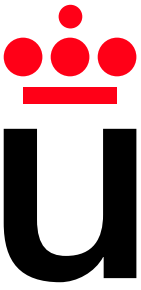

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería Informática

Curso 2022-2023

Trabajo Fin de Grado

APLICACIÓN WEB BASADA EN EL FRAMEWORK DJANGO PARA LA ENSEÑANZA DE LA RECURSIVIDAD A TRAVÉS DE DIAGRAMAS

Autor: Alejandro Valtierra Baños
Tutor: Manuel Rubio Sánchez

Agradecimientos

En primer lugar agradecer a mi tutor del TFG, Manuel Rubio Sánchez, por el apoyo y ayuda prestados, así como la atención al desarrollo del proyecto. También agradecer a todos los profesores de la ETSII, por todos los conocimientos que me han transmitido a lo largo de la carrera que ha sido indispensable para mi desarrollo personal y la realización de este trabajo.

En el plano personal, agradecer a mi familia por todos los ánimos y apoyo a lo largo de todo este tiempo. También a mis amigos por la constante dedicación tanto en buenos como en malos momentos. Finalmente a mis compañeros de la universidad, entre los que he hecho grandes amigos y han hecho que disfrute mucho de este periodo.

Abstract

Recursion is a very powerful and widely used tool in programming. It is also one of the first problems that students generally face due to the change from an iterative approach.

One of the most common methodologies used to teach is the utilization of recursion trees, even so, problems and difficulties in learning continue to be detected. The declarative method has been chosen as an approach for learning, in which students think by exposing an idea instead of carrying out the process that a device would do.

Therefore, the development of a web application is proposed, in which this metodology is developed through informed examples using the selection of options for the generation of a recursive code. The application will also explain the basics of recursion, gamification elements such as obtaining medals have been introduced to encourage the use ande develoment of recursion by using the application.

For the implementation of the application, the development model of incremental software has been used, through this model varios functionalities have been implemented and a deployment has been finally made to verify the aplication would have the capability to be working.

Resumen

La recursividad es una herramienta muy potente de resolución de problemas y ampliamente utilizada en la programación. Los alumnos en los primeros cursos suelen encontrar algunas dificultades debido a que generalmente el enfoque iterativo se explica antes.

Es habitual enseñar recursividad mediante la visualización de árboles de recursión, se ha comprobado en muchas seminarios y trabajos que el centrarse en esta metodología, se piensa a un mayor bajo nivel de lo que en realidad hace falta, alejándose por tanto del paradigma declarativo. En este trabajo se ha optado por desarrollar el método declarativo, en el que los alumnos piensan mediante la exposición de una idea a través de un gráfico declarativo en vez de realizar el proceso que haría un dispositivo.

Se propone por tanto el desarrollo de una aplicación web sobre la que se desarrolle esta metodología a través de ejercicios paso a paso, en los que a través de la selección de opciones, se irá informando al usuario para la generación de un código recursivo. Aprovechando el gráfico en cuatro partes como elemento primario para la enseñanza. La aplicación también explicará las bases de la recursividad y se han introducido elementos de gamificación como la obtención de medallas para fomentar el uso y desarrollo de la recursividad en la aplicación.

Para la implementación de la aplicación se ha utilizado el modelo incremental de desarrollo de *software*, a través de este modelo se han implementado diversas funcionalidades y finalmente se ha hecho un despliegue para la comprobación del funcionamiento y la capacidad de utilización del trabajo.

Palabras clave:

- Recursividad
- Aprendizaje de la recursividad
- Python
- JavaScript
- Django
- Html

- CSS
- Programación declarativa
- Diagramas declarativos

Índice de contenidos

Índice de figuras

Índice de códigos

1 Introducción

En este capítulo se expondrá el tema al que se va a dedicar el trabajo y la estructura del documento.

1.1. Introducción

La recursividad es un pilar fundamental en la programación, hace referencia a una estructura que se contiene a si misma o que se vuelve a llamar en el código interno. Es ampliamente utilizada en funciones para poder resolver problemas. La recursividad consta habitualmente de una serie de pasos fijos. Primero, caso o casos base, es una estructura condicional que se asegura de que en algún momento se pare la recursión, son los casos en los que la resolución del problema es trivial. A continuación se elige una descomposición que reduce el tamaño del problema a un caso más sencillo; con este subproblema es con el que se hace la llamada recursiva para resolverlo. Finalmente, para resolver el problema, se compone el resultado de la llamada recursiva con el valor de la descomposición, utilizando una función lógico matemática todo lo compleja que se necesite [1].

En la recursividad son importantes los siguientes conceptos, que son con los que se compondrá el código. El caso base es imprescindible para finalizar las llamadas recursivas sin generar una recursión infinita, son casos en los que la solución al problema es trivial. La descomposición se utiliza para el resto de casos, aquellos en los que la solución es más compleja, se realiza por tanto una descomposición que reducirá el tamaño del problema a uno más pequeño con una solución un poco más sencilla; al descomponer un problema nos quedará un

elemento o elementos que utilizaremos para componer la solución. La llamada recursiva se hace con la descomposición obtenida, a la que se llama subproblema, entonces existen dos posibles caminos, la generación de un árbol de llamadas o el denominado "salto de fe", en este último se asume que el resultado va a ser una solución válida y correcta de la llamada [1][2].

Lo primero que aprenden los alumnos son métodos iterativos para la resolución de problemas con tareas repetitivas, esto lleva a los alumnos a intentar utilizar la metodología iterativa frente a la recursiva. Además, con frecuencia las soluciones recursivas que utilizan son como un "*goto*", es decir, un salto a una línea específica del código, para recrear una función iterativa volviendo al comienzo del "bucle", en vez de reducir el tamaño de los parámetros de la llamada. Otro de los grandes problemas suele ser que no se enseña bien desde un principio, centrándose en la demostración por árboles en vez de en la idea de recursividad [3].

1.2. Estructura del documento

Este documento contiene cuatro capítulos principales y dos apéndices. A continuación se explica en que consiste cada uno.

1. Objetivos del TFG: Es el segundo capítulo, se expondrán los objetivos e hitos que se esperan alcanzar.

2. Estado del arte: Es el tercer capítulo, se desarrollan los antecedentes a este trabajo, tanto las metodologías como aplicaciones o herramientas ya utilizadas para la enseñanza de la recursividad. También se explicará el marco tecnológico utilizado para el desarrollo del proyecto.

3. Descripción informática: Es el cuarto capítulo, se recoge la selección de la metodología empleada, la especificación de requisitos, se expone el desarrollo según la metodología elegida en el anterior capítulo y finalmente el despliegue de la aplicación.

4. Conclusiones y trabajos futuros: Es el quinto y último capítulo, se detallan tanto las conclusiones del ejercicio así como académicas, también se proponen ideas y consideraciones para ampliar y mejorar la aplicación en un futuro.

5. Apéndice A: En el apéndice A se explican los códigos de los ejercicios implementados en la aplicación.

6. Apéndice B: En el apéndice B se añaden el código de los ficheros de configuración necesario para **Apache** y para **MySql**

2

Objetivos del TFG

El objetivo principal de este trabajo es el desarrollo de una aplicación que ilustre el proceso de generación de casos recursivos empleando los diagramas del libro *introduction to recursive programming*[2]. También se quiere adaptar la aplicación para su uso en la universidad en las asignaturas que requieran el aprendizaje de la recursividad. Gracias a lo cual, en un futuro se permitirá realizar una evaluación del valor pedagógico de esta aplicación.

Otros de los objetivos principales sería la alta disponibilidad de la aplicación y permitir que los alumnos puedan acceder fácilmente a esta herramienta, para ello se propondrá el desarrollo de una aplicación web. De esta forma se facilita el uso por parte de los alumnos dado que los navegadores web se encuentran en la práctica totalidad de los dispositivos de uso personal o público.

Además, sería interesante poder cumplir algunos subobjetivos prácticos en lo que a la aplicación respecta. Los cuales son:

- Inclusión de múltiples ejercicios ofreciendo a los alumnos una selección variada de ejemplos con distintos tipos de entradas y objetivos.
- Sistema de preguntas respuestas para profundizar en el desarrollo declarativo mediante la interacción informada.
- Proposición de código para el desarrollo de ejercicios disponible a los alumnos.
- Utilización de la gamificación con objetivo de mejorar el interés de los alumnos para la adquisición del conocimiento.

- Diseño de un sistema de progreso basado en hitos para la retroalimentación positiva del alumno.
- Introducción a los pasos necesarios para la implementación de un algoritmo recursivo.

3

Estado del arte

Este capítulo incluye un pequeño estudio sobre las dos corrientes principales de enseñar la recursividad, las herramientas desarrolladas anteriormente y el marco tecnológico sobre el que se ha desarrollado este proyecto.

3.1. Estudio sobre enseñanza de la recursividad

Un enfoque habitual para enseñar recursividad es a través de los árboles de llamada, tal y como desarrollaría una computadora. Sin embargo, cuando se sigue un enfoque declarativo, es decir, la formulación con palabras de los ejercicios siguiendo una estructura predefinida, se ha observado que los alumnos conseguían mejores resultados a la hora de resolver ejercicios de implementación de código, teniendo más éxito y mejor desempeño en la resolución de los problemas recursivos; además, los que tenían errores, en general eran de carácter muy leve[3]. Es por ello que en este trabajo se ha seguido la metodología declarativa que se explicará en detalle más adelante.

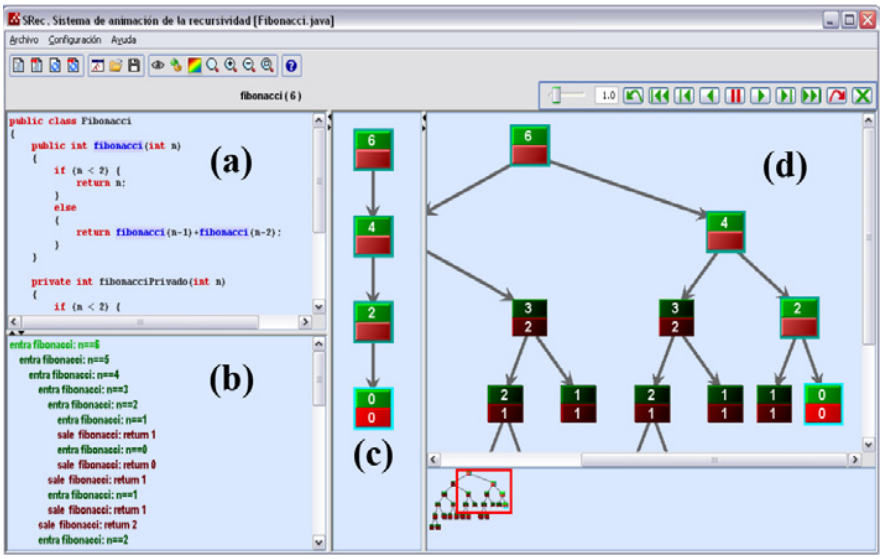

Figura 3.1: Vista de la ventana principal de Srec.

3.2. Herramientas para la enseñanza de la recursividad

3.2.1. Srec

Es un programa hecho originariamente por Antonio Pérez Carrasco para la visualización de árboles de recursión. En el programa, como se puede ver en la figura 3.1, se contemplan una pila de control para ver los datos en cada llamada (c), un modo texto para ver la sucesión de llamadas hasta el momento (b), la vista del árbol de llamadas (d) y una última en la que se ve el código que se está ejecutando (a). Tiene múltiples opciones tanto de exportación e importación como la parada en *breakpoints*. También tiene algoritmos precargados para no necesitar ni crear el código [4].

3.2.2. VisBack

Es un aplicación que también puede usarse como *plugin* para Eclipse con el que se pueden visualizar algoritmos recursivos. Desarrollado en Java por Jesús Francisco Pérez Mena, es una aplicación en la que puedes cargar algoritmos y proporcionar unas entradas para la función, tiene una vista en la que está embebido el código, en la que además te indica que instrucción se está ejecutando; otra de las vistas sirve para ver una representación del árbol de llamadas, y una última en la que se ve el nodo que está siendo ejecutado [5].

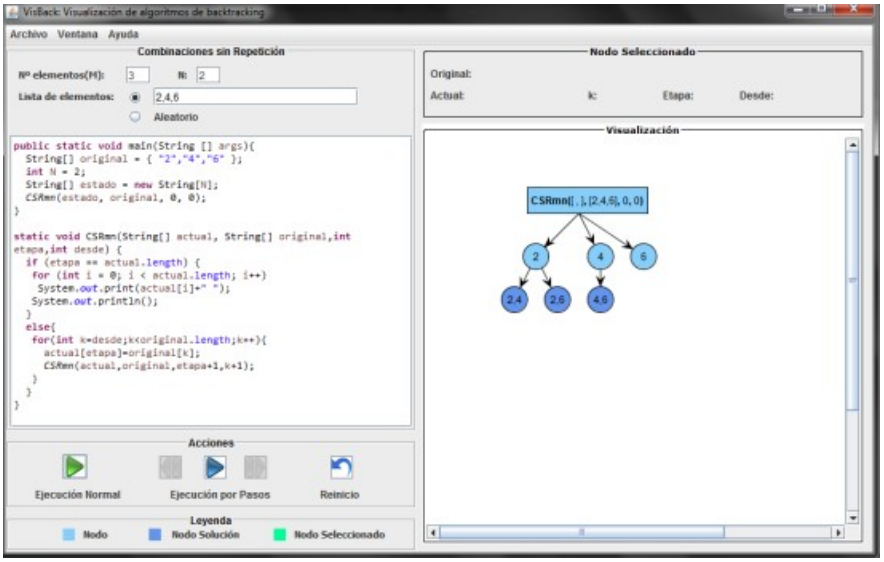

Figura 3.2: Vista de la ventana principal de VisBack.

