ANEXO IV: Diseño del sistema software.

Chatbot aplicado a la resolución de consultas en asignaturas de Bases de Datos

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

Septiembre de 2023

Autor:

Manuel Santa Isabel Mayo

Tutoras:

Ana Belén Gil González

Ana De Luis Reboredo

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

El objetivo de este anexo es buscar una abstracción muy baja y cercana de cómo será nuestro proyecto software final. En las anteriores etapas, hemos ido dividiendo nuestro trabajo en diferentes tareas, subsistemas, en funcionalidades, etc. Aunque todos estos conceptos eran imprescindibles para entender a grandes rasgos cómo es el proyecto para desarrollar, aún en el anexo de análisis teníamos una abstracción muy alta.

Con este anexo buscamos plantear de forma realista como es la arquitectura de nuestro sistema. Acercándonos a los patrones arquitectónicos usados, las bases de datos y su estructura, la distribución de nuestro programa, etc.

Dividiremos el anexo en las siguientes partes:

- Modelo de diseño
    - Patrones y estilos arquitectónicos
    - Subsistemas arquitectónicos
    - Clases de diseño
    - Vista arquitectónica
    - Relación Casos de Uso-Diseño

- Diseño de bases de datos
- Diseño de interfaz

2. MODELO DE DISEÑO

En este apartado del anexo hablaremos de los distintos patrones arquitectónicos usados para la creación de nuestro proyecto web. Hablaremos de dos diseños arquitectónicos los cuales fueron usados: Arquitectura Cliente Servidor y el modelo de Frontend Backend.

2.1. Patrones y estilos arquitectónicos

2.1.1. Arquitectura cliente-servidor

Este estilo de arquitectura de Software es uno de los más usados hoy en día. Esta arquitectura hace una separación en dos componentes dentro del sistema: Clientes (los cuales hacen peticiones) y Servidor (el que responde a dichas peticiones) y es de gran utilidad para sistemas distribuidos.

Estas dos partes trabajan juntas para proporcionar servicios, recursos e información a los usuarios.

Primero tenemos los Clientes, los cuales son las aplicaciones que interactúan con nuestros usuarios y hacen las peticiones a los Servicios.

En el caso de nuestra aplicación web, tenemos Clientes que solicitan al Servicio de ChatGPT con distintos mensajes. Nuestro servicio web (Cliente) interpretará todos los mensajes que nos responda el Servidor de ChatGPT para luego mostrarlos al usuario.

Por otra parte, tenemos al Servidor, que tiene como objetivo resolver las diferentes peticiones que se realizan por parte del Cliente.

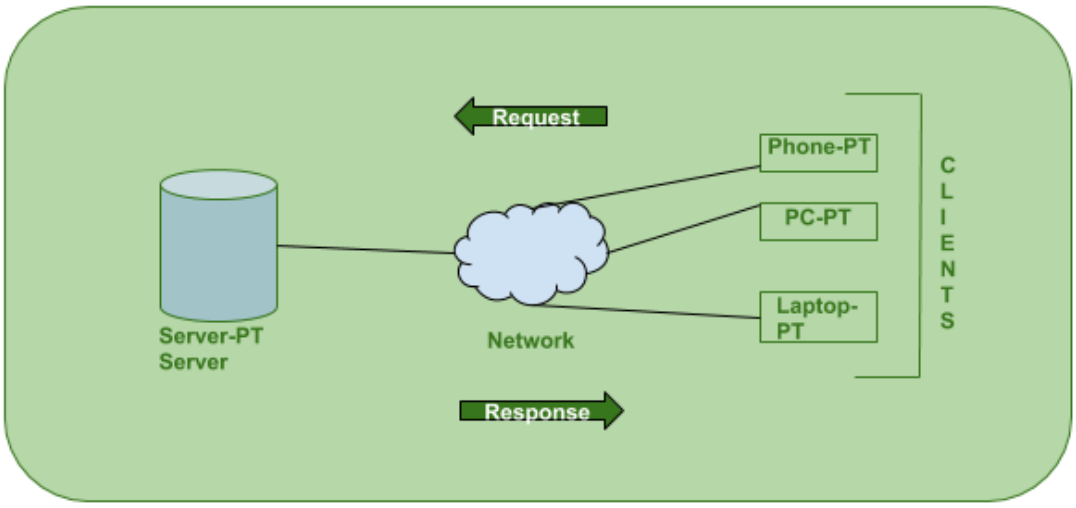

Figura 1. Ejemplo arquitectura cliente-servidor

2.1.2. Backend-Frontend

El siguiente modelo que se usó para crear nuestro servicio es el de Backend Frontend. Su estructura es muy similar a la de Cliente Servidor, pero nos servirá para explicar mejor las herramientas usadas dentro de nuestra Web.

Para empezar, tendremos una estructura que interactúa directamente con nuestro cliente, el Frontend. Dicha estructura está construida con el framework React JS, el cual interactúa con nuestro usuario mostrando una interfaz para la web, enseñando el ChatBot, y administrando diversos datos internos (como por ejemplo para comprobar la autentificación de los usuarios). Podríamos definirlo como una gran estructura en la cual se juntan tantos componentes de Vista Modelo y Controlador. Esto se debe a que se fusionan tanto código HTML, CSS y JavaScript (con diferentes modificaciones) Otra de las funciones que cumple el Frontend son las llamadas a ChatGPT.

Lo siguiente que tendremos será el Backend, que está construido con la herramienta de Django. El Backend funcionara como un servidor FULLRest, el cual solo acepta peticiones de HTTP para comunicarse.
Nuestro Backend de Django también se comunicará con una base de datos, en la que guardaremos datos como los usuarios, los mensajes que escriben y las distintas peticiones que realizan a ChatGPT.

Para comunicarse el Backend y el Frontend, se realizarán llamadas HTTP con la herramienta AXIOS de ReactJS.

Esta estructura es muy similar a la arquitectura Cliente Servidor, no obstante, nuestro sistema no está distribuido en diferentes servidores. En este caso el Frontend se alojará dentro del mismo servidor que el Backend. Para ello simplemente se crea una versión Build del Frontend para almacenar lo dentro del servidor de Django.

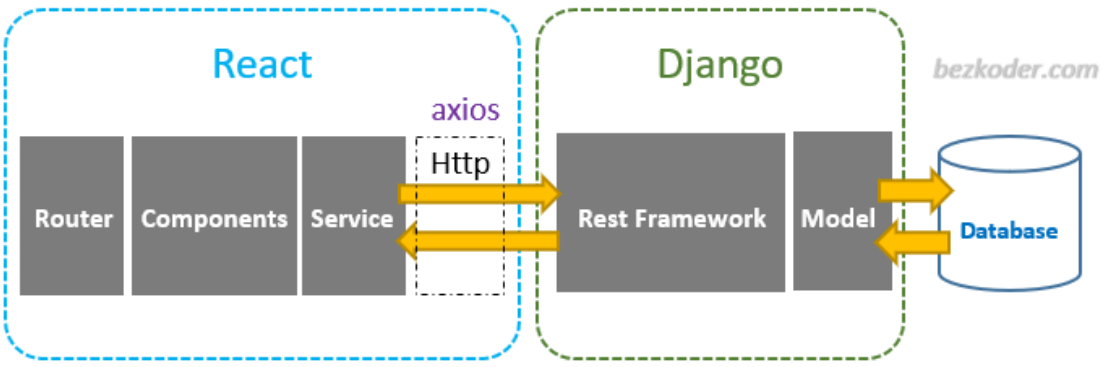

Figura 2. Ejemplo Bakend-Frontend con React y Django

2.2. Subsistemas de diseño

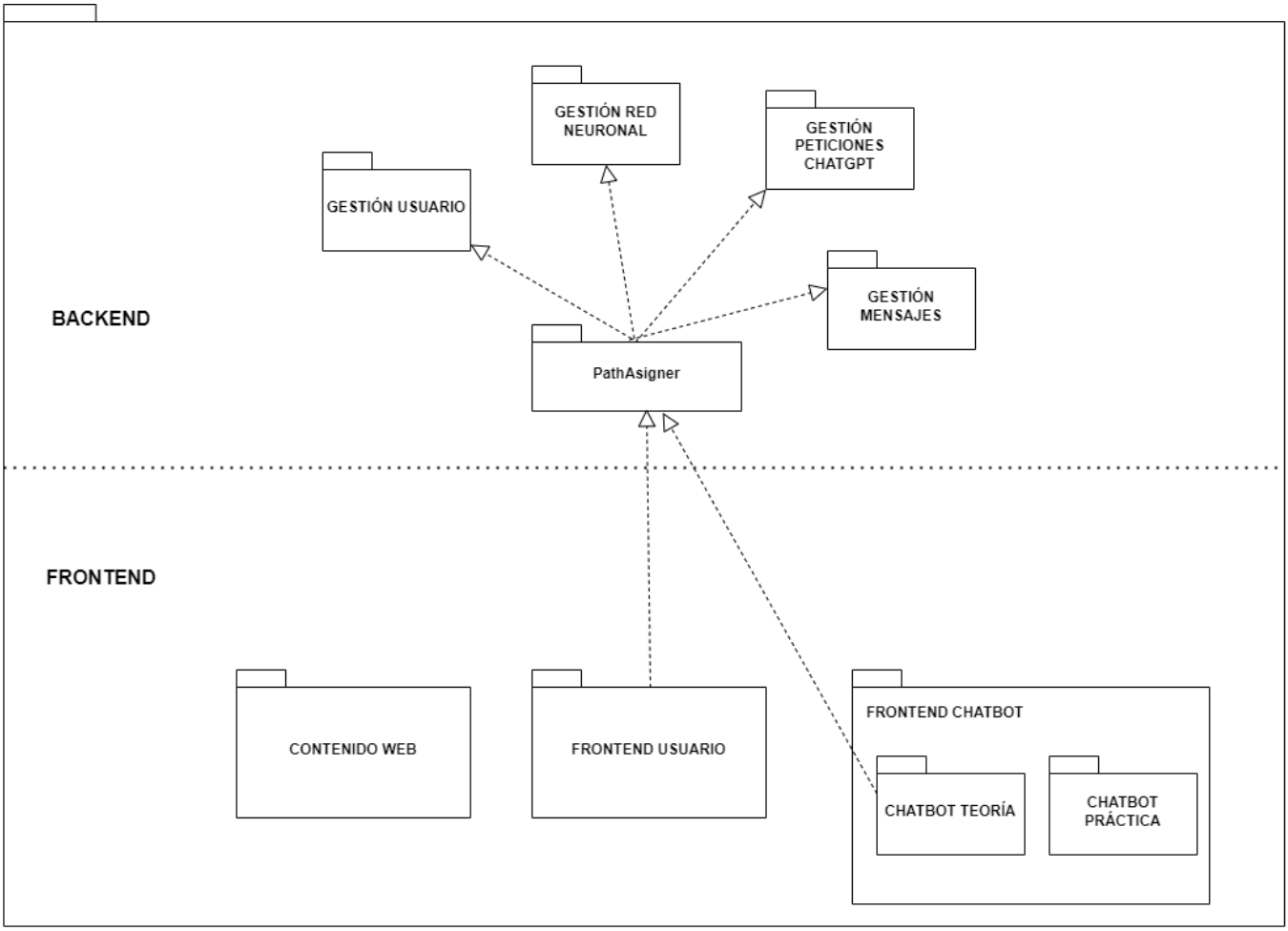

Figura 3. Subsistemas de diseño

2.3. Clases de diseño

En este apartado, describiremos el anterior diagrama en el que se dividía el sistema en subpaquetes diferenciados entre el Backend y el Frontend.

Para ello debemos tener en cuenta que en React, muchos de los archivos pueden tener una estructura que mezcle el modelo VMC(Vista-Modelo-Controlador). Para poder comprender mejor estos archivos y el flujo de información que hay en el diseño, se ha decidido renombrar los paquetes para poder entender mejor los próximos diagramas de flujo. En otras palabras, un mismo archivo puede aparecer en múltiples dominios, pero con el nombre cambiado para diferenciarlo en secciones.

2.3.1. Paquetes Frontend

2.3.1.1. Contenido web

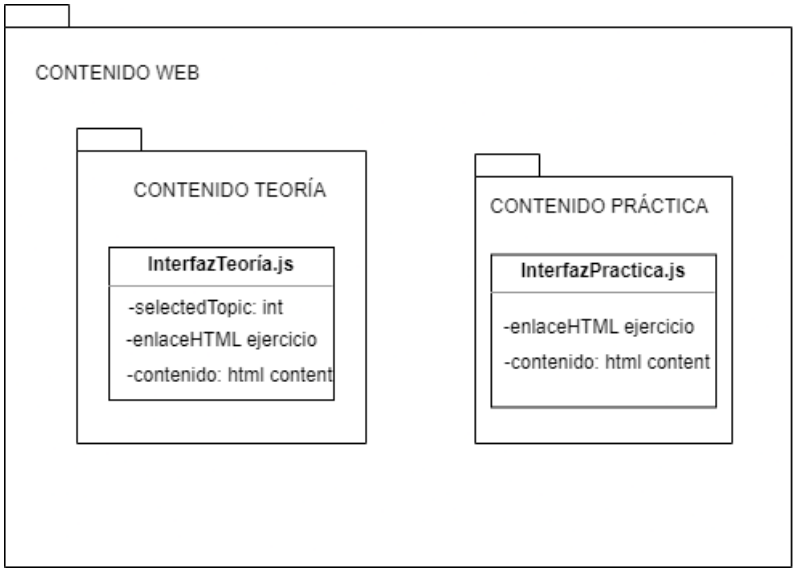

Figura 4. Subpaquete Contenido web

El subpaquete de Contenido Web se divide en otros dos subpaquetes. Uno para el contenido de la parte teórica de la web y la otra para la parte práctica. Estos se encontrarán en archivos de React distintos.

Como se mencionó anteriormente, un mismo archivo puede ser renombrado para diferenciar sus apartados, en este caso Teroria.js y Practica.js se renombrarán por Interfaz.

El contenido de estos archivos es muy simple, solo consta de los elementos de HTML y los enlaces para poder comunicar ambos subpaquetes.

