



TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN



PharmaDock AI: Desarrollo de un asistente conversacional para el docking molecular entre proteínas y fármacos.

Estudiante: Carolina Rey López
Dirección: Carlos Fernández Lozano
Ángel Piñeiro Guillén

A Coruña, septiembre de 2025.

A mi familia y a mis amigos

Agradecimientos

En primer lugar, quiero dar las gracias a mis tutores por su ayuda y apoyo durante el desarrollo del proyecto.

Gracias a mi pareja y a mis amigos, tanto de dentro como de fuera de la carrera, por acompañarme en este camino y enseñarme muchas cosas en estos intensos 4 años.

Gracias a mis padres, por su apoyo incondicional y por estar ahí tanto en mis buenos momentos como en los malos, celebrando mis logros y subiéndome el ánimo en las épocas más duras.

Y por último, quiero darles las gracias a mis abuelos, que sé que estarían orgullosos de lo que he conseguido.

Resumen

El docking molecular es una de las técnicas más usadas en bioinformática, en específico para el descubrimiento de nuevos fármacos para el tratamiento de enfermedades como el cáncer. Sin embargo, la mayoría del software existente para llevar a cabo estos experimentos suele necesitar una formación específica o resulta demasiado complejo para investigadores que no están familiarizados con estas herramientas. Además, muchos de estos programas son de pago, lo que dificulta su acceso a grupos de investigación con recursos limitados.

Con el objetivo de superar estas barreras nace PharmaDock AI, una aplicación web gratuita que permite realizar experimentos de docking molecular de forma sencilla, usando consultas en lenguaje natural. Esto permite que cualquier investigador pueda llevar a cabo los experimentos que quiera de forma sencilla sin necesidad de aprender ningún software específico para avanzar con sus investigaciones.

Abstract

The molecular docking is one of the most widely used techniques in bioinformatics, particularly for the discovery of new drugs to treat diseases such as cancer. However, most existing software for carrying out these experiments usually requires a specific formation or is too complex for researchers who aren't familiar with these tools. Besides, much of this software is paid, which limits their access to research groups with limited resources.

To overcome these barriers, PharmaDock AI was created: a free web application that allows users to realize docking molecular experiments easily using natural language queries. This enables any researchers to conduct the experiments they want without the need to learn any specific software for progress with their researches.

Palabras clave:

- Docking molecular
- Bioinformática
- Procesamiento de lenguaje natural
- Desarrollo web
- Python
- Django

Keywords:

- Molecular Docking
- Bioinformatics
- Natural language processing
- Web development
- Python
- Django

Índice general

1	Introducción	1
1.1	Motivación	1
1.2	Objetivos	2
1.3	Estructura de la memoria	2
2	Estado del arte	4
2.1	Glide	4
2.2	AutoDock Vina	5
2.3	Conclusiones	6
3	Fundamentos tecnológicos	8
3.1	Lenguajes de programación	8
3.1.1	Python	8
3.1.2	HTML	8
3.1.3	CSS	8
3.1.4	JavaScript	8
3.2	Frameworks y librerías	9
3.2.1	OpenAI API	9
3.2.2	RDKit	9
3.2.3	Django	9
3.2.4	Crispy Forms	9
3.3	Software específico	10
3.3.1	AutoDock Vina	10
3.4	Entorno de programación y control de versiones	10
3.4.1	Git y GitHub	10
3.4.2	VS Code	10
3.4.3	Sistema Operativo	10

4	Metodología	12
4.1	Metodología usada	12
4.2	Planificación inicial	12
4.3	Seguimiento de la planificación	14
4.4	Gestión de riesgos	15
4.5	Costes	16
4.5.1	Costes humanos	16
4.5.2	Costes materiales	16
4.5.3	Costes totales	17
5	Diseño y desarrollo	18
5.1	Casos de uso	18
5.1.1	Definición de los actores	18
5.1.2	Casos de uso del sistema	19
5.2	Diagramas de componentes	24
5.3	Diagramas de clases	27
5.4	Diagramas de estados	28
5.5	Patrones de diseño	29
5.6	Incrementos	30
5.6.1	Incremento 1	30
5.6.2	Incremento 2	31
5.6.3	Incremento 3	32
5.6.4	Incremento 4	34
5.6.5	Incremento 5	37
5.6.6	Incremento 6	41
5.6.7	Incremento 7	44
5.6.8	Incremento 8	46
5.6.9	Incremento 9	49
5.6.10	Incremento 10	53
6	Pruebas	55
7	Conclusiones	56
7.1	Trabajo futuro	56
7.2	Lecciones aprendidas	57
A	Ejecución de la aplicación	59
A.1	Requisitos previos	59
A.2	Instalación	59

A.3 Ejecución de la aplicación	60
Lista de acrónimos	61
Glosario	62
Bibliografía	64

Índice de figuras

2.1	Resultado de un docking molecular en la interfaz de Glide	4
2.2	Resultado de un docking molecular usando AutoDock Vina junto a una herramienta externa para el uso de una interfaz gráfica	6
4.1	Diagrama de Gantt de la planificación inicial	14
4.2	Diagrama de Gantt del seguimiento de la planificación	14
5.1	Diagrama de casos de uso	18
5.2	Diagrama de componentes del backend	25
5.3	Diagrama de componentes del frontend	26
5.4	Diagrama de clases de la estructura de las aplicaciones	27
5.5	Diagrama de clases de la estructura interna del chatbot	28
5.6	Diagrama de estados sobre los estados por los que va pasando el chatbot	28
5.7	Diagrama de estados de las diferentes páginas que componen la web	29
5.8	Diseño de la interfaz del chat en pantallas grandes	33
5.9	Diseño de la interfaz del chat en pantallas pequeñas	33
5.10	Ejemplo del resumen que proporcionará el chatbot ante una petición del usuario	35
5.11	Ejemplo de la molécula resultado del docking entre el Abemaciclib y el gen ABL1	43
5.12	Contenedor con el botón de descarga del fichero de los resultados	44
5.13	Página de inicio de PharmaDock AI	46
5.14	Página de inicio de sesión de PharmaDock AI	47
5.15	Página de registro de PharmaDock AI	48
5.16	Página del chatbot con la cabecera	49
5.17	Error que aparece al intentar registrar una cuenta con un correo ya existente	52
5.18	Página de verificación de PharmaDock AI	52
5.19	Mensaje flotante al verificar la cuenta con éxito	53

Índice de tablas

4.1	Riesgos técnicos del proyecto	15
4.2	Riesgos de recursos humanos del proyecto	16
4.3	Costes totales del proyecto	17
5.1	Definición del ACT-01	19
5.2	Definición del ACT-02	19
5.3	Definición del ACT-03	19
5.4	Caso de uso CU-01: Registrarse	20
5.5	Caso de uso CU-02: Iniciar sesión	21
5.6	Caso de uso CU-03: Verificar cuenta	22
5.7	Caso de uso CU-04: Interactuar con el chatbot	23
5.8	Caso de uso CU-05: Gestionar usuarios	24
5.9	Resumen de conformaciones del experimento de docking molecular	40

Introducción

1.1 Motivación

El docking molecular es un tipo de modelización molecular que facilita la orientación del **Binding** entre un **Receptor** y un **Ligando** cuando interactúan entre ellos con el objetivo de formar un compuesto estable. El objetivo principal de esta modelización es obtener una conformación acoplada optimizada con el fin de lograr una disminución de energía libre de todo el sistema [1]. Este proceso es ampliamente usado alrededor del mundo para tanto el hallazgo de nuevos compuestos con efectos terapéuticos como en el diseño de fármacos, siendo una alternativa a los procesos masivos de ensayo en laboratorios [2].

A pesar de ser muy usado, a lo largo de los años se ha visto que presenta ciertas limitaciones, como lo puede ser la necesidad de experiencia técnica o la baja precisión en **Scoring**. Para solucionarlas, se han propuesto diferentes métodos basados en inteligencia artificial. En *Molecular Docking: Classical and AI-Based Approaches*[3] se nos hace un repaso de los enfoques clásicos de este proceso y nos habla de estos nuevos métodos, los cuales involucran el **Machine Learning** y las **Convolutional Neural Networks (CNN)**.

Además, también se están empezando a desarrollar herramientas basadas en **Large Language Model (LLM)** para realizar estas modelizaciones. En *ChatMol Copilot: An Agent for Molecular Modeling and Computation Powered by LLMs* [4] se desarrolla un modelo de inteligencia artificial basado en estos modelos de lenguaje, el cual permite diseñar proteínas, generar moléculas e incluso realizar experimentos de docking molecular. En *Large language models open new way of AI-assisted molecule design for chemists* [5] se habla de ChatChemTS, una aplicación basada en LLMs para que los usuarios puedan realizar tareas de predicciones, análisis de generación de moléculas y otras gracias al uso del framework ChemTSv2.

PharmaDock AI se enmarca dentro de las nuevas tendencias en el uso de los **Modelo de lenguaje** para tareas de **Bioinformática** mediante el desarrollo de un chatbot basado en la **Application Programming Interface (API)** de OpenAI para la realización de experimentos de

docking molecular usando consultas en lenguaje natural.

1.2 Objetivos

El objetivo principal del proyecto es desarrollar una plataforma web donde se puedan realizar experimentos de docking molecular a partir de consultas de lenguaje natural. Esto garantiza que se puedan realizar de manera sencilla, facilitando que aquellos investigadores que no están muy familiarizados con otro software más complejo puedan realizar sus investigaciones. Además, el usuario podrá interactuar con el resultado, pudiendo hacer zoom o girarlo y descargar el fichero con los resultados.

Los objetivos concretos son los siguientes:

- **Desarrollar un asistente conversacional capaz de comprender consultas biomédicas en lenguaje natural.**
- **Integrar bases de datos para la recuperación de estructuras moleculares.**
- **Automatizar la preparación de ficheros necesarios para realizar el experimento.**
- **Ejecutar experimentos de docking.**
- **Presentar los resultados en una página web con una representación visual interactiva del resultado.**

1.3 Estructura de la memoria

La memoria se compone de los siguientes capítulos:

- **Introducción.** Se presentará la motivación del proyecto, los objetivos a cumplir en el desarrollo del proyecto y la estructura de la memoria.
- **Estado del arte.** Se hablará de distinto software que ya existe para abordar el problema y se hará un análisis de los riesgos y las oportunidades del proyecto.
- **Fundamentos tecnológicos.** Se hablará de las tecnologías utilizadas para el desarrollo del proyecto.
- **Metodología.** Se hablará de la metodología usada, la planificación diseñada al inicio del desarrollo y su seguimiento. Además, se añadirá un desglose de los costes del desarrollo del proyecto.

- **Diseño y desarrollo.** Se mostrarán los correspondientes diagramas que mostrarán la organización del proyecto, casos de uso del sistema, estados por los que va pasando y se mostrarán los patrones de diseños empleados en el desarrollo. También se hablará más en detalle de lo desarrollado en cada iteración y su proceso.
- **Conclusiones.** Se hablará de todas aquellas funcionalidades que se podrían añadir a la aplicación en un futuro y se hará un repaso por lo aprendido a lo largo del desarrollo del proyecto.

Capítulo 2

Estado del arte

2.1 Glide

Glide [6] es un software desarrollado por la compañía Schrödinger ampliamente utilizado en el ámbito farmacéutico y en la investigación académica para el desarrollo de nuevos fármacos. Se caracteriza por el uso conjunto de algoritmos de búsqueda avanzados y modelos de energía precisos para la realización de experimentos de docking molecular.

Las ventajas de este software son las siguientes:

- **Interfaz gráfica fácil de usar.**
- **Alta precisión en el docking con diversos tipos de receptores.**
- **Restricciones personalizables para el cálculo del docking.**

Podemos ver cómo es la interfaz de Glide en la Figura 2.1.

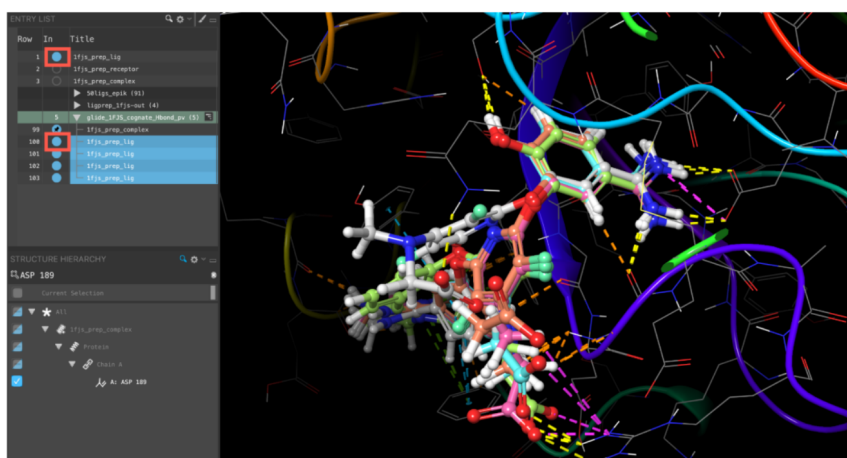


Figura 2.1: Resultado de un docking molecular en la interfaz de Glide

Sin embargo, a pesar de ser un software muy utilizado, es una herramienta propietaria, lo que significa que limita su accesibilidad para muchos usuarios y restringe su integración directa en aplicaciones personalizadas o entornos abiertos.

2.2 AutoDock Vina

AutoDock Vina [7] es un programa de código abierto para la realización de experimentos de docking molecular. Fue desarrollado como una mejora de [AutoDock 4](#).

Algunas de sus características principales son las siguientes:

- **Mejora la precisión media de los resultados que proporcionaba AutoDock 4.**
- **Usa el mismo formato de ficheros tanto para la entrada como para la salida.**
- **Desarrollado para ser usado fácilmente, sin necesidad de que el usuario entienda sus detalles de implementación.**
- **Posibilidad de usar múltiples [Central Processing Unit \(CPU\)](#).**

La principal desventaja de esta herramienta es que no proporciona una interfaz gráfica nativa, por lo que se debe combinar con otras herramientas externas como [PyMol](#) o [AutoDockTools](#) para la preparación y visualización de los resultados.

Podemos ver un ejemplo de una interfaz gráfica usada con este software en la [Figura 2.2](#).