3.2.3. ChiQat

Es una aplicación desarrollada por Davide Fossati, es más didáctica que las anteriores dado que busca enseñar recursividad a través de ejemplos; la vista principal es un árbol de llamadas con los valores de cada nodo y los parámetros utilizados; también tiene una vista en la que se describe el problema y en el que se hacen unas preguntas básicas del código; en la última vista se propone una solución en código del problema.[6].

3.2.4. Conclusión a las alternativas

Prácticamente todas las aplicaciones para la enseñanza de la recursividad utilizan diagramas de expansión de árboles para enseñar conceptos o el desarrollo del problema. En general no enseñan el "salto de fe"de hacer una llamada recursiva; esto es asumir que el subproblema ya está resuelto al hacer la llamada recursiva, en realidad no se resuelve solo por hacer la llamada; pero si damos por sentado que lo va a hacer, solo necesitamos componer la solución a partir del resultado de la llamada y el valor de la descomposición que hayamos elegido. Siguiendo estas premisas se propone en este trabajo una aplicación web para la enseñanza de recursividad sin utilizar árboles, solo la composición de código a partir de las nociones: caso base, descomposición, llamada recursiva y solución del problema. Con ello mediante ejercicios guiados para la iniciación en los que alumnos podrán elegir entre diversas opciones, se irá componiendo un código según vayan eligiendo. Paso a paso se explica la motivación de las elecciones correctas y el de las incorrectas del alumno. Se han añadido también elementos de gamificación para

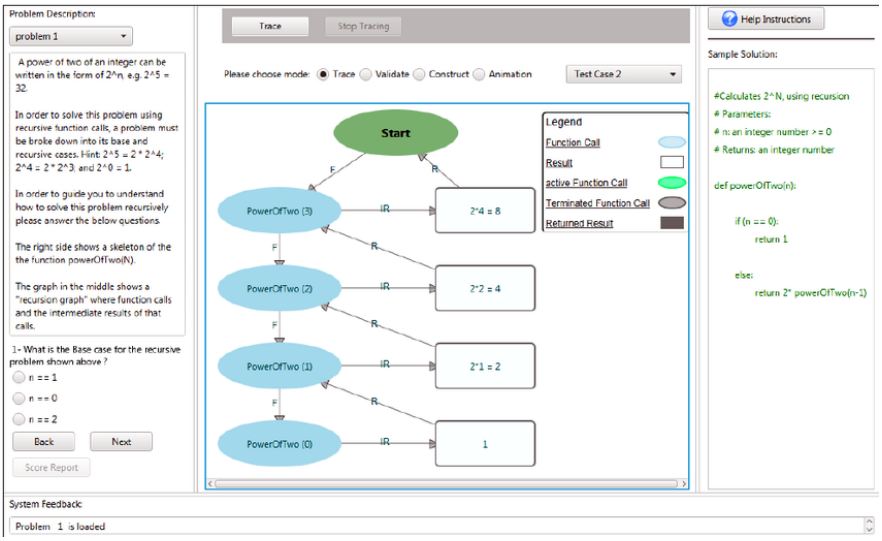

Figura 3.3: Vista de la ventana principal de ChiQat.

el refuerzo positivo de los alumnos e intentar aumentar las ganas de acceder y usar la aplicación.

3.3. Marco tecnológico

Primero se indicarán todas las herramientas o soluciones que se han empleado en el desarrollo del trabajo. Siendo las siguientes:

- **Python**: Es un lenguaje de alto nivel interpretado con licencia de código abierto. Apareció en 1991 y se ha convertido en uno de los lenguajes más utilizados en el mundo. Sigue en constante desarrollo y cuenta con miles de paquetes con muchísimas funcionalidades[7].

- **Django**: Es un *framework* de desarrollo web de los más utilizados hoy en día junto a **Flask** para el lenguaje **Python**. Está en constante revisión y desarrollo en el momento de escribir este trabajo. **Django** se preocupa más de la seguridad de base que **Flask**, además es muy escalable y permite fácilmente la conexión con bases de datos. También está pensado para desarrollar rápidamente aplicaciones. Generalmente es muy utilizado para páginas web especificas como el portal que se va a realizar. En cambio **Flask** sobresale si se van a realizar muchos microservicios especializados o experimentación con bibliotecas y arquitecturas, siendo algo más tedioso el desarrollo. Dada la focalización del proyecto en la enseñanza así como la buena seguridad base y la compatibilidad con bases de datos se va a utilizar **Django**[8].

- **Visual Studio Code**: Es un IDE (*Integrated Development Enviroment*) para el desarrollo de proyectos en múltiples lenguajes, esto es gracias a la alta personalización y *plugins* disponibles para el entorno. También permite utilizar la consola del sistema operativo de base para poder lanzar comandos desde el propio entorno. La motivación principal para utilizarlo es la compatibilidad con **Python**, **Django** y **GitHub** mediante *plugins* así como un entorno muy fácil de cargar[9].

- **MySQL**: Es una base de datos de tipo relacional y una de las más utilizadas en el mundo. Tiene una licencia dual, incorporando una con un método de pago y otra de código abierto en la versión *community*, el mantenimiento lo hace principalmente **Oracle Corporation**. Se ha escogido esta base de datos dado que es una de las soportadas directamente por **Django** y al ser relacional permite generar las relaciones deseadas entre usuarios y ejercicios. Además de que permite más escalabilidad de cara al futuro que la **SQLite** que viene por defecto en **Django**[10].

- **GitHub**: Es una herramienta de control de versiones a nivel mundial, en la que se crean repositorios manejados y se comparten proyectos por la comunidad. Es uno de los principales servicios de código abierto en el mundo y es ampliamente utilizado para llevar el sistema de versiones de muchos proyectos[11].

- **JavaScript**, **HTML** y **CSS**: Son algunas de las herramientas más comunes utilizadas para el desarrollo *frontend*. **JavaScript** es un lenguaje de programación basado en prototipos, aunque se puede realizar programación orientada a objetos, imperativa y declarativa[12]. **HTML** es el lenguaje básico para las web, define el significado y la estructura de la misma. **CSS** es un lenguaje de estilos utilizado para la representación de **HTML** describiendo como se debe ver el componente[13].

4

Descripción informática

En este capítulo se va a explicar tanto la elección de la metodología de *software*, como recoger los requisitos, explicar el desarrollo de la aplicación a través de la metodología y para acabar lo necesario para el despliegue.

4.1. Metodología empleada

El modelo incremental es un planteamiento tradicional del desarrollo de software basado en un enfoque modular, generando nueva funcionalidad con cada entrega. Estas son iteraciones del código basadas en una primera habitualmente conocida como núcleo, sobre la que se van forjando el resto[14].

Cabe destacar que muchas de las metodologías ágiles beben directamente de este modelo, por ejemplo el modelo *extreme*[15] o *devops*[16][17], algunos de los modelos más utilizados hoy en día. Siendo una buena base centrada en la entrega y revisión continua de código[14].

Algunas de las ventajas que otorga trabajar con este modelo son[14]:

- En primera instancia reduce riesgo de que algunas de las funciones no sean relevantes para el proyecto y se puedan descartar rápidamente. Permitiendo a su vez enfocarse en las más interesantes.
- No hay que esperar al final para poder ver la funcionalidad, si no que se va desarrollando progresivamente.

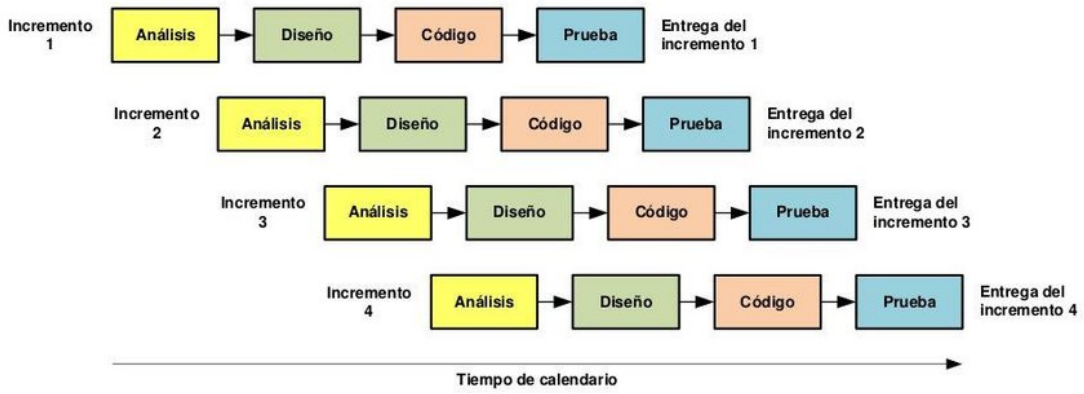

Figura 4.1: Ejemplo del modelo incremental en el que se ven los pasos en una escala temporal.

- Lo más importante se desarrolla primero por tanto hay más pruebas y se disminuye el riesgo de fallos.

El modelo tiene una serie de etapas por cada iteración[14]:

- Análisis: Primero se analizan los requisitos y se elabora un esquema o modelo simple del desarrollo que se va a realizar.
- Diseño: A continuación se elabora un modelo más complejo, con mayor grado de detalle, de los requisitos para plantearlo como objetivo.
- Programación: Después se comienza a desarrollar el código para cumplir los requisitos a través de los modelos ya realizados en las fases anteriores.
- Pruebas: Finalmente se procede a probar las funcionalidades generadas y revisar que se hayan cumplido los objetivos.

Como se ha explicado anteriormente se ha escogido el modelo incremental como la metodología a utilizar para este proyecto. Dado que se adecua casi por completo a los requisitos planteados en los que se pueden ir corrigiendo y añadiendo funcionalidades desde el principio intentando adaptar la aplicación y mejorando con cada iteración[14].

4.2. Especificación de requisitos

Para comenzar con la metodología incremental y el desarrollo de la aplicación se deben especificar los requisitos que tiene que cumplir el proyecto. Para ello se

usará como base algunos de los puntos explicados en el capítulo de objetivos 2 que presentaremos a continuación.

En primer lugar, a través de reuniones con el profesor se han establecido algunos de los requisitos fundamentales de la aplicación, así como cuestiones de diseño y presentación de la información; también se ha tenido en cuenta la opinión de compañeros y amigos tanto de grado como de otros campos para perfeccionar algunos puntos de diseño, usabilidad y presentación de la información.

Como segundo punto importante, el estudio de alternativas 3.2 nos ha permitido obtener como información relevante que muchas aplicaciones están basadas en los árboles de llamadas. Por otra parte teniendo en cuenta algunas de las ideas fundamentales dadas en la descripción del problema **??**, tales como las distintas metodologías empleadas para el aprendizaje de la recursividad. Con la mezcla de ambos puntos se va a plantear una aplicación basada en una metodología declarativa, es decir explicar el problema y ayudar a que los alumnos hagan un planteamiento teórico mediante preguntas, respuestas y explicaciones elaboradas a través de la gráfica de la metodología declarativa. Simultáneamente se va a desarrollar el código en base a las elecciones del alumno[1][2].

Finalmente, se ha decidido que la aplicación sea un portal web, dado que permite un acceso mucho más fácil para los estudiantes y profesores que crear una aplicación de escritorio, dado que es más fácil acceder a través de un explorador web que desarrollos que puedan limitar el número de dispositivos en los que ejecutar la aplicación, ya sea por falta de requisitos técnicos, diversidad de soluciones o sistemas operativos o falta de espacio en disco.

Utilizando estás especificaciones se procederá a indicar los requisitos tanto funcionales como no funcionales que van a regir el desarrollo de la aplicación.

4.2.1. Requisitos funcionales

Los requisitos funcionales son la declaración de como debe comportarse un sistema. Serán las características apreciables por los usuarios mediante la interacción con la aplicación a través de las funcionalidades implementadas.

Los requisitos funcionales son los siguientes:

- Selección de ejercicios: Se ofrecerán múltiples problemas con distintos tipos de valores de entradas y descomposiciones de forma que los usuarios tengan variedad para elegir.
- Visualización de los ejercicios realizados: Se permitirá que los usuarios puedan llevar un progreso de los ejercicios que realizan y los que les falta por hacer.

- Generación de código: Se ofrecerá un código de ejemplo funcional en **Python** que los alumnos podrán copiar o utilizar si así lo desean y que demuestre el avance en cada paso del ejercicio.
- Selección de parámetros: Se podrá introducir parámetros a los ejercicios con objetivo de que los alumnos puedan ver las diferencias de ejecución entre distintas entradas.
- Sistema de preguntas y respuestas: Durante la resolución de los ejercicios se utilizará elementos de tutelaje para hacer preguntas al alumno y la explicación de escoger una opción u otra.
- Explicación global: Se permitirá un acceso a un recurso que permita explicar el proceso de la recursividad general.
- Gamificación: Se dispondrá de elementos de gamificación para incentivar al alumno a seguir utilizando la aplicación y para que tengan más interés en aprender.Se utilizarán elementos como el progreso, lo cual se ha demostrado que sirve de incentivo y como elemento motivador en la enseñanza para los alumnos[18].
- Sistema de acceso: Para poder llevar el progreso y los elementos de gamificación se utilizará un sistema de acceso con credenciales.

4.2.2. Requisitos no funcionales

Los requisitos no funcionales son aquellas restricciones a las que se ve impuesta la aplicación, no suelen ser apreciables a simple vista, aunque son consecuencia directa de la calidad final de la aplicación. Por lo tanto son los descriptores del funcionamiento final del sistema.