2.3.1.2. Frontend usuario

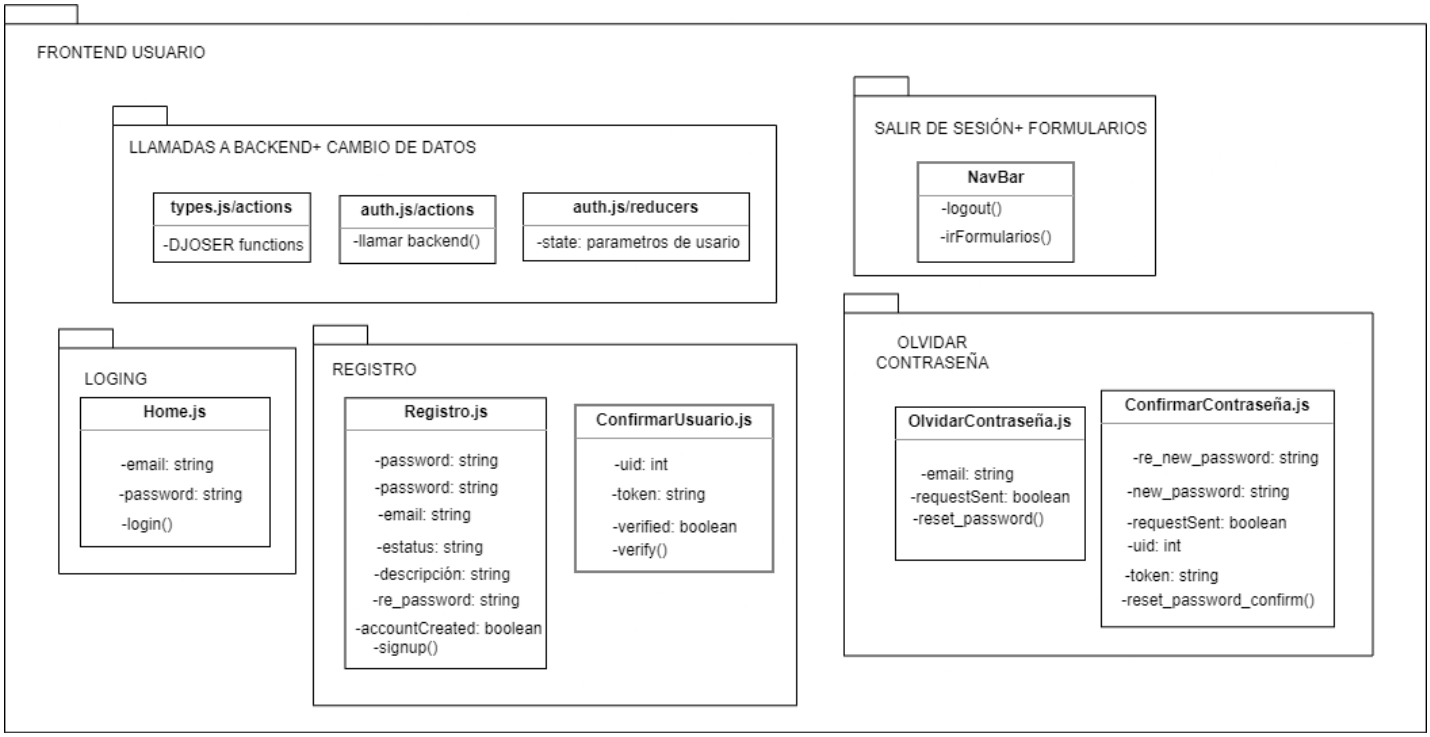

Figura 5. Subpaquete Frontend usuario

Este paquete se dividirá en los siguientes subpaquetes:

- **Llamadas a Backend+ Cambio de datos**: subpaquete que se encargará de manejar todas las acciones de llamada al Backend y los cambios que ellos suponen. Para poder realizar las llamadas al Backend se realizarán las llamadas del archivo auth.js/actions. Dichos cambios se verán afectados en los datos del archivo auth.js/reducers. La variable que se guardan en ese último archivo serán los datos del usuario que este interactuando con el Frontend.
- **Loging**: Subpaquete encargado de ser la interfaz de entrada a los usuarios.
- **Registro**: Subpaquete encargado de ser la interfaz de registro y validación de los nuevos usuarios a la web.
- **Olvidar Contraseña**: Subpaquete encargado de ser la interfaz de recuperación de una contraseña de un usuario ya registrado y restablecer una nueva contraseña.
- **Salir de la sesión+ formularios**: Subpaquete encargado de la interfaz para que un usuario salga de su sesión y de llevar a los usuarios al formulario de valoración de la web.

2.3.1.3. Chatbot teoría

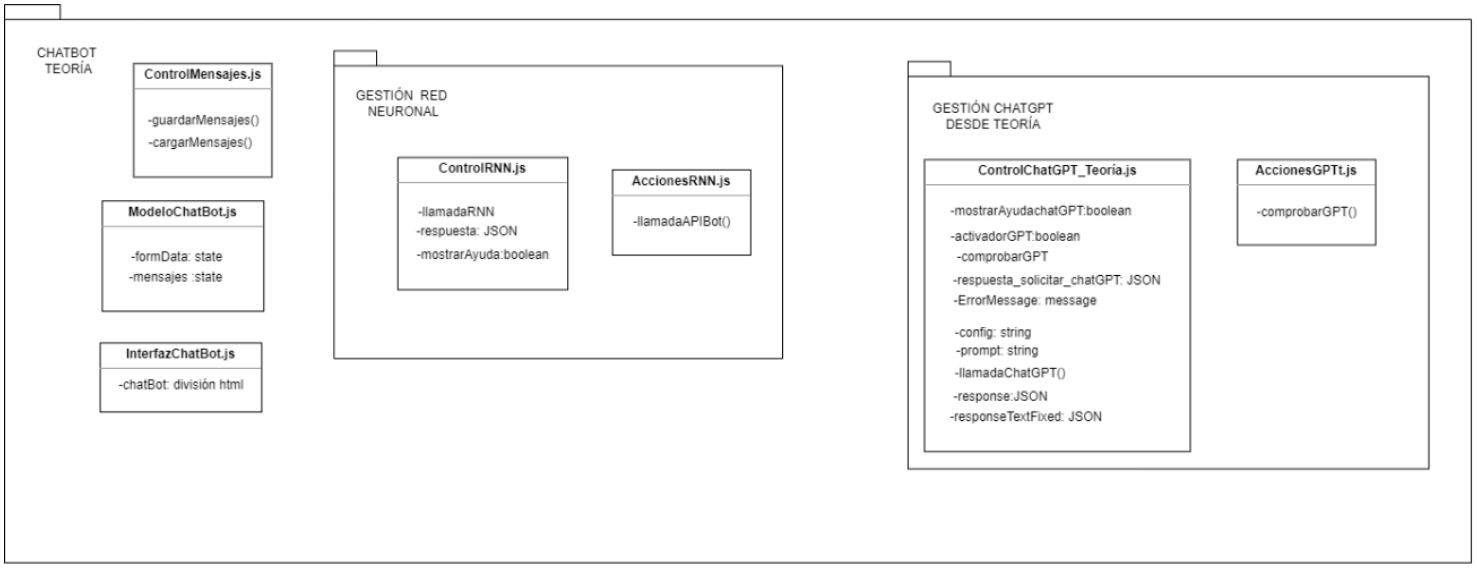

Figura 6. Subpaquete Chatbot teoría

Primero de todo tenemos tres archivos encargados del manejo de los distintos mensajes que aparecen dentro del Chatbot. Todos ellos pertenecen al archivo Teoría.js, que se ha renombrado de esta forma para poder entender mejor cada una de sus partes.

Estos tres archivos se dividirán en Interfaz, Modelo y Controlador de los mensajes que hay en el chatbot.

Después nos encontramos con el subpaquete de Gestión de red neuronal. Dicho paquete este compuesto por dos archivos, el primero, **ControlRNN.js**, un controlador para las diferentes funciones de enviar mensajes de usuarios a la red neuronal y el procesamiento de dichas respuestas. El segundo archivo, **AccionesRNN.js**, tiene la función para llamar al Backend para pedir una respuesta de la red neuronal.

El último subpaquete dentro del ChatBot de Teoría se trata del paquete de Gestión de ChatGPT desde Teoría. Como se mencionó anteriormente en la descripción de Backend y Frontend, el propio React es el que se encarga de hacer la llamada a ChatGPT, no obstante, antes de hacer una llamada a esta API, debemos comprobar los límites de un usuario.
El paquete se divide en dos subpaquetes. El primero **ControlChatGPT_Teoria.js** se encarga de todo lo respectivo a la llamada y el control de la función de preguntar a ChatGPT. El segundo archivo, AccionesGPT.js, solo tiene la función para preguntar al Backend si un usuario con un determinado mensaje puede hacer la llamada a la API.

2.3.1.4. Chatbot práctica

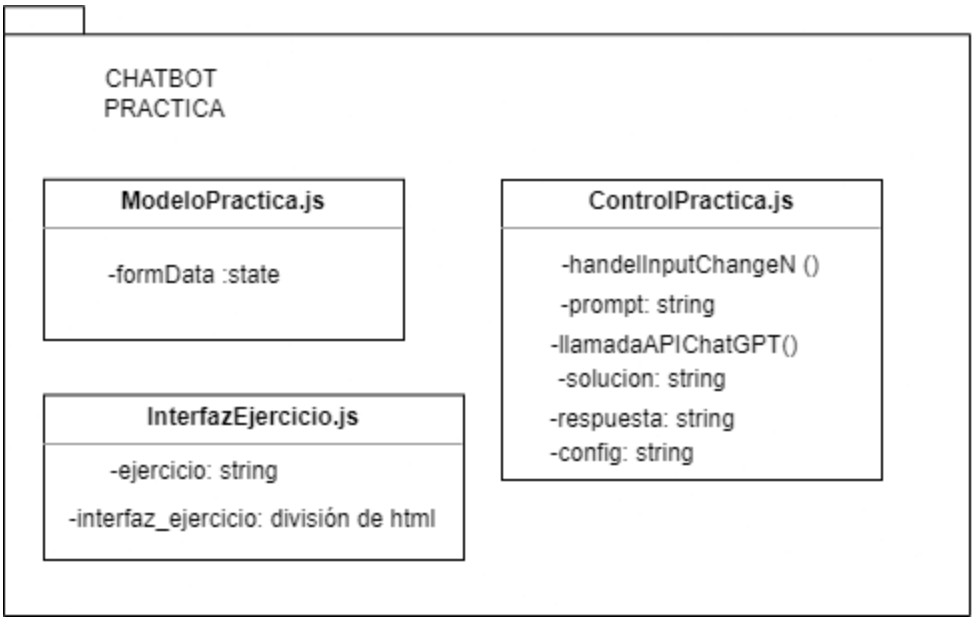

Figura 7. Subpaquete Chatbot práctica

Este paquete se dividirá en tres archivos, que realmente solo es el archivo Practica.js renombrado para diferenciar cada una de sus secciones.

- ModeloPractica.js: Encargado de guardar las distintas soluciones que un usuario hace para cada una de las respuestas.
- InterfazEjercicio.js: Interfaz para que un usuario pueda escribir la respuesta a un ejercicio y también el espacio donde se verá la corrección de dicho ejercicio.
- ControlPractica.js: Se compone de los distintos parámetros para poder hacer las llamadas a ChatGPT. El objetivo de este archivo es comprobar si la solución de un usuario es correcta y darle una explicación del resultado.

En este paquete no hay ninguna llamada hacia el Backend de nuestro servicio.

2.3.2. Paquetes Backend

2.3.2.1. Gestión de Red Neuronal

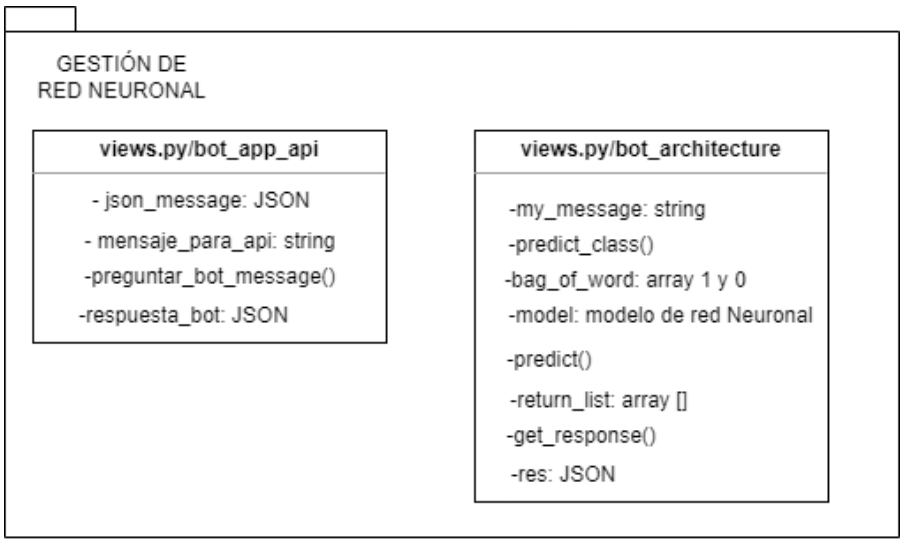

Figura 8. Subpaquete Gestión de Red Neuronal

En este paquete que se encuentra dentro del Backend, se diferencian dos archivos:

- Views.py/bot_app_api: Encargado de procesar y responder los mensajes que se reciben por parte del Frontend. También está encargado de hacer las llamadas a la propia Red Neuronal para que resuelva el mensaje.
- Views.py/bot_archietecture: Es el archive encargado de poder realizar la resolución de un mensaje con la red neuronal. Dentro de este archivo se encuentra el modelo de la red neuronal y otras funciones para poder resolver las distintas llamadas de los usuarios.

2.3.2.2. Gestión Peticiones ChatGPT

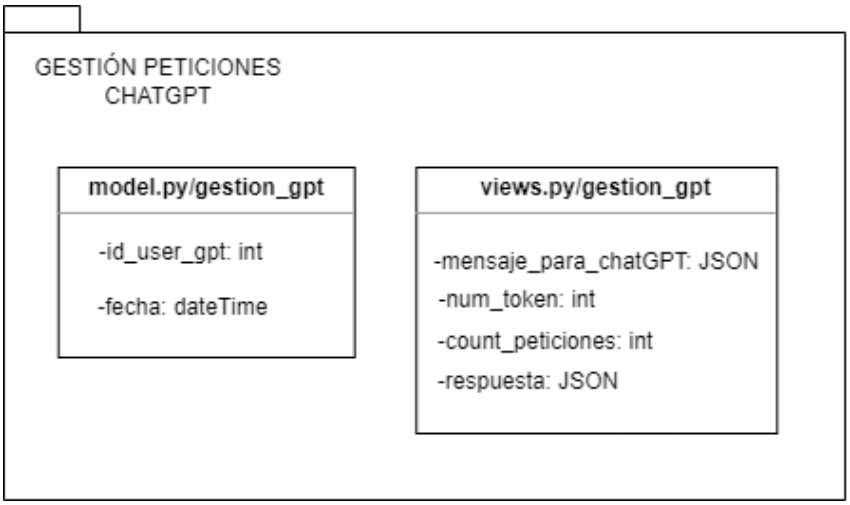

Figura 9. Subpaquete Peticiones ChatGPT

El siguiente subpaquete que vamos a tratar tiene las funcionalidades de registro y control de las diferentes peticiones de usuarios para llamar a ChatGPT. El objetivo es validar los límites de cada usuario respecto a la longitud de su mensaje y el número de llamadas diarias que hace a la API de ChatGPT.

Para poder cumplir estos objetivos, el paquete se divide en dos archivos:

- Model.py/gestión_gpt: archivo que será considerado el Modelo donde se guardan las diferentes peticiones de los usuarios.
- Views.py/gestión_gpt: archive que funciona como control para las diferentes llamadas de registro y validación de mensajes de usuarios. En este archivo se realiza la validación de la longitud de un mensaje y el número de peticiones diarias de un usuario. Como se ha ido explicando en varias veces a lo largo de los anexos, dependiendo del estatus del usuario los limites pueden variar.

2.3.2.3. Gestión Guardado de mensajes

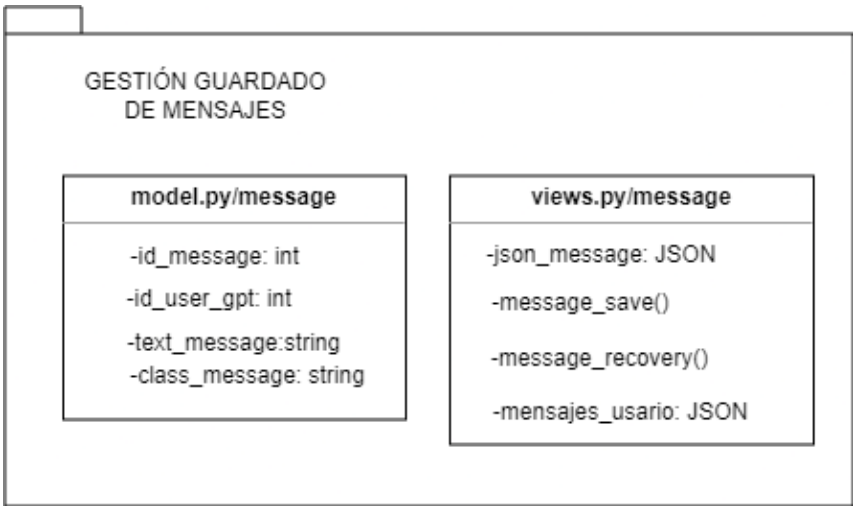

Figura 10. Subpaquete Gestión guardado de mensajes

Este paquete es muy similar al anterior. Ahora en vez de guardar las peticiones para las llamadas a ChatGPT, se van a guardar los mensajes entre el usuario y la API de ChatGPT.

Este paquete se divide en dos archivos: el Modelo (donde vamos a guardar los mensajes) y la Vista, que funcionará como Controlador para realizar las funciones de guardado de mensajes y de recuperación de mensajes.