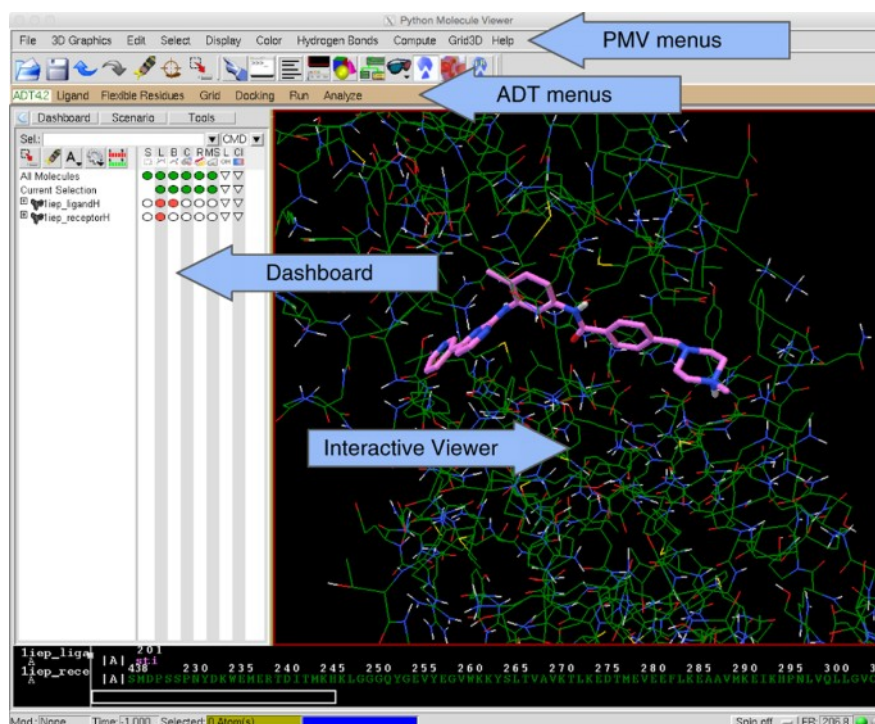


Figura 2.2: Resultado de un docking molecular usando AutoDock Vina junto a una herramienta externa para el uso de una interfaz gráfica

2.3 Conclusiones

Ambas tecnologías son herramientas consolidadas en el ámbito del docking molecular, presentando una muy buena precisión y fiabilidad a la hora de predecir interacciones ligando-receptor.

No obstante, ambas presentan barreras que pueden limitar su uso. Glide requiere de una licencia comercial, limitando su disponibilidad para estudiantes e investigadores con recursos limitados. A su vez, como AutoDock Vina no posee una interfaz gráfica nativa, supone un reto para aquellos usuarios que no tienen experiencia en entornos de línea de comandos o sin conocimientos técnicos en preparación de los ficheros necesarios para la realización de dichos experimentos.

PharmaDock AI pretende mitigar las desventajas de estos dos software, proporcionando una solución accesible, intuitiva y libre de restricciones de uso. Para ello, se han diseñado las siguientes características clave:

- **Proyecto de código abierto.** Cualquiera podrá usarlo sin necesidad de usar ninguna licencia, lo que garantiza su libre uso, modificación y distribución.

- **Interfaz web amigable con el usuario.** La plataforma cuenta con una interfaz web diseñada para ser clara, accesible y fácil de usar, de modo que cualquier usuario pueda familiarizarse rápidamente con su funcionamiento.
- **Implementación de herramientas de código abierto para la realización de los experimentos.** El usuario no tendrá que interactuar directamente con el software, eliminando la necesidad de tener experiencia técnica.
- **Realización de experimentos mediante la interacción con un chatbot.** Los investigadores pueden realizar proyectos de forma más sencilla, ya que no necesitan aprender ninguna herramienta nueva para utilizarla ni aprender a manejar ficheros con las entradas.
- **Visualización de los resultados.** Una vez finalizado el experimento, el chatbot proporciona al usuario una visualización interactiva del resultado, permitiéndole rotar la molécula, hacer zoom y explorar la estructura en detalle de forma sencilla e intuitiva.

Fundamentos tecnológicos

3.1 Lenguajes de programación

3.1.1 Python

Python [8] es un lenguaje de programación interpretado multiparadigma, ya que soporta la orientación a objetos, la programación imperativa, entre otros. Este ha sido el lenguaje escogido para el desarrollo del backend, debido a que es el lenguaje más usado para el desarrollo de aplicaciones de [Inteligencia Artificial \(IA\)](#) gracias a que cuenta con un gran número de librerías para ello.

3.1.2 HTML

[HyperText Markup Language \(HTML\)](#) [9] es el lenguaje de marcado utilizado en la creación de las páginas web. Se ha usado para desarrollar la estructura del frontend.

3.1.3 CSS

[Cascading Style Sheets \(CSS\)](#) [10] es un lenguaje usado para definir el renderizado de los elementos que se muestran en pantalla. Se ha usado para dar estilos a las distintas páginas que componen el frontend.

3.1.4 JavaScript

JavaScript [11] es un lenguaje interpretado, basado en objetos e imperativo. Es el lenguaje de scripting seleccionado para el desarrollo del frontend.

3.2 Frameworks y librerías

3.2.1 OpenAI API

OpenAI API [12] es una librería de pago que proporciona acceso a los modelos de inteligencia artificial desarrollados por la compañía de OpenAI, permitiendo integrar capacidades de [Natural Language Processing \(NLP\)](#) en diversas aplicaciones.

Esta API se utiliza en el desarrollo del chatbot. En casi todas las etapas de la generación del experimento, se realizan dos consultas: en la primera se obtienen los elementos necesarios para realizar el docking (el ligando, el receptor, etc.) y en la segunda, se proporciona un intermediario entre el usuario y el sistema de docking molecular.

En concreto, se ha utilizado el modelo *gpt-4o* debido a su capacidad de generar salidas estructuradas, ideal para generar respuestas con el formato [JavaScript Object Notation \(JSON\)](#) adecuado para obtener los datos necesarios para el docking.

3.2.2 RDKit

RDKit [13] es un framework de código abierto especializado en química computacional y quimioinformática. Posee varias funcionalidades como manipular estructuras moleculares, generación de modelos 3D, calcular propiedades moleculares, entre otras.

Este framework se usa para la generación del fichero de la molécula receptora, basándose en su secuencia [SMILES](#), obtenible a través de una consulta a la base de datos.

3.2.3 Django

Django [14] es un framework de desarrollo web en Python que permite crear aplicaciones de forma rápida, segura y escalable.

Se ha utilizado como backend principal en este proyecto para la gestión de la lógica de negocio, las interacciones con el sistema de almacenamiento y la comunicación con la API de OpenAI. Además, proporciona una estructura robusta para la creación de rutas, controladores y modelos de datos, permitiendo una integración eficiente con el chatbot conversacional y el sistema de interacción con el resultado del docking molecular.

3.2.4 Crispy Forms

Crispy Forms [15] es una librería de Django que facilita la creación y personalización de formularios de forma más sencilla. Ofrece un sistema de plantillas flexible que permite integrarlo con varios frameworks, evitando tener que escribir manualmente el HTML de cada formulario.

Se ha utilizado para agilizar la creación de los formularios de registro, inicio de sesión y verificación.

3.3 Software específico

3.3.1 AutoDock Vina

Como se ha mencionado en el apartado 2, AutoDock Vina es un software de código abierto que permite la realización de experimentos de docking molecular entre un ligando y un receptor.

En este proyecto se emplea una imagen de [Docker](#) que contiene este software, simplificando mucho el número de parámetros a usar y su funcionalidad. De esta forma, se pueden generar los ficheros con los resultados de forma sencilla usando la función `run` de la librería `subprocess`

3.4 Entorno de programación y control de versiones

3.4.1 Git y GitHub

Git [16] es una herramienta de código abierto para el control de versiones diseñada pensando en la eficiencia, la confiabilidad y la compatibilidad, sirviendo tanto para gestionar proyectos grandes como pequeños.

Este ha sido el software escogido para el control de versiones de proyecto, usado en conjunto con GitHub, plataforma diseñada para el alojamiento de proyectos en la nube.

3.4.2 VS Code

VS Code [17] es un editor de código de licencia libre desarrollado por Microsoft. Soporta numerosos lenguajes de programación y es totalmente customizable, pudiendo configurarlo para que resalte la sintaxis, añada autocompletado o se integre con herramientas como Git.

Este ha sido el [Integrated Development Environment \(IDE\)](#) usado para el desarrollo de todo el proyecto debido a su flexibilidad y al gran número de extensiones que posee. Esto facilita diversas tareas como la detección de errores, formateado de código u optimización de librerías.

3.4.3 Sistema Operativo

El sistema operativo usado para el desarrollo de este proyecto ha sido Linux, concretamente la distribución de Ubuntu. Esta elección se debe a la facilidad que ofrece para la gestión

e instalación de bibliotecas y dependencias, así como a su alto grado de personalización, lo que permite adaptar el entorno de desarrollo a las necesidades específicas del proyecto.

Capítulo 4

Metodología

4.1 Metodología usada

Se ha realizado el desarrollo del proyecto usando la metodología incremental. Esta metodología consiste en estructurar el proyecto en etapas o iteraciones que se integrarán de forma progresiva. Las ventajas de usar esta metodología son:

- Se genera software operativo de manera rápida.
- Es un modelo flexible.
- Se realizan iteraciones pequeñas y fáciles de probar.
- Los riesgos son fáciles de administrar.
- Cada iteración es decisiva para la obtención del producto final y es fácil de gestionar.

Además de esto, se han tenido reuniones con los tutores para el seguimiento y revisión del avance del desarrollo.

4.2 Planificación inicial

Se ha realizado una planificación inicial para tener definidos cada uno de los pasos a seguir durante el desarrollo de la aplicación.

El desarrollo comenzó el 20 de febrero y se organizó en los siguientes incrementos:

- **Incremento 1: 20 febrero - 1 marzo**
 - Realización del análisis de requisitos del sistema.
 - Adquisición de conocimientos y práctica con las tecnologías necesarias para el proyecto que no se habían utilizado previamente.

- Ejecución de pruebas con ejemplos proporcionados por los tutores para comprender en mayor profundidad el funcionamiento de dichas tecnologías.
- **Incremento 2: 2 marzo - 4 marzo**
 - Implementación del módulo del chatbot encargado de extraer, a partir de la entrada del usuario, los datos necesarios para el experimento y devolverlos en un formato estructurado.
 - Procesamiento y preparación de los datos obtenidos del chatbot para su uso en las siguientes fases del flujo de trabajo.
- **Incremento 3: 5 marzo - 14 marzo**
 - Implementación de la interfaz web del chatbot para facilitar las pruebas más adelante durante el desarrollo.
- **Incremento 4: 15 marzo - 30 marzo**
 - Implementación del módulo del chatbot encargado de interactuar con el usuario y proporcionarle los datos necesarios mediante consultas a las bases de datos.
 - Implementación del flujo completo de los estados por los que pasa el chatbot hasta realizar el experimento.
- **Incremento 5: 31 marzo - 6 abril**
 - Preparación de los ficheros necesarios para la realización del experimento.
 - Ejecución del experimento usando los ficheros preparados y las opciones proporcionadas por el usuario.
- **Incremento 6: 7 abril - 15 abril**
 - Implementación de la visualización interactiva del resultado del docking molecular.
 - Incorporación de la funcionalidad para que el usuario pueda descargar un archivo con los resultados obtenidos.
- **Incremento 7: 16 abril - 18 abril**
 - Implementación de la página inicial de la aplicación web.
- **Incremento 8: 19 abril - 25 abril**
 - Desarrollo e integración de los formularios de registro e inicio de sesión de la aplicación.

- Implementación de las vistas de la aplicación web asociadas a las funcionalidades de registro e inicio de sesión.
- **Incremento 9: 26 abril - 3 mayo**
 - Desarrollo del sistema de verificación para la creación de nuevos usuarios, incluyendo la generación y envío por correo electrónico del código de verificación.
 - Implementación de la vista de la aplicación web destinada al proceso de verificación de usuarios.
- **Incremento 10: 4 mayo - 11 mayo**
 - Revisión y optimización de módulos existentes del código.
 - Elaboración de la documentación de las clases y funciones que conforman el proyecto.

Podemos ver de forma más visual la planificación del proyecto en la Figura 4.1. Los incrementos se identifican mediante las siglas In-X, siendo X el número del incremento. Con esta planificación, la fecha prevista de finalización del proyecto se estableció para el 10 de mayo.

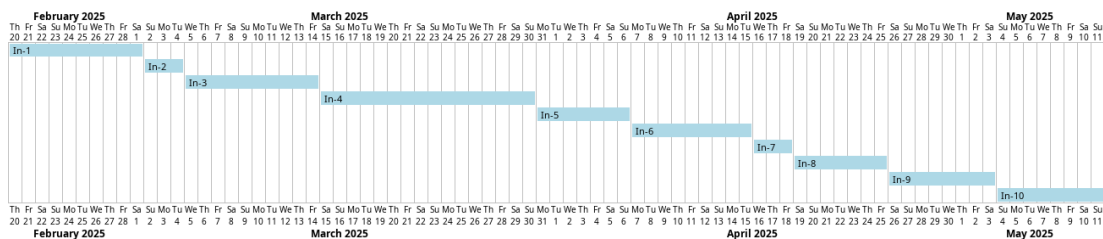


Figura 4.1: Diagrama de Gantt de la planificación inicial

4.3 Seguimiento de la planificación

En la Figura 4.2 podemos ver el seguimiento de la planificación mostrada en el apartado anterior. El color azul representa los incrementos que no se han retrasado; el rojo, los que se han atrasado; y el verde, los que han finalizado antes de lo previsto.

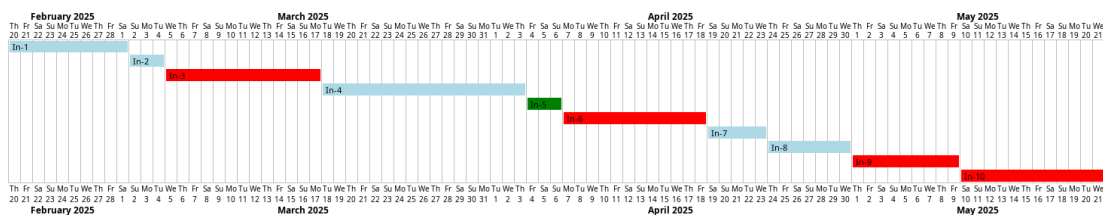


Figura 4.2: Diagrama de Gantt del seguimiento de la planificación

Ha habido 4 incrementos que han tardado más de lo previsto. Estos son:

- **Incremento 3.** El retraso se debió a las dificultades surgidas durante la integración de Django con el backend encargado de la generación de respuestas.
- **Incremento 6.** Este retraso estuvo relacionado con la integración de la librería 3DMol.js en Django. El principal problema fue la carga de las rutas de los ficheros para la visualización, ya que la ruta base proporcionada por defecto por Django no resultaba válida.
- **Incremento 9.** El retraso en este incremento se originó al preparar el servicio de correo electrónico. Inicialmente, se asumió que el proceso sería más sencillo de lo que resultó ser en la práctica.
- **Incremento 10.** El retraso se produjo por la decisión de última hora de implementar un gestor de recursos para la obtención de los ficheros necesarios para el experimento. El objetivo era abstraer la funcionalidad de obtención y/o creación de dichos ficheros del estado final del chatbot que ejecutaba el experimento.

Sin embargo, hubo un incremento que llevó menos tiempo del esperado, el cual fue el incremento 5. La preparación de los ficheros necesarios para el experimento resultó más sencilla de lo estimado, especialmente durante la integración de la librería RDKit para generar el fichero del ligando, cuya implementación fue más sencilla de lo esperado inicialmente.