- Seguridad: Para una mayor seguridad las contraseñas en la base de datos deberían estar cifradas y se debería de proteger de ataques como los de CSRF (*Cross-site request forgery*).
- Fiabilidad: Sería indispensable que los usuarios del sistema puedan recuperar sus credenciales o cambiar las mismas.
- Interfaz gráfica: Se buscará ofrecer una interfaz amigable y sencilla en la que los elementos sean fáciles de diferenciar y se puedan reconocer de un vistazo.
- Lenguaje de desarrollo: Se utilizará **Python** para la realización del portal, más concretamente el *framework* de **Django**. El código de los ejercicios estará formulado en **Python**, dado que es el lenguaje utilizado en la asignatura de Diseño y Análisis de Algoritmos.

- Base de datos: Para guardar el progreso y de los usuarios se utilizará una base de datos relacional, en este caso se ha usado **MySQL**.

4.3. Modelo incremental para el desarrollo

A continuación se procederá a explicar los distintos incrementos que se han realizado. Se van a explicar en cada apartado las cuatro fases, primero el análisis, a continuación se hará una propuesta de diseño, en tercer lugar se procederá a implementar y finalmente se harán las pruebas sobre la funcionalidad realizada.

4.3.1. Primer incremento: Base

El primer incremento en la metodología siempre es la base sobre la que se asentarán el resto de las funcionalidades. En este primer incremento por tanto se planteará la estructura base de la aplicación final.

Dado que se va a utilizar el *framework* de **Django** se creará primero la aplicación. Está se compone de un directorio principal en el que se especifican los *settings* del proyecto, entre ellos están el directorio en el que está localizado el proyecto; las aplicaciones que lo componen, las cuales ampliaremos para las diversas funcionalidades, el *middleware* que se ocupa de cuestiones de seguridad, autentificación, sesiones o las señales de la aplicación; también la configuración de directorios de las plantillas **HTML**; el tipo de aplicación, en este caso utilizará wsgi para la presentación de los **HTML**, no se requiere de métodos asíncronos para generar plantillas o entregar datos en otro momento. También guarda las configuraciones de las bases de datos; la validación de contraseñas; la internacionalización, como el idioma o el horario; el control de los ficheros estáticos, como las imágenes, **JavaScript** o **CSS**. Se va a generar también la aplicación *core* en la que se implementará la base de la aplicación.

Una vez detallada la arquitectura inicial del proyecto procederemos a analizar, mediante un diagrama de casos de uso, las funcionalidades a realizar para la base de la aplicación.

Como se muestra en el diagrama de casos de uso 4.2, las funcionalidades principales a implementar son:

- Visualización de descripciones: Esta funcionalidad afectará a la página principal, el usuario podrá a acceder a unas descripciones de los distintos pasos de la recursividad que irán apareciendo dinámicamente con objetivo de no alarmar a los usuarios con una gran cantidad de texto inicial.

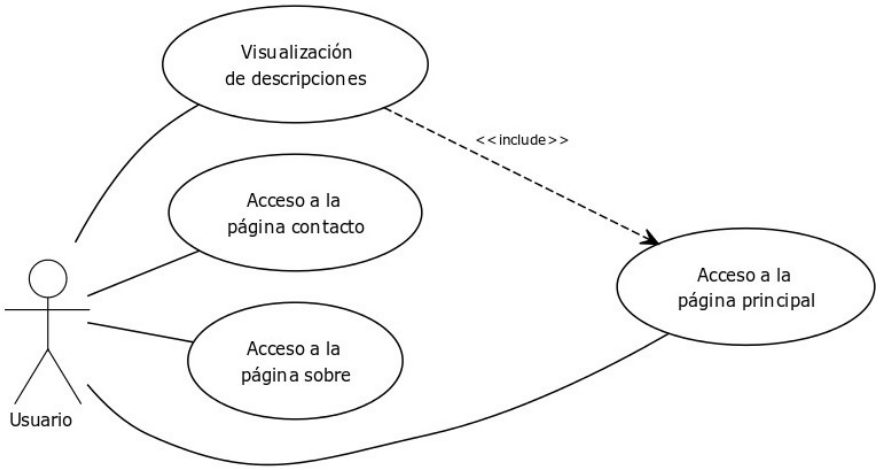

Figura 4.2: Diagrama de casos de uso de la base de la aplicación.

- Acceso a página principal: Esta funcionalidad permitirá en cualquier momento o desde cualquier otra dirección interna a la aplicación para alcanzar la página principal.
- Contacto: Esta funcionalidad será una pantalla con detalles de como ponerse en contacto con un administrador.
- Sobre: Esta funcionalidad tendrá como objetivo crear una página con algunos datos importantes sobre el uso de la aplicación.

Una vez explicados los casos de uso se habrá terminado la etapa de análisis y se comenzará con la de diseño. Inicialmente se procederá a realizar un boceto inicial del aspecto que podría tener la página principal. Estos bocetos pueden resultar muy interesantes en las etapas tempranas del desarrollo de las funcionalidades. Permiten en la mayoría de casos que tanto usuarios finales como desarrolladores tengan una buena idea del producto final; también ofrecen la capacidad de adelantarse a posibles cambios sobre el programa o funcionalidades no contempladas, su identificación antes de comenzar con el desarrollo genera la posibilidad de evitar parches futuros o correcciones adicionales ofreciendo además una mayor interacción, compromiso y nivel de involucramiento para con los usuarios.

En el boceto 4.3 podemos observar una barra de navegación superior, escalable al resto de funcionalidades tanto presentes como futuras, en esta barra encontraremos actualmente a la izquierda una figura de una casa indicativa de regreso al inicio. Por otra parte una barra inferior de navegación, denominada *footer*, en la cual se añadirán un logotipo y una sección con enlaces a la página de contacto y la de sobre. Finalmente, el espacio principal de la página se dedica a un elemento gráfico estilo diagrama; se comprende de 4 cuadrados en las esquinas unidos con flechas indicando los pasos de la recursividad. Finalmente como anotación se in-

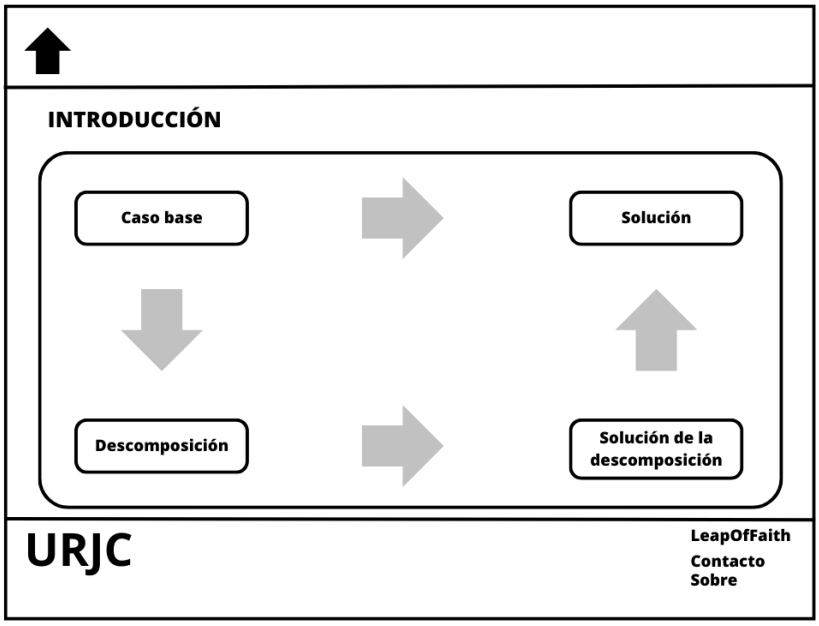

Figura 4.3: Boceto de la página principal.

dica que al interactuar con cada uno de estos cuadrados aparecerá un elemento estilo bocadillo en medio de la pantalla con la información descriptiva del paso.

Terminada ya la etapa de diseño se procederá con la etapa de implementación de código.

Durante esta etapa se crearán tres *views*, estas son funciones encargadas de componer la plantilla **HTML** con unos datos pasados como contexto, aunque para estas plantillas sencillas no nos hace falta pasar ningún contexto adicional al *middleware* encargado componer los documentos **HTML**. Para llamar a las *views* se utiliza un iterable de urls, de esta forma en el momento en el que se haga una petición, **Django**, compara internamente en el iterable y llama a la *view* responsable, la cual a su vez genera la plantilla **HTML** final y la devuelve al usuario.

Como las *views* necesitan plantillas, se han generado cuatro plantillas diferentes. Una de ellas es la plantilla base sobre la que se van a componer todas, con esto conseguimos no tener que reescribir constantemente elementos ya escritos para el resto de documentos **HTML**. Esta plantilla tendrá los encabezados y un cuerpo en el que se van a añadir las barras de navegación superior e inferior; entre ambas se ha creado una etiqueta de contenido la cual juntará con las plantillas solicitadas y creará el documento **HTML** final. También se ha agregado otro bloque de contenido en el encabezado para poder añadir documentos **CSS** distintos según la plantilla que se va a renderizar en el explorador y poder dividir en distintos módulos los documentos. Con respecto a las plantillas específicas, se ha utilizado

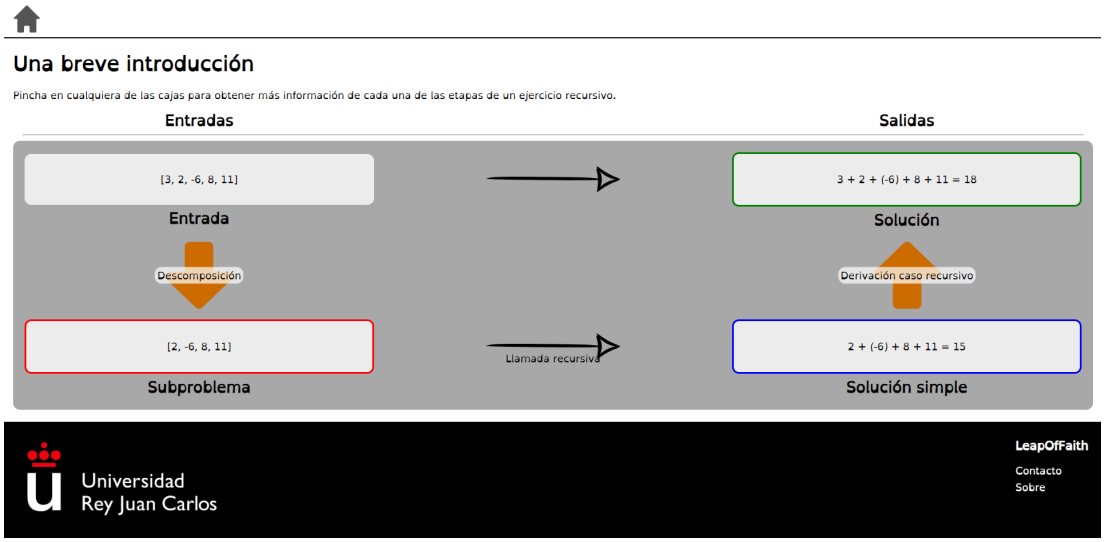

Figura 4.4: Interfaz final página principal.

para la que va a ser la página principal un gráfico con código **JavaScript** para que en caso de que se haga clic con el cursor en los cuadrados aparezca el bocata asociado. Para la plantilla de contacto y sobre se han utilizado plantillas similares con reutilización de hojas de estilos para mejorar la legibilidad y mantener una coherencia para los usuarios de la aplicación.

Como se puede apreciar en la interfaz de la página principal 4.4 se ha optado por usar un diagrama con los pasos esquematizados de la recursividad. El tercio izquierdo hace referencia a las entradas, el cuadro superior está dedicado a los parámetros con los que llamamos a la función recursiva en primera instancia; el cuadro inferior es para los parámetros que se han reducido mediante la descomposición. El tercio derecho representa las salidas esperadas de las llamadas, el cuadro superior es el resultado objetivo al llamar a la función, mientras el cuadro inferior es el resultado de la llamada recursiva. Entre los cuadros se han puesto flechas verticales para indicar la dirección y los dos pasos más importantes de la recursividad, en el tercio izquierdo la flecha indica el paso de la descomposición del problema, reduciendo el tamaño de un parámetro; la flecha del tercio derecho expone el paso de la derivación del caso recursivo, esto es utilizando el resultado parcial de la llamada recursiva con el objeto de la descomposición, componer mediante una función lógico matemática más o menos compleja para obtener el resultado final. Las flechas horizontales hacen referencia a los resultados de aplicar la llamada recursiva al abstraerte del resto de pasos.

Se van a crear plantillas personalizadas de errores, los documentos por defecto que genera **Django** son funcionales pero rompen con la estética de la aplicación. Los cuatro errores a los que se va a hacer referencia son el error 400 (*Bad Request*), el error 403 (*Forbidden*), el error 404 (*Not Found*) y el error 500 (*Internal Server Error*). Para estructurar estás plantillas también se tienen que configurar unas

vistas, una por cada error, aunque en contraposición para las el archivo de urls no lo añadiremos dentro del iterable si no que se adjuntan como variables a las vistas. Vamos a generar dos plantillas para estas vistas, una común para los errores tres errores 400 dado que todos pueden añadir un comentario o contexto con el que poder informar al usuario de las posibles causas, será una plantilla personalizada con el tipo de error y el motivo; por otra parte tenemos el error 500 indicando que el servidor ha fallado durante la ejecución de instrucciones, además notifica a los correos de los usuarios administradores cuando se produce uno de estos errores con la traza del fallo.