2.3.2.4. Gestión Usuarios

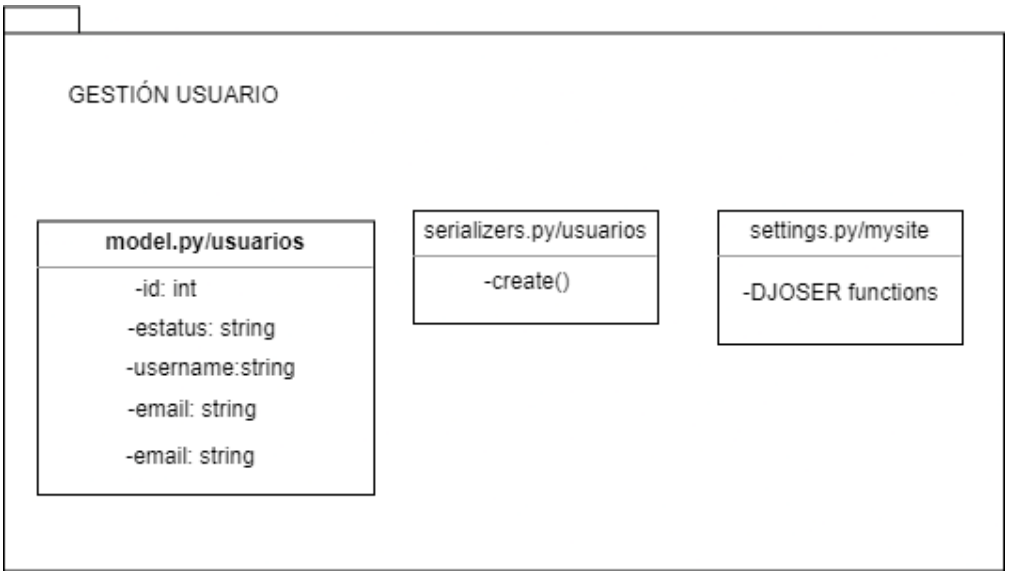

Figura 11. Subpaquete Gestión usuarios

El siguiente paquete que analizamos trata de la gestión que se tiene de los usuarios desde el Backend. Para empezar, debemos aclarar que muchas de las funciones vienen dadas de un paquete instalado para Django llamado DJSOER, dicho paquete está ubicado dentro del archivo **settigs.py/mysite**. Los otros dos archivos tendrán las siguientes funcionalidades:

- Model.pyt/usuarios: Este paquete contiene los datos que hemos personalizado para guardar a los usuarios de la web. Es importante mencionar que dentro de este archivo se ha especificado cuales son los diferentes tipos de estatus y que el email será la variable con la que un usuario se identifique a la hora de entrar a la web.
- Serializers.py/usuarios: es el archivo controlador encargada de realizar la función de crear un usuario validando sus parámetros.

2.3.2.5. PathAssigner

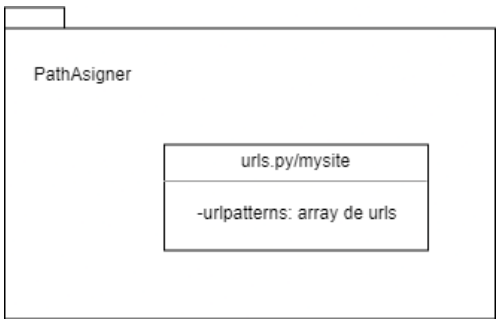

Figura 12. Subpaquete PathAssigner

Se trata del último paquete que tendremos dentro de nuestro dominio. Dicho paquete se encarga de redirigir todas las llamadas que se hacen desde el Frontend hacia el Backend, redirigiéndolas hacia el archivo correspondiente. Este paquete solo tiene un archivo que está compuesto por las diferentes rutas de los archivos del Backend.

2.4. Realización de casos de uso-diseño

Llegados a este punto de la etapa de diseño, vamos a proceder con la reinterpretación de los distintos casos uso. En este casó nuestros diagramas de secuencia venían de un contexto muy abstracto y solo nos centramos en el flujo de los datos que se transmitían.
Ahora con nuestro diseño mucho más desarrollado, podemos mostrar las secuencias de los datos desde un punto más fidedigno a la realidad. Vamos a reinterpretar los distintos diagramas de secuencia de la etapa de Análisis.

2.4.1. Contenido web

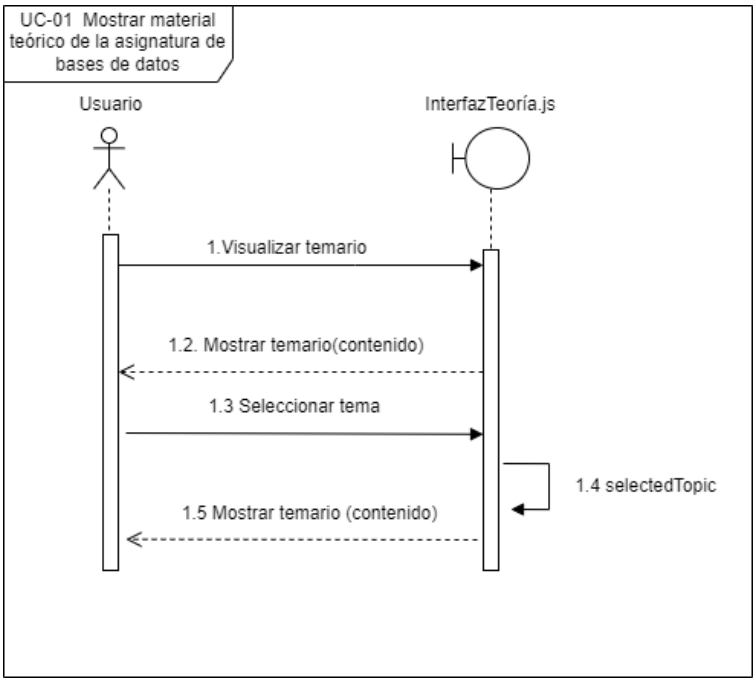

Figura 13. UC-Diseño-01 Mostrar material teórico de la asignatura de bases de datos

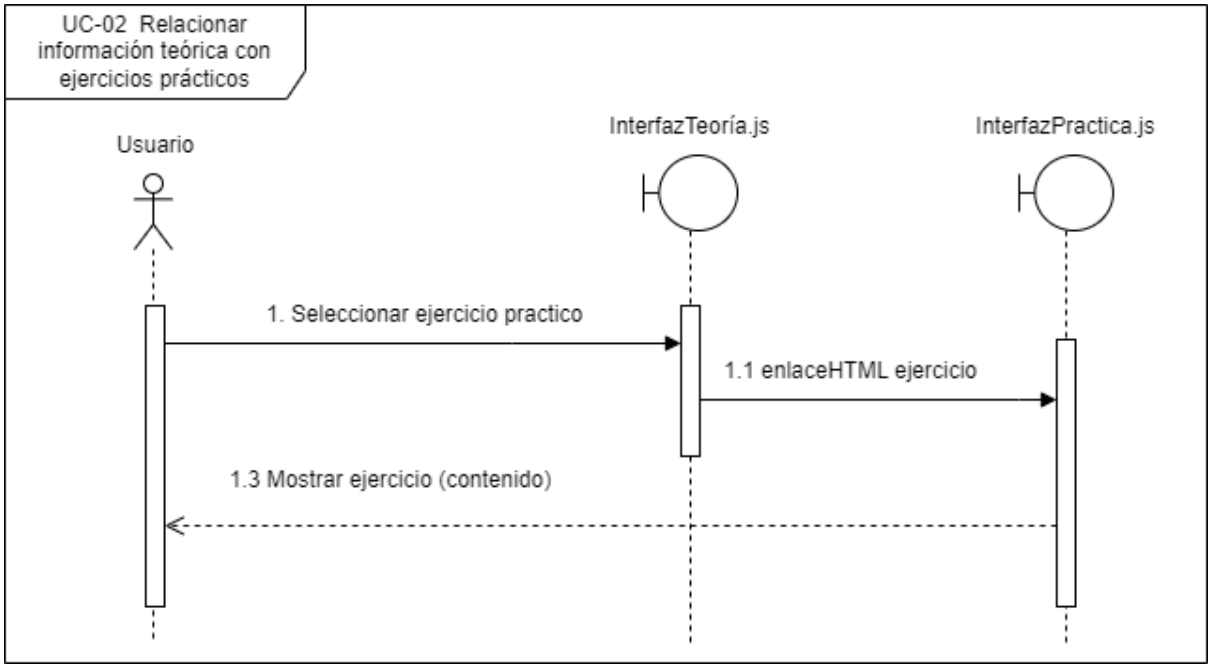

Figura 14. UC-Diseño-02 Relacionar información teórica con ejercicios prácticos

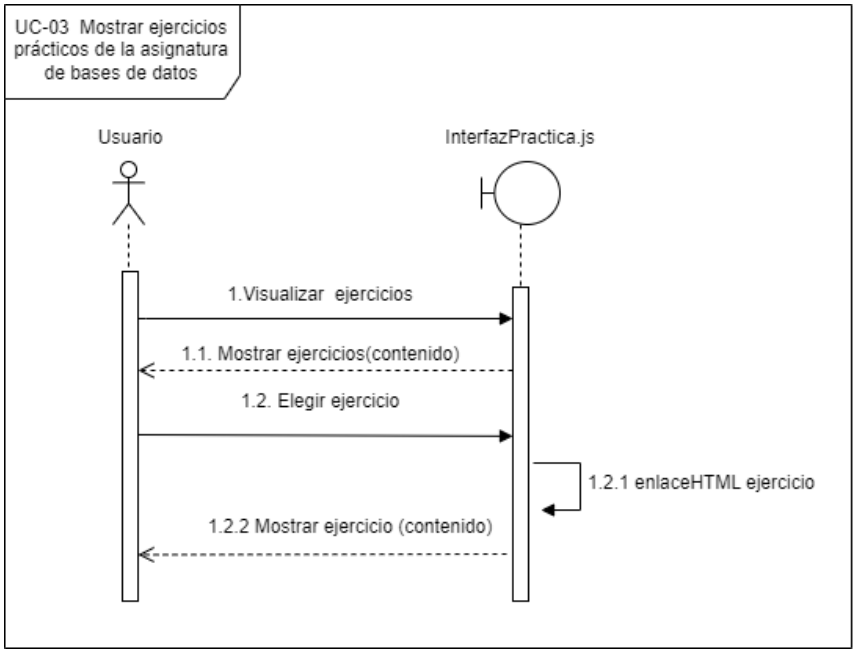

Figura 15. UC-Diseño-03 Mostrar ejercicios prácticos de la asignatura de bases de datos