4.4 Gestión de riesgos

Se ha realizado una gestión de riesgos para detectar posibles problemas antes del desarrollo del proyecto. Para ello, se detectaron dos tipos de riesgos: técnicos y de recursos humanos.

Podemos ver los riesgos técnicos en la Tabla 4.1 y los de recursos humanos, en la Tabla 4.2.

Nombre	Probabilidad	Impacto
Errores en el código	Media	Bajo
Fallo del equipo informático	Baja	Alto
Falta experiencia en las tecnologías usadas	Media	Medio
Errores de diseño	Media	Baja

Tabla 4.1: Riesgos técnicos del proyecto

Nombre	Probabilidad	Impacto
Indisponibilidad temporal del personal (por motivos de salud o personales)	Bajo	Bajo
Sobrecarga de trabajo	Bajo	Bajo

Tabla 4.2: Riesgos de recursos humanos del proyecto

Para los riesgos con más impacto, se han propuesto las siguientes medidas:

- **Fallo del equipo informático.** Para mitigarlo, se ha usado software de control de versiones y se hacen guardados cada vez que se termina una pequeña tarea. De esta forma, si el ordenador falla, se puede retomar el proyecto con otro equipo.
- **Falta de experiencia en las tecnologías usadas.** Para mitigarlo, se han realizado tutoriales y se ha leído la documentación oficial con el objetivo de saber un poco más a fondo cómo funcionan dichas tecnologías.

4.5 Costes

Para la evaluación de los costes, se tuvieron en cuenta los recursos materiales y humanos. De esta forma, tendríamos los siguientes costes:

4.5.1 Costes humanos

En este proyecto han participado dos perfiles diferentes:

- **Desarrollador junior.** El salario medio de este perfil son unos 9,62€/h [18].
- **Profesor universitario.** El salario medio de este perfil son unos 20,87€/h [19].

4.5.2 Costes materiales

Dentro de los costes materiales de este proyecto, tenemos:

- **Equipos informáticos.** Este coste se ha estimado en función de su amortización. Para ello, se ha considerado que cada ordenador tiene un costo de 1200€ y que posee una vida útil estimada de 4 años (48 meses).

Para realizar el cálculo, usaremos el coste por hora usada. Para calcularlo, primero obtendremos el **coste mensual**, el cual se obtendrá usando la siguiente fórmula:

$$CosteMensual = \frac{CosteInicial}{Vidatil}$$

En total, nos saldría un costo de 25€/mes.

Para calcular el **costo por hora**, usaremos la siguiente fórmula:

$$CosteHora = \frac{CosteMensual}{HorasUsoMensual}$$

El coste de cada equipo será calculado de acuerdo a las horas trabajadas por cada miembro del equipo de desarrollo.

- **OpenAI API.** El modelo usado para el desarrollo ha sido *gpt-4o*, cuyo coste es de 1,25\$ (1,07€) por millón de tokens de entrada y 10\$ (8,59€) por millón de tokens de salida [20].

El coste de realizar una petición a la API se calculará atendiendo a la siguiente fórmula:

$$CostePetición = \frac{TokensEntrada}{1000000} * 1.07 + \frac{TokensSalida}{1000000} * 8,59$$

Para ello, tomaremos de media que, por petición, se tienen 15000 tokens de entrada y 2000 tokens de salida por petición. Atendiendo a la fórmula anterior, se obtendría un costo de 0,03€ por petición aproximadamente.

4.5.3 Costes totales

Podemos ver el desglose del coste total del desarrollo del proyecto en la Tabla 4.3.

Recurso	Cantidad	Uso	Coste	Total
Programador junior	1	400 horas	9,62€/h	3848€
Profesor universidad	2	20 horas	20,87€/h	834,80€
Equipo informático	1	400 horas	0,25€/h	100€
Uso API	1	210 peticiones	0,03€/petición	6,3€
Total	-	-	-	4789,10€

Tabla 4.3: Costes totales del proyecto

Diseño y desarrollo

5.1 Casos de uso

Podemos ver este diagrama en la Figura 5.1, donde se pueden ver los actores y los casos de uso que puede realizar cada actor.

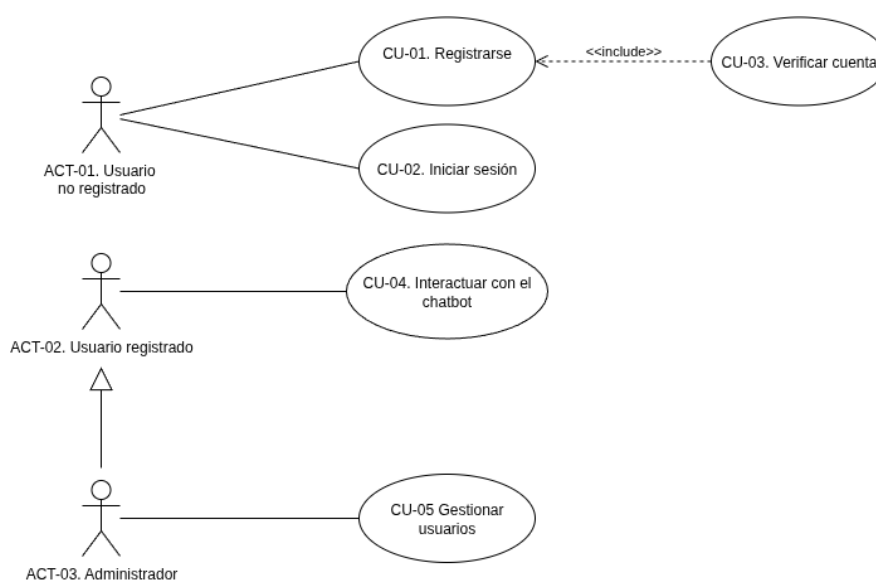


Figura 5.1: Diagrama de casos de uso

5.1.1 Definición de los actores

Podemos ver una descripción más detallada de los actores en las Tablas 5.1, 5.2 y 5.3.

ACT-01	Usuario no registrado
Descripción	Actor que representa a un usuario que no está identificado en el sistema
Sinónimos	N/A

Tabla 5.1: Definición del ACT-01

ACT-02	Usuario registrado
Descripción	Actor que representa a un usuario que está identificado en el sistema
Sinónimos	N/A

Tabla 5.2: Definición del ACT-02

ACT-03	Administrador
Descripción	Actor que gestiona la web.
Sinónimos	N/A

Tabla 5.3: Definición del ACT-03

5.1.2 Casos de uso del sistema

Podemos ver una descripción más detallada de los casos de uso en las Tablas [5.4](#), [5.5](#), [5.6](#), [5.7](#) y [5.8](#).

CU-01	Registrarse
Descripción	Permite crear una cuenta nueva en el sistema
Precondición	N/A
Secuencia normal	<ol style="list-style-type: none"> 1. El sistema mostrará un formulario con los siguientes campos: <ul style="list-style-type: none"> – Nombre de usuario – Nombre – Apellidos – Correo electrónico – Contraseña 2. El usuario introducirá sus datos. 3. El sistema verificará que todos los datos son correctos. 4. El sistema enviará un código de verificación al correo del usuario una vez el administrador haya validado la cuenta. Para ello, ver CU-03 Verificar Cuenta.
Postcondición	N/A
Excepciones	<ol style="list-style-type: none"> 3'. Si ya existe un usuario con el mismo nombre de usuario, el sistema lanzará una excepción y se volverá al paso 1. 3". Si ya existe un usuario con el mismo correo electrónico, el sistema lanzará una excepción y se volverá al paso 1. 3'''. Si la contraseña no cumple con las restricciones indicadas, el sistema lanzará una excepción y se volverá al paso 1.

Tabla 5.4: Caso de uso CU-01: Registrarse

CU-02	Iniciar sesión
Descripción	Permite iniciar sesión en una cuenta ya existente.
Precondición	N/A
Secuencia normal	<ol style="list-style-type: none"> 1. El sistema mostrará un formulario con los siguientes campos: <ul style="list-style-type: none"> – Nombre de usuario – Contraseña 2. El usuario introducirá sus datos. 3. El sistema verificará que todos los datos correspondan con el de una cuenta existente e iniciará la sesión.
Postcondición	N/A
Excepciones	<ol style="list-style-type: none"> 3'. Si el nombre de usuario no coincide con el de algún usuario registrado, el sistema lanzará una excepción y se volverá al paso 1. 3". Si la contraseña no es correcta, el sistema lanzará una excepción y se volverá al paso 1.

Tabla 5.5: Caso de uso CU-02: Iniciar sesión

CU-03	Verificar cuenta
Descripción	Permite realizar una verificación de dos pasos al registrar una nueva cuenta.
Precondición	N/A
Secuencia normal	<ol style="list-style-type: none"> 1. El sistema enviará un correo al usuario con el código una vez el registro sea aprobado por el administrador. 2. El sistema mostrará un formulario con un campo donde el usuario deberá introducir el código de verificación. 3. Opcionalmente, el usuario podrá pedir que se le reenvíe un código nuevo. 4. El usuario introducirá el código. 5. El sistema verificará que el código sea correcto.
Postcondición	N/A
Excepciones	<ol style="list-style-type: none"> 4'. Si el código no es correcto, el sistema lanzará una excepción y se volverá al paso 2. 4". Si el código ha caducado, el sistema lanzará una excepción y se volverá al paso 2.

Tabla 5.6: Caso de uso CU-03: Verificar cuenta

CU-04	Interactuar con el chatbot
Descripción	Permite realizar la interacción con el chatbot con el objetivo de realizar un experimento de docking molecular.
Precondición	N/A
Secuencia normal	<ol style="list-style-type: none">1. El usuario enviará un mensaje al chatbot.2. El sistema interactuará con el usuario hasta obtener la información suficiente para realizar el experimento del docking.3. El usuario podrá interactuar con la molécula resultante del docking.4. El usuario podrá descargar el fichero con los resultados del docking.
Postcondición	N/A
Excepciones	<ol style="list-style-type: none">2'. Si ha ocurrido algún error interno, el sistema lanzará una excepción y se volverá al paso 1.

Tabla 5.7: Caso de uso CU-04: Interactuar con el chatbot

CU-05	Gestionar usuarios
Descripción	Permite crear, eliminar y modificar a los usuarios registrados y validar a un usuario que se acaba de registrar.
Precondición	N/A
Secuencia normal	<ol style="list-style-type: none"> 1. El sistema muestra una lista donde se muestran los usuarios ya registrados y un botón que permite crear uno nuevo. 2. El usuario elige la acción a realizar. 3. El sistema muestra un formulario con los siguientes campos: <ul style="list-style-type: none"> – Nombre de usuario – Nombre – Apellidos – Correo electrónico – Es un usuario verificado. – Es un usuario verificado por OTP (verificación en dos pasos). – Código de verificación de dos pasos. – Fecha de creación del código. – Fecha de expiración del código. 4. El usuario rellena los campos. 5. El sistema verifica que todos los campos sean correctos
Postcondición	N/A
Excepciones	5'. Si alguno de los campos está mal formado, el sistema lanza una excepción y se vuelve al paso 3.

Tabla 5.8: Caso de uso CU-05: Gestionar usuarios

5.2 Diagramas de componentes

Podemos ver el diagrama de componentes del backend en la Figura 5.2. Como se puede observar, en el módulo `web` tenemos los ficheros encargados de la configuración de la web. En `apps` tenemos las vistas, modelos y URLs de cada aplicación (en este caso serían dos: la parte del chat y la parte de gestión de usuarios). En `services`, tenemos todo lo referente al backend de ambas aplicaciones: los hilos que se utilizan en el chatbot, los estados por los que va pasando y los formularios de la parte de usuario.

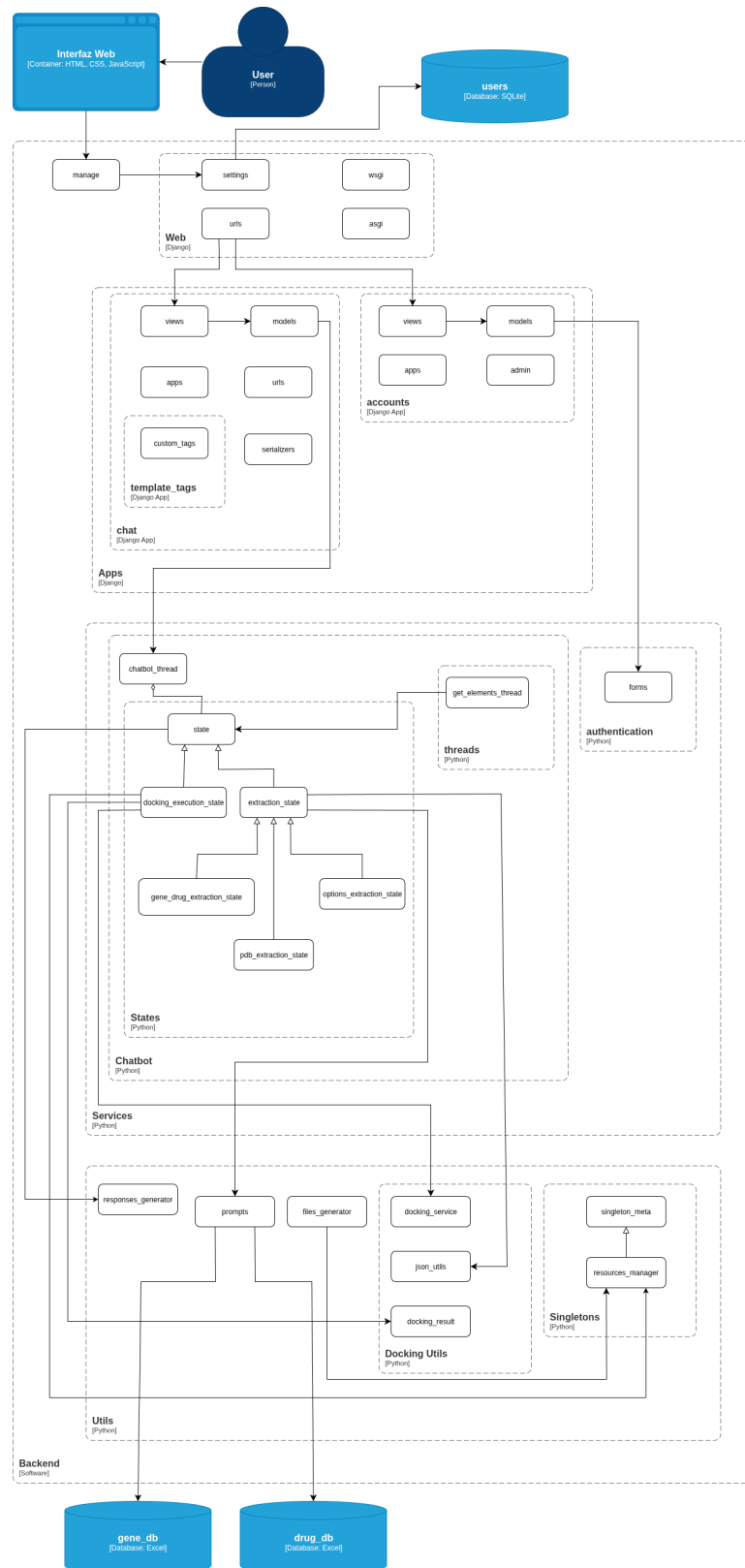


Figura 5.2: Diagrama de componentes del backend

Por último, podemos ver el diagrama de componentes del frontend en la Figura 5.3. En `templates` tenemos los HTML de todas las páginas que componen la web, mientras que en `static` tenemos los ficheros JavaScript y CSS de cada plantilla.