De cara a la realización de pruebas se valoró la utilización de la librería de pruebas estándar unittest. Es un módulo de **Python** con soporte por **Django** que permite la realización de pruebas unitarias. Al final se ha descartado su uso porque la mayoría de las interacciones realizadas por el usuario se resuelven por una combinación de **CSS** y **JavaScript**. En base a que gran parte de la funcionalidad es a través del *frontend* y el código tan modular se ha optado por realizar *User testing*, un modelo de pruebas en el cual los propios usuarios realizan las pruebas y proporcionan retroalimentación a los desarrolladores, los principios de este método es dejar libertad a los usuarios y no guiarles en el uso de la aplicación. Las pruebas han sido realizadas por cuatro usuarios pertenecientes a distintos campos educativos; la primera una estudiante doctoral bioquímica con algunos conocimientos de programación, el segundo un compañero de carrera, el tercero un estudiante de ingeniería civil con solo conocimiento matemático, el último un estudiante de IT con conocimiento básico en programación.

En esta primera iteración las pruebas han sido relativamente cortas. Los principales puntos modificados son en base a los comentarios realizados:

- En primer lugar se redujo el tamaño de la introducción por consejo general de todos los usuarios, indicando que era muy larga y explicaba demasiado perdiendo atracción al leerlo, se intercambió por bocadillos que aparecen al tocar en los recuadros de entrada, subproblema, solución simple y solución, tras lo cual recibió comentarios mucho más positivos.

- En segundo lugar se ha cambiado el color de los bocadillos por una borde de color dado que confundía a los usuarios, estos bordes se utilizarán principalmente en la vista de ejercicios del segundo incremento.

- En tercer lugar y por recomendación del profesor y apreciado por el compañero de la universidad se intercambiaron las flechas horizontales, esto es en base a que los pasos que se han de realizar por un usuario son la descomposición y la derivación del caso recursivo. La llamada recursiva se hace automáticamente.

- Finalmente se sugirió utilizar una letra para disléxicos “OpenDislexic”. Tras un estudio realizado, se ha comprobado que no existen letras para personas

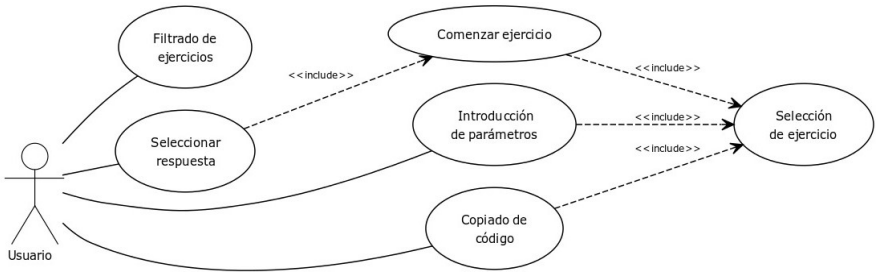

Figura 4.5: Diagrama de casos de uso de los ejercicios de la aplicación.

con esta dificultad, aunque algunas ayudan más que otras al enfoque o por lo menos reducen ligeramente los tiempos requeridos para la lectura [19] [20].

Una vez hemos completado el primer incremento tenemos la estructura principal sobre la que se va a crear el resto de funcionalidades.

4.3.2. Segundo incremento: Ejercicios

El segundo incremento es una de las funcionalidades más importantes para la aplicación. Se servirá de ejemplos para ayudar a los alumnos con casos prácticos, de esta forma no solo será un desarrollo teórico, si no que se dedicarán a realizar ejercicios con los que mejorar su comprensión y desarrollo de la materia.

Primero es necesario identificar los requisitos, como en el primer incremento, que se espera desarrollar durante esta fase. Para el análisis se procederá a utilizar un diagrama de casos de uso como en el primer incremento.

Como se puede apreciar en el diagrama 4.5, las funcionalidades a implementar serán las siguientes:

- Selección de ejercicio: Esta funcionalidad tendrá como objetivo poder elegir entre múltiples ejercicios para mostrar un mayor número de posibilidades y tipos de entradas como para poder aprender adecuadamente los conceptos.
- Filtrado de ejercicios: Para facilitar la navegación entre ejercicios se implementará un filtro que permita hacer selecciones o buscar por nombre a los usuarios.
- Introducción parámetros: Un requisito indispensable sería que los usuarios puedan elegir los parámetros de llamada para que puedan observar el desarrollo que prefieran en el ejercicio en vez de utilizar unos parámetros determinados.

- Copiado código: El código generado debería poder copiarse sin necesidad de que los usuarios tengan que seleccionarlo.
- Diálogo progresivo: Con objetivo de facilitar el aprendizaje declarativo sobre el desarrollo del árbol de recursión, se utilizará una guía interactiva en la que se hagan preguntas al usuario y pueda seleccionar las respuestas devolviendo retroalimentación de la elección hasta la finalización del ejercicio.

Los ejercicios que se implementarán en para esta aplicación son los siguientes:

- Potencia.
- Sumar los elementos de una lista
- Misma cadena
- Decimal a binario
- Dígito en un número
- Palíndromo
- Decimal a base n
- Factorial
- Mayor de los elementos de una lista
- Invertir una cadena
- Ordenar una lista

Una vez que se ha terminado la etapa de análisis de requisitos continuaremos con la etapa de diseño. En este caso se harán dos bocetos diferentes, dado que se planea separar las funcionalidades de selección y filtrado de las demás; esto es debido a que se desea realizar una implementación más sencilla y cargar menos la visualización, pudiendo así solo enfocarse en el ejercicio planteado.

Como se puede observar en el boceto del buscador 4.6, es una página sencilla en la cual habrá un cuadro en la parte superior en el que elegir los filtros y los parámetros de búsqueda. Como elementos principales tendrá un campo introductorio de texto para poder buscar por nombre, un botón para iniciar la búsqueda y otro para poder restablecer filtros. En la parte inferior se dispondrá una tabla en la que aparecerán los ejercicios con enlaces a los mismos.

Tal y como se puede ver en el boceto de la página de ejercicios 4.7, se dispondrá de cuatro cuadros para distintas funcionalidades. El cuadro superior se utilizará

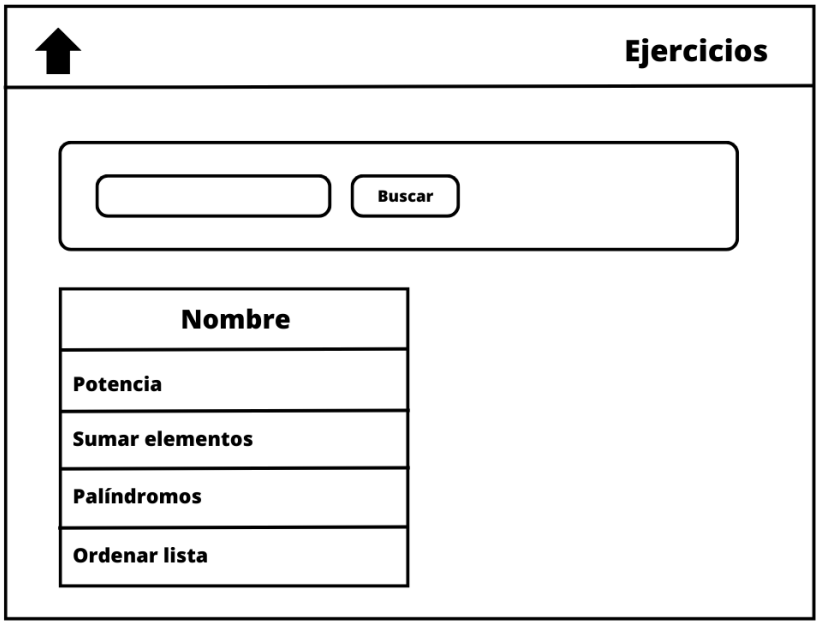

Figura 4.6: Boceto del buscador de ejercicios.

para añadir una descripción con ejemplos de los parámetros y la solución del ejercicio. Inmediatamente debajo del cuadro con la descripción pondremos dos, uno a la izquierda que ocupe la mitad y otro a la derecha que ocupe el resto del espacio. El cuadro de la izquierda contendrá un diagrama muy similar al de la página principal; también tendrá cuatro cuadrados pero solo con las representaciones de los valores que tome el ejercicio. El cuadro de la derecha contendrá el código que se genere dinámicamente durante la resolución del ejercicio, así como el botón de copiar para poder extraerlo al portapapeles. Finalmente, debajo de estos cuadros pondremos un último, en el cual aparecerán los bocadillos con los diálogos en los que se muestran las explicaciones y las opciones a elegir por los usuarios.

Cabe destacar que en la etapa de diseño se modificó la vista final, la cual procederemos a explicar más adelante.

Una vez finalizada la etapa de diseño, el siguiente paso es iniciar la fase de desarrollo. Esta es una de las fases más largas debido a que cada ejercicio requiere de una serie de datos en específico, siendo estos datos la descripción del problema y los bocadillos que se mostrarán a los usuarios.

Primero se desarrolló la *view* del buscador con la interfaz 4.8; para esta vista se requiere recibir un dato como parámetro enviado en la petición, por defecto vacío en caso de que no se escriba nada y el contenido del cuadro de búsqueda si el usuario ha escrito algo; se buscará entonces en la base de datos con el parámetro encaso de que no se vacío, tras lo cual se devolverá un iterable con todos los que

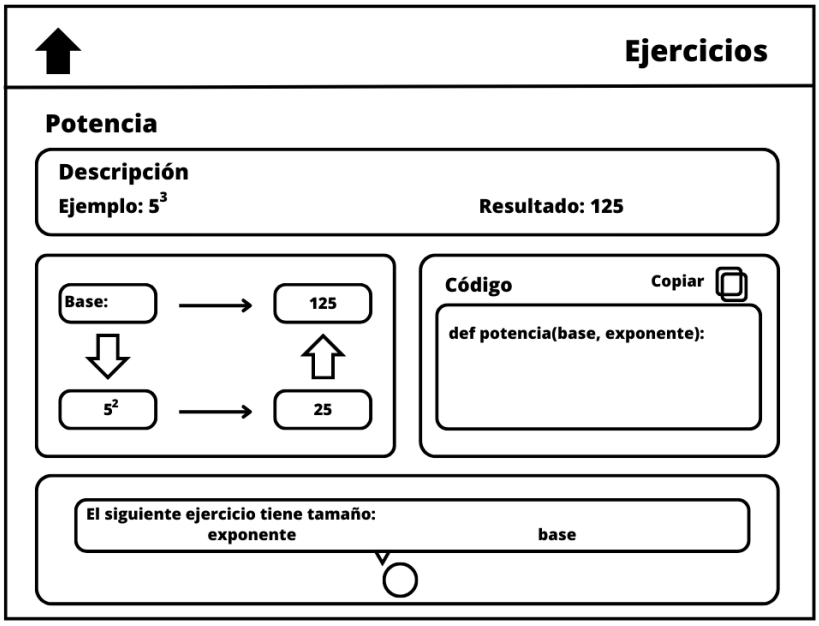

Figura 4.7: Boceto de la página de ejercicios.

cumplan los requisitos; también devolveremos el valor recibido en la petición para ponerlo en el campo de búsqueda. En la plantilla se utilizará un formulario de tipo *GET*, será el que utilicemos para enviar el campo de búsqueda al *backend*. Se añadirá un botón con el que se hace una petición con los campos vacíos que será el botón de borrar filtros. Justo debajo del buscador se creará una tabla en la que estarán los nombres de los ejercicios con enlaces para acceder al propio problema. Además añadiremos arriba, en la barra de navegación un enlace a esta vista de búsqueda.

A continuación se implementará la vista de los ejercicios como se ve en la interfaz de los ejercicios 4.9. Antes de empezar es necesario explicar que se utilizará el identificador principal (*id*) de la base de datos en las urls públicas. En general no es conveniente usar el *id* en cualquier dato que sea publico, ya que puede imponer un riesgo en la seguridad. Sin embargo, en esta aplicación dado que los ejercicios no son un recurso de con información crítica o confidencial de ningún tipo por lo que usar una url pública no supone ningún riesgo. Aún así, una alternativa a una url pública sería añadir un campo *slug* (es un campo tipo texto con la propiedad *UNIQUE*) en la base de datos, sustituyéndolo en la url por el *id*.

Con el objetivo de utilizar los datos pertinentes al ejercicio, tales como la descripción del problema, los párrafos explicativos y opciones de los diálogos, se ha implementado una función de selección en la *view*. Esta función de selección actúa como una petición a la base de datos para extraer la información del ejercicio y enviar como contexto a la plantilla los siguientes datos: Por un lado, una

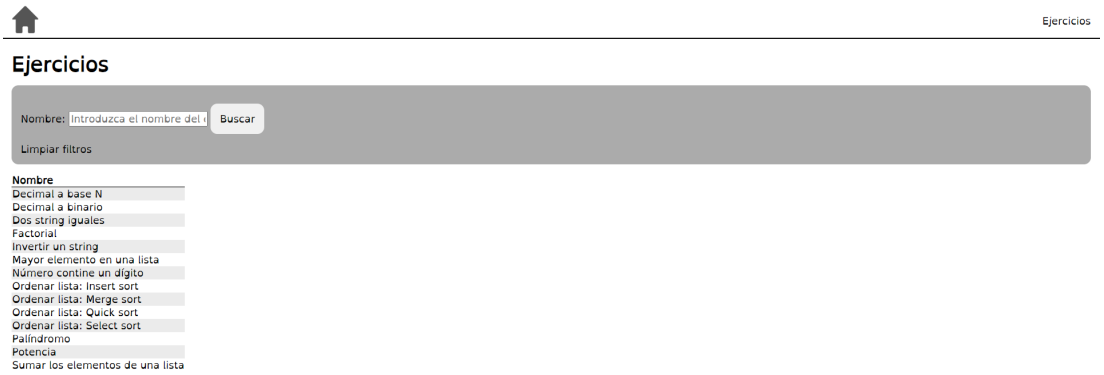

Figura 4.8: Interfaz de la página del buscador.

descripción del problema con un ejemplo y su solución; por otro lado, un formulario para los datos que se enviarán al *backend* para calcular el resultado, la solución del subproblema y la descomposición; además, el código en Python con el que se resuelve el ejercicio; finalmente, los textos y las opciones que aparecen dinámicamente en el bocadillo del profesor virtual.