2.4.2. Gestión del chatbot

2.4.2.1. Gestión del chatbot desde teoría

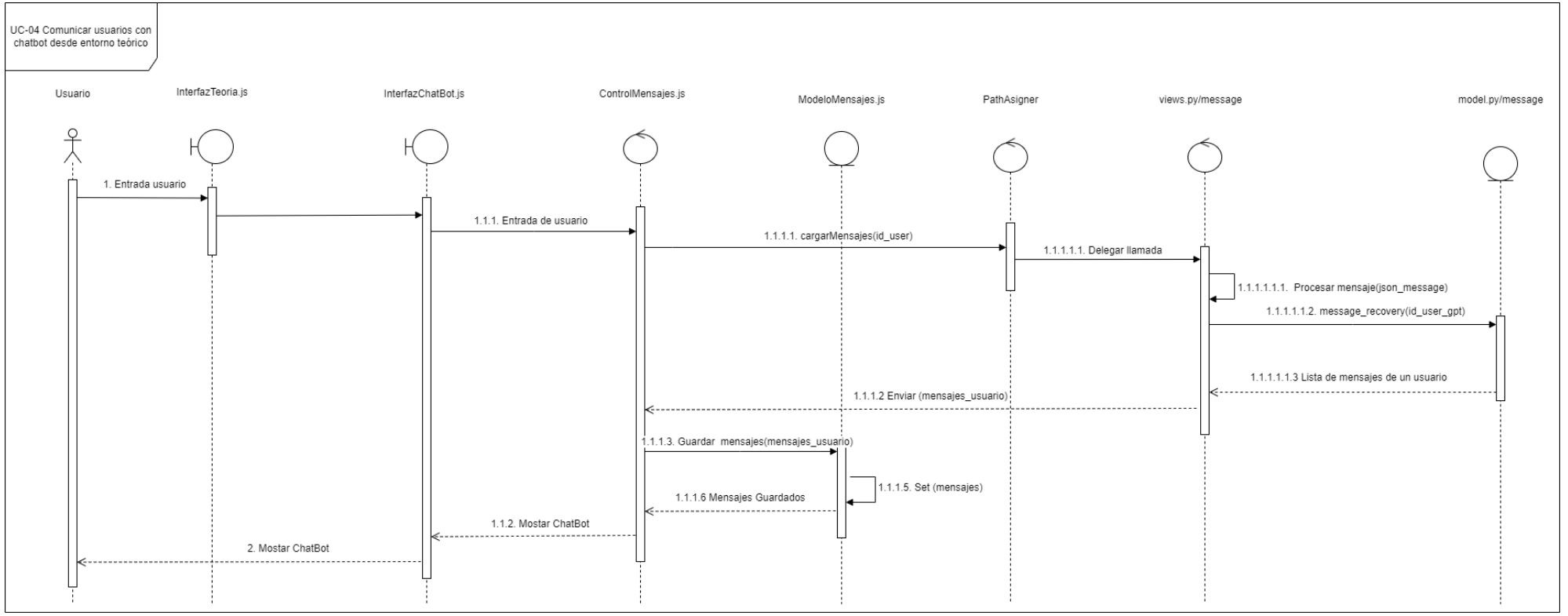

Figura 16. UC-Diseño-04 Comunicar usuarios con chatbot desde entorno teórico

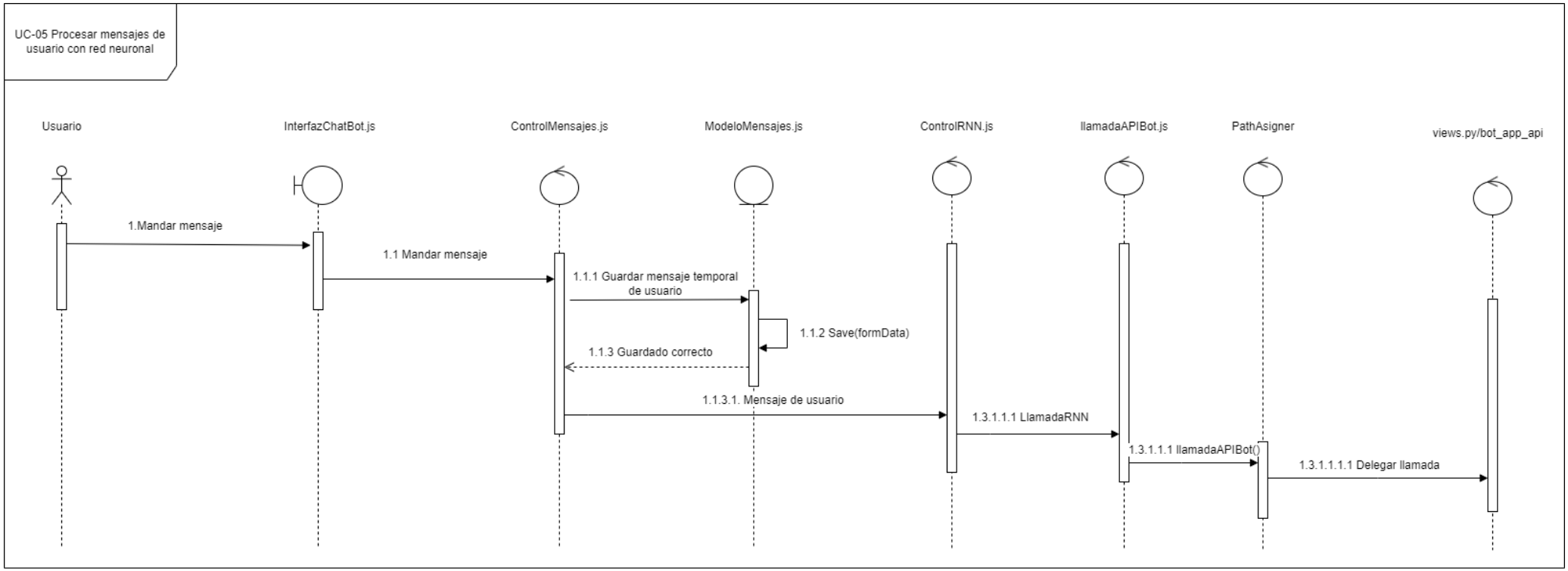

Figura 17. UC-Diseño-05 Procesar mensajes de usuario con red neuronal

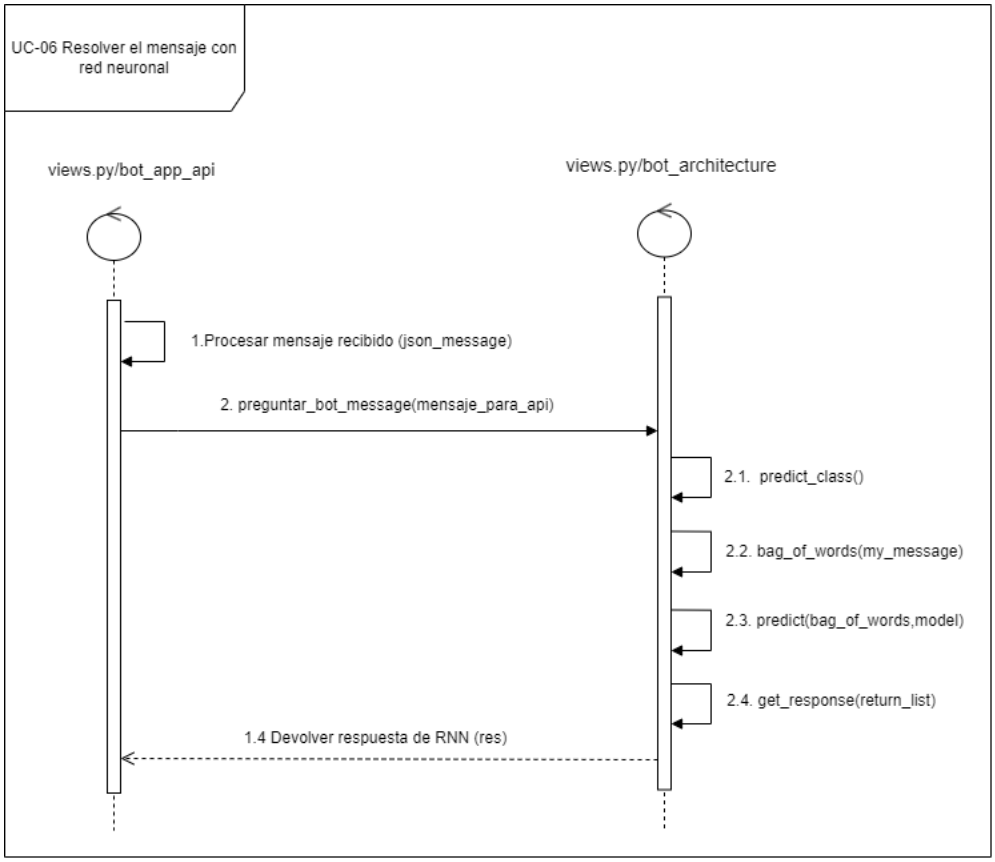

Figura 18. UC-Diseño-06 Resolver el mensaje con red neuronal

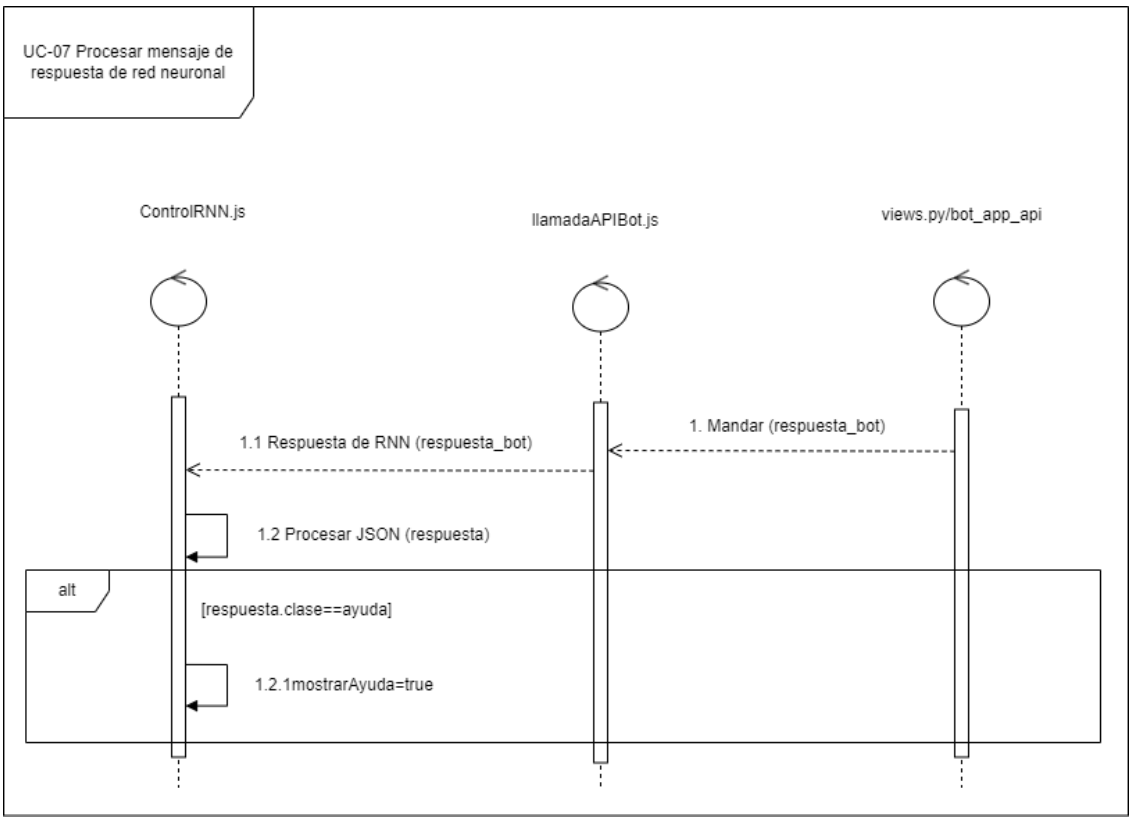

Figura 19. UC-Diseño-07 Procesar mensaje de respuesta de red neuronal

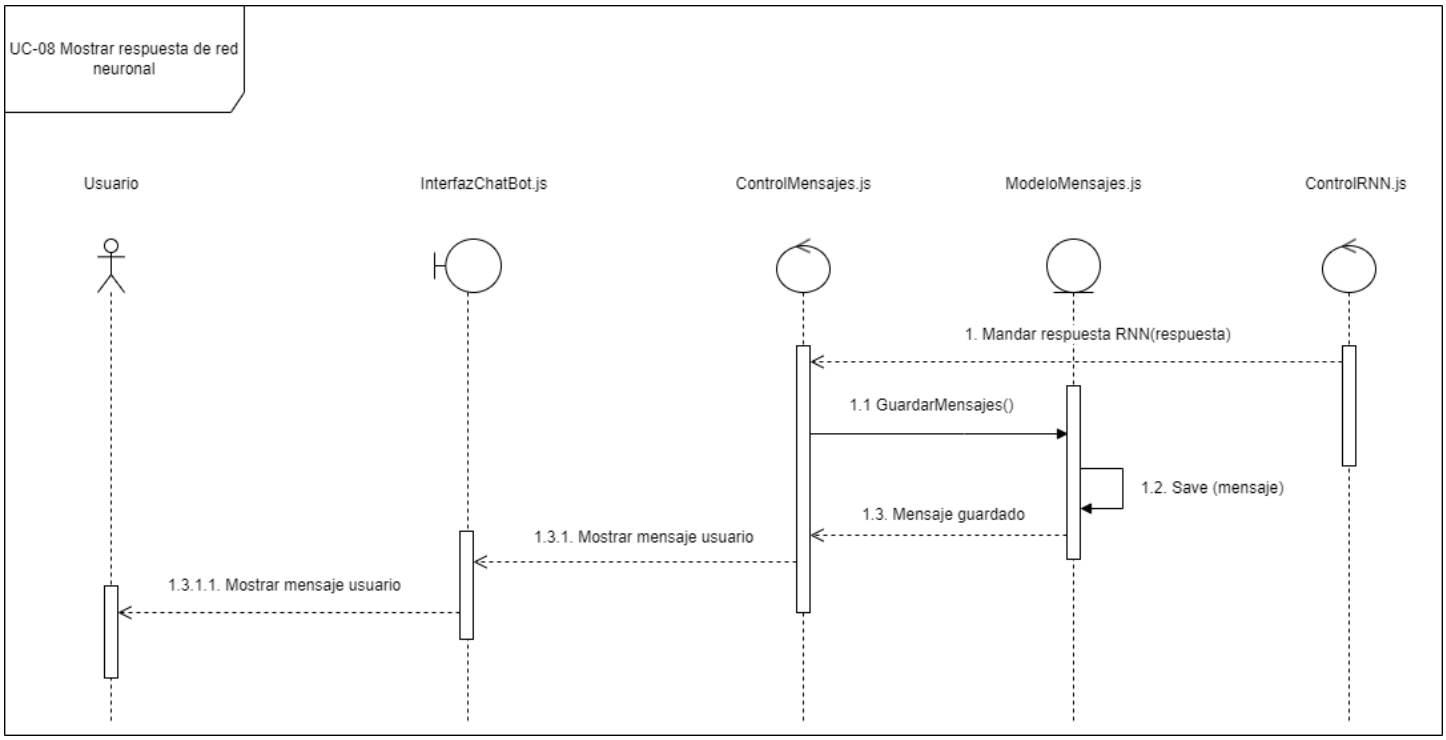

Figura 20- UC-Diseño-08 Mostrar respuesta de red neuronal

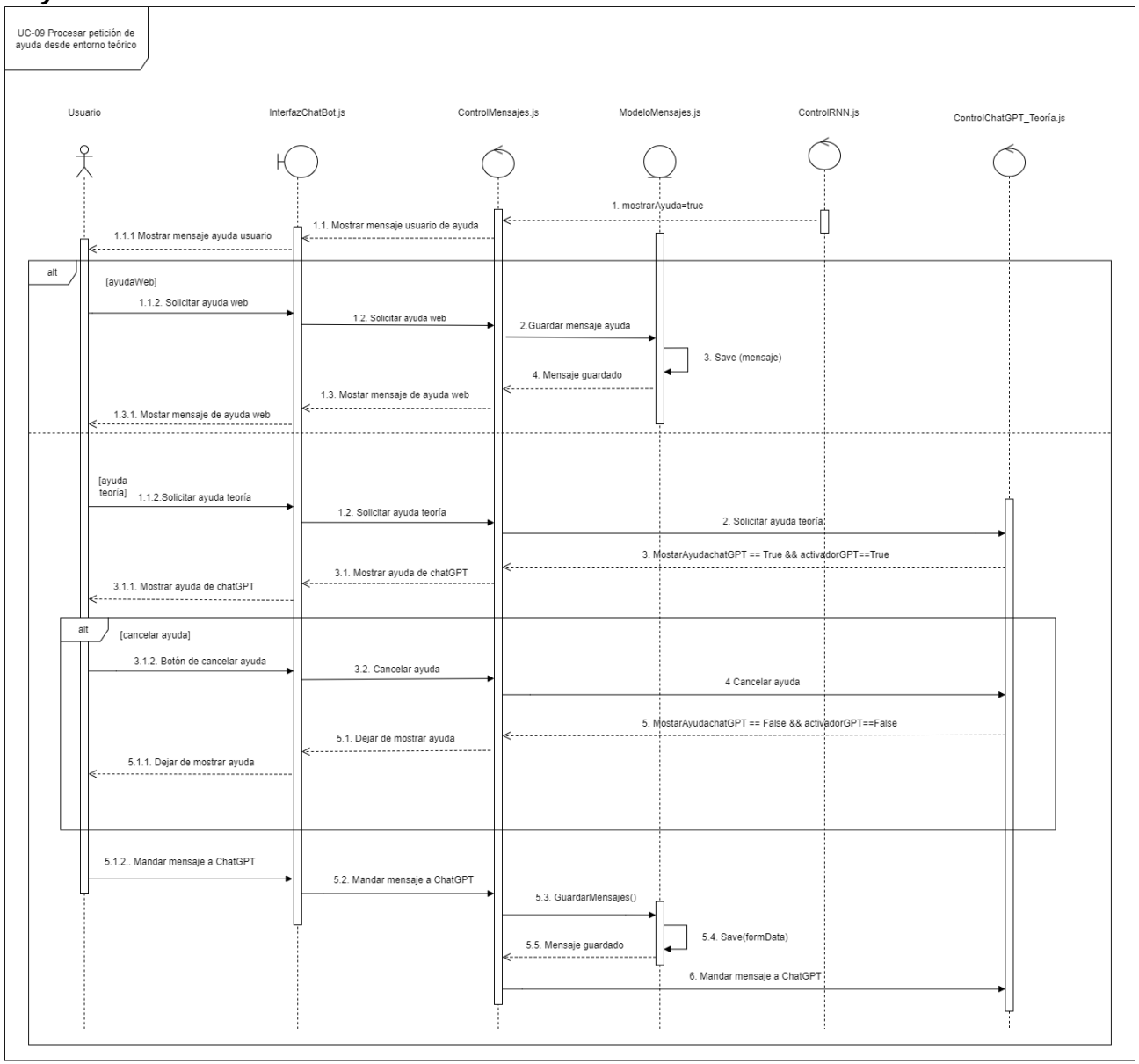

Figura 21. UC-Diseño-09 Procesar petición de ayuda desde entorno teórico