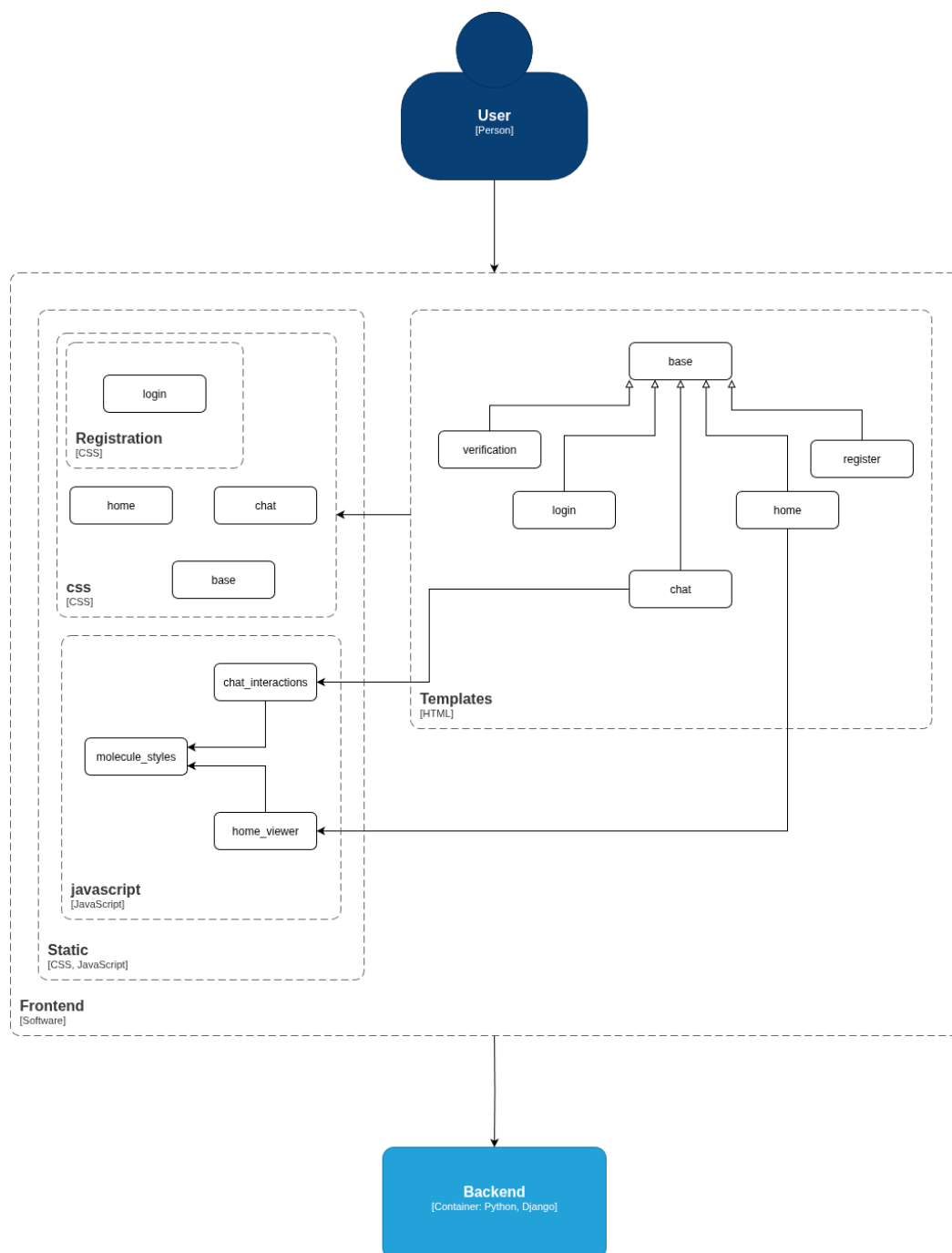


Figura 5.3: Diagrama de componentes del frontend

5.3 Diagramas de clases

En la Figura 5.4 se pueden ver las clases que componen la aplicación de usuario (donde se define un nuevo User y Admin) y la aplicación del chat (donde cada conversación tiene un ID que lo identifica). En la Figura 5.5 podemos ver la estructura del backend. En el hilo que interactúa con el usuario, ChatbotThread, se guarda el estado del chatbot, el cual dependerá del punto en el que estemos de la conversación con el usuario.

Además, se ha implementado un gestor de recursos para separar la lógica de obtención de los ficheros de la funcionalidad del chatbot. Por último, tenemos DockingResult, que posee las rutas de los ficheros de los resultados y DockingService, que ejecutará el docking en caso de que los ficheros con los resultados no estén en caché.

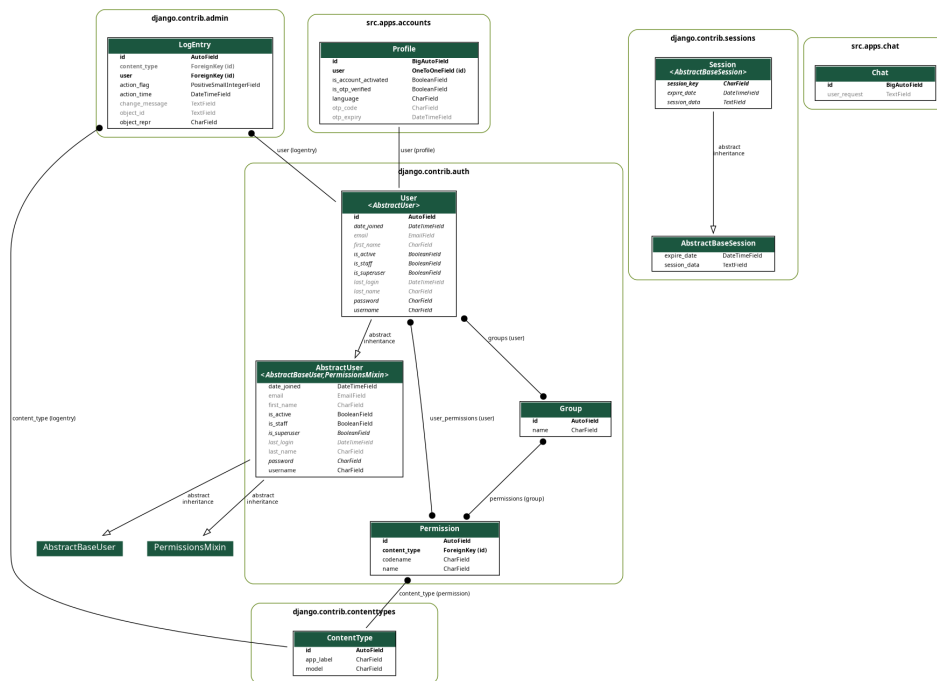


Figura 5.4: Diagrama de clases de la estructura de las aplicaciones

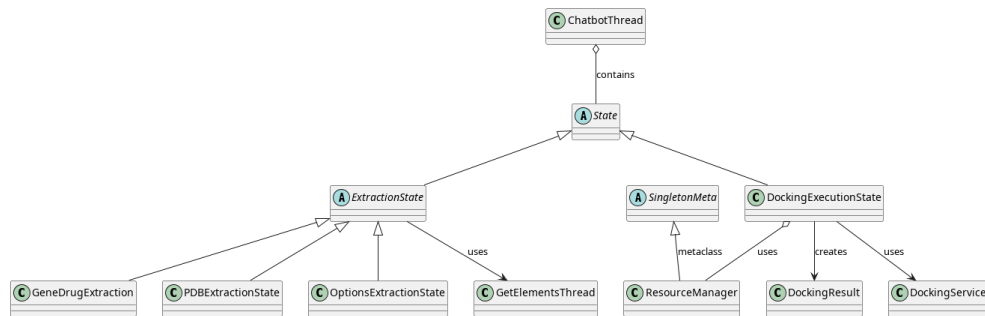


Figura 5.5: Diagrama de clases de la estructura interna del chatbot

5.4 Diagramas de estados

En la Figura 5.6, podemos ver los estados internos del chatbot. Se empieza en `GeneDrugExtractionState`, donde se extrae el gen y el fármaco de la entrada del usuario. En `PDBExtractionState` se obtiene el `PDB` que quiere usar el usuario para el docking a partir de su entrada. Una vez obtenidos estos datos, se pasa a `OptionsExtractionState`, donde se obtienen las opciones con las que el usuario quiere realizar el docking. Por último, se pasa a `DockingExecutionState`, donde se ejecuta el docking. Una vez terminado, se pasa al estado inicial para estar preparado para realizar otro experimento de docking.

En la Figura 5.7, se visualiza la secuencia de páginas web a las que va accediendo el usuario.

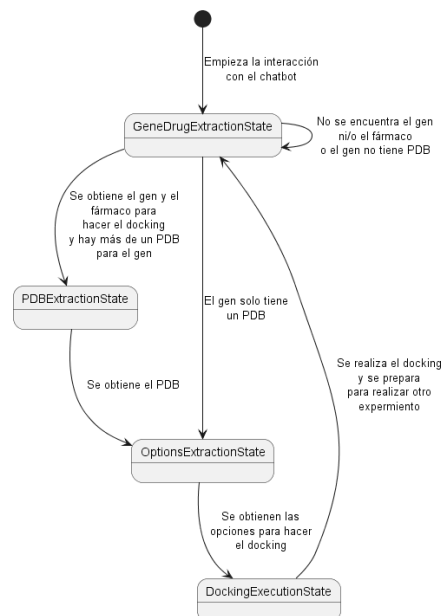


Figura 5.6: Diagrama de estados sobre los estados por los que va pasando el chatbot

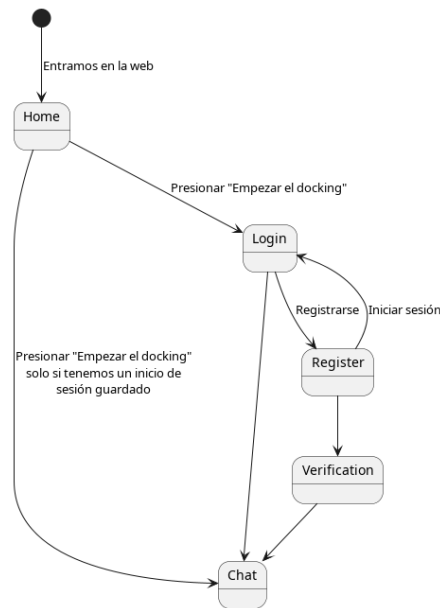


Figura 5.7: Diagrama de estados de las diferentes páginas que componen la web

5.5 Patrones de diseño

Los patrones de diseño usados en este proyecto son los siguientes:

- **Patrón Plantilla.** Este patrón define el esqueleto de un algoritmo en un método, pudiendo dejar algunos pasos específicos a ser implementados por las subclases. El método de la clase base proporciona una secuencia de pasos fijos, pero permite que las subclases sobrescriban ciertos métodos o pasos sin cambiar la estructura general del algoritmo, promoviendo la reutilización de código y permitiendo que se sigan pasos específicos.

Este patrón se utiliza en aquellos estados que heredan de `ExtractiOnState`, donde se redefinen todos sus métodos ya que, dependiendo del estado, se cambiará el `Prompt` (puesto que cada uno debe devolver un JSON con distintos cambios) y, después de la extracción, se deben hacer ciertas acciones dependiendo del estado. Por ejemplo, en `GeneDrugExtractiOnState` se deben obtener el número de PDBs para saber a qué estado se debe transicionar.

- **Patrón Estado.** Este patrón permite que un objeto altere su comportamiento cuando cambia su estado interno, delegando la lógica de cada estado a clases concretas. Es útil cuando una entidad puede tener muchos estados y su comportamiento varía dependiendo del estado en el que se está.

Esto se aplica directamente en el chatbot, ya que dependiendo del estado en el que se

encuentra, se extraerán diferentes datos o se realizarán diferentes tareas. Por ejemplo, en `PDBExtractionState`, se obtiene el PDB seleccionado por el usuario a partir de su entrada, mientras que en `DockingExecutionState` ejecuta el docking molecular y le pasa los ficheros con los resultados al frontend para mostrarlos en pantalla.

- **Patrón Singleton.** Este patrón garantiza que una clase tenga una única instancia en todo el programa, proporcionando un punto de acceso global a ella. Si no hay ninguna instancia creada al llamarlo, se crea y se devuelve. Es útil cuando se necesita un único objeto compartido.

Este patrón se utiliza únicamente en `ResourceManager`.

- **Patrón MVT (Model-View-Template).** Patrón similar al patrón [Model-View-Controller \(MVC\)](#) usado en diseño web. Consta de las siguientes partes:

- **Model.** Define la estructura y comportamiento de los datos.
- **View.** Contiene la lógica de negocio y controla qué datos se muestran.
- **Template.** Capa de presentación, normalmente un HTML.

Permite la separación de responsabilidades y la reutilización de componentes.

Este patrón de diseño es usado por el framework de Django. A partir de una URL, Django busca en `urls.py` la vista correspondiente a esa URL. Desde la vista, accede al modelo para obtener los datos y renderiza el HTML pasándole dichos datos.

5.6 Incrementos

5.6.1 Incremento 1

En este primer incremento se ha realizado el análisis de requisitos del sistema. Este incremento se realizó con el objetivo de sentar las bases del desarrollo de las siguientes iteraciones. Esta etapa incluyó:

- **Requisitos funcionales.** Se identificaron las funcionalidades esenciales del sistema. Estas, en un inicio, serían:
 - Interacción con el chatbot.
 - Obtención de los parámetros para la realización del experimento a partir de la entrada del usuario.
 - Visualización del resultado final del experimento.

- **Requisitos no funcionales.** Se identificaron los criterios de rendimiento, usabilidad y compatibilidad del sistema. Estos serían:
 - Compatibilidad con la mayoría de navegadores.
 - Interfaz sencilla e intuitiva.
 - Peticiones asíncronas para evitar que la página se congele mientras se procesan.

A continuación, se realizó una adquisición de conocimientos de las tecnologías que se iban a usar en el proyecto. Para ello, se hizo uso tanto de documentación oficial como de tutoriales completos, lo que permitió tener una pequeña base de aquellas tecnologías que no se habían usado antes, como Django.