En relación al funcionamiento del *frontend* se ha utilizado **JavaScript** para el bocadillo. Se utilizan unos eventos definidos en base a la clase de **CSS** del botón con el que se interactúe, el comportamiento del evento hará que aparezca el párrafo del bocadillo que le sigue, con las opciones asociadas y en caso de ser necesario, el código correspondiente. Por ejemplo, una decisión errónea llevará a un párrafo distinto que la correcta, también pueden coexistir múltiples elecciones correctas o incorrectas. Por tanto, los párrafos tienen opciones personalizadas, en ocasiones hay varias soluciones posibles a un mismo ejercicio y el código se genera dinámicamente según las respuestas que escoja el usuario.

Además, se han implementado distintos tipos de formularios según el tipo de los parámetros y la cantidad. Los formularios hacen referencia a los campos que han de rellenar los usuarios. Con el fin de calcular los datos aportados por los usuarios en los campos del formulario del diagrama de entradas y salidas 4.9 se hará una petición *GET* a una **API** propia desde **JavaScript**, enviando los datos mediante **JSON** al *backend* y devolviendo las respuestas en el mismo formato para representarlas en la página. A continuación se indicarán los ejercicios elegidos, en los que se explicarán los formularios utilizados, así como las posibles vías según la descomposición elegida. Los catorce ejercicios que se han elegido son:

- Decimal a base N: El ejercicio tiene un formulario con un campo de tipo numérico para el dígito del que se quiere transformar y otro campo numérico para introducir la base a la que se quiere convertir; en el *backend* se comprobará que los datos introducidos sean correctos y válidos generando una alerta en caso contrario. Tiene un único camino que utiliza como

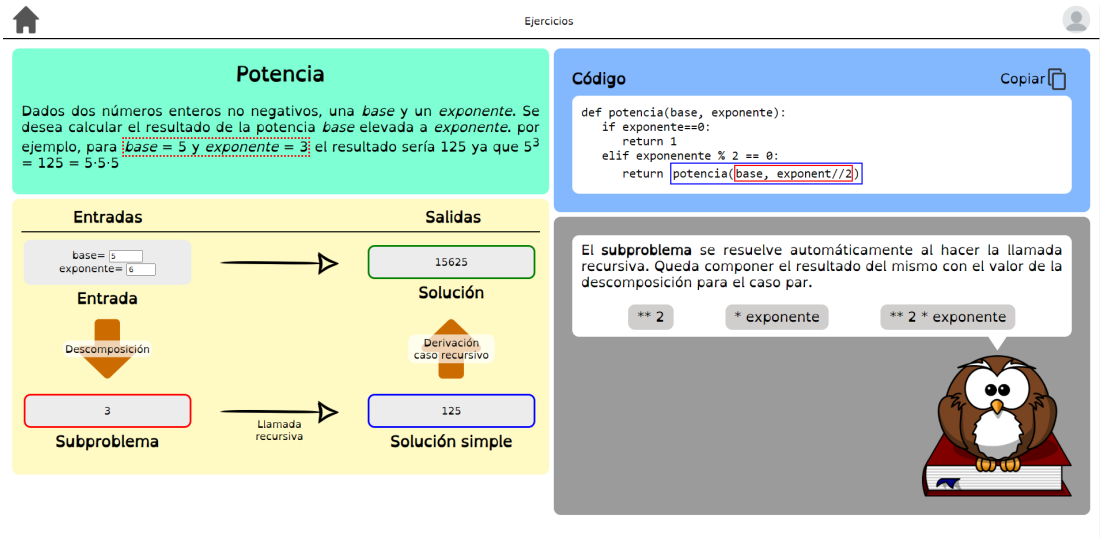

Figura 4.9: Interfaz de la página del ejercicio de potencias.

descomposición haciendo la división entera del dígito entre la base.

- Decimal a binario: El ejercicio utiliza un formulario con un único campo de tipo numérico para el dígito que se quiere transformar a base 2; también se validará el valor enviado en el *backend*. Tiene un único camino en el que la descomposición será la división entera del dígito.

- Dos *string* iguales: Para el ejercicio se usa un formulario con dos campos de tipo cadena de caracteres. En este caso puede seguir tres caminos independientes, las descomposiciones válidas serían quitar el primer elemento, el último elemento o dividir a la mitad las cadenas.

- Factorial: El formulario utilizado en este ejercicio contiene un único campo tipo numérico para el dígito, se hará validación por *backend* del valor. Tiene un único camino posible, siendo este la descomposición restando uno al dígito.

- Mayor elemento en una lista: Este ejercicio utiliza un formulario con un campo de cadena de caracteres, validando el campo en el *backend*. El ejercicio tiene tres caminos posibles, la descomposición sería eliminando el primer elemento de la lista, el último o dividirla a la mitad.

- Número contiene un dígito: El formulario que utiliza este ejercicio tiene dos campos numérico, uno será el número sobre el que se buscará el dígito. El ejercicio tiene un único camino, este será utilizando la descomposición de la división entera entre diez.

- Ordenar lista *Insert sort*: Es uno de los ejercicios de ordenar una lista. El formulario utilizado es un campo de cadena de caracteres, el cual se validará

en *backend*. Solo tiene un camino posible, que es quitar el último elemento para la descomposición. En realidad hay muchas descomposiciones posibles, pero se explican distintos algoritmos en ejercicios diferentes.

- Ordenar lista *Merge sort*: Otro de los ejercicios de ordenar una lista. El formulario también usa un campo de caracteres validado por backend para la lista. Tiene un único camino posible, la descomposición que se emplea es dividir la lista a la mitad.

- Ordenar lista *Quick sort*: El tercero de los ejercicios de ordenar una lista. El formulario usa un campo de caracteres que se validará en el *backend*. Tiene un solo camino posible, la descomposición implica escoger un pivote y todos los elementos que sean menores se pongan en la izquierda y todos los que sean mayores a la derecha.

- Ordenar lista *Select sort*: El último de los ejercicios de ordenar una lista. El formulario al igual que los anteriores es un campo de caracteres validado por *backend*. La descomposición tiene un único camino posible que seguir, en este caso la descomposición es extraer el mínimo de la lista.

- Palíndromo: Este ejercicio usa un formulario con un campo de caracteres, el cual se validará en el *backend*. La descomposición que se utilizará sera eliminar el primer y el último elemento, permitiendo un único camino posible.

- Potencia: El formulario utilizado en el ejercicio tiene dos campos de tipo numérico, uno será la base y el otro el exponente, ambos validados en el *backend*. En este ejercicio hay dos caminos posibles, la operación del primer camino será restar uno al exponente y la segunda descomposición posible será dividir a la mitad el exponente.

- Sumar los elementos de una lista: Finalmente, el último ejercicio tiene un campo de tipo cadena de caracteres, el cual será validado en el *backend*. Hay tres caminos posibles, siendo las descomposiciones eliminar el último elemento, el primero o dividir a la mitad la lista.

Una vez terminada la etapa de desarrollo procederemos con la etapa de pruebas, tal y como hemos hecho anteriormente en el primer incremento vamos a utilizar la metodología de *User testing*. Dado que este es el apartado más importante de la aplicación se ha realizado más tarde un pequeño formulario a los usuarios. Las preguntas y la valoración y modificaciones hechas se añaden a continuación:

- Valore lo intuitivo que ha sido el buscador (Nota media: 8.5): El principal problema con el buscador ha sido que uno de los usuarios se ha equivocado

pensando que el campo de introducir era para volver a introducir su nombre dado que no se veía la parte del ejercicio.

- Valore la facilidad de uso del buscador (Nota media: 9): En general todo comentarios positivos salvo el hecho de que hay pocos campos, solo el nombre.

- Valore lo intuitivo que ha sido el ejercicio (Nota media: 7): La única valoración negativa en este apartado, aunque por todos los usuarios, han sido los diálogos y la descripción. Los textos resultaron un tanto complicados y enrevesados a la mayoría de usuarios, se han modificado prácticamente en su totalidad y cuando se les ha vuelto a enseñar demostraron mayor comprensión y lo valoraron más positivamente. También cabe destacar que los usuarios valoraron el añadir un primer párrafo de comienzo y de final.

- Valore la facilidad de uso del ejercicio (Nota media: 7.5): El principal problema con el que se encontraron los usuarios era la disposición del ejercicio, no se veía bien con los bocadillos abajo y la descripción ocupando tanto espacio arriba, se ha modificado poniendo la descripción a la izquierda arriba, el gráfico abajo a la izquierda. A la derecha arriba se ha puesto el código que se va generando y abajo a la derecha el profesor con los bocadillos. También se alteraron los colores del gráfico, tras estos cambios estaban más contentos los usuarios.

- Comentarios adicionales: Algunas de las modificaciones generadas en base a los comentarios fueron alinear el botón de copiar e indicar que el texto ha sido copiado, otra de las modificaciones fue añadir el triángulo al bocadillo para dar mayor apariencia de que era un bocadillo del buho, finalmente el ajuste de colores gracias a un usuario daltónico.

4.3.3. Tercero incremento: Usuarios

El tercer incremento se dedicará a la creación de un sistema de cuentas. Cada usuario tendrá que tener una cuenta asociada para poder identificarse en la aplicación.

En primer lugar como indica el modelo incremental, se tienen que analizar los requisitos impuestos para esta fase. Utilizaremos un diagrama de casos de uso para realizar este análisis.

Como podemos apreciar en el diagrama 4.10, los elementos a implementar serán los siguientes:

- Inicio de sesión: Se plantea permitir que los usuarios inicien sesión a través de un formulario para que puedan acceder a los ejercicios.

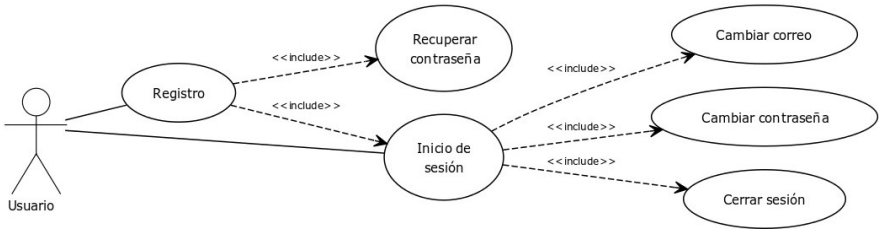

Figura 4.10: Diagrama de casos de uso de los usuarios.

- Registro: Se propone que un futuro usuario pueda registrarse en la página especificando unas credenciales para proteger la pertenencia de la cuenta.
- Cambiar contraseña: Es interesante que un usuario sea capaz de cambiar su contraseña una vez ha accedido a la aplicación o poder recuperarla en caso de que se olvide.
- Cambiar correo: Un usuario podría alterar el correo por si necesita que las comunicaciones se hagan a través de otra cuenta.
- Cerrar sesión: Se pretende que un usuario pueda cerrar sesión una vez este autorizado dentro de la aplicación para mayor seguridad.

Una vez se han establecido los requisitos a implementar, se procederá con la etapa de diseño de la aplicación. En un principio se han de generar múltiples *views* para cumplir con los requisitos expuestos en el punto anterior 4.10. Todas las pantallas de usuario tendrán la misma estructura y estética, por lo que se ha decidido utilizar vistas similares con modificaciones pequeñas para que el diseño resulte familiar e intuitivo al usuario. En conclusión, se utilizará un boceto de la página de inicio de sesión como único diseño, por consiguiente los demás bocetos deberían de seguir la misma estructura con distinto contenido 4.11.

Como se puede apreciar en el boceto de inicio de sesión 4.11, todas las plantillas que se van a generar tendrán un cuadro central sobre el que aparecerá la información, en caso de que requiera de introducir datos por parte del usuario aparecerán los campos, en cambio si representa información aparecerá dichos datos en vez del formulario.

Una vez terminada la etapa de diseño, se procederá a la de implementación de todas las funcionalidades relacionadas con los usuarios. Se va a utilizar el modelo de usuario genérico que implementa **Django** por defecto. Dado que las vistas son muy similares solo se utilizará de ejemplo la interfaz de inicio de sesión 4.12 Los campos de interés para este trabajo son:

- El nombre de usuario: Es un campo varchar con un tamaño permitido de ciento cincuenta caracteres y con la propiedad *UNIQUE*, la cual obliga a que los usuarios no puedan repetir nombre.