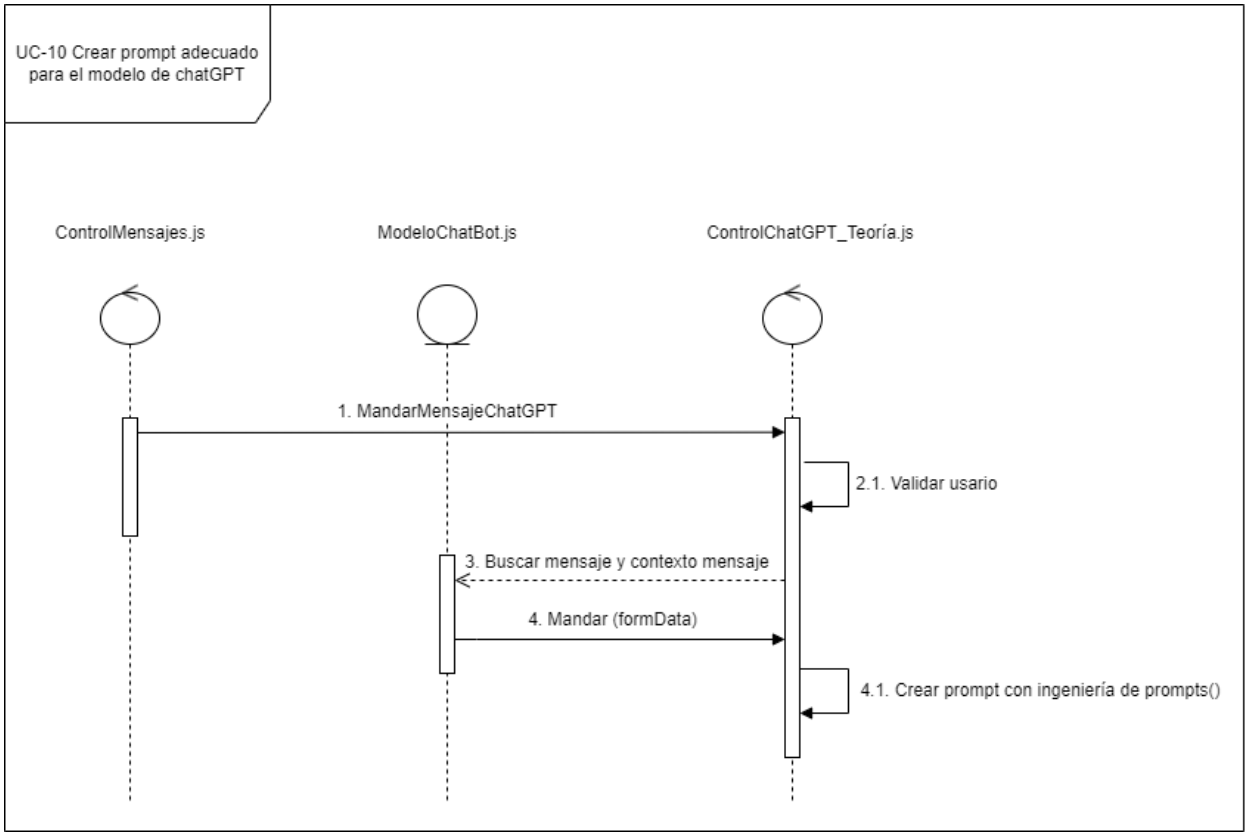

Figura 22. UC-Diseño-10 Crear prompt adecuado para el modelo de ChatGPT

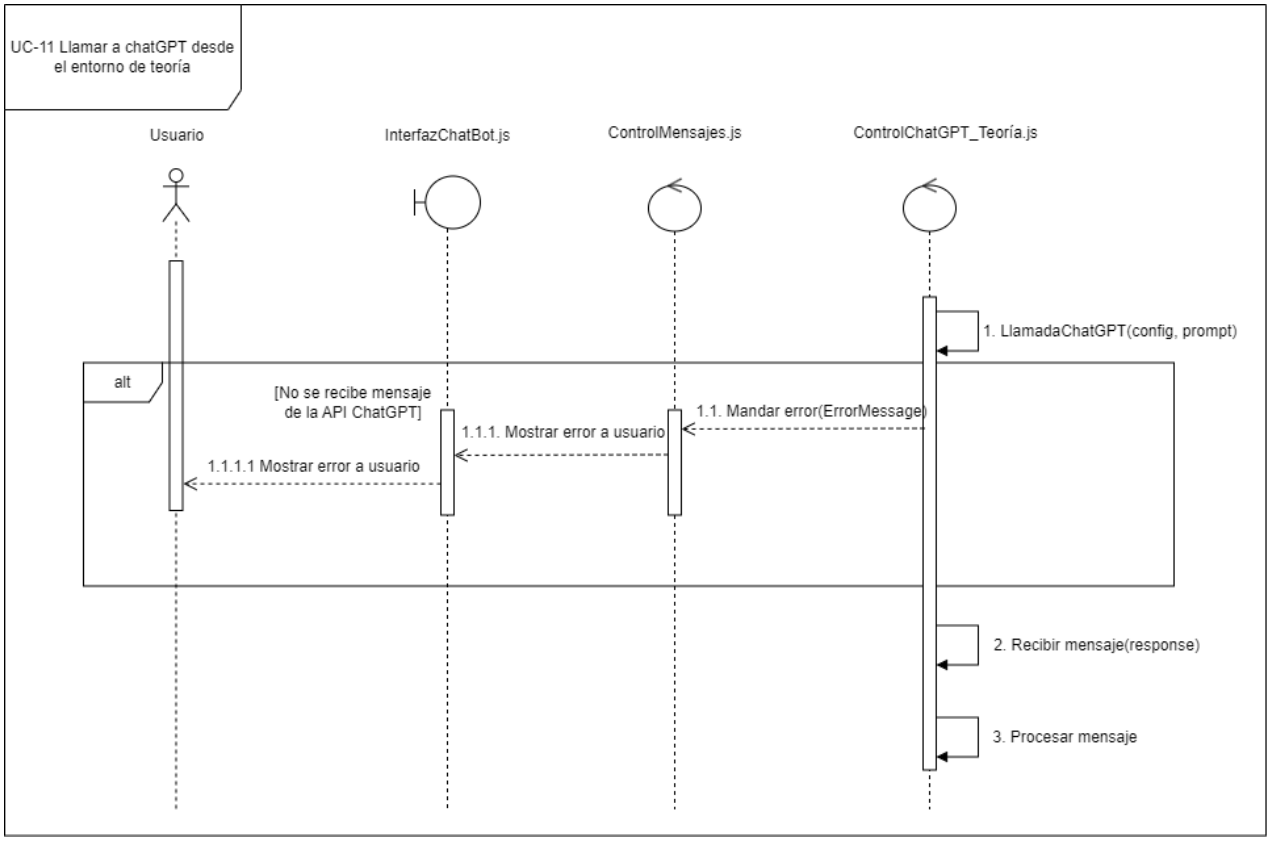

Figura 23. UC-Diseño-11 Llamar a ChatGPT desde el entorno de teoría

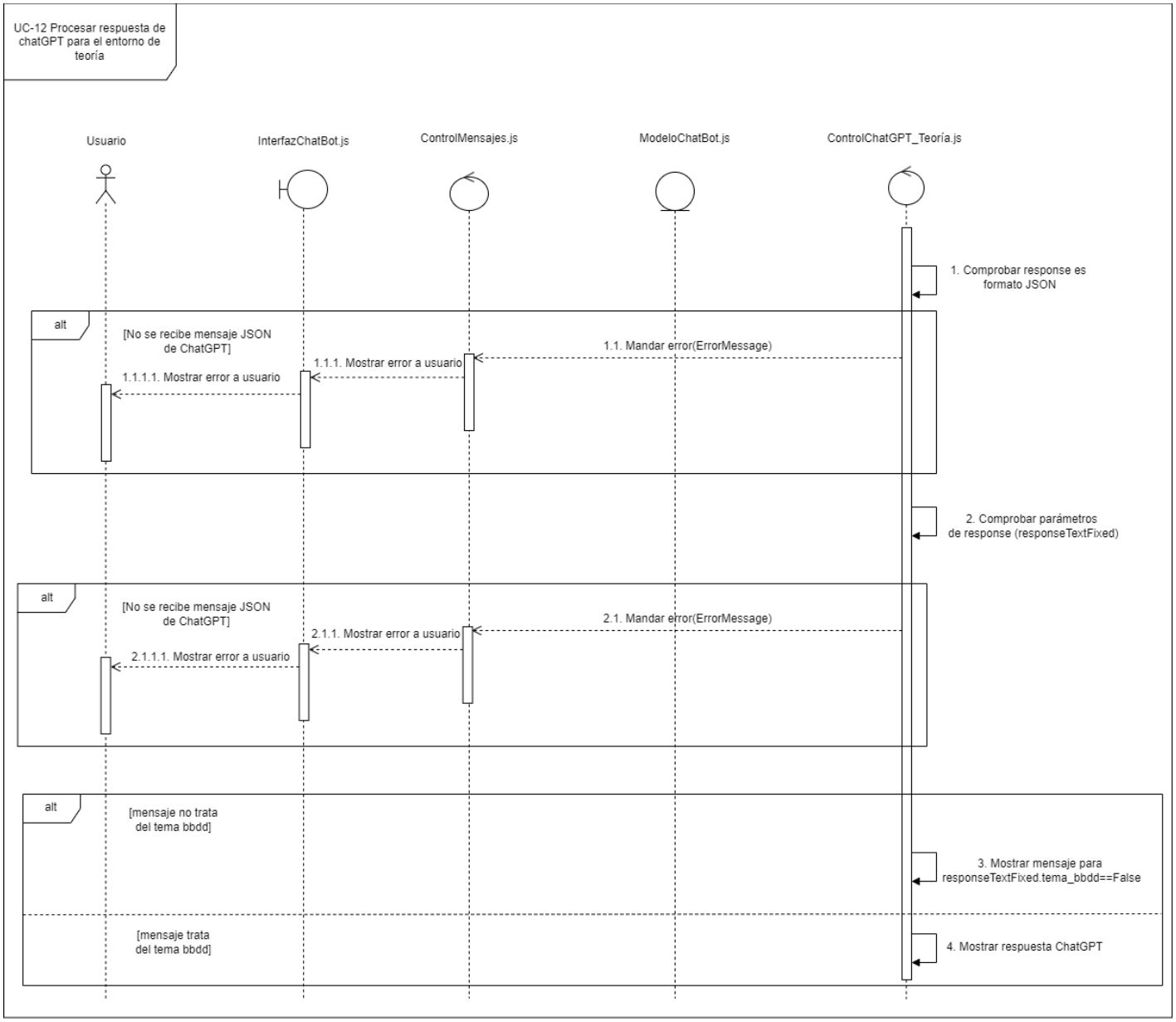

Figura 24. UC-Diseño-12 Procesar respuesta de ChatGPT para el entorno de teoría

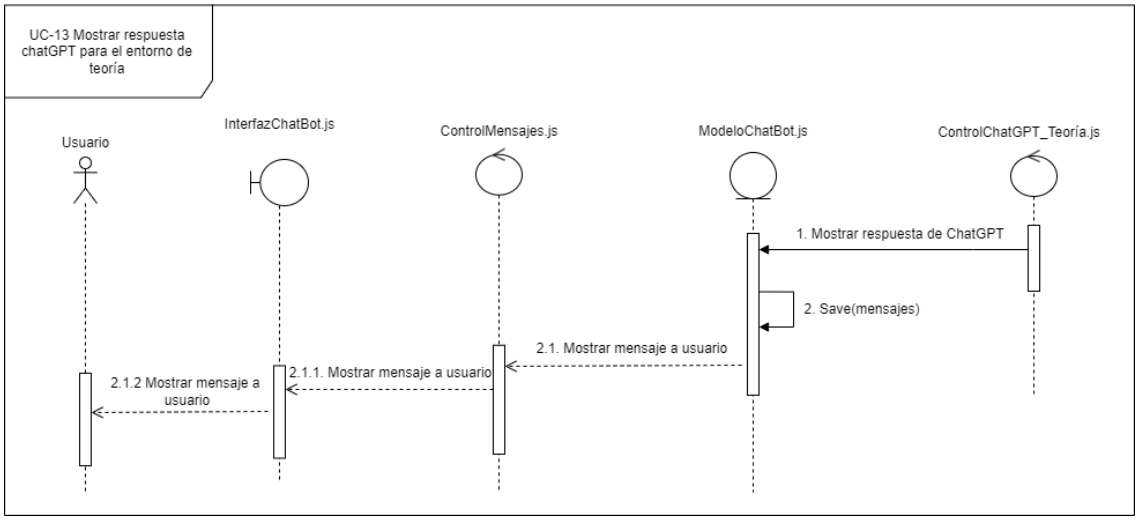

Figura 25. UC-Diseño-13 Mostrar respuesta ChatGPT para el entorno de teoría

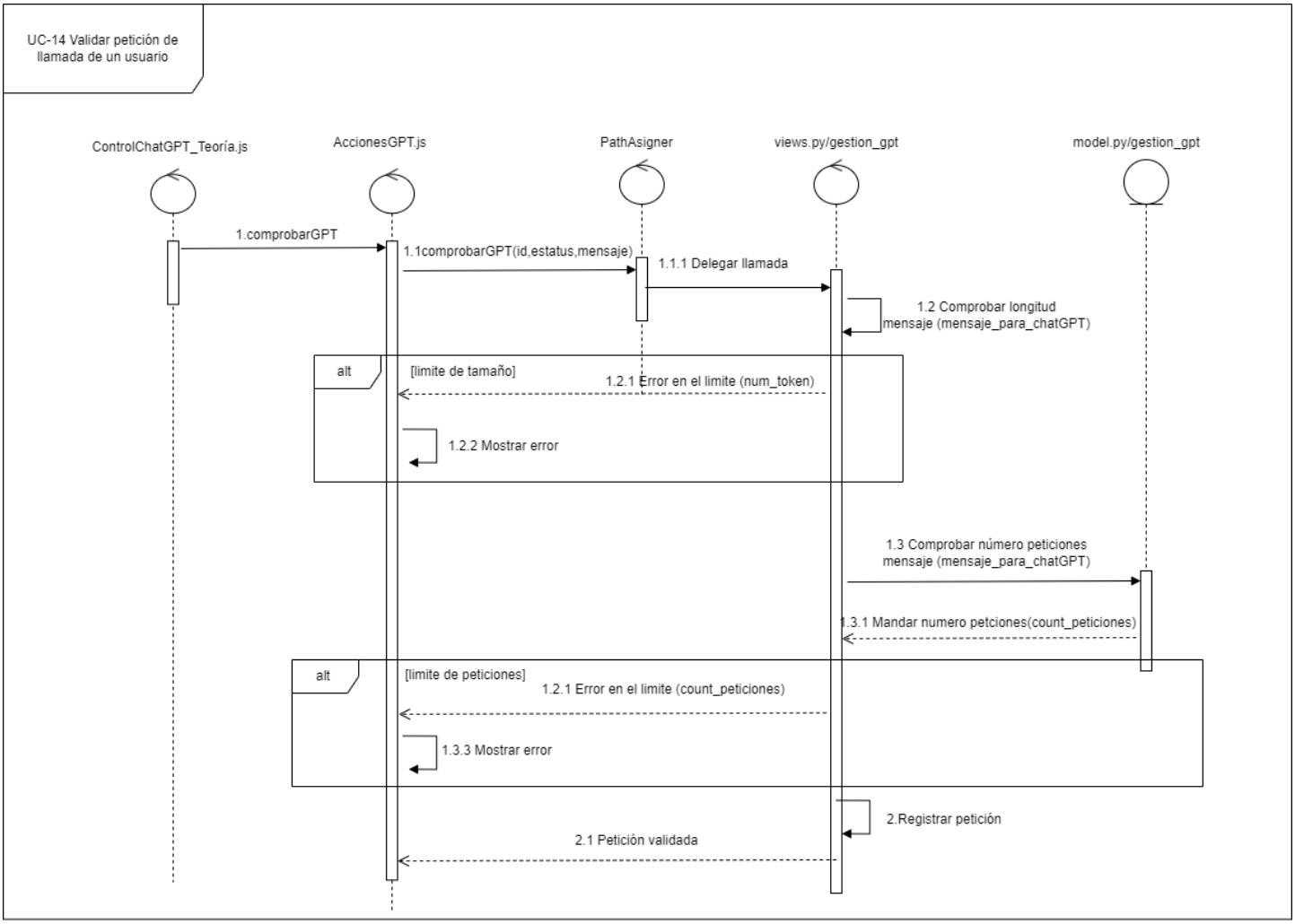

Figura 26. UC-Diseño-14 Validar petición de llamada de un usuario

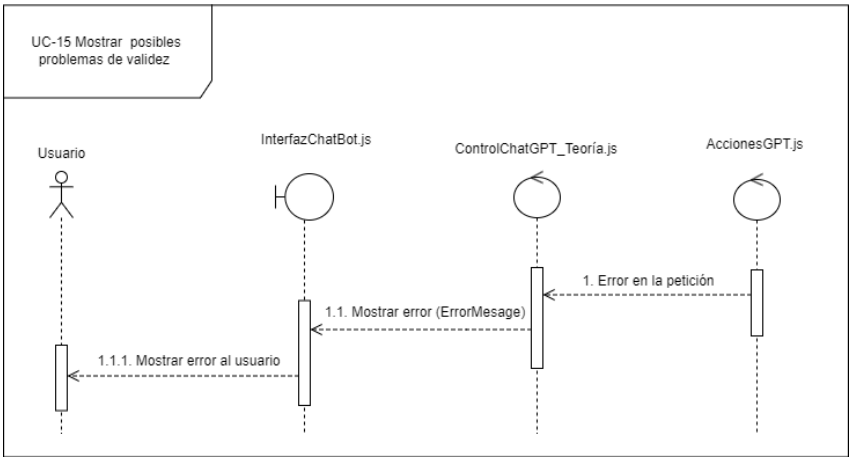

Figura 27. UC-Diseño-15 Mostrar posibles problemas de validez

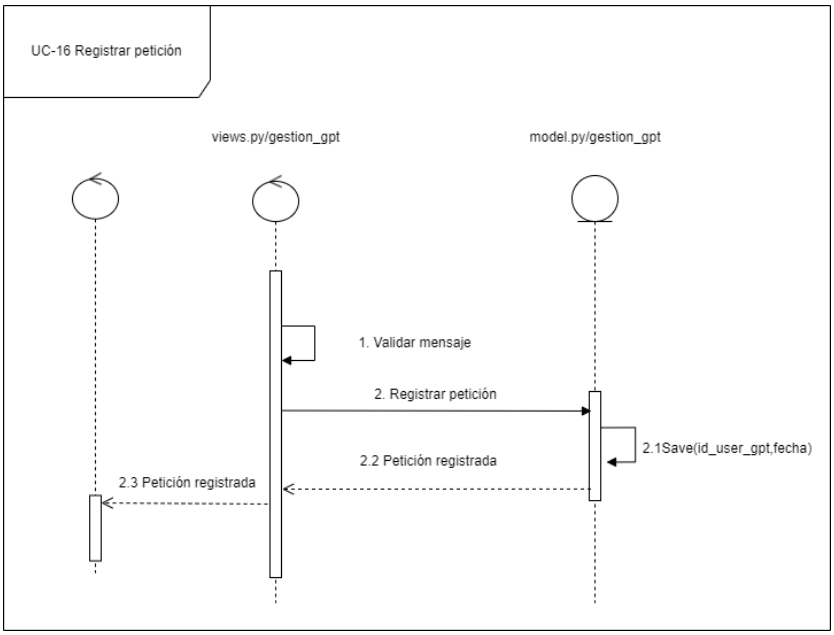