Por último, en la primera reunión con los tutores se obtuvo un conjunto de ejemplos con la API de OpenAI para comprender su funcionamiento, por lo que se realizó un conjunto de pruebas para entender el funcionamiento y aprender a configurar la API correctamente para no tener problemas al comienzo del desarrollo de la aplicación.

5.6.2 Incremento 2

En este incremento se abordó una característica esencial del sistema: la capacidad de que, a partir de la entrada del usuario, se obtengan los datos necesarios para el docking.

Como primer paso, se decidió obtener el gen de la proteína y el fármaco que se utilizarían en el experimento. Para ello, se diseñó un prompt específico para esta tarea. Además de ello, se le propuso un formato de salida, en este caso en formato JSON, ya que sería más fácil obtener los datos más adelante para realizar el experimento.

El prompt del sistema sería este:

Analyze the user's input and extract only the specified protein/gene and drug. Disregard any additional details. Return the extracted values in the exact JSON format below:

```
1 {  
2   "protein": "protein",  
3   "drug": "drug"  
4 }
```

Una vez se realiza la petición al chatbot usando la API de OpenAI, se obtuvo la respuesta con el formato deseado de forma satisfactoria. Por ejemplo, para la entrada *Quiero realizar el docking entre el gen ABAT y el fármaco Didox*, el sistema devolvería el siguiente JSON:

```
1 {  
2   "protein": "ABAT",  
3   "drug": "Didox"  
4 }
```

Una vez obtenida la respuesta del chatbot, era necesario extraer el contenido de cada campo del objeto JSON. Para ello, se creó el fichero `json_utils` en el que se emplea la librería `json` para transformar la cadena de texto devuelta por el chatbot en un objeto JSON y acceder a sus valores.

En el caso de que la respuesta contuviese un JSON mal formado o caracteres ajenos a la estructura esperada, se procedería a limpiar la cadena o a reconstruir dicho objeto, según correspondiese. Para ello, se intentaría transformar la cadena de texto en un objeto JSON directamente usando el método `json.loads`; si salta una excepción, se corregiría.

El resultado de este incremento sería un sistema capaz de analizar la entrada del usuario, extraer la información indicada en el formato establecido y ponerla a disposición del experimento de forma eficiente.

5.6.3 Incremento 3

En este incremento se abordó la implementación de la interfaz web del chatbot. Aunque inicialmente se planeaba desarrollar toda la interfaz web una vez finalizado el backend, se decidió adelantar esta parte para facilitar las pruebas en los incrementos posteriores.

Para el diseño de la interfaz se tomó la decisión de dividirla en dos mitades:

- En la primera mitad se mostraría la visualización interactiva del resultado del docking molecular.
- En la segunda mitad estaría la parte del chat, donde se tendría la barra para escribir el mensaje del usuario, el botón para enviarlo y el historial de los mensajes enviados o recibidos hasta ese momento.

El desarrollo comenzó con la creación de los dos contenedores que representarían a cada funcionalidad de la interfaz. Luego, se definieron la barra donde el usuario escribiría la petición y el botón de envío. En cuanto la apariencia, se escogió una paleta de colores basada en verde y blanco como colores principales, con botones en rojo para generar contraste.

Para garantizar que la web fuera *responsive*, se estableció que en pantallas grandes cada contenedor ocuparía la mitad del ancho de la pantalla (Figura 5.8), mientras que en pantallas de menos de 992 px ocuparían la mitad del largo de la pantalla (Figura 5.9).

Para cargar la vista, se creó una aplicación nueva en Django llamada `chat`, que contiene todas las vistas y modelos correspondientes a esta funcionalidad.

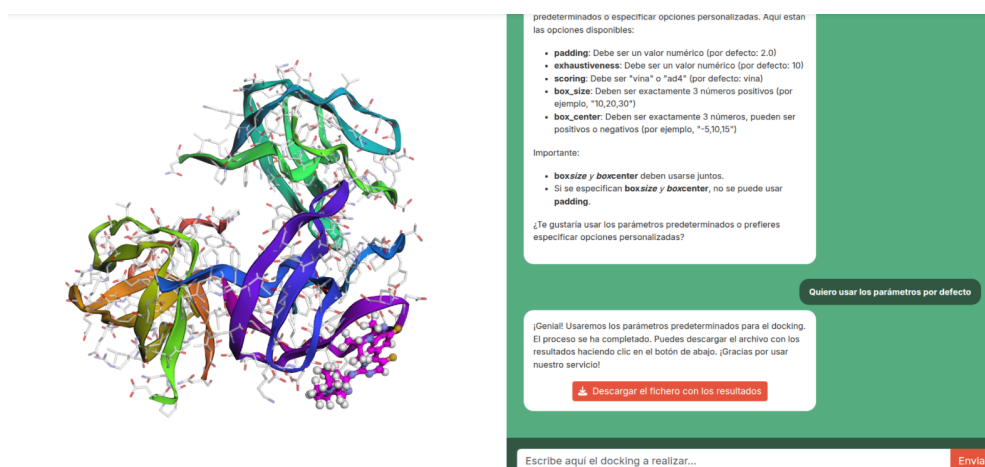


Figura 5.8: Diseño de la interfaz del chat en pantallas grandes

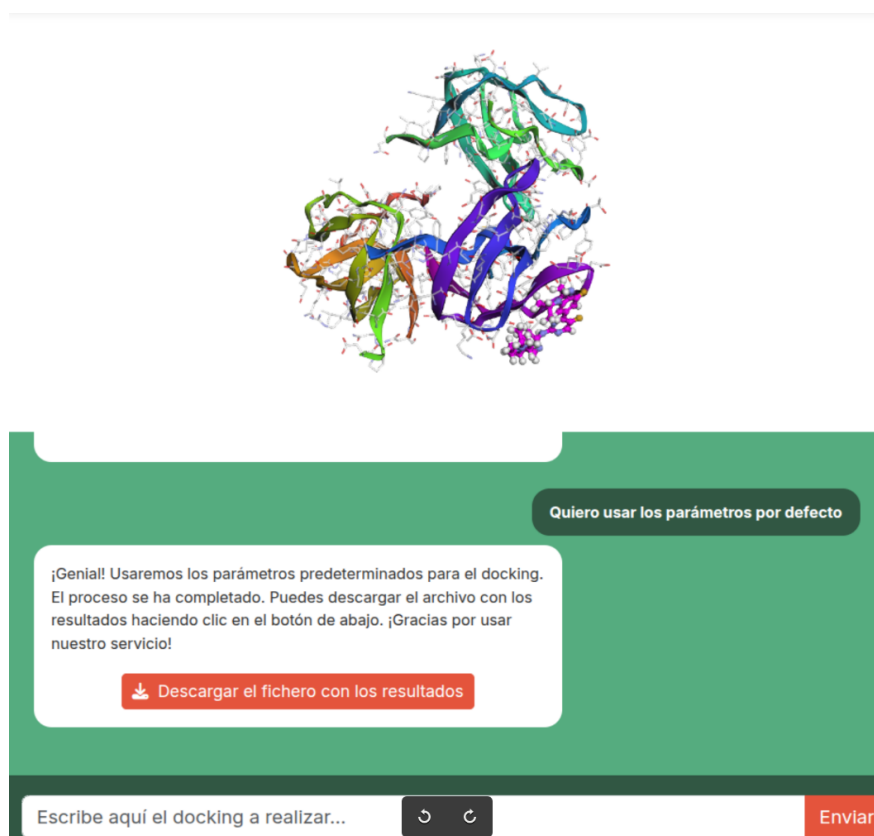


Figura 5.9: Diseño de la interfaz del chat en pantallas pequeñas

En el backend, se implementó un conjunto de conversaciones activas, donde se guardaba el identificador (ID) de cada una. Cuando se recibe un nuevo mensaje, se comprueba si el ID existe, creándose uno en caso contrario. Además, con el objetivo de optimizar el uso de

memoria en el servidor, se estableció que si un usuario no envía ningún mensaje en un plazo de 5 minutos, la conversación finaliza y se elimina.

Para la interacción en la interfaz, se creó un fichero en JavaScript llamado `chat_interaction`, donde se implementó la función `sendMessage`. Esta función se encarga de la generación dinámica de las cajas de mensaje del usuario y del chatbot a medida que se vayan generando. Para mejorar la experiencia del usuario, durante la espera de la respuesta del chatbot se muestra un spinner dentro de su contenedor, evitando que la interfaz se quede congelada mientras se genera la respuesta.

El resultado de este incremento es un sistema que posee una interfaz simple que le permite escribir, enviar y visualizar los mensajes que ha enviado como los generados por el chatbot.

5.6.4 Incremento 4

En este incremento se abordó la implementación del flujo completo del chatbot de forma que el chatbot, además de obtener los datos del usuario, interactuara de forma dinámica con él a lo largo de todo el proceso experimental.

Para desarrollar esta funcionalidad, tras una reunión con los tutores se decidió implementar esta funcionalidad usando concurrencia, de forma que dos hilos actuaran en paralelo:

- Un hilo se encargaría de obtener los datos correspondientes de la entrada del usuario.
- El otro hilo se encargaría de interactuar con el usuario, guiándolo en cada paso.

Para ello, se implementaron dos clases que heredarían de la clase `Thread` del módulo `threading`:

- `ChatbotThread`. Hilo principal del chatbot encargado de almacenar los datos introducidos por el usuario, mantener el historial de la conversación, guardar el estado actual del chatbot e interactuar directamente con el usuario.
- `GetElementsThread`. Hilo secundario responsable de obtener los datos necesarios a partir de la entrada del usuario cuando sea necesario.

Para comenzar el desarrollo, se empezó diseñando el prompt principal del sistema, que define las directrices que debe seguir el modelo de lenguaje. Inicialmente, primero debería proporcionarle al usuario un resumen de la descripción del gen y el fármaco que el usuario había escogido.

Para ello, se proporcionó dos bases de datos en Excel para el desarrollo del proyecto: una para genes y otra para fármacos. Inicialmente, se intentó pasar las dos bases de datos enteras al modelo para que buscara el nombre del gen y el fármaco manualmente y generara el resumen.

Sin embargo, esta estrategia superaba el límite máximo de caracteres de entrada del modelo debido al gran tamaño de los ficheros.

Se optó entonces por un enfoque más eficiente: que el sistema primero obtuviera los datos de entrada del usuario y luego, usando la librería `pandas`, se obtuvieran solo los datos del gen y el fármaco seleccionados por el usuario. En caso de que uno o ninguno de ellos estuviera en las bases de datos, se solicitaba al usuario que introdujera otro.

Además de proporcionar el resumen, se le indicará al usuario las estructuras PDB correspondientes a ese gen. Un mismo gen puede tener ninguna, una o varias estructuras.

Por ello, se tuvieron en cuenta los siguientes casos:

- **Ninguna estructura PDB.** Se le solicita al usuario un gen diferente.
- **Solo una estructura PDB.** Se continuará con la configuración del experimento.
- **Varias estructuras PDB.** El chatbot se las mostrará al usuario y le pedirá que escoja la estructura que quiera.

La Figura 5.10 muestra un ejemplo de respuesta del chatbot al consultar por el gen ABL1 y el fármaco Abemaciclib.

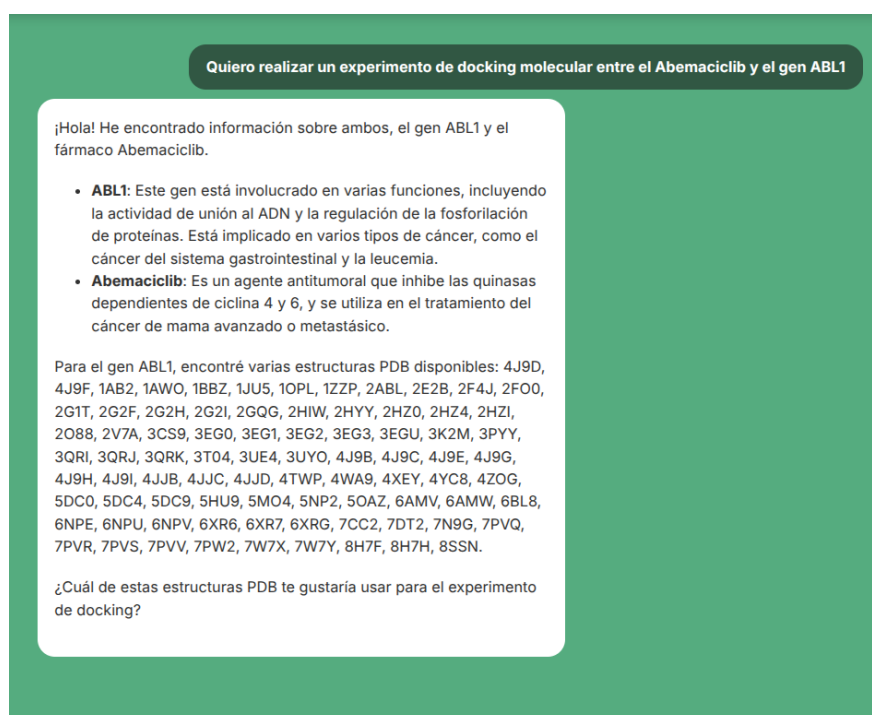


Figura 5.10: Ejemplo del resumen que proporcionará el chatbot ante una petición del usuario

El siguiente paso sería obtener la estructura PDB en el caso necesario. Sin embargo, a estas

alturas del desarrollo, se observó que el chatbot siempre seguiría la misma secuencia a la hora de realizar un experimento:

- Obtener el gen y el fármaco.
- Obtener la estructura PDB que quiere usar el usuario en el caso de que el gen escogido tenga varias.
- Solicitar al usuario las opciones que quiere usar para realizar el experimento.
- Ejecutar el experimento y mostrar el resultado al usuario.

Para estructurar este flujo, se haría uso del patrón de diseño Estado. Para ello, primero se definió la clase abstracta `State`, con el método abstracto `process_user_input`, que cada estado implementa según la lógica que le corresponda.

Además, para aquellos estados encargados de extraer los datos del usuario, se creó la clase abstracta `ExtractionState`, que hereda de `State` y encapsula la lógica común:

- Generar el *prompt* de extracción.
- Lanzar el hilo `GetElementsThread` para procesar la información.
- Ejecutar la acción correspondiente una vez obtenidos los datos, como por ejemplo transicionar de estado.

Podemos ver el desarrollo de este paso en el siguiente fragmento de código:

```

1  def process_user_input(self) -> None:
2      """
3      Processes user input by running an extraction thread with
4      timeout handling.
5
6      Creates and starts an extraction thread to process user
7      input,
8      waits for completion or timeout, and handles the aftermath.
9      """
10     prompt = self._get_extraction_prompt()
11     extraction_done = threading.Event()
12
13     def extraction_callback(result: Dict[str, Any]) -> None:
14         self._handle_extraction_result(result)
15         extraction_done.set()
16
17     extractor_thread = GetElementsThread(
18         prompt, self.context.user_prompt, extraction_callback

```

```
17         )
18         extractor_thread.start()
19
20         if not extraction_done.wait(timeout=self._timeout):
21             self._logger.error("Extraction thread timeout")
22             if self.context.callback:
23                 self.context.callback({"error": _("Response
24                                     timeout, Try again later")})
25             return
26         self._after_extraction()
```

El resultado de este incremento es un sistema capaz de guiar al usuario a través de todos los pasos del experimento y extraer los datos necesarios para su realización.

