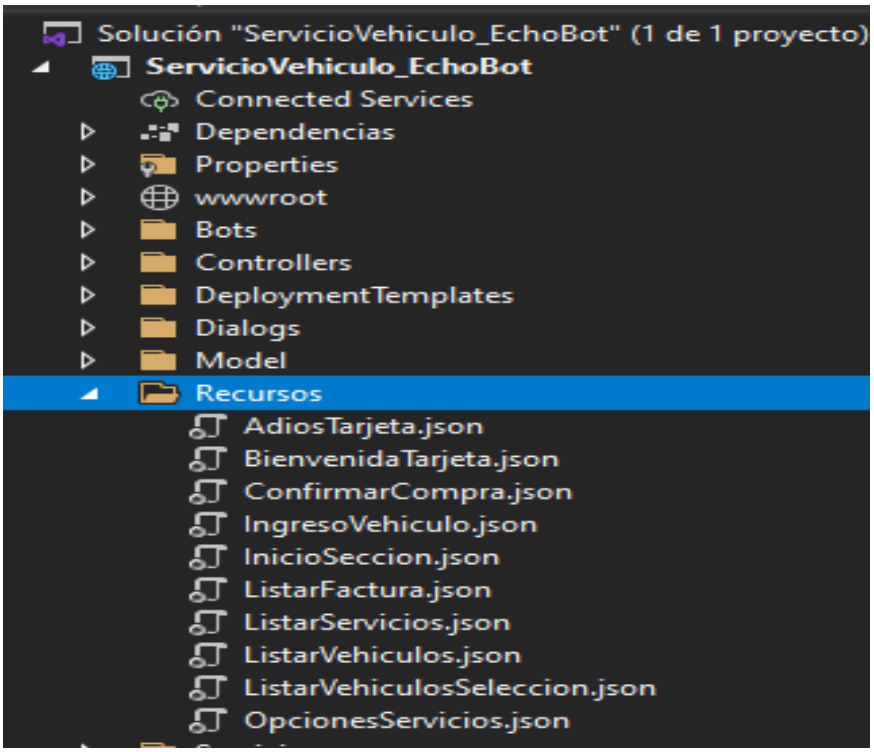


Figura C.9: Templates de Adaptive Cards Utilizados en el Cliente.
Fuente: Elaborado por el investigador.

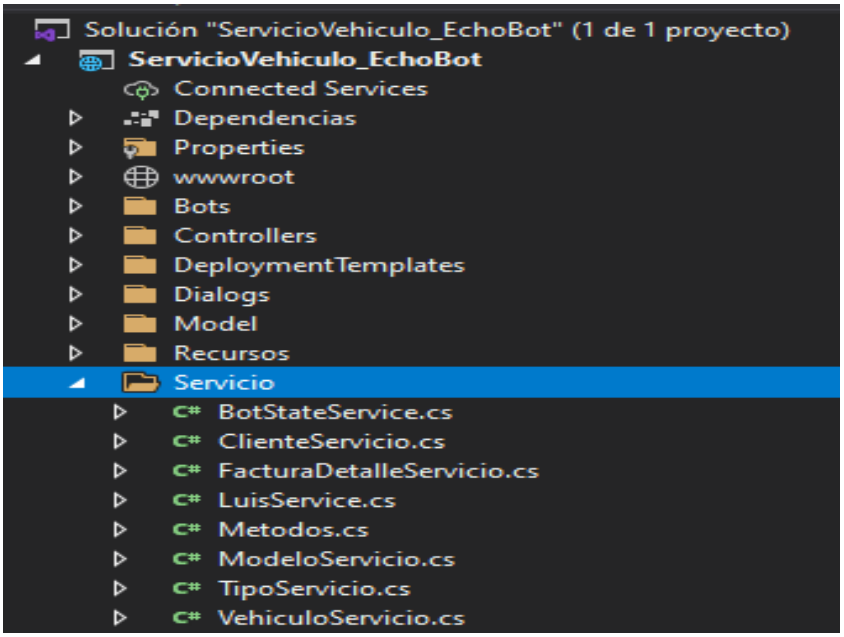


Figura C.10: Servicios Utilizados en el Cliente.
Fuente: Elaborado por el investigador.

Anexo D.-Ejecución del Sistema desde Entorno Local

- **Listando los Vehículos**

Para poder visualizar los vehículos registrados, primero se debe ingresar cualquier texto, inmediatamente el bot responderá con el Dialogo de Ingreso al Sistema.