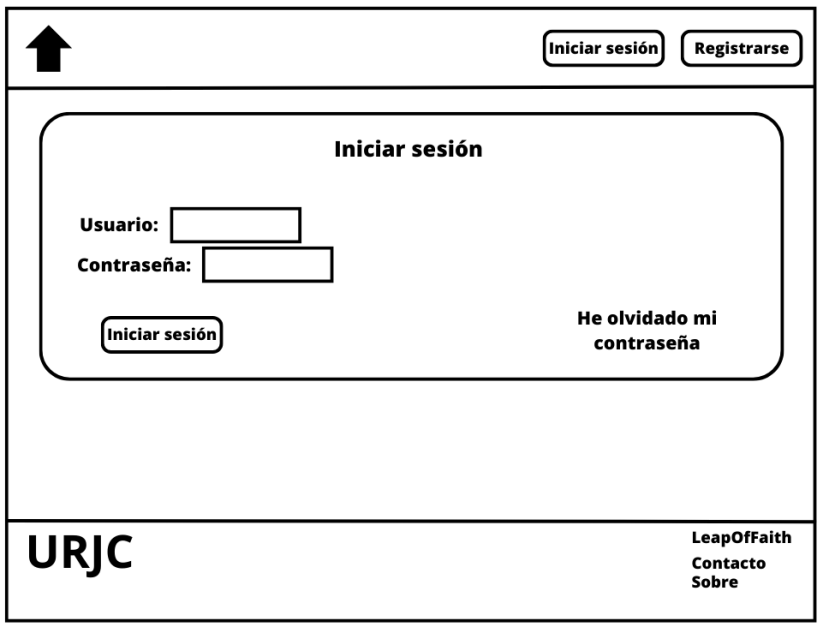

Figura 4.11: Boceto de la página de inicio de sesión.

- El correo: Es un campo varchar con un tamaño permitido de doscientos cincuenta y cuatro caracteres.
- Contraseña: Es un campo varchar con un tamaño permitido de ciento veintiocho caracteres. Utiliza un cifrado basado en *hashing*, el algoritmo usado es PBKDF2 por defecto, lo suficientemente seguro para la aplicación.

Se va a añadir un modelo adicional, el cual va a representar el perfil. Los campos que tendrá serán el usuario al que pertenece con una relación uno a uno, como si fuese una extensión de la tabla y un código de cambio que se utilizará en caso de que olviden la contraseña. Para este último caso se enviará un correo al usuario con el código de recuperación, el cual cambiará cada vez que lo solicite y solo se podrá utilizar una vez. La probabilidad de acertar este código es de una entre cien millones y no se puede descifrar por métodos de fuerza bruta.

En caso de que un usuario quiera cambiar la contraseña se requerirá introducir la antigua, el mismo caso ocurre con el correo aunque se solicitará el antiguo. Esto evitará que en caso de que un usuario se deje la sesión abierta puedan modificarle fácilmente las credenciales. También se utilizarán validadores de contraseñas de **Django**, el objetivo es que los usuarios utilicen contraseñas un poco más seguras. Los criterios serán: utilizar una contraseña distinta a los atributos del usuario, por ejemplo que no sea similar al nombre; que tenga un tamaño mínimo de 8 caracteres; que no sea una contraseña muy común, como por ejemplo 1234 y que no sea una contraseña solo numérica.

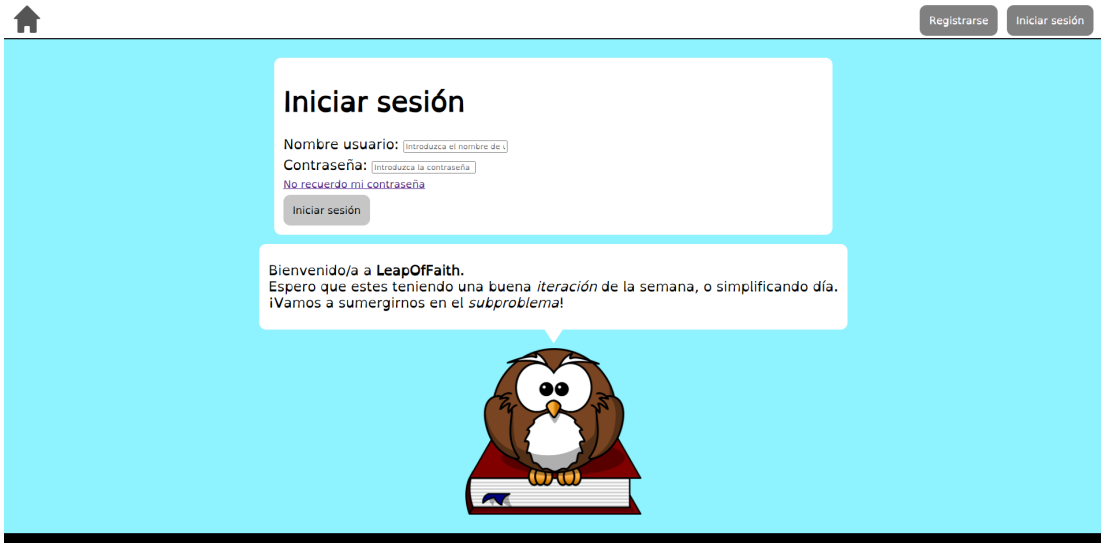

Figura 4.12: Interfaz de la página de inicio de sesión.

Finalmente indicar que tal y como se explicó todas las vistas son muy similares, la mayoría presentan datos o formularios para las páginas solicitadas. Sobre todas las vistas cabe destacar el método de recuperación de contraseña a través del correo. Primero se enviará un mensaje de correo con el código de recuperación en caso de que exista el correo especificado, si no existe se llevará aún así a la página indicando que se entregará en caso de que exista. A continuación se añadirán a la sesión unos parámetros identificadores para que no se pueda acceder a las vistas sin estos datos, además de llevar el control de que pasos ha realizado el usuario, todos los parámetros son temporales y se van eliminando según se utilizan. En caso de completar todos los pasos con éxito se restablecerá la contraseña con la nueva que introduzca el usuario.

Una vez terminada la etapa de desarrollo se procederá a la etapa de pruebas, se volverá a utilizar la metodología de *User testing* para que los usuarios prueben de primera mano las capacidades y la funcionalidad así como obtener resultados directamente de posibles usuarios.

Este ha sido uno de los incrementos más positivos en cuanto a valoración, la mayoría de valoraciones han sido positivas, aunque listaremos también algunas modificaciones que se han realizado.

- En primer lugar los usuarios tardaron en registrarse porque empezaron pensando que ya tendrían un usuario, se les recordó que es como si entrasen por primera vez a la aplicación o no la hubiesen visto nunca. En ese momento se registraron sin problemas aunque dos de ellos intentaron poner contraseñas muy simples y les toco hacerlas más complejas ante el error controlado de campos que les apareció.

- En segundo lugar a los usuarios no les costo nada acceder a través del inicio de sesión aunque uno de ellos se olvido de la contraseña y le incitamos a intentar recuperarla, lo cual discutiremos en un punto siguiente, después entro sin problemas.
- En tercer lugar todos accedieron fácilmente al perfil y fueron capaces tanto de modificar su correo como su contraseña sin problemas. No hubo ningún comentario negativo salvo el ajuste de unos botones.
- En cuarto lugar todos consiguieron cerrar sesión sin problemas mediante el botón del desplegable. Tras lo cual se les solicito qu intentasen recuperar la contraseña. No le costo a ninguno seguir los pasos y además uno de los usuarios se sorprendió mucho de que de verdad le llegase el correo y funcionase, tras lo cual intento hacerlo mal y comprobó que no podía cambiar la contraseña quedando muy sorprendido e indicando que le gustaba mucho.
- En último lugar se han corregido algunos errores visuales como botones que aparecían pegados a los bordes. También se añadió el buho con un diálogo en el inicio de sesión y el registro a petición de un usuario; todos los usuarios apreciaron y el cambio aunque no lo valoraron necesario e imprescindible pero tuvo valoración positiva por todos.

4.3.4. Cuarto incremento: Gamificación

El cuarto paso busca generar motivación en los usuarios para que quieran utilizar la página y desarrollar las competencias. El público objetivo de este trabajo es el público joven, por tanto la gamificación es uno de los aspectos más relevantes. El éxito de la gamificación se basa en el sentimiento de progreso y de haber cumplido objetivos, además queda patente en la popularidad de los videojuegos entre el espectro más joven de la población que implementan ampliamente ambos conceptos. En base a estos argumentos se implementarán métodos de gamificación que modificando mínimamente algunas de las funciones explicadas en los anteriores incrementos. Estas modificaciones añaden funcionalidad sin modificar el comportamiento de las funciones[18].

Primero realizaremos un diagrama de casos de uso 4.13 para analizar los requerimientos que se buscará realizar en esta fase.

- Obtener medalla: Al terminar un ejercicio se quiere añadir una funcionalidad que permita al usuario obtener una medalla virtual indicando que ha completado el ejercicio.
- Visualización del progreso: Un usuario podrá visualizar el porcentaje de ejercicios resuelto, de esta forma será capaz de estimar cuanto ha hecho y cuanto le falta para acabar.

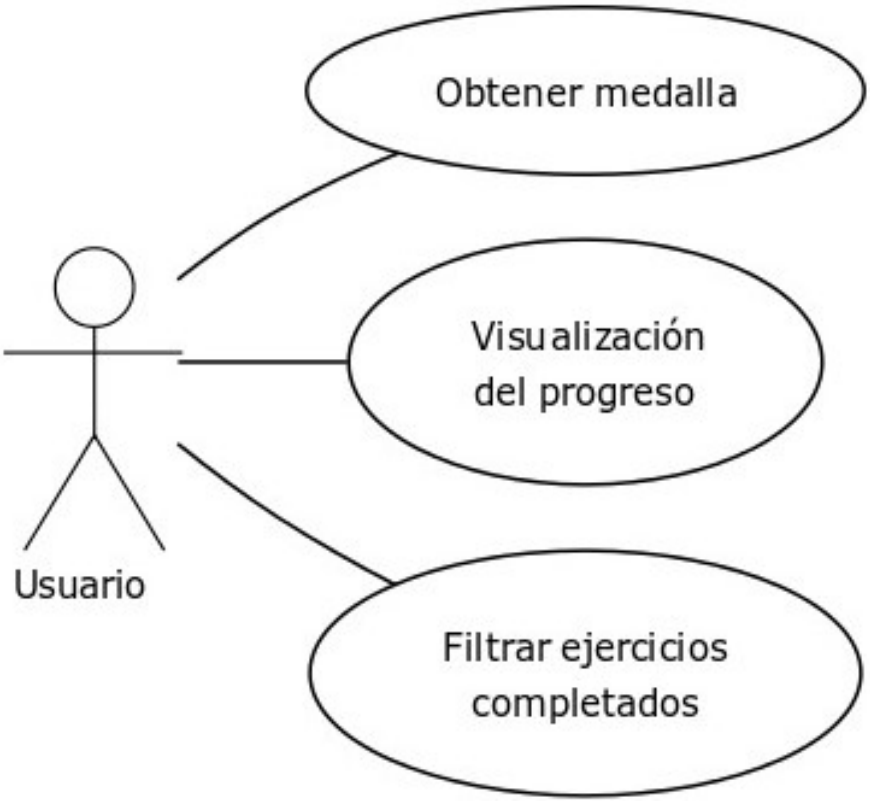

Figura 4.13: Diagrama de casos de uso de los usuarios.

- Filtrar ejercicios completados: Un usuario tendrá la capacidad de filtrar en el buscador utilizando como criterio si ha realizado o no los ejercicios.

A continuación, una vez establecidos los requisitos, se comenzará con el diseño de la nuevas funcionalidades. Realizaremos dos bocetos, el primero 4.14 será la vista de la medalla, la cual se hará sobre la vista de los ejercicios dado que se entrega al completar uno. El segundo boceto 4.15 realizará una modificación de la vista del buscador con objetivo de que los usuarios puedan ver el progreso fácilmente y en la misma vista en los que aparecen.

Como se aprecia en el boceto de la vista de la medalla 4.14, tal y como se ha explicado en el anterior párrafo, se ha modificado la vista para que una vez se complete el ejercicio aparezca una imagen de una medalla con un texto indicando que has completado el ejercicio, también se añadirán elementos dinámico como rayos o líneas de fondo rotando para que sea algo más vistosa y no quede estática.

En el boceto de la vista del progreso 4.15 se puede observar que se ha modificado el buscador. Los elementos nuevos que aparecen son una barra de progreso con una oración que variará en base al número de ejercicios realizados. También se añadirá un selector para permitir que el usuario filtre por ejercicios realizados y no realizados.

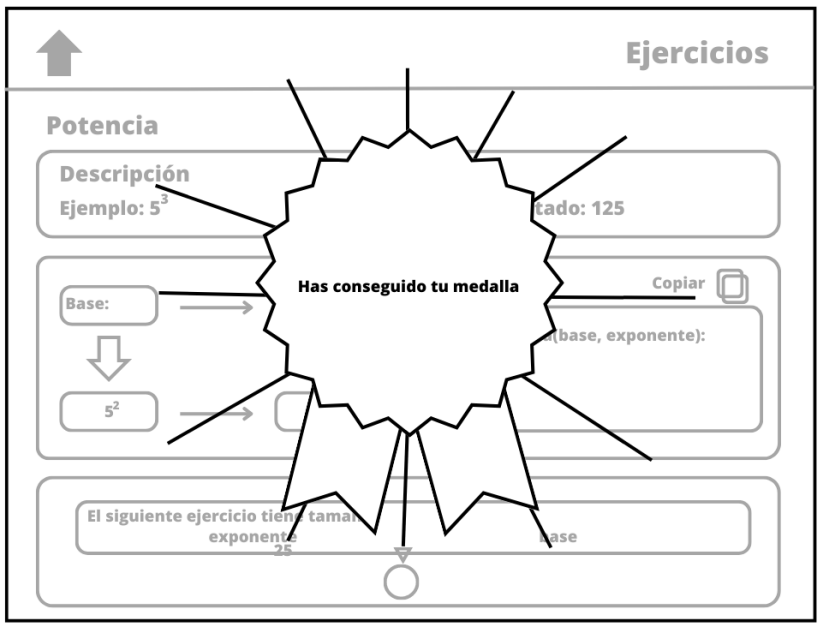

Figura 4.14: Boceto de la vista de la medalla.