Figura 28. UC-Diseño-16 Registrar petición

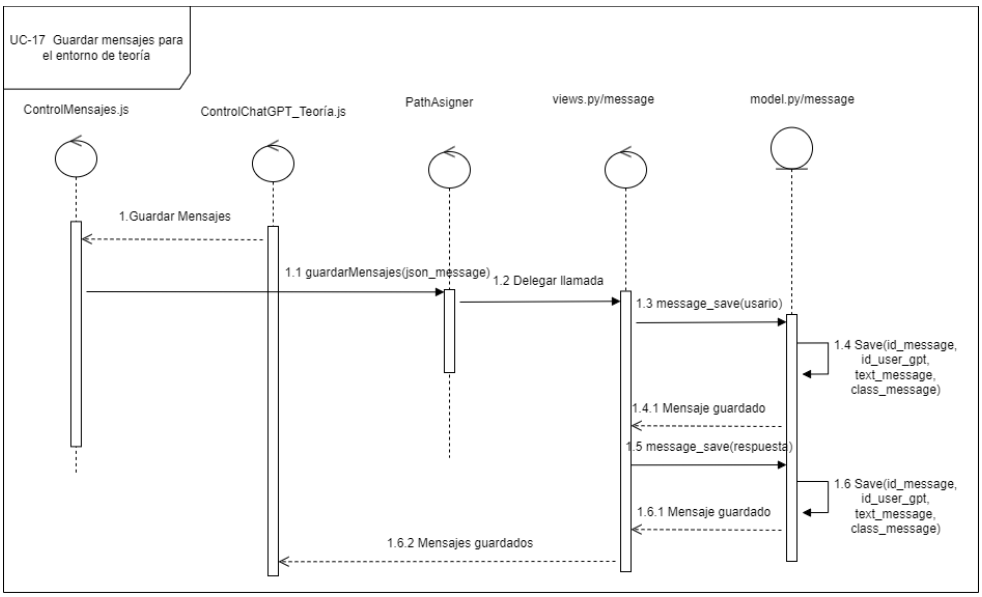

Figura 29. UC-Diseño-17 Guardar mensajes para el entorno de teoría

2.4.2.2. Gestión del chatbot desde práctica

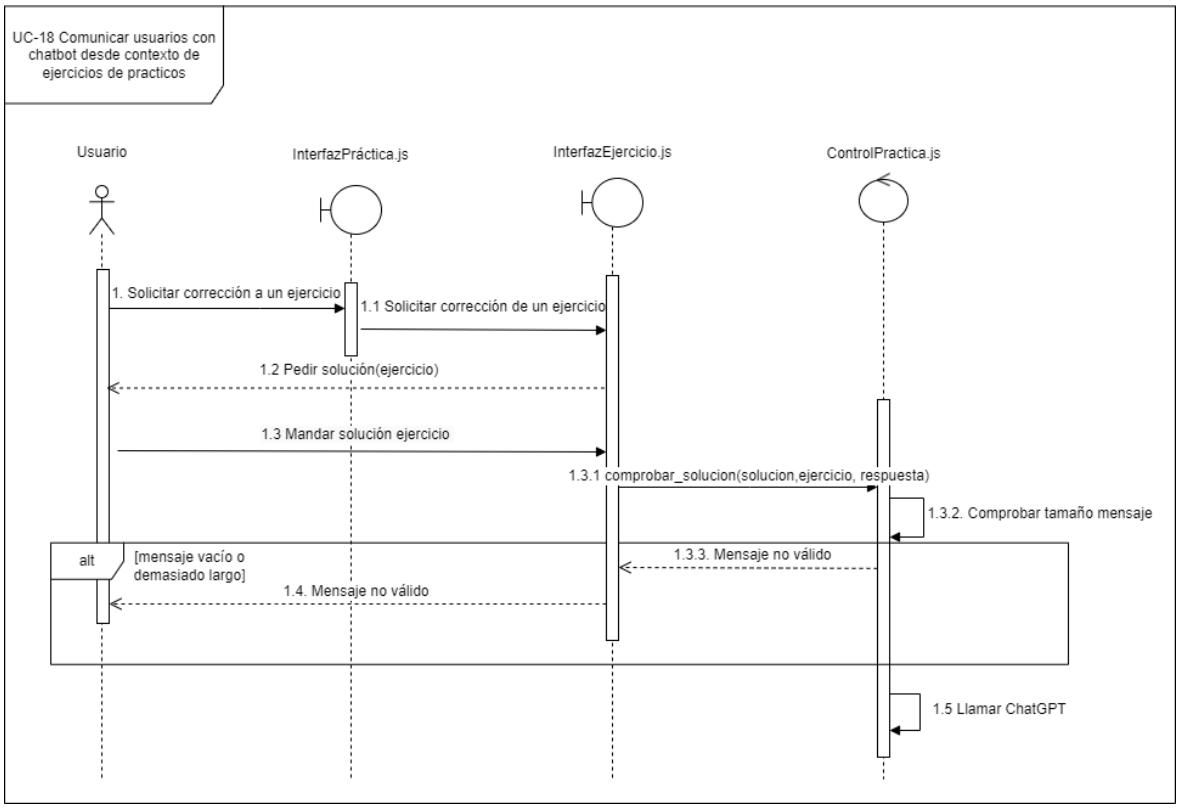

Figura 30. UC-Diseño-18 Comunicar usuarios con chatbot desde contexto de ejercicios prácticos

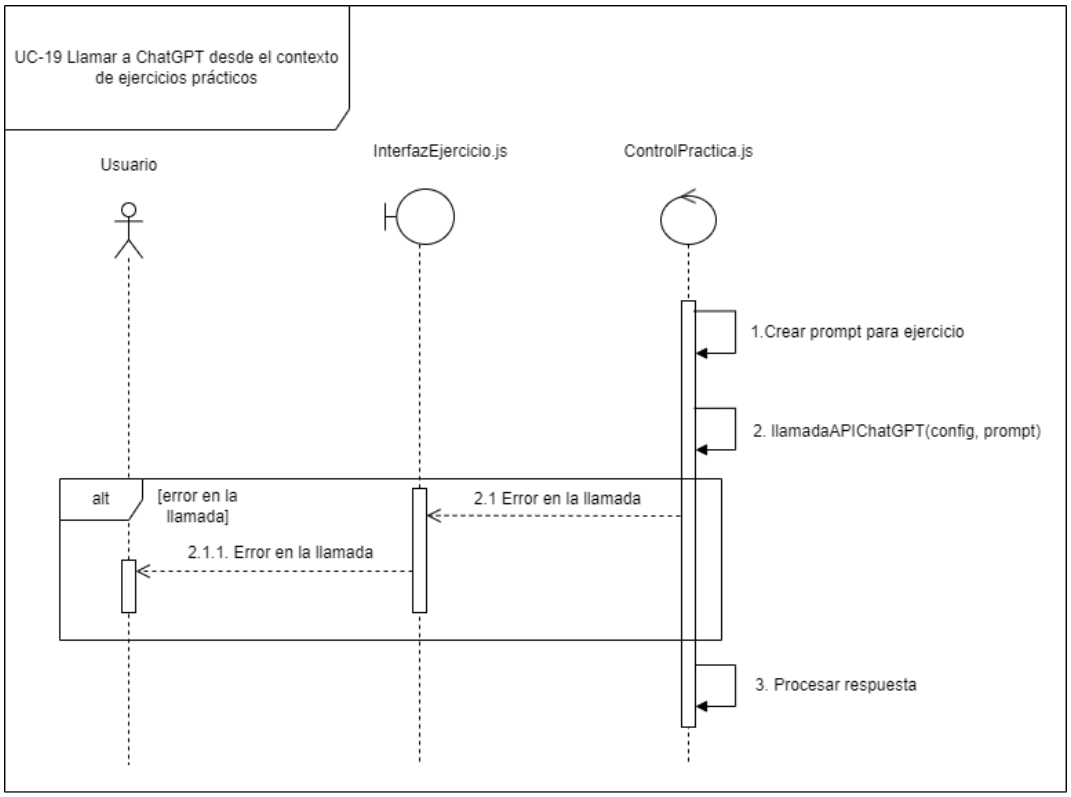

Figura 31. UC-19 Llamar a ChatGPT desde el contexto de ejercicios prácticos

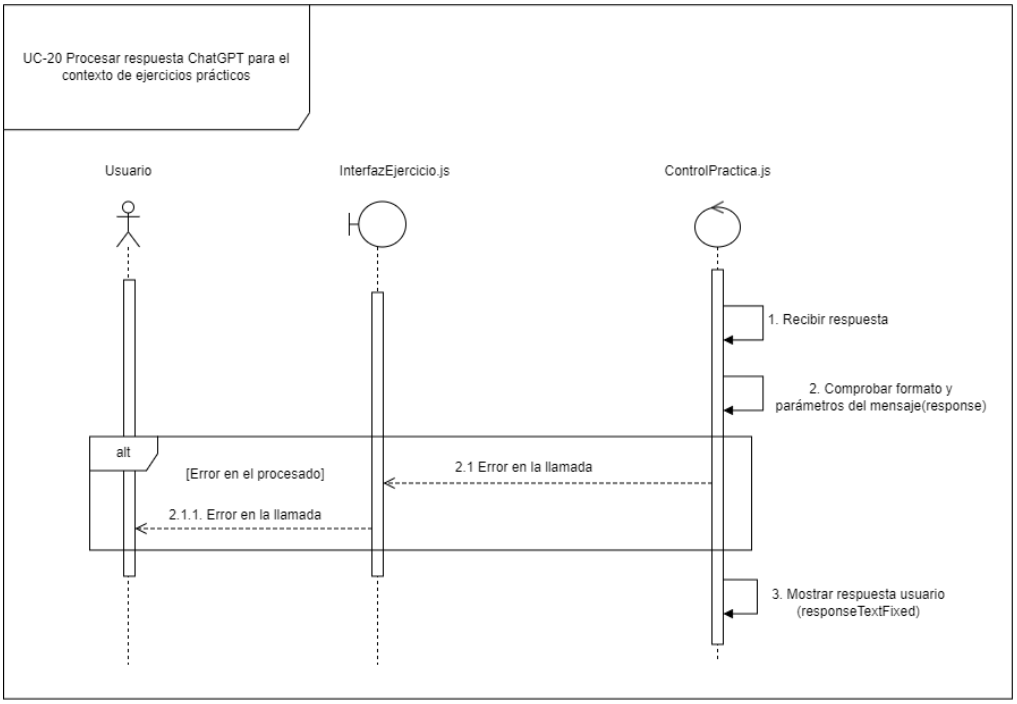

Figura 32. UC-20 Procesar respuesta ChatGPT para el contexto de ejercicios prácticos

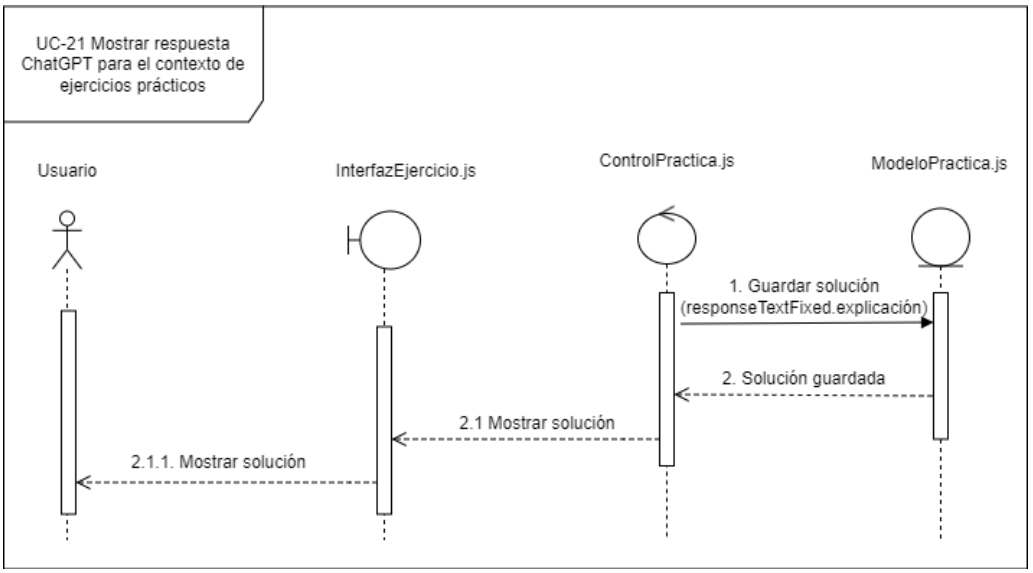

Figura 33. UC-Diseño-21 Mostrar respuesta ChatGPT para el contexto de ejercicios prácticos