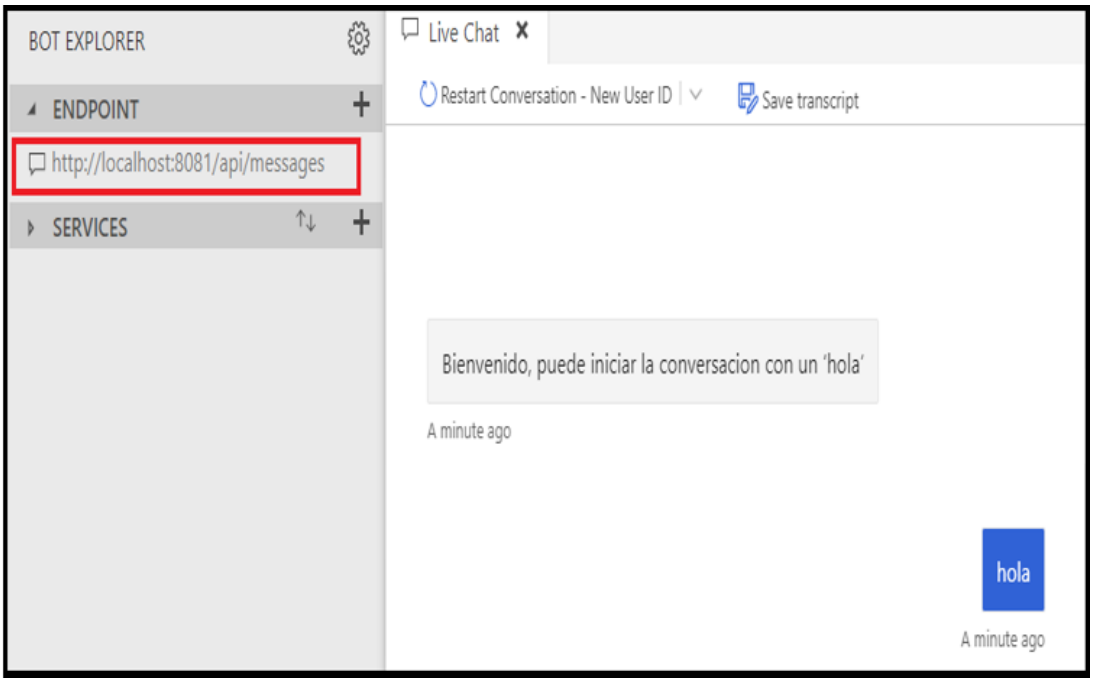


Figura D.1: Interfaz del Dialogo de Inicio.
Fuente: Elaborado por el investigador.

Una vez ingresado cualquier texto se cargará automáticamente el dialogo de Ingreso al Sistema, el usuario deberá ingresar sus credenciales correctamente, caso contrario el bot mostrar un mensaje notificando que los datos son incorrectos.

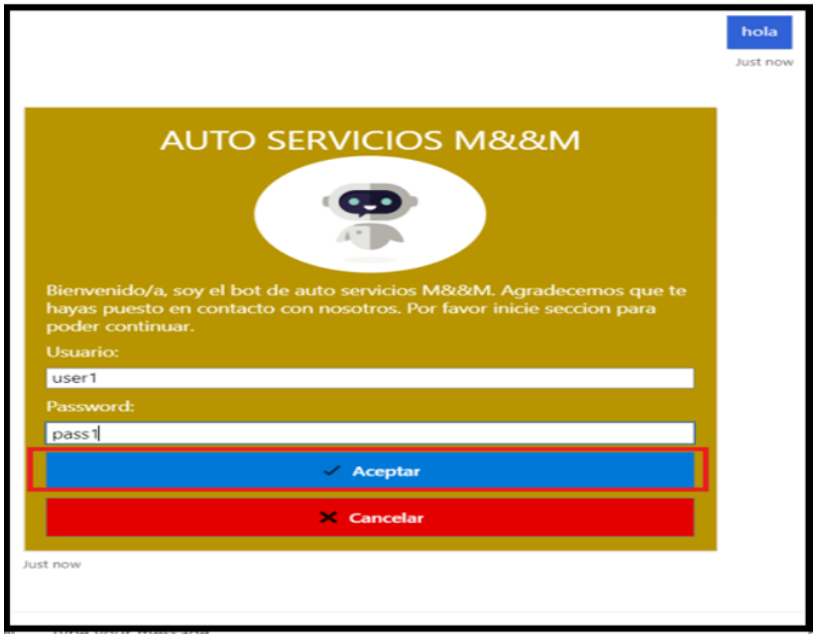


Figura D.2: Interfaz del Dialogo de Ingreso al Sistema.
Fuente: Elaborado por el investigador.