5.6.5 Incremento 5

En este incremento se trabajó en la preparación de los ficheros necesarios para el experimento y en su posterior ejecución, lo que implicó el desarrollo del último estado del flujo del chatbot.

La primera tarea consistió en preparar los ficheros necesarios, lo cual se realizó de la siguiente forma:

- Para obtener el fichero del ligando (PDB), se descargará directamente de la RCSB Protein Data Bank [21].
- Para obtener el fichero del receptor (fármaco), se usará la librería RDKit para generarlo. Para ello, se obtiene la secuencia SMILES del fármaco y se exporta a formato SDF usando el método `MolFromSmiles`.

A continuación, se llamará a una imagen de Docker que contiene AutoDock Vina, pasando las rutas de los ficheros y las opciones de configuración indicadas por el usuario, tales como:

- **box_enveloping**. Define la caja de búsqueda envolviendo automáticamente el ligando o el receptor. Es la opción por defecto.
- **padding**. Agrega un margen adicional a la caja alrededor del ligando. Debe usarse en conjunto con *box_enveloping*. Su valor por defecto es 2.
- **box_size**. Define el tamaño de la caja. Deben ser 3 números positivos. Si se usa esta opción, no se puede usar *box_enveloping* y *padding*.

- **box_center.** Define la posición central de la caja. Deben ser 3 números. Debe ir junto a *box_size*.
- **exhaustiveness.** Controla cuántas búsquedas se ejecutan. El valor por defecto es 10.
- **scoring.** Indica la función de puntuación. Por defecto es *vina*, pero puede ser *vina* o *ad4*.

Esto se realiza de la siguiente manera:

```
1  def run_vina_docking(  
2      ligand_file: str,  
3      drug_file: str,  
4      options : str,  
5      input_dir="data/input",  
6      output_dir="out/docking_result",  
7  ) -> bool:  
8      """  
9      Execute AutoDock Vina docking through Docker.  
10     """  
11     input_dir = os.path.abspath(input_dir)  
12     output_dir = os.path.abspath(output_dir)  
13  
14     cmd = [  
15         "docker",  
16         "run",  
17         "-it",  
18         "--rm",  
19         "-v",  
20         f"{input_dir}:/input",  
21         "-v",  
22         f"{output_dir}:/output",  
23         "cafernandezlo/dock-tools:v1.0",  
24         "vina",  
25         ligand_file,  
26         drug_file,  
27     ]  
28  
29     # Add options as individual arguments  
30     if options:  
31         options_list = options.split()  
32         cmd.extend(options_list)  
33  
34     logger.debug(f"Executing docking command: {' '.join(cmd)}")  
35  
36     try:
```



```
37         result = subprocess.run(cmd, capture_output=True,
38         text=True, check=False)
39
40         if result.returncode != 0:
41             logger.error(f"Error executing docking:
42             {result.stderr}")
43             return False
44
45         logger.debug(f"Docking completed successfully:
46         {result.stdout}")
47         return True
48
49     except Exception as e:
50         logger.error(f"Exception during docking execution:
51         {str(e)}")
52         return False
```

Al terminar el experimento, se generan 4 ficheros:

- **pdb_farmaco_vina.log**. Registro del experimento de docking, que incluye los parámetros utilizados, la energía final obtenida, la afinidad entre moléculas y otros datos relevantes. Al final de este fichero, podemos ver una tabla resumen como la de la Tabla 5.9

mode	affinity (kcal/mol)	dist from best mode (rmsd l.b)	dist from best mode (rmsd u.b)
1	0	0	0
2	0	5.0709	9.2093
3	0.0001	4.3368	9.9457
4	0.0002	5.0944	10.0467
5	0.0007	6.0471	10.1564
6	0.0014	3.8377	5.9957
7	0.0017	3.3784	5.4403
8	0.0024	4.1826	7.7967
9	0.0030	4.7598	7.0196

Tabla 5.9: Resumen de conformaciones del experimento de docking molecular

En este ejemplo, las afinidades resultan próximas a cero. Esto se debe a que el docking se realizó con la molécula ya posicionada y con parámetros que no optimizaron energéticamente la unión, por lo que el valor carece de significado físico. Sin embargo, la tabla sigue siendo útil para analizar la variabilidad estructural entre las conformaciones propuestas.

- **pdb_farmaco_out.pdbqt.** Contiene la estructura tridimensional de la proteína (receptor) en formato [PDBQT](#).
- **pdb.pdbqt.** Contiene las coordenadas de la colocación del fármaco en el experimento en formato PDBQT.
- **pdb_farmaco_out.pdbqt.sdf.** Contiene la estructura tridimensional del fármaco en formato SDF.

Para reducir tiempos de procesamiento, antes de ejecutar un nuevo experimento se verifica si los ficheros del ligando, receptor y los generados al terminar el experimento ya estaban en caché. Si existían, se reutilizaban directamente y se enviaban al frontend para su posterior procesado.

La comprobación se realiza mediante el nombre de los ficheros. Para los ficheros con los resultados, al generarlos se les asigna un nombre que incluye el nombre del fármaco, la estructura PDB y las opciones de ejecución empleadas. Por ejemplo, para los ficheros generados por el experimento de docking molecular entre el Abemaciclib, el PDB 2mm3 y la opción box_enveloping, los ficheros se llamarán *2mm3_Abemaciclib_box_enveloping_ligand.pdbqt*, *2mm3_Abemaciclib_box_enveloping_receptor.pdbqt.sdf*, *2mm3_Abemaciclib_box_enveloping_pos.pdbqt* y *2mm3_Abemaciclib_box_enveloping_vina.log*.

En este caso, la respuesta del chatbot incluiría las rutas de los ficheros en un diccionario con la siguiente estructura:

```
1 interaction = {  
2     "role": "assistant",  
3     "content": interaction["content"],  
4     "receptor_file": receptor_file_path,  
5     "pos_file": pos_file_path,  
6     "ligand_file": ligand_file_path,  
7     "docking_result_log": log_file_path,  
8 }
```

El resultado de este incremento es un sistema capaz de generar automáticamente los ficheros requeridos para el experimento y llevar a cabo su ejecución.

5.6.6 Incremento 6

En este incremento se realizó la implementación de la visualización interactiva de la molécula resultante del experimento, así como la funcionalidad que permite al usuario descargar el fichero con los resultados del experimento.

Para la visualización, se usó la librería *3DMol* de JavaScript, que permite cargar ficheros con extensión PDB o PDBQT y mostrarlos en una animación 3D interactiva. De esta manera, el usuario podrá no solo observar la molécula resultante del docking molecular, si no también rotarla, desplazarla y hacer zoom para examinarla con detalle.

En la aplicación chat de Django se desarrollaron dos nuevas vistas:

- **get_docking_file.** Encargada de servir un fichero con extensión PDBQT o SDF.
- **get_docking_log.** Vista encargada de servir el fichero con los resultados del experimento en formato log, permitiendo su descarga directa.

Las implementaciones de estas vistas se hicieron de la siguiente manera:

```
1 def get_docking_file(request: HttpRequest, file_path: str) ->  
  HttpResponse:  
2     base_path = os.path.dirname(settings.BASE_DIR)
```

```
3     full_path = os.path.join(base_path, file_path)
4
5     if not os.path.exists(full_path):
6         return HttpResponseNotFound(_(f"File not found:
7 {file_path}"))
8
9     try:
10         file_ext = os.path.splitext(file_path)[1].lower()
11
12         # Select the appropriate open mode and content type
13         based on the file extension.
14         if file_ext == ".sdf":
15             mode = "rb"
16             content_type = "chemical/x-mdl-sdfile"
17         else:
18             mode = "r"
19             content_type = "chemical/x-pdb"
20
21         with open(full_path, mode) as f:
22             content = f.read()
23
24         return HttpResponse(content, content_type=content_type)
25     except Exception as e:
26         return HttpResponseNotFound(_(f"Docking file not found:
27 {str(e)}"))
```

```
1 def get_docking_log(request: HttpRequest, file_path: str) ->
2     HttpResponse:
3     """
4     Retrieves and serves molecular docking log files.
5     """
6     base_path = os.path.dirname(settings.BASE_DIR)
7     full_path = os.path.join(base_path, file_path)
8
9     if not os.path.exists(full_path):
10         return HttpResponseNotFound(_(f"Log file not found"))
11
12     try:
13         with open(full_path, "r") as f:
14             content = f.read()
15
16         response = HttpResponse(content, content_type="text/plain")
17         response["Content-Disposition"] = (
18             f'attachment; filename="{os.path.basename(file_path)}"'
19         )
20         return response
```

```
20     except Exception as e:  
21         return HttpResponseNotFound(f"Error: {str(e)}")
```

Para integrar esta funcionalidad en el frontend, se modificó la función `sendMessage`, de forma que, si los datos recibidos contienen los ficheros de resultados, estos se carguen automáticamente.

El flujo de visualización empieza obteniendo el contenedor donde se mostrará el resultado, el cual permanece oculto hasta finalizar la carga, aplicando una transición suave para mejorar la experiencia del usuario. Posteriormente, usando la vista mostrada anteriormente, se obtendrán los ficheros y se cargarán en el contenedor.

En la representación visual, se optó por resaltar el ligando con un color más intenso que el receptor ya que, al ser significativamente más pequeño, había ocasiones en las que no se podía visualizar correctamente.

Podemos ver en la Figura 5.11 la molécula resultado entre el fármaco Abemaciclib y el PDB 4J9D, perteneciente al gen ABL1, usando los parámetros por defecto. La estructura magenta en forma de diamante correspondería al ligando, mientras que la estructura más grande correspondería al receptor.

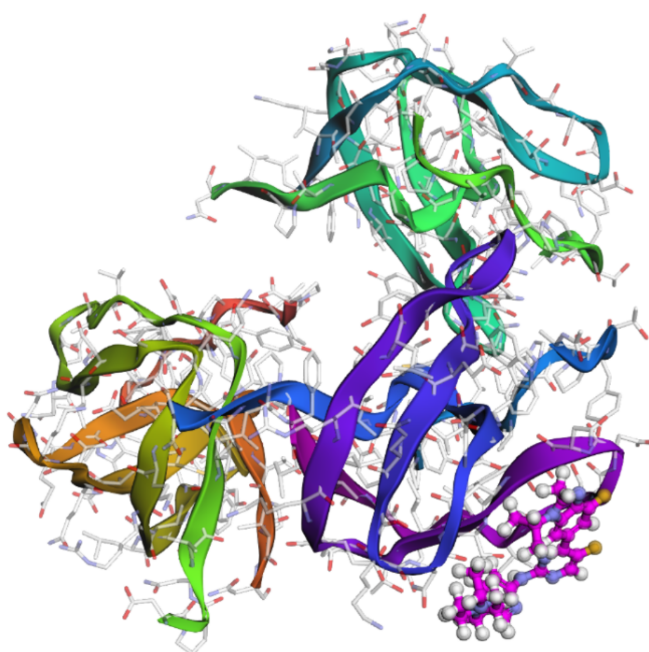


Figura 5.11: Ejemplo de la molécula resultado del docking entre el Abemaciclib y el gen ABL1

Finalmente, para la descarga del fichero log con los resultados, se decidió que la mejor forma de hacerlo era diseñar un botón que aparece en la respuesta del chatbot cuando el

experimento ha terminado. El enlace de la descarga se genera a partir de la ruta proporcionada por la vista `get_docking_log`.

En la Figura 5.12 se muestra un ejemplo del contenedor con el botón de descarga.

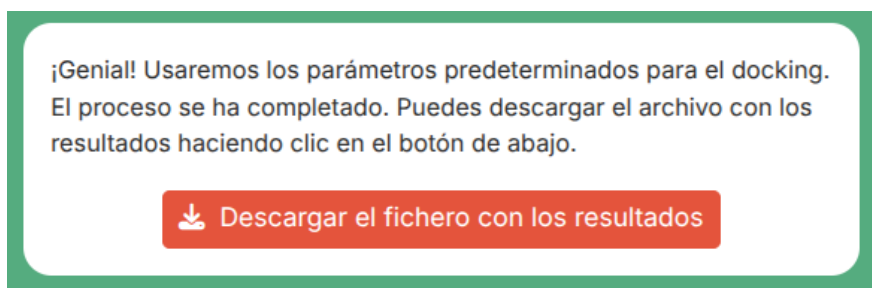


Figura 5.12: Contenedor con el botón de descarga del fichero de los resultados

El resultado de este incremento es un sistema que es capaz de mostrar al usuario una visualización interactiva de la molécula resultante del experimento y de proporcionarle la opción de descargar el desglose de los resultados de dicho experimento.

5.6.7 Incremento 7

En este incremento se desarrolló la página inicial de la aplicación, concebida como la carta de presentación del proyecto.

Tras una reunión con los tutores, se decidió el diseño de la web: esta pantalla integraría dos elementos clave para captar la atención del usuario y mostrar de forma directa las capacidades de la aplicación:

- Una terminal animada que escribiría y borraría mensajes explicativos sobre la finalidad de la web.
- Una visualización interactiva de una molécula, con la que el usuario podría rotar, mover y hacer zoom, familiarizándose así con una de las funcionalidades principales de la aplicación.

Además de ello, para unificar el diseño visual de toda la aplicación, se desarrolló la plantilla `base.html`, de la cual heredarían todas las páginas de la aplicación. Esta incorporaría únicamente una cabecera común, que contendría:

- El nombre de la aplicación, que actuaría como enlace a la página inicial.
- El logotipo del equipo de investigación.

En cuanto a su estética, se escogió el mismo tono verde empleado en la interfaz del chat para conservar la coherencia cromática.

Para la página principal se creó la plantilla `home.html`. Esta contendría dos contenedores principales:

- La terminal animada.
- La visualización molecular, que inicialmente se mantendría oculta.

Además, se incorporó un botón de acceso directo al chatbot para facilitar la navegación.

Durante el desarrollo, se tomó en cuenta la internalización de la web. Para ello, se usó la librería `il8n`, que permite marcar los textos traducibles. La web se encuentra en castellano, gallego e inglés, siendo el idioma principal este último. La selección del idioma se realiza automáticamente en función de la configuración del navegador del usuario, evitando que lo tenga que cambiar manualmente.

También se tomó en cuenta la accesibilidad de la web, por lo que se añadió la funcionalidad de poder navegar por la web usando las teclas. Para moverse entre los botones se usará el tabular y, para hacer clic, el botón Enter. Esto también se aplica a la página del chat.

La animación de la terminal se implementó en JavaScript, especificando:

- La secuencia de mensajes a mostrar.
- El tiempo entre transiciones.
- La acción a ejecutar al finalizar la secuencia.