Una vez terminada la fase de diseño se continuará con la fase de desarrollo e implementación de las funcionalidades. En este caso se van a modificar algunas vistas de las implementadas en anteriores funcionalidades, aunque las modificaciones no afectan al comportamiento ya definido en las funciones, solo añadirán cambios simples sin modificar los comportamientos ya establecidos.

Primero se implementará la funcionalidad de la medalla como se puede ver en la la interfaz de la vista de la medalla 4.16. Para ello se añadirán código **HTML** para crear el elemento de la medalla y se ocultará con **CSS**. Tal y como aparece el contenido de los bocadillos en los ejercicios, se sustituirá la última llamada por la aparición de la medalla y la última llamada será hacer clic sobre la medalla. Cabe destacar que se ha implementado un objeto **CSS** con animación para que rote lentamente alrededor de la medalla sin llegar a marear. También indicar que en caso de que el ejercicio ya haya sido completado se modificará el texto escrito en la medalla indicando que ya la ha conseguido.

Finalmente, se creará la funcionalidad del progreso como se puede observar en la interfaz modificada del buscador 4.17. Para ello se va a modificar el modelo del perfil para generar una relación N:M con los ejercicios, dado que el perfil es una extensión de los usuarios sería como hacer una relación entre los modelos de ejercicios y usuarios. Lo primero será añadir una función que guarde en la base de datos el ejercicio realizado cuando el usuario haya acabado, con eso ya se sabría que ejercicios ha hecho cada usuario. En segundo lugar se hará una petición a la base de datos en la *view* del buscador para averiguar que ejercicios ha realizado un usuario. Se añadirá a la tabla el campo de realizado y se rellenará. En tercer

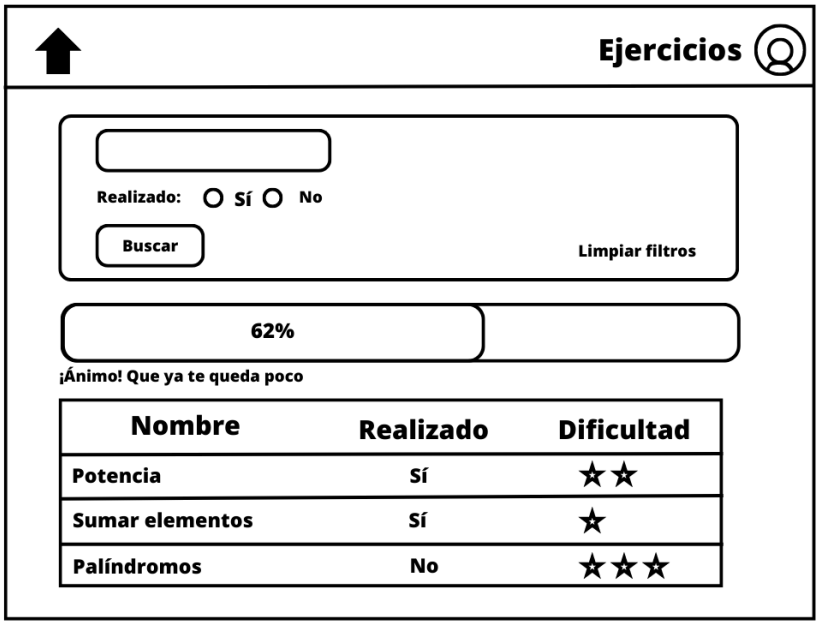

Figura 4.15: Boceto de la vista de progreso.

lugar, en la misma *view* se añadirá el filtro y se hará la búsqueda por los campos de nombre y de ejercicio realizado. Finalmente, se creará una barra de progreso en base al número de ejercicios realizados entre el total; se añadirá a través de **JavaScript** un código que presente un mensaje distinto en base al porcentaje de ejercicios realizados.

Una vez se ha terminado con el desarrollo se procederá a la etapa de pruebas. Por última vez se utilizará la metodología de *User testing*, con la cual obtendremos la última valoración de la aplicación.

Esta etapa ha sido la que más valoraciones positivas ha tenido, se han puntualizado algunos detalles para hacer algunas mejoras finales, pero en general ha habido una recepción muy positiva.

- En primer lugar un comentario que se repitió entre los usuarios fue que el botón de enlace al ejercicios buscador de ejercicios, el cual se encontraba al lado del usuario, podía parecer que era el nombre de la persona y confundirlo o no prestar atención, tras lo cual se puso en el centro de la barra de navegación para que se diferenciase más claramente.
- En segundo lugar todos fueron capaces de entender la barra de progreso y hubo un usuario que pregunto por si el comentario cambiaba, a lo cual se le respondió afirmativamente pero también se indicó que lo mejor para ello es probarlo, tras lo cual comprobó que en efecto cambiaba en base al porcentaje realizado.

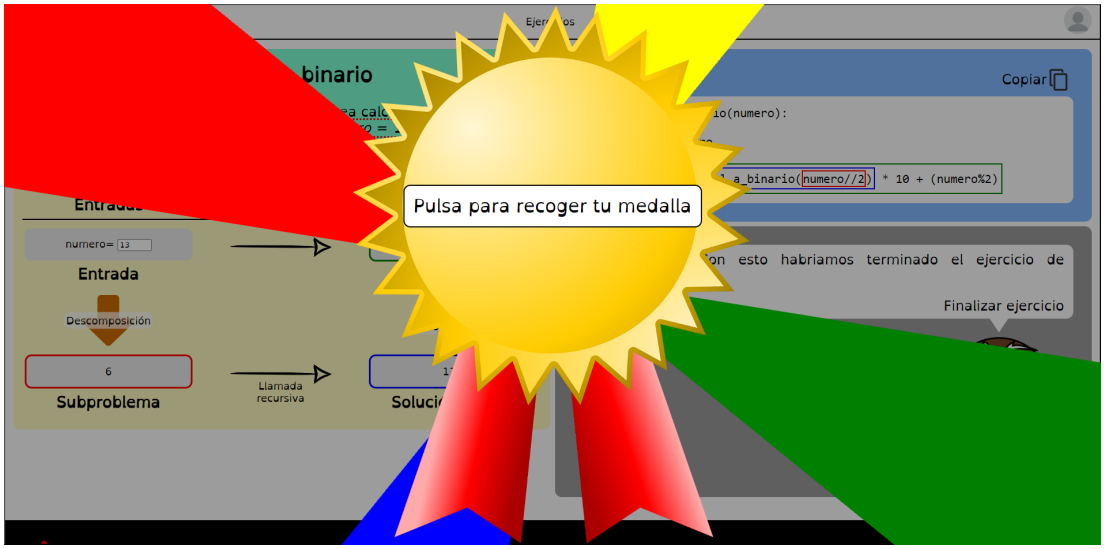

Figura 4.16: Interfaz de la vista de la medalla.

- En tercer lugar la mayoría de los usuarios apreciaron tanto el añadir un grado de dificultad a los problemas como la columna indicativa de si has realizado o no el ejercicio.
- En último lugar al ver la mayoría de usuarios expreso su conformidad y entusiasmo por el logro. También indicaron que la animación giraba un poco rápido y que podía marear, tras lo que se redujo la velocidad algo más lo, cual apreciaron los usuarios que expresaron dicha opinión.

4.4. Valoración de usabilidad

Para la evaluación de usabilidad se ha decidido utilizar el decálogo de principios heurísticos de Jackob Nielsen, dado que son uno de los enfoques más utilizados en el desarrollo de aplicaciones no especializadas[21]. La evaluación del cumplimiento de los diez principios se verá a continuación.

- Visibilidad del estado del sistema: Siempre se informa al usuario del estado de las peticiones que se realizan, por ejemplo si introduce incorrectamente sus credenciales se le informa del error. Otro ejemplo es que si los datos que introduce el usuario son incorrectos durante la resolución del ejercicio se le explicará en una alerta el formato de los datos a introducir, así como poniendo un ejemplo en el diálogo. Además en caso de tener mala conexión se indicará en el bocadillo que se está realizando el cálculo. También se mantiene informado al usuario cuando ha cambiado el correo o contraseña y el número de ejercicios que lleva realizados.

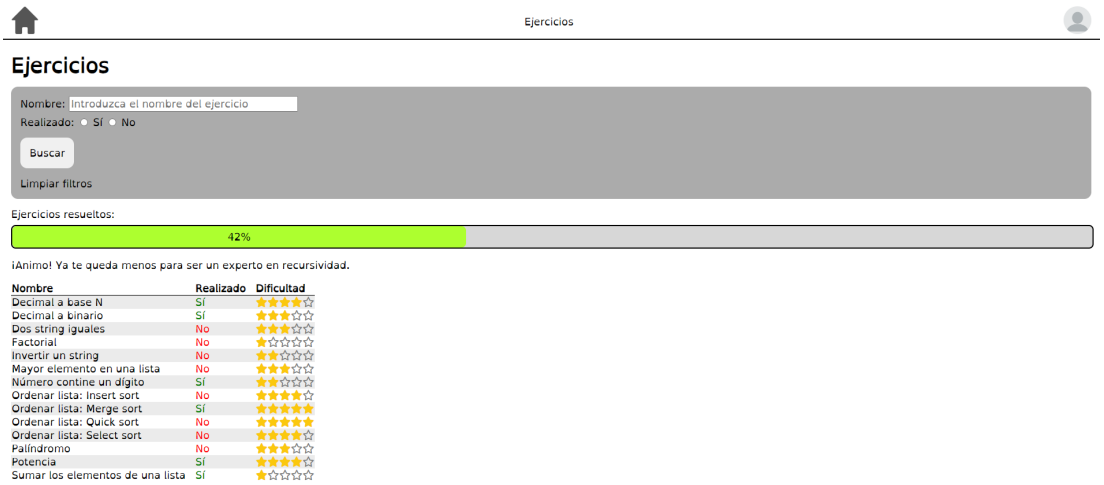

Figura 4.17: Interfaz de la vista modificada del buscador.

- Coincidencia entre el sistema y el mundo real: Se presenta por ejemplo el código real que los usuarios van desarrollando en base a sus respuestas, también se indica el progreso que llevan y si han realizado un ejercicio u otro. También se enseñan los datos relacionados con el usuario aunque ligeramente ofuscados por motivos de seguridad.

- Control y libertad del usuario: En general es difícil equivocarse, pero si accede a páginas que no quiere como cambiar la contraseña, el correo o el inicio de sesión en vez de registrarse o al revés, existe una barra de navegación para acceder al resto de opciones o los botones de cancelar en los dos primeros casos.

- Coherencia y estándares: La barra de navegación se mantiene a lo largo del sistema salvo cuando el usuario no ha iniciado sesión, manteniendo la funcionalidad en todo momento. Se han utilizado también opciones de *hover* para que los enlaces tengan el mismo color al pasar el puntero por ellos y todos tienen el mismo estilo.

- Prevención de errores: En todo momento se representan los distintos elementos con distintos colores para mantener una clara separación entre ellos y se mantiene informado al usuario mediante descripciones o diálogos del estado para que pueda proceder con la siguiente instrucción. Por ejemplo durante el restablecimiento de contraseña solicitará primero el correo al que enviar el código de recuperación, a continuación la solicitud del propio código de recuperación, seguido del cambio de contraseña y finalizando con la indicación de que se ha restablecido.

- Reconocimiento antes que memoria: La aplicación dispone de un número pequeño de páginas aunque siempre utilizan los mismos elementos y una

disposición similar para que no cueste al usuario reconocer dónde se encuentra.

- Flexibilidad y eficiencia de uso: Dado que es una aplicación web
- Estética y diseño minimalista: Se ha reducido al mínimo los elementos visuales en todas las pantallas, dejando lo mínimo e indispensable para transmitir toda la información. Como ejemplo podrían ser las vistas de usuario que tienen solo el recuadro con los campos a rellenar y una breve descripción o título de la pantalla. Otro ejemplo es la vista principal, donde se dispone el diagrama de la metodología declarativa, donde se ocultan las descripciones de los recuadros hasta que el usuario interactúe con el diagrama.
- Ayuda al usuario a reconocer, diagnosticar y recuperarse de los errores: En los distintos formularios utilizados en la aplicación se devuelven los distintos errores que el usuario haya podido cometer al rellenarlos. Por ejemplificar, se indica si la contraseña no coincide o si es muy similar al nombre de usuario. También en la vista de ejercicios se genera una alerta en caso de que los campos estén mal escritos. También se informa al usuario que la elección a la pregunta pueda estar equivocada.
- Ayuda y documentación: Se ha creado una vista en la que se explica el funcionamiento de la página y algunos de los errores más comunes. Además, las vistas tienen descripciones o botones de ayuda con *tooltips* asociados.

4.5. Despliegue

En esta sección se describe el despliegue realizado de la aplicación, es de consideración que ante la falta de un servidor en el que poder mantener dicho despliegue y la falta de un certificado de confianza, se ha utilizado un equipo propio como servidor temporal para este apartado y un certificado autofirmado. Tras la comprobación del funcionamiento de todas las características de la aplicación se ha cerrado el despliegue del servidor.