Luego de la autenticación exitosa, el bot mostrar el Dialogo de Opciones, esta interfaz presentara las funcionalidades que dispone el bot, como se muestra a continuación:

Figura D.3: Interfaz del Dialogo de Opciones.
Fuente: Elaborado por el investigador.

Se debe seleccionar la primera opción “Listar Vehículos” o también se puede escribir “ver vehículos” inmediatamente se desplegará un dialogo con la lista de vehículos registrados.

Figura D.4: Interfaz del Dialogo para Listar Vehículos.
Fuente: Elaborado por el investigador.

- **Ingresando un Vehículo**

Se debe seleccionar la segunda opción “Ingresar Vehículos” o escribir la palabra “ingresar”.

Figura D.5: Seleccionando la opción Ingreso de Vehículos.
Fuente: Elaborado por el investigador.

Luego se carga un dialogo con el formulario de ingreso del vehículo, se debe llenar los datos: Placa, Modelo, Chasis, Color y Modelo, posteriormente se debe “Guardar”.

Figura D.6: Formulario de Ingreso de un Vehículo.
Fuente: Elaborado por el investigador.

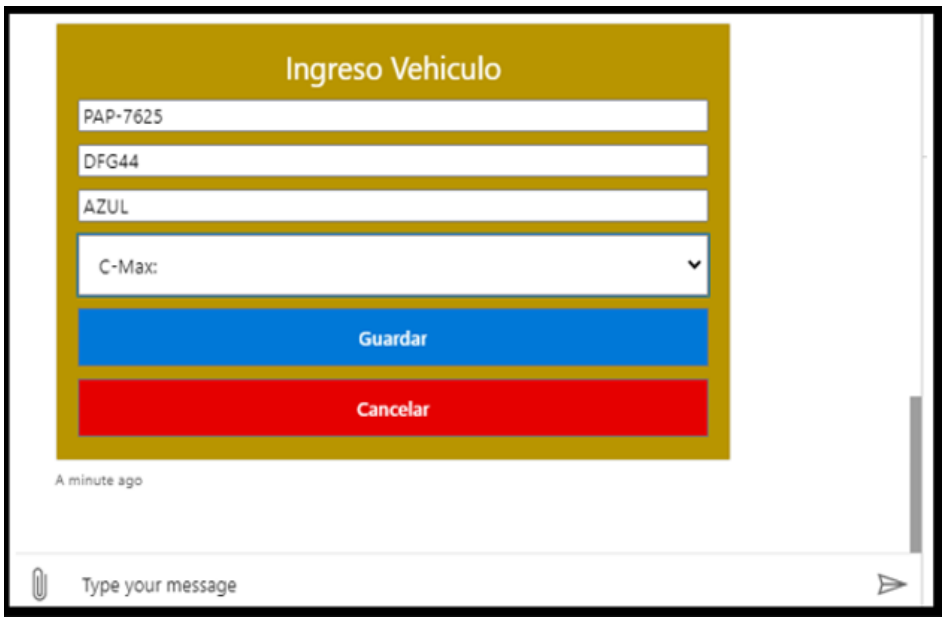


Figura D.7: Formulario con Datos del Vehículo.
Fuente: Elaborado por el investigador.

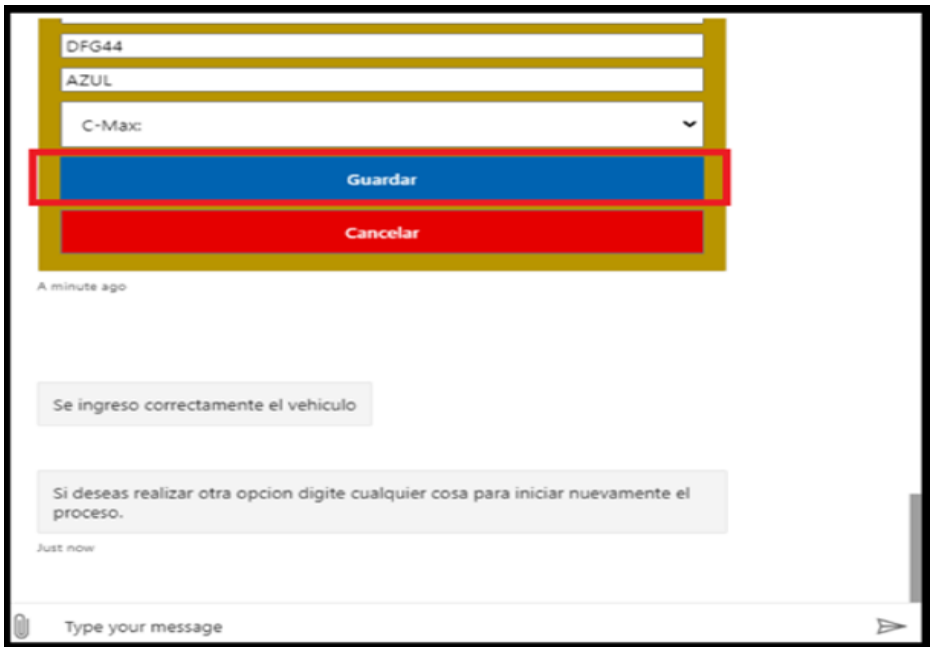


Figura D.8: Confirmación del Registro.
Fuente: Elaborado por el investigador.