Los mensajes serán los siguientes:

- *Welcome to PharmaDock AI!*
- *Here you can do the docking of your choice*
- *For example:*
- *Give me the docking between the drug Abemaciclib and the gene 2mm3*

Una vez se muestre este último mensaje, la visualización molecular aparece mediante una transición suave, generando un efecto de presentación progresiva.

La página inicial quedaría como se muestra en la Figura 5.13. Esta ofrece una introducción clara a la finalidad de la aplicación y permite al usuario interactuar con una molécula desde el primer contacto, sirviendo como demostración inmediata de las capacidades de la plataforma.

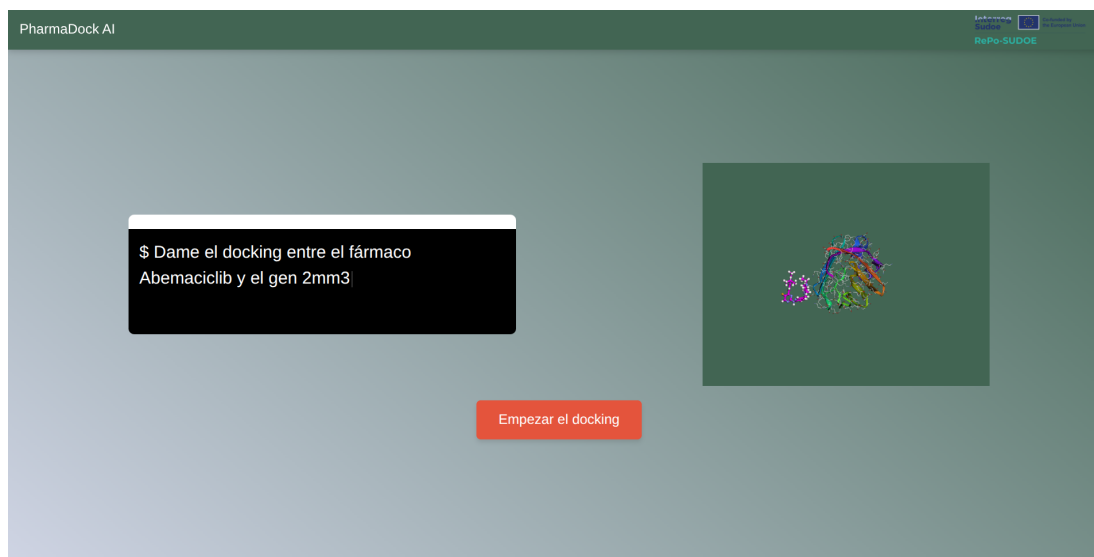


Figura 5.13: Página de inicio de PharmaDock AI

El resultado de este incremento es un sistema con una página de inicio simple, funcional y coherente con el diseño global de la aplicación, proporcionando una combinación entre información y demostración práctica con el objetivo de mejorar la experiencia del usuario desde el primer momento que tiene contacto con la aplicación.

5.6.8 Incremento 8

En este incremento se realizó la implementación de los formularios de registro e inicio de sesión, así como toda la lógica necesaria para su funcionamiento.

Para ello, se creó una nueva aplicación en Django llamada `accounts`, que se encargaría de la gestión del registro, inicio de sesión y cierre de sesión de los usuarios.

En primer lugar, se creó la clase `ChatbotUserCreationForm`, que define los campos que el usuario deberá cubrir para registrarse en la web. Estos serían:

- Nombre de usuario.
- Nombre.
- Apellidos.
- Correo electrónico.
- Contraseña.
- Confirmación de la contraseña.

También se implementaron las vistas correspondientes al logout, registro y login.

Para la creación de los formularios en la interfaz web, se usó la librería Crispy Forms para simplificar la creación de las vistas manteniendo un diseño uniforme, además de añadir la cabecera común con todas las páginas de la aplicación.

En cuanto a la paleta de colores, se conservaron los tonos empleados en el resto de la interfaz para garantizar la uniformidad estética.

Las Figuras 5.14 y 5.15 muestran, respectivamente, las vistas de inicio de sesión y de registro.

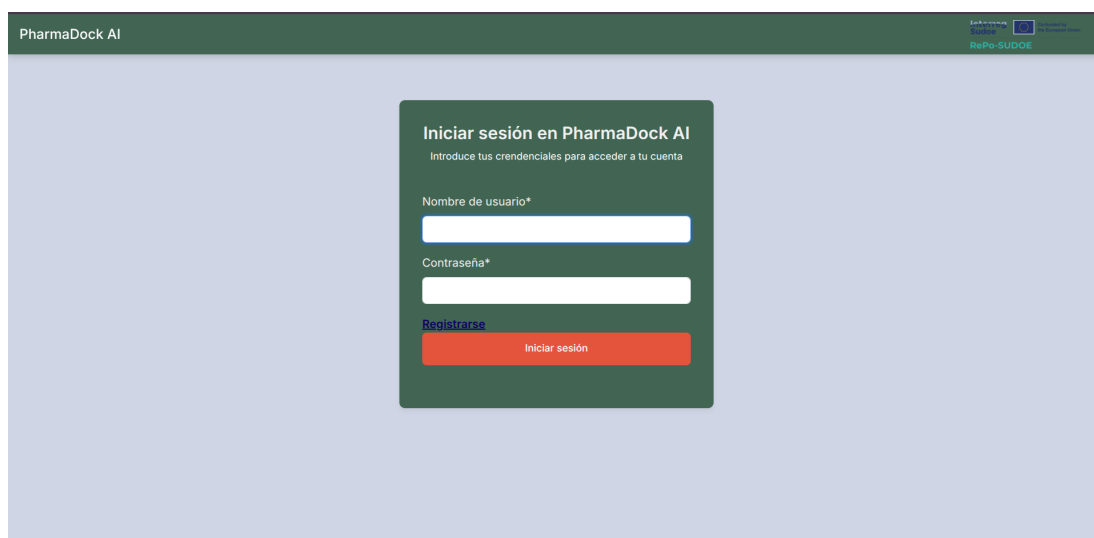


Figura 5.14: Página de inicio de sesión de PharmaDock AI

PharmaDock AI

Regístrate en PharmaDock AI

Introduce tus datos para registrar una nueva cuenta

Nombre de usuario*

Requerido: 150 caracteres como máximo. Únicamente letras, dígitos y @/./+/-/_

Nombre

Apellidos

Dirección de correo electrónico

Contraseña*

Contraseña (confirmación)*

Para verificar, introduzca la misma contraseña anterior.

[Iniciar sesión](#)

Registrarse

- Su contraseña no puede asemejarse tanto a su otra información personal.
- Su contraseña debe contener al menos 8 caracteres.
- Su contraseña no puede ser una clave utilizada comúnmente.
- Su contraseña no puede ser completamente numérica.

Figura 5.15: Página de registro de PharmaDock AI

Una vez el usuario se haya registrado, será redirigido automáticamente a la página del chatbot.

Para añadir la opción de cierre de sesión desde esta vista, se modificó ligeramente la cabecera

- El logotipo del equipo de investigación se situó en el centro.
- El botón de cierre de sesión se colocó en la esquina derecha de la cabecera.

Podemos ver cómo quedaría finalmente la página del chatbot en la Figura 5.16.

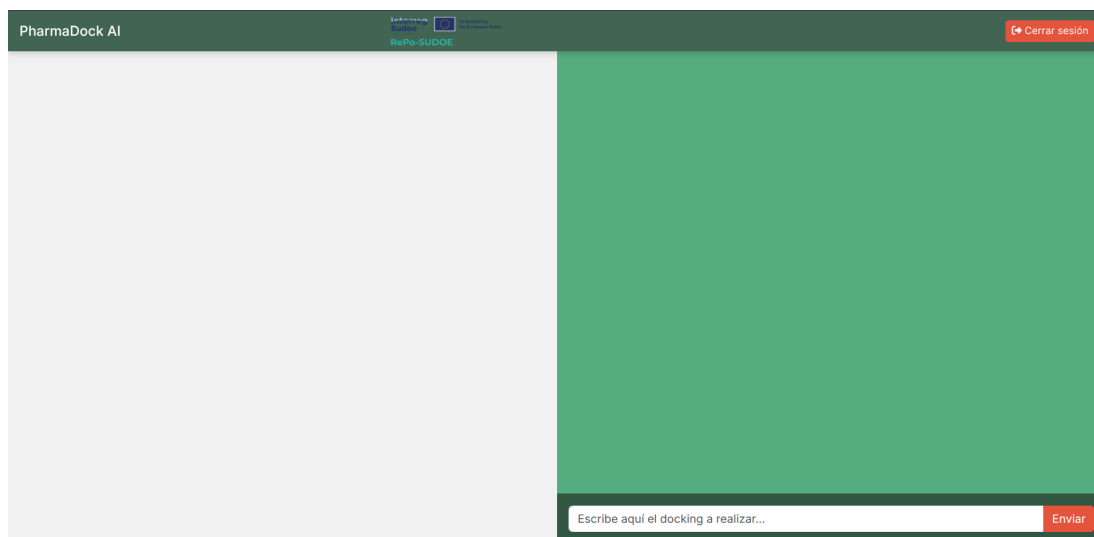


Figura 5.16: Página del chatbot con la cabecera

El resultado de este incremento es un sistema que permite registrar nuevos usuarios, iniciar sesión para acceder al chatbot y cerrar sesión en cualquier momento, asegurando una gestión de usuarios completa y coherente con el resto de la aplicación.

5.6.9 Incremento 9

En este incremento se implementó el sistema de verificación de usuarios. El objetivo es garantizar que, tras registrarse en la aplicación, solo aquellos usuarios autorizados por el administrador puedan acceder a la aplicación.

Este proceso se realiza en 3 fases:

- El administrador revisa y aprueba el registro.
- Una vez aprobado, se envía al usuario un correo electrónico con un código de verificación de 6 dígitos.
- El usuario introduce este código en el formulario de verificación, con la posibilidad de que se le pueda enviar otro código en caso necesario. Si el código es correcto y no ha caducado, la cuenta queda activada y el usuario puede iniciar sesión cuando lo desee.

Para ello, en la aplicación `accounts`, se creó el modelo `Profile`, que extiende de la clase `User` de Django. Aquí se añadirían los siguientes campos al perfil:

- Estado de verificación por parte del administrador
- Estado de verificación mediante el código de verificación (**One-Time Password (OTP)**).

- Código OTP generado para verificarse.
- Fecha de expiración de dicho código.

Se desarrolló la lógica de la generación y verificación del código, junto con dos métodos adicionales:

- **Envío del código.** Genera un OTP de seis dígitos con caducidad de 30 minutos y lo envía al usuario:

```

1      message = (
2          _(
3              ""Hello %(name)s,
4
5              Your PharmaDock AI account has been activated. To
complete
6              the process, please enter the following
7              verification code on the verification page:
8
9              %(otp)s
10
11             This code will expire in 30 minutes. If you did not
request
12             this activation, please contact our support
13             team immediately.
14
15             Regards,
16             The PharmaDock AI team""
17         )
18         % {
19             "name": profile.user.first_name or
profile.user.username,
20             "otp": otp_code,
21         }
22     )
23

```

- **Envío del correo de bienvenida.** Notifica al usuario que su cuenta fue verificada con éxito:

```

1      subject = _("Welcome to PharmaDock AI")
2
3      message = (
4          _(
5              ""Hello %(name)s,
6

```

```

7         Your PharmaDock AI account has been successfully
verified.
8         You now have full access to our platform.
9
10        Regards,
11        The PharmaDock AI team"""
12    )
13    % {"name": profile.user.first_name or
profile.user.username}
14    )
15

```

La generación del código OTP se implementó de la siguiente forma:

```

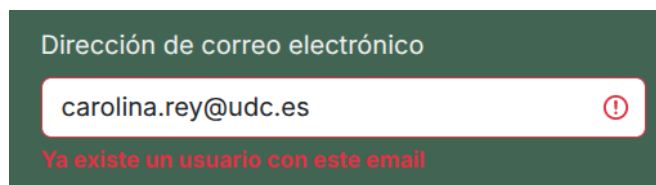
1  def generate_otp(self) -> str:
2      """
3      Generate a 6-digit OTP code with 30-minute expiry.
4      """
5      otp_code = "".join(random.choices(string.digits, k=6))
6      expiry = timezone.now() + timedelta(minutes=30)
7
8      # Update database directly to avoid recursion
9
10     Profile.objects.filter(pk=self.pk).update(otp_code=otp_code,
otp_expiry=expiry)
11
12     self.otp_code = otp_code
13     self.otp_expiry = expiry
14
15     return otp_code

```

Para la internacionalización, el idioma de los correos se adapta automáticamente al idioma principal del navegador del usuario.

Además, se añadió una validación extra en `ChatbotUserCreationForm` para evitar que un usuario pueda registrarse con un correo electrónico ya existente en la base de datos. Esto se hizo con la finalidad de prevenir conflictos en el envío de correos, ya que en caso contrario el sistema podría asociar la dirección a más de un usuario.

Podemos ver el mensaje de error correspondiente en la Figura 5.17.



Dirección de correo electrónico

carolina.rey@udc.es

Ya existe un usuario con este email

This is a screenshot of a registration form. It has a dark green header with the text 'Dirección de correo electrónico'. Below it is a white input field containing the email 'carolina.rey@udc.es'. To the right of the input field is a red circle with a white exclamation mark. Below the input field, the text 'Ya existe un usuario con este email' is displayed in red.

Figura 5.17: Error que aparece al intentar registrar una cuenta con un correo ya existente

Se crearon también las vistas correspondientes al envío, reenvío y verificación del código OTP, al igual que la vista del formulario de verificación. Para ello, en el fichero `forms` se creó la clase `OTPVerificationForm`, que indicaba los campos de este formulario. En este caso, simplemente consistiría en un campo donde se escribiría un código de 6 dígitos.

Para la creación del formulario, se volvería a usar la librería `Crispy Forms` para que el formato fuera consistente con el resto de formularios, manteniéndose su estilo y paleta de colores.

En la Figura 5.18 se muestra cómo se vería este formulario.



PharmaDock AI

Verifica tu cuenta

Introduce el código que ha sido enviado a tu correo electrónico

Código de verificación*

Introduce el código de 6 dígitos

Reenviar el código

Verificar la cuenta

This is a screenshot of a web page for 'PharmaDock AI'. The page has a light blue background. In the center, there is a dark green box with white text. The box contains the title 'Verifica tu cuenta', a subtitle 'Introduce el código que ha sido enviado a tu correo electrónico', a label 'Código de verificación*', an input field with the placeholder 'Introduce el código de 6 dígitos', a link 'Reenviar el código', and a red button labeled 'Verificar la cuenta'. The top of the page has a dark green header with 'PharmaDock AI' on the left and some logos on the right.

Figura 5.18: Página de verificación de PharmaDock AI

Una vez la cuenta se ha verificado con éxito, se muestra un mensaje flotante como el de la Figura 5.19. Este formato también se utiliza para mandar errores, como que el código de verificación introducido es incorrecto.