2.4.2. Gestión de estadísticas

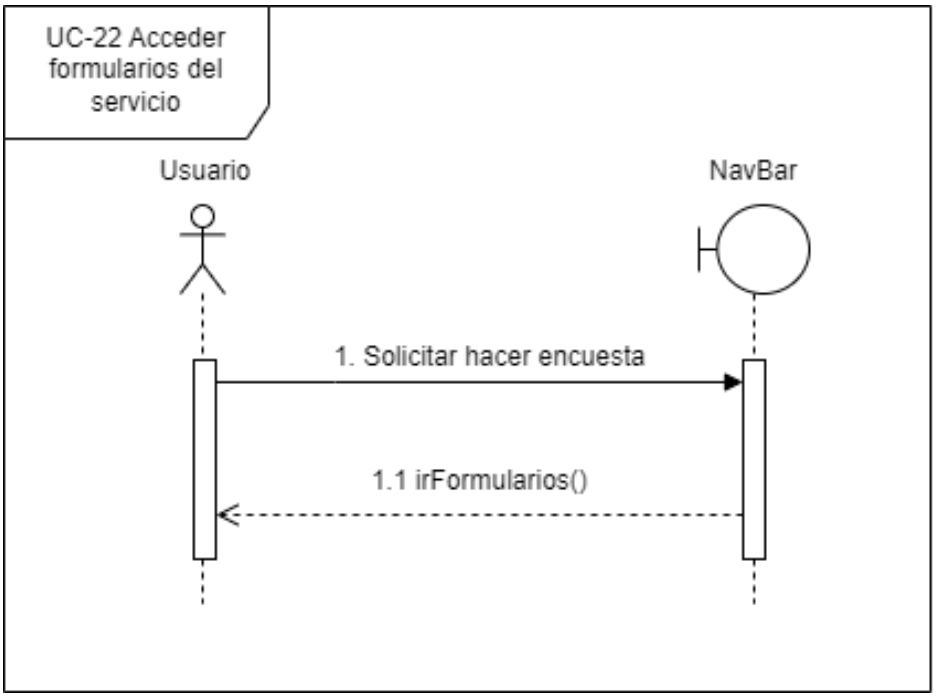

Figura 34. UC-Diseño-22 Acceder formularios del servicio

2.4.2. Gestión de usuarios

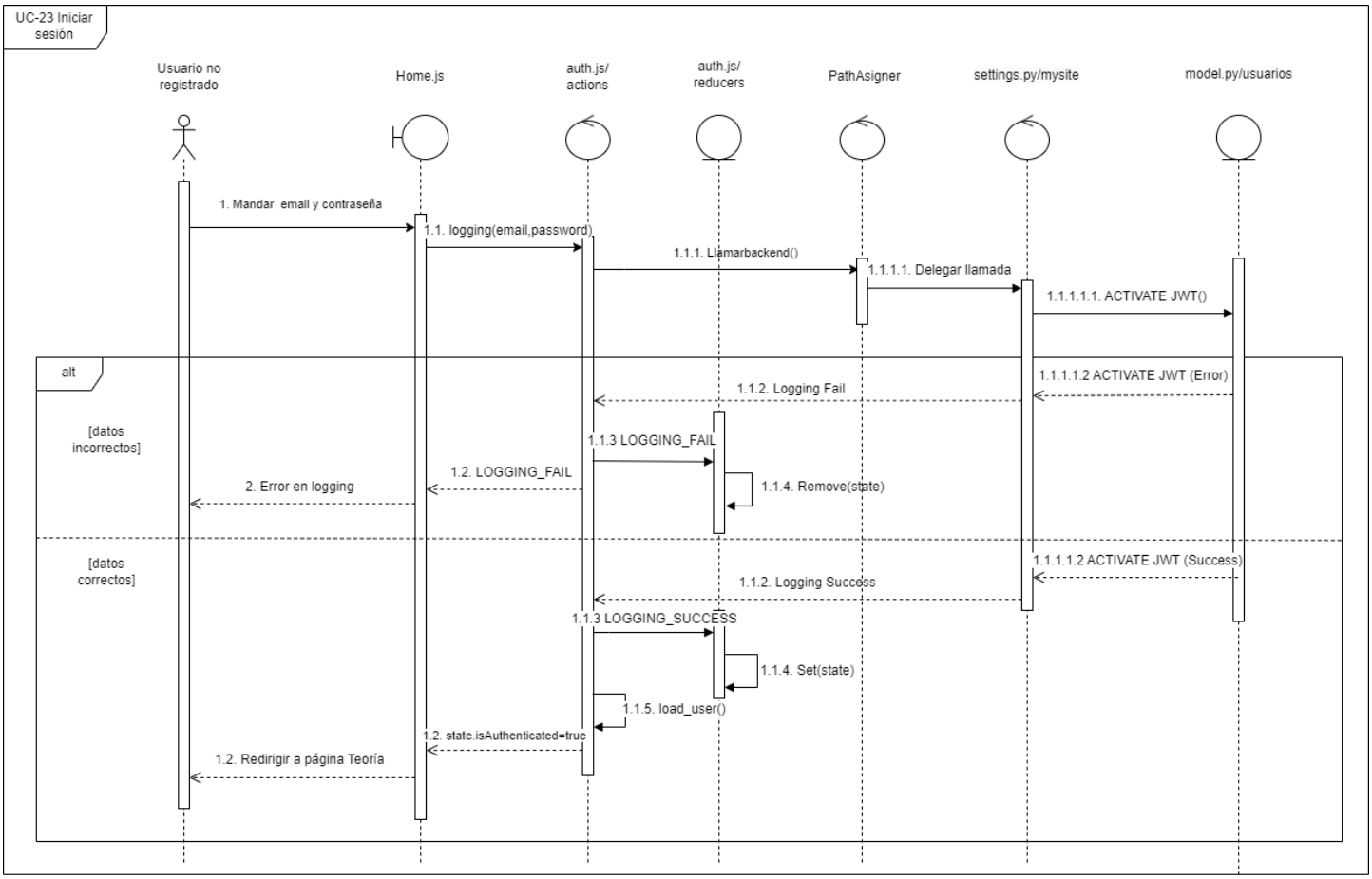

Figura 35. UC-Diseño-23 Iniciar sesión

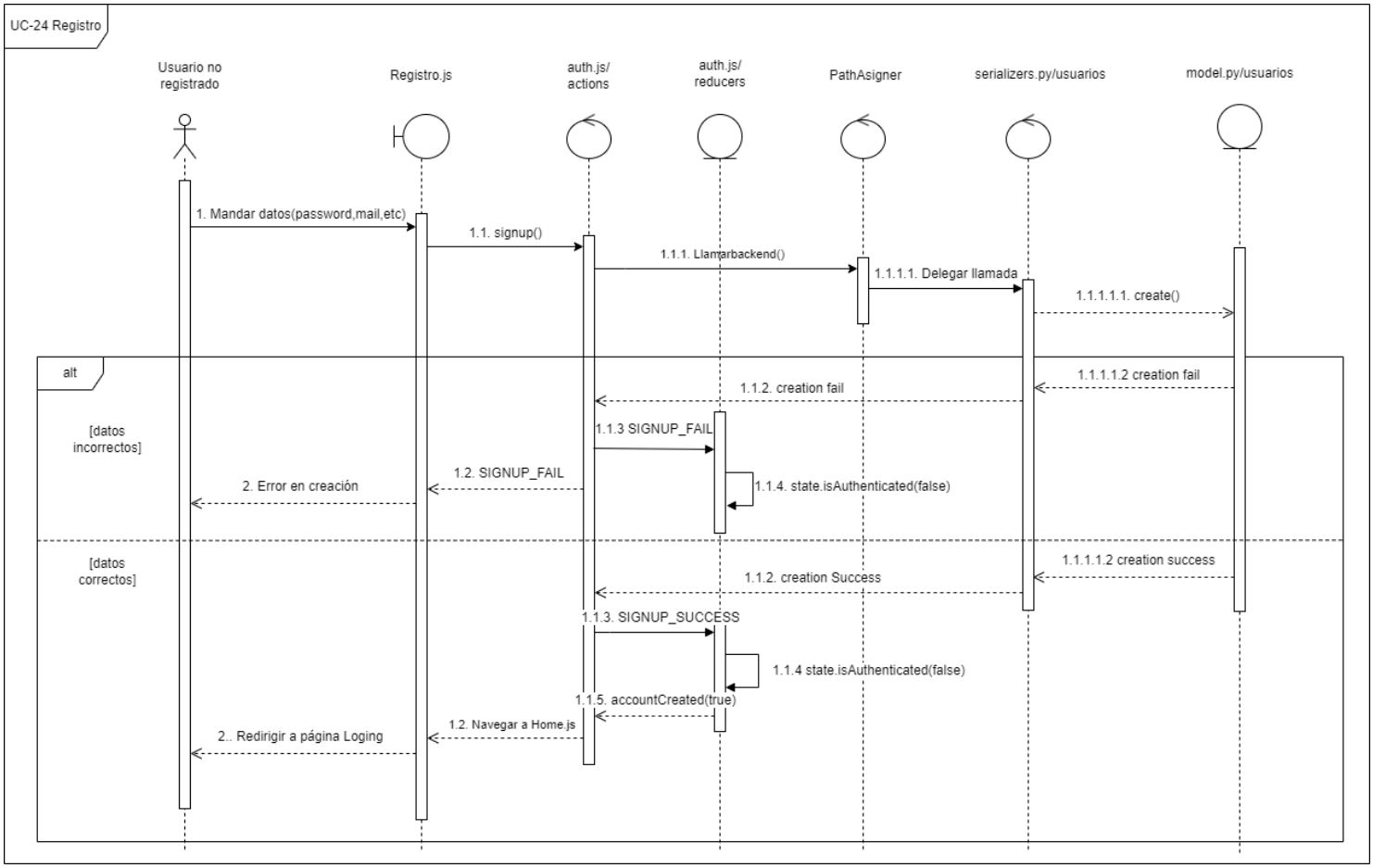

Figura 36. UC-Diseño-24 Registro

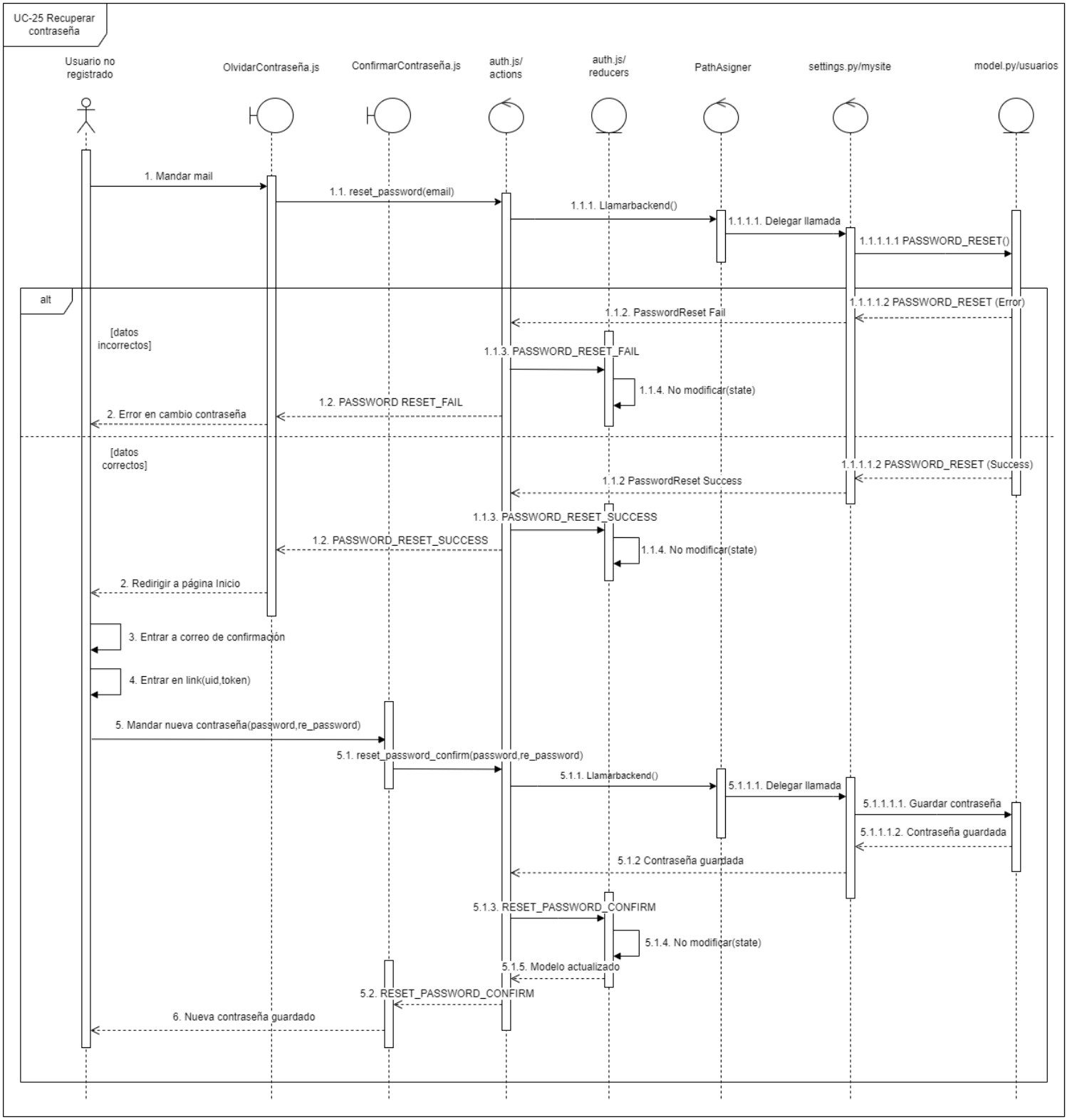

Figura 37. UC-Diseño-25 Recuperar contraseña

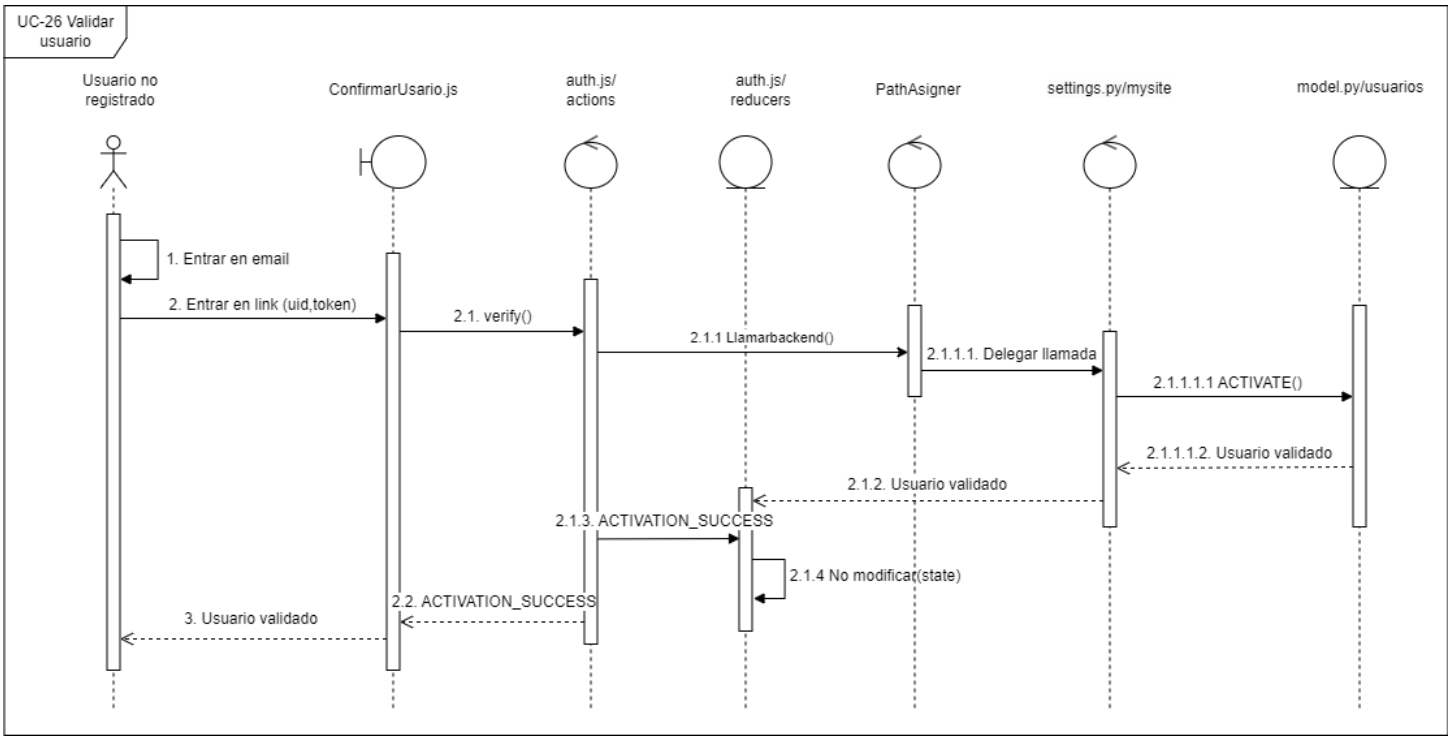

Figura 38. UC-Diseño-26 Validar usuario

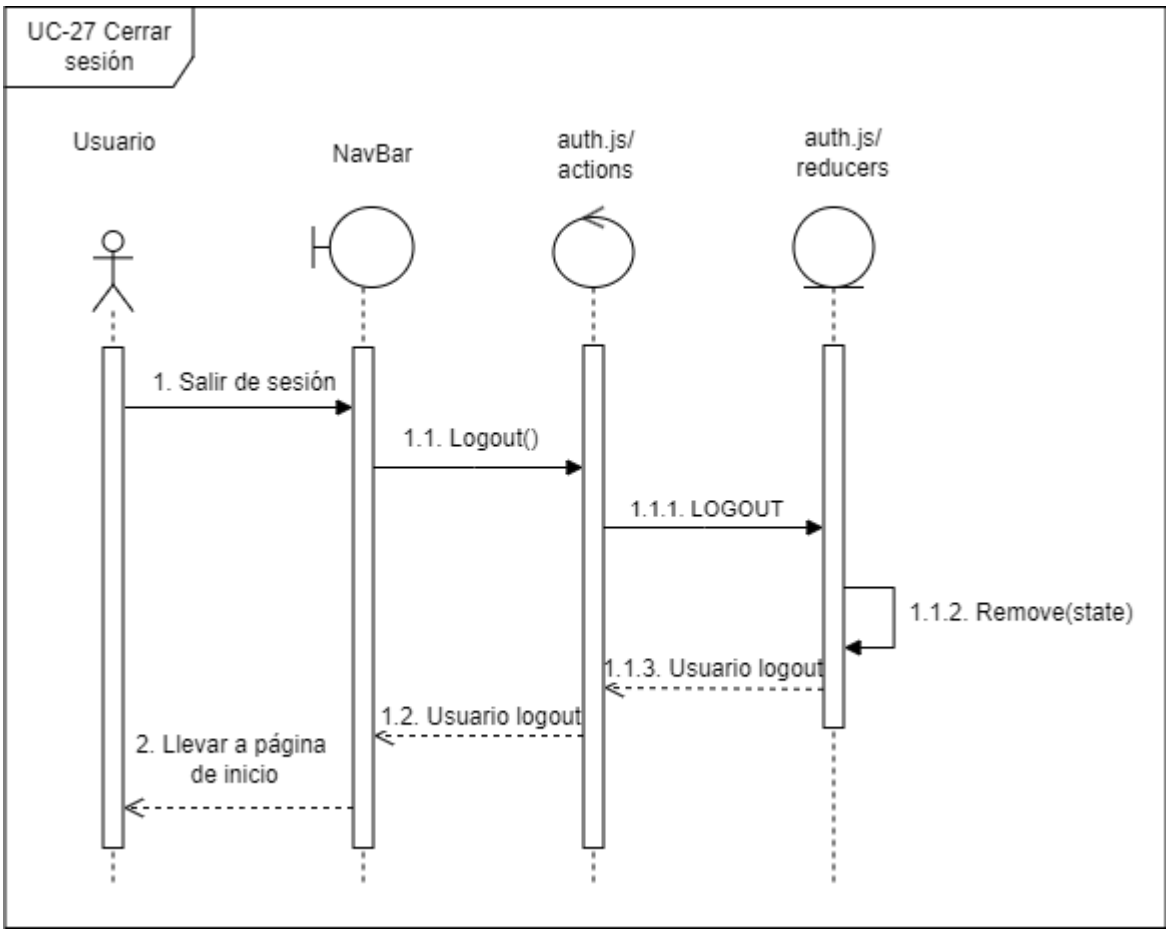

Figura 39. UC-Diseño-27 Salir sesión

3. MODELO DE DESPLIEGUE

Ya que en nuestro sistema web este compuesto por diferentes componentes Abstract, vamos a mostrarlos de una manera gráfica con el modelo de despliegue.