- **Realizando una Compra**

Para realizar una compra se debe seleccionar la opción “Comprar” o escribir la palabra “comprar”

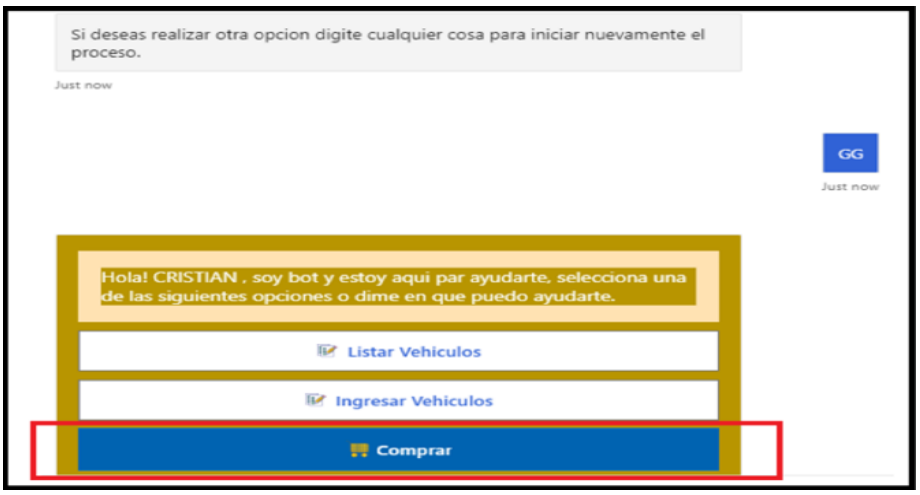


Figura D.9: Seleccionando la Opción Comprar.
Fuente: Elaborado por el investigador.

Luego se deberá selecciona un vehículo de la lista:

Figura D.10: Selección de un Vehículo.
Fuente: Elaborado por el investigador.

Una vez seleccionado el vehículo se debe seleccionar uno más Servicios Vehiculares de la lista:

Figura D.11: Seleccionando un Servicio Vehicular.
Fuente: Elaborado por el investigador.

Una vez seleccionado el Servicio Vehicular, el bot preguntara si desea agregas más servicios, si se dese agregar más se debe confirmar el mensaje con un “si”

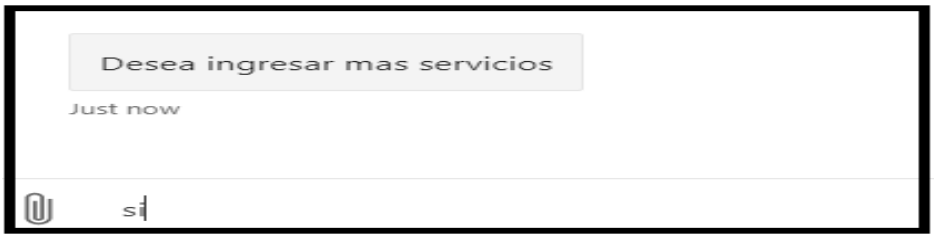


Figura D.12: Sugerencia del Chatbot.
Fuente: Elaborado por el investigador.

Automáticamente se volverá a presentar el dialogo de Servicios Vehiculares para poder seleccionar otro servicio:

Figura D.13: Seleccionado otro Servicio Vehicular.
Fuente: Elaborado por el investigador.