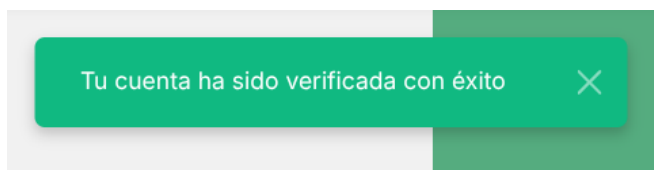


Figura 5.19: Mensaje flotante al verificar la cuenta con éxito

El resultado de este incremento es un sistema capaz de verificar usuarios mediante un código OTP enviado por correo y permitiendo su reenvío, garantizando un acceso más seguro y controlado a la plataforma.

5.6.10 Incremento 10

En esta última iteración se realizó la revisión, optimización y documentación de todos los módulos del código.

La optimización más importante fue en uno de los estados del chatbot: el encargado de realizar el experimento de docking molecular.

Este estado, inicialmente, se encargaba de:

- Cargar o preparar los ficheros necesarios para realizar el experimento.
- Ejecutar el experimento en caso de no existir resultados en caché o cargar los ficheros con los resultados.
- Enviar los ficheros al frontend para su posterior procesado.

Dado que su complejidad era considerablemente mayor que la del resto de estados, se decidió refactorizarlo de forma que su función quedara limitada a:

- Llamar a la clase encargada de realizar el experimento.
- Pasar los resultados al frontend

La primera mejora consistió en encapsular la lógica de preparación de los ficheros necesarios para el experimento. Para ello, se creó un fichero de utilidades llamado `files_generator`, que contendría las funciones responsables de generar dichos ficheros de forma independiente.

Posteriormente se creó un gestor de recursos, encargado de centralizar la obtención de ficheros y la carga de bases de datos, facilitando su integración en el prompt del chatbot. Para implementarlo, se hizo uso del patrón Singleton, garantizando que solo exista una instancia de esta clase en toda la aplicación.

Este gestor de recursos cuenta con un método específico para obtener cada tipo de fichero, aplicando la siguiente lógica.

- Comprobar en su diccionario interno si el fichero ya está disponible.
- Si no, revisar si está en caché. Si está, lo añade al diccionario para un acceso más rápido en el futuro y se devuelve.
- Si el fichero no existe, se ejecuta la acción necesaria para generarlo

La funcionalidad de ejecución del experimento se encapsuló en la clase `DockingService`, cuya única responsabilidad sería lanzar el experimento y devolver un booleano indicando si se completó con éxito.

Además, se creó la clase `DockingResult`, que actuaría como contenedor para almacenar los resultados. Esta clase almacena el nombre de los ficheros generados. Adicionalmente, posee una propiedad que indica si el experimento fue exitoso. Como resultado, el último estado del chatbot únicamente debe enviar esta información al frontend, donde será procesado.

En el frontend no hubo cambios significativos: simplemente se centralizó la configuración de estilos de la visualización de las moléculas en un fichero aparte. Esto permitió eliminar código duplicado, ya que tanto en la página de inicio como en la del chatbot se emplea el mismo estilo.

Por último, se procedió a documentar el código. Para ello, se añadieron *docstrings* a las clases, métodos y funciones. Estos son un tipo especial de comentarios cuyo propósito es documentar qué hace la función o clase, siendo accesibles simplemente poniendo el cursor encima o en tiempo de ejecución a través de atributo `.__doc__`. Además, se incluyeron comentarios aclaratorios en aquellas secciones cuyo funcionamiento pudiera resultar ambiguo para terceros.

Además de ello, se añadieron anotaciones de tipo en todos los parámetros de entrada y en los valores de retorno de las funciones de Python. Esto mejora la legibilidad del código, facilita su mantenimiento y aporta mayor claridad a la documentación ya existente.

El resultado de este incremento es un sistema más modular y eficiente, con un gestor de recursos que facilita la obtención de datos y una arquitectura más clara. Además, el código final es más limpio y más fácil de mantener gracias a la refactorización y la documentación.

Capítulo 6

Pruebas

Para la comprobación del correcto funcionamiento de la aplicación, se optó por la realización de pruebas manuales sobre cada funcionalidad implementada en cada incremento. Esto se decidió ya que el sistema no presenta gran complejidad en cuanto a dependencias internas, por lo que sería suficiente para garantizar su correcto funcionamiento.

Las pruebas se centraron en:

- **Gestión de usuarios.** Se verificó la creación de nuevas cuentas, verificación, inicio y cierre de sesión.
- **Chatbot.** Se interactuó con el asistente comprobando la coherencia de las respuestas y otros aspectos internos, como la transición de estados, correcta obtención de los datos y comprobación de errores.

Para facilitar la detección de errores, se implementó un sistema de logs que cubría todo el backend. En dichos logs se registraban qué datos obtenía el chatbot, a qué estado transicionaba en cada momento y los errores ocurridos durante la interacción con el chatbot.

En el caso de la verificación de usuarios y gestión de cuentas, la administración de Django sirvió como apoyo para confirmar que los registros se almacenaban correctamente en la base de datos.

Si bien no se desarrollaron pruebas unitarias automatizadas, estas pruebas manuales fueron suficientes para garantizar la calidad del sistema en el contexto de esta aplicación. No obstante, en un desarrollo futuro en el que se diseñaran más aplicaciones, sería recomendable añadirlas.

Conclusiones

7.1 Trabajo futuro

A pesar de que la aplicación cumple su propósito y está en un estado totalmente funcional, hay múltiples funcionalidades que se podrían añadir en un futuro para ampliar sus capacidades y mejorar la experiencia de usuario:

- **Consulta de experimentos previos.** Se podría añadir al chatbot un estado adicional donde el usuario podría consultar qué experimentos realizó anteriormente, con la posibilidad de que pueda repetir un experimento ya realizado directamente, saltándonos todo el flujo.
- **Cambio del software de docking.** Se podría adaptar el sistema para ofrecer la opción al usuario de seleccionar entre diferentes programas de docking, pudiendo añadir una clave en caso necesario.
- **Integración con tecnologías de realidad aumentada.** Se podría añadir la posibilidad de visualizar la molécula resultante con gafas de realidad aumentada, añadiendo inmersividad. El usuario podría interactuar con ella usando las manos.
- **Accesibilidad para personas con discapacidad.** La aplicación podría incorporar tanto un sistema de lectura en voz alta de las respuestas del chatbot como un sistema que permita realizar consultas por comandos de voz. Estas mejoras facilitarían el uso de la plataforma por parte de personas con discapacidad visual.
- **Notificación por correo al finalizar el experimento.** Dado que algunos experimentos pueden tardar varios minutos en completarse, sería útil implementar un sistema de notificaciones por correo electrónico que informe al usuario una vez finalizada la ejecución, permitiéndole cerrar la aplicación mientras tanto.

7.2 Lecciones aprendidas

Este proyecto ha supuesto una oportunidad para poner en práctica todo lo aprendido durante estos 4 años de carrera. Principalmente, ha permitido profundidad en la rama de procesamiento de lenguaje natural, un área que no solo resulta muy interesante, sino que también es una puerta hacia el futuro de múltiples aplicaciones que facilitarán la labor de investigación y la vida de los usuarios en general.

Además, la realización de este proyecto ha servido para aprender nuevas tecnologías como lo son Django y afianzar conocimientos en el desarrollo de aplicaciones web. También se ha reforzado la comprensión y la aplicación de metodologías de proyectos.

Otro aspecto a mencionar es que este proyecto ha sido la oportunidad perfecta para adentrarse en la rama de la bioinformática, una rama que siempre ha despertado un interés personal desde que tuve la oportunidad de abordarlo en proyectos anteriores durante la carrera.

En definitiva, este proyecto no solo ha servido para adquirir nuevos conocimientos, sino que también ha servido para consolidar los ya aprendidos, suponiendo un paso decisivo a nivel de desarrollo profesional, ayudando a clarificar el camino que deseo seguir en el futuro.

Apéndices

Ejecución de la aplicación

A.1 Requisitos previos

Los requisitos previos para ejecutar la aplicación son los siguientes:

- **Python 3.13.**
- **python3.13-venv** para la gestión de entornos virtuales
- **pip** para la gestión de paquetes.
- **make** en Linux, para facilitar la instalación.

A.2 Instalación

Una vez descargado el proyecto desde el repositorio, accede a la carpeta del proyecto, abre una terminal y escribe:

- **Linux.**

```
make install
```

- **Windows.**

```
python -m venv .venv  
.venv/Scripts/activate  
pip install -r requirements.txt
```

A.3 Ejecución de la aplicación

Desde terminal, ejecutamos el comando:

- **Linux.**

```
make run
```

- **Windows.**

```
.venv\Scripts\activate  
python manage.py runserver
```

Algunas funcionalidades tales como el envío de correos de verificación o el funcionamiento del chatbot no funcionarán inicialmente.

Para que funcionen correctamente, dentro de la carpeta del proyecto se debe crear un directorio llamado `.config` con un fichero `config.env` donde se indiquen las variables de entorno necesarias. Este fichero debe escribirse de la siguiente forma:

```
OPENAI_API_KEY=tu_api_KEY  
EMAIL_HOST=email_host  
EMAIL_PORT=email_port  
EMAIL_HOST_USER=email  
EMAIL_HOST_PASSWORD=contraseña_email
```

Lista de acrónimos

API Application Programming Interface. 1

CNN Convolutional Neural Networks. 1

CPU Central Processing Unit. 5

CSS Cascading Style Sheets. 8

HTML HyperText Markup Language. 8

IA Inteligencia Artificial. 8

IDE Integrated Development Environment. 10

JSON JavaScript Object Notation. 9

LLM Large Language Model. 1

MVC Model-View-Controller. 30

NLP Natural Language Processing. 9

OTP One-Time Password. 49

Glosario

AutoDock 4 Software que predice las interacciones de unión de moléculas pequeñas, como fármacos candidatos, a receptores con estructuras 3D conocidas. [5](#)

AutoDockTools Interfaz gráfica de usuario para configurar, iniciar y analizar las ejecuciones de AutoDock. [5](#)

binding Proceso en el que una molécula pequeña se acopla en una posición específica de una más grande. [1](#)

bioinformática Disciplina que combina la biología con la informática para analizar y gestionar grandes cantidades de datos biológicos. [1](#)

Docker Proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software. [10](#)

ligando Moléculas pequeñas que se unen al receptor. [1](#)

Machine Learning Rama de la inteligencia artificial centrada en el uso de datos y algoritmos para permitir que una IA imite la forma en la que aprenden los humanos. [1](#)

modelo de lenguaje Sistema que analiza, comprende y genera texto de forma similar a como lo haría un ser humano. [1](#)

PDB Siglas de Protein Data Bank. Describe la estructura tridimensional de una proteína, la cual es clave para comprender su funcionamiento y sus interacciones con otras moléculas. [28](#)

PDBQT Siglas de Protein Data Bank, partial charge (Q), and atom type (T). Es un formato de fichero generado por AutoDock que extiende del formato PDB pero añadiendo información sobre la carga parcial de los átomos y el tipo de átomo. [40](#)

prompt Instrucción, solicitud o texto que se proporciona a un sistema de inteligencia artificial para obtener una respuesta específica o realizar una tarea. [29](#)

PyMol Visor molecular de código abierto. [5](#)

receptor Molécula que recibe una más pequeña. [1](#)

scoring Proceso de evaluar y clasificar las posibles poses (orientaciones y conformaciones) de un ligando dentro del sitio de unión de una proteína mediante una función de puntuación. [1](#)

SDF Siglas de Structured Data File. Es un tipo de archivo utilizado para almacenar datos de manera organizada y accesible. [37](#)

SMILES Siglas de Simplified Molecular Input Line Entry System. Es la especificación para delinear sin ambigüedades una estructura molecular. [9](#)

Bibliografía

- [1] S. Agarwal and S. Raghava, “Mini review: An overview of molecular docking,” *International Journal of Advanced Research*, vol. 4, no. 8, pp. 601–609, 2016. [En línea]. Disponible en: <https://www.researchgate.net/publication/303897563>
- [2] A. R. Domínguez, “Simulación del reconocimiento entre proteínas y moléculas orgánicas (o docking): Aplicación al diseño de fármacos,” 2014. [En línea]. Disponible en: <https://www.researchgate.net/publication/267779223>
- [3] S. Banerjee, S. Mukherjee, S. Das, and S. Das, “Molecular docking in drug discovery: Techniques, applications and advancements,” <https://chemrxiv.org/engage/api-gateway/chemrxiv/assets/orp/resource/item/66301b6f21291e5d1d0b5e1a/original/molecular-docking-in-drug-discovery-techniques-applications-and-advancements.pdf>, 2024, preprint, ChemRxiv.
- [4] J. Sun, A. Li, Y. Deng, and J. Li, “Chatmol copilot: An agent for molecular modeling and computation powered by llms,” in *Proceedings of the 1st Workshop on Language + Molecules (L+M 2024)*. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 55–65. [En línea]. Disponible en: <https://aclanthology.org/2024.langmol-1.7/>
- [5] S. Ishida, T. Sato, T. Honma, K. Terayama, and et al., “Large language models open new way of ai-assisted molecule design for chemists,” *Journal of Cheminformatics*, vol. 17, no. 36, 2025. [En línea]. Disponible en: <https://jcheminf.biomedcentral.com/articles/10.1186/s13321-025-00984-8>
- [6] “Official website for glide,” <https://www.schrodinger.com/platform/products/glide/>.
- [7] “Official website for autodock vina,” <https://vina.scripps.edu/>.
- [8] “The python programming language – official website,” <https://www.python.org/>.
- [9] “Official website for html,” <https://html.spec.whatwg.org/>.

- [10] “Official website for css,” <https://www.w3.org/Style/CSS/Overview.en.html>.
- [11] “Official documentation for javascript,” <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- [12] “Official website for openai api,” <https://platform.openai.com/docs/api-reference/introduction>.
- [13] “Official website for rdkit,” <https://www.rdkit.org/>.
- [14] “Official website for django,” <https://www.djangoproject.com/>.
- [15] “Official website for crispy forms,” <https://django-crispy-forms.readthedocs.io/en/latest/>.
- [16] “Official website for git,” <https://git-scm.com/>.
- [17] “Official website for vs code,” <https://code.visualstudio.com/>.
- [18] Glassdoor. (2025) Sueldo de desarrollador junior en españa. [En línea]. Disponible en: https://www.glassdoor.es/Sueldos/espa%C3%B1a-desarrollador-junior-sueldo-SRCH_IL.0%2C6_IN219_KO7%2C27.htm
- [19] Jobted España. (2025) Sueldo de profesor de universidad. [En línea]. Disponible en: <https://www.jobted.es/salario/profesor-universidad>
- [20] OpenAI. Precio de los modelos de openai. [En línea]. Disponible en: <https://platform.openai.com/docs/pricing>
- [21] RCSB Protein Data Bank, “PDB File Download Services,” <https://files.rcsb.org/>, servicio de descarga de archivos del RCSB PDB (consultado del sitio).