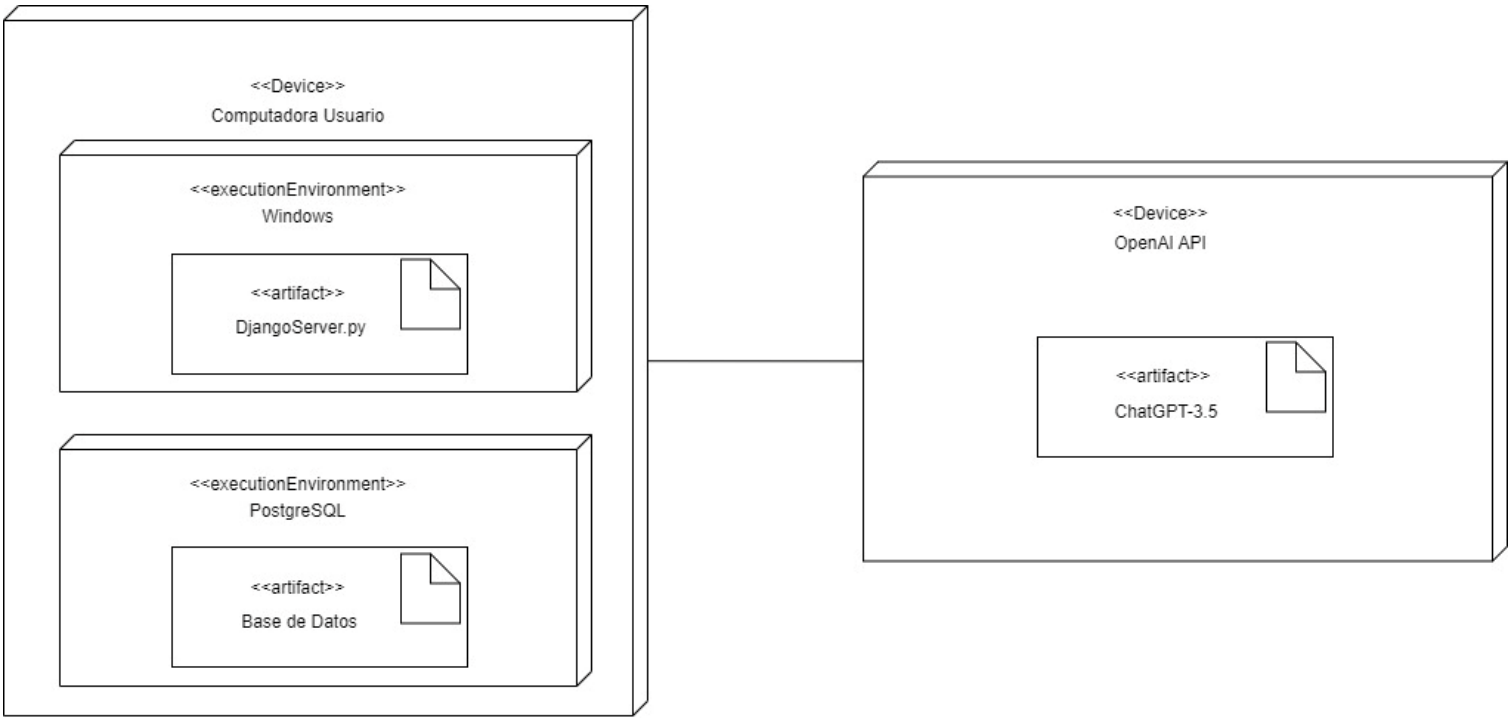

Figura 40. Modelo de despliegue

Nuestro sistema está compuesto solo por un computador de un Usuario, el cual contiene tanto nuestro servidor de Django y nuestra Base de Datos de PostgreSQL. Dicho computador tiene una relación con el servicio que ofrece OpenAI para el uso de la API de ChatGPT.

Este proyecto web solo se ha planteado de forma local, por tanto, si nuestro sistema web evoluciona y es exportado fuera de nuestro entorno local, debemos evolucionar también este modelo de despliegue.

4. DISEÑO DE LA INTERFAZ

En este último apartado se muestran las distintas interfaces que proporciona nuestra web. Debemos plantear esta lista de llamadas como las distintas funciones que nuestro Servicio de Django ofrece al Frontend de React. Primero se listará de forma gráfica todas estas llamadas, para posteriormente detallarlas con sus características y los elementos que las componen.

3.1. Diagrama de componentes

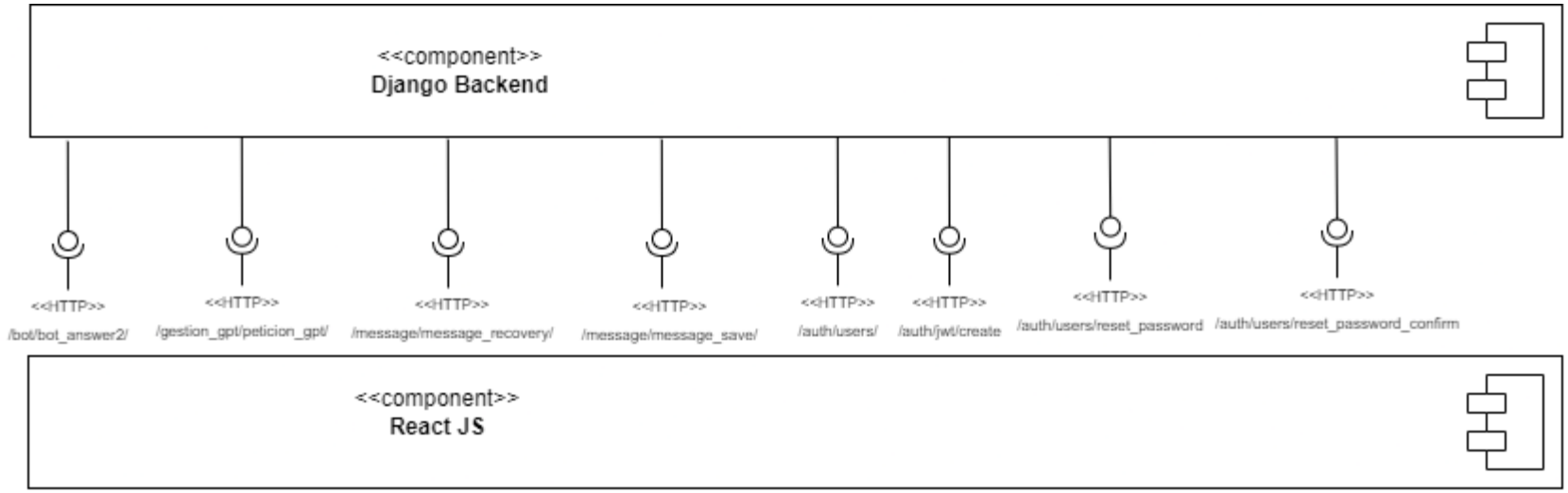

Figura 41. Diagrama de componentes

3.2. Definición de las interfaces

Tabla 1. Endpoint - /bot/bot_answer2

Endpoint	**/bot/bot_answe2**		
Módulo	Django		
Descripción	Realiza la obtención de una respuesta de la red neuronal para un mensaje mandado por un usuario.		
Operación	POST		
Ubicación de los parámetros	\DJANGO_BACKEND\mysite\bot_app_api \DJANGO_BACKEND\mysite\bot_arquitecture		
Formato de marshalling	JSON		
Parámetros	**Nombre**	**Tipo**	**Descripción**
	message	String	Mensaje mandado por el usuario por el chatbot
Retorno	**Nombre**	**Tipo**	**Descripción**
	clase	String	Predicción de la clase que se ha determinado para el mensaje
	message	String	Respuesta predefinida para la Clase que ha sido elegida.

Tabla 2. Endpoint - /gestion_gpt/peticion_gpt

Endpoint	**/gestion_gpt/peticion_gpt**		
Módulo	Django		
Descripción	Realiza las funciones de validación y guardado de una petición para la API de ChatGPT.		
Operación	POST		
Ubicación de los parámetros	\DJANGO_BACKEND\mysite\gestion_gpt		
Formato de marshalling	JSON		
Parámetros	**Nombre**	**Tipo**	**Descripción**
	message	String	Mensaje mandado por el usuario para preguntar a ChatGPT
	id	Int	Identificador de usuario que mando el mensaje
	estatus	String	Estatus del usuario que mandó el mensaje.
Retorno	**Nombre**	**Tipo**	**Descripción**
	message	String	Se devuelve o el mensaje que se ha enviado (petición validada y registrada) o el mensaje “Error”, para indicar un fallo dentro de la validación.
	data	String/Int	Se devuelve o el tipo de error en el caso de producirse un error en la validación. Si la validación es correcta, se

			manda un entero con el número de tokens del mensaje.

Tabla 3. Endpoint - /message/message_recovery

Endpoint	**/message/message_recovery**		
Módulo	Django		
Descripción	Realiza la recuperación de los mensajes guardados de un usuario. Se devolverá un JSON con los distintos mensajes entre un usuario y ChatGPT.		
Operación	POST		
Ubicación de los parámetros	\DJANGO_BACKEND\message\message_recovery		
Formato de marshalling	JSON		
Parámetros	**Nombre**	**Tipo**	**Descripción**
	id	Int	Identificador del usuario que quiere recuperar los mensajes.
Retorno	**Nombre**	**Tipo**	**Descripción**
	id_message	Int	Identificador del mensaje
	text_message	String	Contenido de un mensaje
	class_message	String	Tipo de mensaje que se devuelve, puede ser de un usuario o la respuesta de la API de ChatGPT.

Tabla 4. Endpoint - /message/message_save

Endpoint	**/message/message_save/**		
Módulo	Django		
Descripción	Realiza el guardado de un mensaje. Se guarda tanto la pregunta de un usuario como la respuesta obtenida.		
Operación	POST		
Ubicación de los parámetros	\DJANGO_BACKEND\message\message_save\		
Formato de marshalling	JSON		
Parámetros	**Nombre**	**Tipo**	**Descripción**
	id	Int	Identificador del usuario que quiere guardar un mensaje
	text	String	Contenido de un mensaje.
	class_name	String	Tipo de mensaje que se guarda. Puede ser tanto de usuario como la respuesta de ChatGPT:
Retorno	**Nombre**	**Tipo**	**Descripción**
	message	String	Validador si un mensaje se ha guardado.
	data	String	Se devuelve o el mensaje guardado o

			el error que se ha producido.

Los siguientes dos endpoints, a pesar de compartir el mismo nombre, realizan funciones diferentes dependiendo de si el usuario se está registrando o si está haciendo logging. Es el propio Django el que se encarga de hacer esto en su propia arquitectura.

Tabla 5. Endpoint - /auth/users (Registro)

Endpoint	**/auth/users/**		
Módulo	Django		
Descripción	Realiza la validación y el guardado de un registro de un nuevo usuario.		
Operación	POST		
Ubicación de los parámetros	-		
Formato de marshalling	JSON		
Parámetros	**Nombre**	**Tipo**	**Descripción**
	username	String	Nombre del usuario
	email	String	Correo electrónico del usuario.
	estatus	String	Tipo de usuario que se registra en la web. Dependiendo del tipo de estatus los limites en la llamada a la API cambian.
	password	String	Contraseña del usuario.
	re_password	String	Confirmación de las contraseñas.
	description	String	Descripción opcional de un usuario.
Retorno	**Nombre**	**Tipo**	**Descripción**
	username	String	Confirmación del nombre del usuario registrado.
	estatus	String	Confirmación del estatus.
	descriptionr	String	Confirmación de la descripción.
	email	String	Confirmación del email.
	id	Int	Identificación del usuario creado.

Tabla 6. Endpoint - /auth/users (Logging)

Endpoint	**/auth/users/**		
Módulo	Django		
Descripción	Realiza la validación de un usuario que se ha registrado. Después de la validación podrá entrar a la Web cuando se loguee.		
Operación	POST		
Ubicación de los parámetros	-		
Formato de marshalling	JSON		
Parámetros	**Nombre**	**Tipo**	**Descripción**
	uid	String	Uid de validación de la validación
	tokenl	String	Token mandado dentro de la validación de un usuario.
Retorno	**Nombre**	**Tipo**	**Descripción**
	username	String	Confirmación del nombre del usuario registrado para indicar que el usuario fue validado.

Tabla 7. Endpoint - /auth/users/reset_password

Endpoint	**/auth/users/reset_password**		
Módulo	Django		
Descripción	Realiza la petición de olvidar la contraseña.		
Operación	POST		
Ubicación de los parámetros	-		
Formato de marshalling	JSON		
Parámetros	**Nombre**	**Tipo**	**Descripción**
	email	String	Correo electrónico del usuario que quiere recuperar su contraseña.
Retorno	**Nombre**	**Tipo**	**Descripción**
	-	-	-

Tabla 8. Endpoint - /auth/users/reset_password_confirm

Endpoint	**/auth/users/reset_password_confirm**		
Módulo	Django		
Descripción	Realiza el cambio de una contraseña y su guardado dentro de la base de datos.		
Operación	POST		
Ubicación de los parámetros	-		
Formato de marshalling	JSON		
Parámetros	**Nombre**	**Tipo**	**Descripción**
	uid	String	Uid de validación de la petición para guardar la nueva contraseña
	token	String	Token de validación de la petición para guardar la nueva contraseña.
	new_password	String	Nueva contraseña.
	re_new_password	String	Repetir nueva contraseña por seguridad
Retorno	**Nombre**	**Tipo**	**Descripción**
	-	-	-

4. BIBLIOGRAFÍA

Aliyya, M. (24 de 5 de 2021). *Entendimiento de Backend y Frontend con ReactJS y Django*. Obtenido de https://mahdiaaliyya.medium.com/software-architecture-bb44325bf0cf

GeeksForGeeks. (s.f.). *Entendimiento de la arquitectura cliente-servidor*. Obtenido de https://www.geeksforgeeks.org/client-server-model/