Una vez agregado el segundo Servicio Vehicular, el bot volverá hacer la misma pregunta, si el usuario responde “no”, se carga automáticamente un dialogo con una vista previa de la factura, luego el bot hará otra pregunta pidiendo autorización para finalizar la compra, para que se registre la compra el usuario deberá confirmar con un “si”:

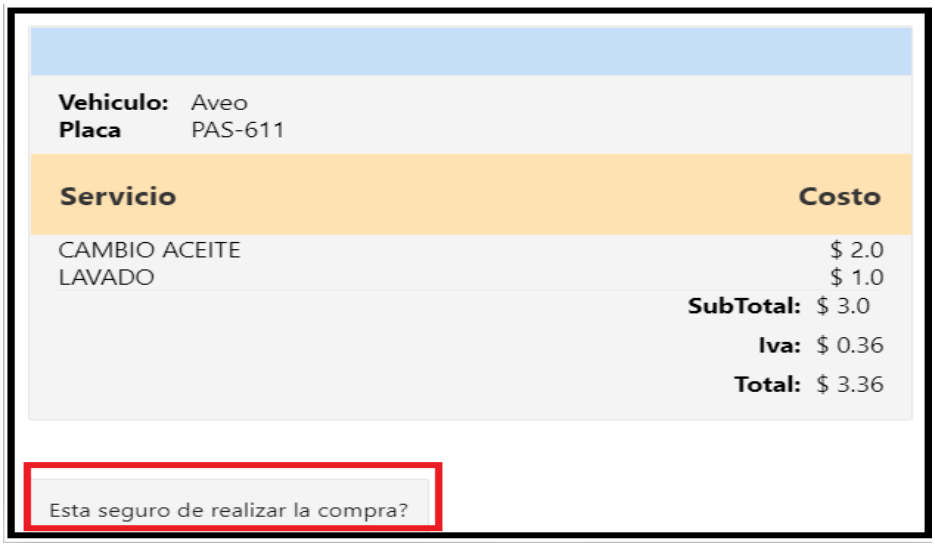


Figura D.14: Vista previa de la Compra.
Fuente: Elaborado por el investigador.

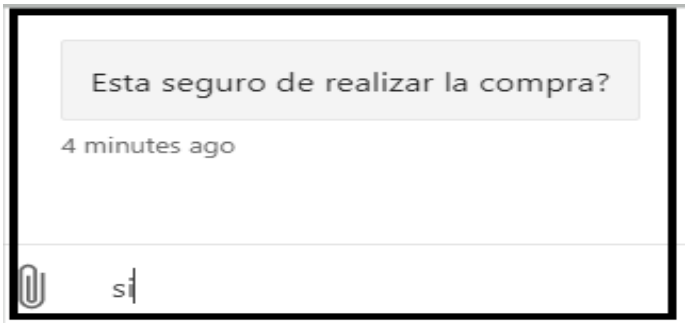


Figura D.15: Confirmación de la Compra.
Fuente: Elaborado por el investigador.

Finalmente, si el usuario confirma la finalización de la compra se mostrará automáticamente el dialogo de la Compra.

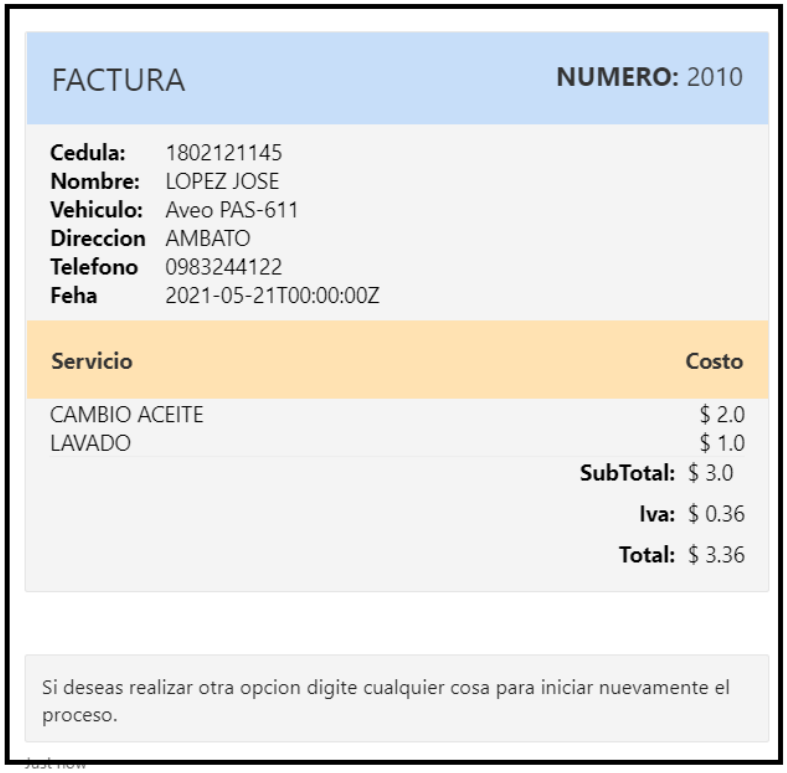


Figura D.16: Compra Finalizada.
Fuente: Elaborado por el investigador.