4.5.1. Consideraciones de seguridad

En primer lugar se han de tener ciertas consideraciones de seguridad. Para el correcto despliegue de **Django** se han de realizar varios cambios al archivo de settings.py. En primer lugar, la variable **SECRET_KEY** tenemos que eliminarla del código. Para ello se ha creado un fichero tipo **.env** el cual va a guardar dicha llave, además se hace un acceso a este archivo a través del código para su utilización por parte de la aplicación. Al realizar este cambio cualquier persona que pueda ver el código no podrá encontrar la llave expuesta, a continuación

se han cambiado los permisos del archivo para que solo sea capaz de leerlos el usuario creado para correr la aplicación y el administrador. Cabe destacar que también se podría guardar en el almacén de llaves de **Ubuntu** y usarla en el código extrayendo el secreto del mismo.

El siguiente cambio realizado en settings.py es cambiar la variable **DEBUG** a *False*. En este caso se ha modificado el modo de utilización de **Django** para que deje de estar en el modo "pruebas". Al quitar este modo **Django** no ofrecerá las trazas de error de aplicación durante el uso por parte de los usuarios; en cambio se enviarán las trazas por correo a los miembros administradores. La desactivación del modo de pruebas también hará que la aplicación deje de hacerse cargo de servir archivos, tales como imágenes, hojas de estilo **CSS** o archivos **JavaScript**.

4.5.2. Apache

Se procederá con la configuración de un servidor **Apache**, este servicio será el encargado de crear el tráfico https, por lo que se tiene que proporcionar un certificado para poder cifrar el tráfico mediante **SSL**. A continuación en el archivo settings.py de la aplicación se tendrá que modificar la variable **ALLOWED_HOSTS**, en la que se incluirá el dominio utilizado por el servidor y su ip. Con objetivo de que **Apache** pueda enviar las imágenes, hojas de estilos y **JavaScript** es mandatorio ejecutar el comando collectstatic en la aplicación **Django**, dicha orden copiará todos los archivos para el uso del servidor. Finalmente se procederá con la creación de un fichero .conf en el que se añadirán el administrador, el nombre del servidor, el alias, la ruta del proyecto, la redirección de los errores a un fichero, la ruta en la que están copiados los ficheros de **Django**, se establecerán los permisos sobre estos ficheros y sobre el archivo wsgi.py de **Django**, el cual es el encargado de recibir las peticiones y servir los documentos **HTML** a **Apache**.

4.5.3. MySql

Se va a utilizar una base de datos **MySql** para guardar la información, cabe destacar de cara al proyecto entregado se entregará con una **SqLite** para poder ejecutar el proyecto sin necesidad de configurar una base de datos propia. Primero se creará la base de datos en la máquina pertinente, siendo en este caso el mismo equipo en el que va a correr la aplicación. Por consiguiente, se creará un usuario al que le se le otorgarán los permisos necesarios y mínimos para manejar la base de datos. A continuación para establecer la conexión entre la aplicación y la base de datos se creará un archivo .conf, en el que se añadirá el nombre de la base de datos, el *host* dónde estará ejecutándose (*localhost* en este caso), el usuario y contraseña de la base de datos creados con anterioridad y el tipo de caracteres por

defecto (en este caso utf8). Finalmente se le dará permisos de lectura al usuario que se va a ejecutar la aplicación para poder acceder a este archivo.

5

Conclusiones y trabajos futuros

En este capítulo se detallan las conclusiones obtenidas de la realización del trabajo de fin de grado así como una serie de propuestas de posibles trabajos futuros con los que ampliar la aplicación.

5.1. Conclusiones

Una vez completado el proyecto se han visto cumplidos los objetivos marcados durante las distintas fases de análisis en cada uno de los incrementos para el modelo.

La aplicación creada es una herramienta para la enseñanza de recursividad a usuarios, se han realizado múltiples ejercicios guiados en los que se genera código mientras el usuario va eligiendo entre una serie de opciones; todas las decisiones que tomen los usuarios recibirán una retroalimentación por parte de la página de los motivos por los que es correcto o incorrecto proceder con la opción seleccionada. Al irse generando código dinámicamente durante el ejercicio, los usuarios podrán copiarlo en cualquier momento y continuar por si mismos en un IDE de programación o podrán, una vez terminado el ejercicio, extraer dicho código y utilizarlo.

Se ha logrado hacer un despliegue en local en una máquina con un sistema operativo **Linux** tipo **Ubuntu**. Se ha accedido a la aplicación desde la propia red local utilizando otros equipos distintos, no se ha probado desde la web debido a

que no se dispone de una configuración funcional, segura y fiable en la instalación de red doméstica ante la falta de certificado y el peligro que puede conllevar exponer puertos, cabe destacar que se podría realizar ajustando las políticas del cortafuegos, las reglas de red y el monitoréo de equipos y red interna.

Gracias a la metodología de *User testing* se han comprobado requisitos y recomendaciones de usabilidad. Esto a permitido que la aplicación sea intuitiva, funcional y fácil de usar en los casos estudiados con una retroalimentación positiva por parte de estos usuarios. También gracias a las consideraciones se han podido mejorar ampliamente los apartados gráficos y el entendimiento de los textos. Cabe destacar que no se ha podido probar con futuros o presentes alumnos de las asignaturas de Diseño y Análisis de Algoritmos o su valor pedagógico.

Con especial atención al trabajo desarrollado cabe destacar el aprendizaje y uso del *framework* de desarrollo utilizado **Django**, con el cual se ha mejorado la utilización del lenguaje de **Python** ya desarrollado en un par de asignaturas de la carrera. También es notorio la puesta en marcha del servicio y los conocimientos adquiridos para la puesta en producción de un *software*. También se han adquirido conocimientos mucho más avanzados en relación al desarrollo de *frontend*, aprendiendo manejo de eventos **JavaScript** y la modificación en dinámico de objetos en **HTML**, también se han desarrollado tanto animaciones como transiciones utilizando **CSS** para el enriquecimiento visual de las páginas.

5.2. Trabajos futuros

Una vez expuestas las conclusiones de los trabajos, se ha pensado en mejoras que podrían aumentar la calidad del *software* desarrollado en este trabajo.

En primer lugar una de las mejoras más interesantes podría ser la generación de grupos para el uso de profesores y alumnos. De esta forma utilizando la base de datos y utilizando el propio modelo de grupos ya implementado en **Django** se podría crear un grupo de profesores. Este grupo se incluiría desde el servicio de administración aprovechando el campo *staff* del usuario básico de **Django**, con el cual se crearía una vista específica para dar de alta a profesores utilizando el servicio de envío de correos para que el usuario reciba las credenciales. Los profesores a continuación podrían crear sus propios grupos para separar los cursos de alumnos; una vez creados estos grupos podrían subir un archivo, por ejemplo **JSON**, para crear un conjunto de usuarios en la aplicación y enviarles las credenciales automáticamente a través del servicio de correo.

En segundo lugar aprovechando la funcionalidad descrita en el anterior apartado se podrían generar estadísticas de los aciertos/fallos de los usuarios durante el desarrollo de los ejercicios, también se podría recopilar el tiempo utilizado para el desarrollo de cada uno por parte de los alumnos. Para esto se podría utilizar

JavaScript aprovechando los eventos ya creados para registrar estos datos en un formulario de **Django** aprovechando la posibilidad de utilizar múltiples formularios en una vista. De esta forma se podrían registrar los datos al final y enviarlos al *backend*, el cual registraría en la base de datos estas estadísticas. Esto podría servir para que los profesores pudiesen comprobar si el uso correcto de esta aplicación ayuda a los alumnos a obtener mejores resultados.

En tercer lugar se propondrá sustituir los campos de información de los ejercicios actuales de la aplicación, creando un modelo específico en la base de datos para el guardado de datos de alta capacidad o usando ficheros para guardar esta información. Además, con dicha implementación se podría crear un formulario para que los profesores pudiesen crear nuevos ejercicios aprovechando esta *feature*, sería un trabajo con formularios múltiples para el árbol de desarrollo de los textos informativos. Respecto a la validación de datos a través de las llamadas a la **API** se podrían crear distintos tipos de formularios dinámicos de **Django** para personalizar cada ejercicio distinto, seleccionando el tipo de formulario que se usará. Para la generación de datos del subproblema y la llamada recursiva se podría aprovechar el uso del método exec, aunque con extremo cuidado, debido a que al permitir la ejecución de código a través de texto impone un riesgo de seguridad muy alto; una opción para salvar la peligrosidad de estos códigos sería utilizar una evaluación de riesgos de la cadena, acompañado de el visto bueno por parte del personal administrativo a través de la propia aplicación.

En cuarto lugar, una mejora sencilla podría ser aunque solo en caso de añadir nuevos problemas o la implementación de la creación de nuevos problemas de forma dinámica; se recomienda utilizar el método paginator de **Django**, este método permite seleccionar el tamaño de la lista de datos enviados al *frontend* de modo que el buscador de ejercicios no tuviese una cantidad abrumadora de problemas, no se ha implementado actualmente dado que al solo haber catorce ejercicios no se considera necesario.

En quinto lugar se propone mejorar la personalización de la página creando en el perfil con campos adicionales que guarden las opciones de personalización como el tipo de letra o fondo de los usuarios. También se podría aprovechar para crear fondos personalizados, los cuales se desbloqueasen al resolver ejercicios, ofreciendo así premios a los usuarios.

Bibliografía

[1] E. S. Roberts, *Thinking Recursively: With Examples in Java.* John Wiley & Sons, Inc., 2005.

[2] M. Rubio-Sánchez, *Introduction to recursive programming.* CRC Press, 2017.

[3] D. Ginat and E. Shifroni, "Teaching recursion in a procedural environment—how much should we emphasize the computing model?" *SIGCSE Bull.*, vol. 31, no. 1, p. 127–131, mar 1999. [Online]. Available: https://doi.org/10.1145/384266.299718

[4] A. Pérez Carrasco and A. Velázquez Iturbide, "Sistema de animación interactiva de la recursividad srec 1.1," 2010.

[5] J. F. Pérez, "Visback: Herramienta para la visualización de algoritmos de backtracking."

[6] D. Fossati, "Chiqat: An intelligent tutoring system for learning computer science," *Qatar Foundation Annual Research Forum Proceedings*, p. ICTP 020, 11 2013.

[7] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace, 2009.

[8] D. S. Foundation, "Django." [Online]. Available: https://djangoproject.com

[9] kjo msft, "Introduction to visual studio code - training — microsoft learn," 2023. [Online]. Available: https://learn.microsoft.com/en-us/training/modules/introduction-to-visual-studio-code/

[10] M. M. W. David Axmark, Allan Larsson, "Mysql," 1995. [Online]. Available: https://www.mysql.com/

[11] *¿Qué es Git? - Azure DevOps — Microsoft Learn.*

[12] L. L. Svekis, M. van Putten, and C. By Rob Percival, *JavaScript from Beginner to Professional: Learn JavaScript quickly by building fun, interactive, and dynamic web apps, games, and pages.* Packt Publishing, 2021.

[13] B. Frain, *Responsive Web Design with HTML5 and CSS: Build future-proof responsive websites using the latest HTML5 and CSS techniques.* Packt Publishing, 2022.

[14] P. J. Maida Esteban Gabriel, "Metodologías de desarrollo de software," Ph.D. dissertation, Cátedra: Seminario de Sistemas, 2015.

[15] Y. Zhou, "Unix process, merging unified process and extreme programming to benefit software development practice," in *2009 First International Workshop on Education Technology and Computer Science*, vol. 3, 2009, pp. 699–702.

[16] M. Krief, *Learning DevOps: A comprehensive guide to accelerating DevOps culture adoption with Terraform, Azure DevOps, Kubernetes, and Jenkins*, 2022.

[17] *¿Qué es DevOps? - Azure DevOps — Microsoft Learn.*

[18] I. Caponetto, J. Earp, and M. Ott, "Gamification and education: A literature review," in *European Conference on Games Based Learning*, vol. 1. Academic Conferences International Limited, 2014, p. 50.

[19] L. Rello and R. Baeza-Yates, "The effect of font type on screen readability by people with dyslexia," *ACM Trans. Access. Comput.*, vol. 8, no. 4, may 2016. [Online]. Available: https://doi.org/10.1145/2897736

[20] ——, "Good fonts for dyslexia," in *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2513383.2513447

[21] E. T. Hvannberg, E. L.-C. Law, and M. K. Lérusdóttir, "Heuristic evaluation: Comparing ways of finding and reporting usability problems," *Interacting with Computers*, vol. 19, no. 2, pp. 225–240, 2007.

Anexos

Código A.1: Código de la potencia con descomposición exponente - 1.

```
1     def power_method(base, exponent):
2         if exponent == 0:
3             return 1
4         return base * power_method(base, exponent-1)
```

A

Apéndice A: Códigos implementados en los ejercicios

A.1. Potencia

La potencia es la operación matemática utilizada para multiplicar un número, siendo este denominado base, tantas veces como indique un número denominado exponente.

A.1.1. Restar uno

El código A.1 es el utilizado para resolver la potencia en tiempo lineal, restando uno al número hasta alcanzar el exponente cero, momento en el que terminamos.

A.1.2. Dividir entre dos

El código A.2 resuelve el ejercicio de potencias en tiempo logarítmico, la descomposición es dividir a la mitad, en este caso elevaremos el exponente al cuadrado y si es impar por la base

Código A.2: Código de la potencia con descomposición exponente / 2.

```
    def power_half(base, exponent):
        if exponent == 0:
            return 1
        else:
            if exponent % 2 == 1:
                return power_half(base, exponent//2) ** 2 *
                    base
            else:
                return power_half(base, exponent//2) ** 2
```

Código A.3: Código de la suma de una lista extrayendo el primer elemento como descomposición.

```
    def add_list_ini(lista):
        if len(lista) == 0:
            return 0
        return lista[0] + add_list_ini(lista[1:])
```

Código A.4: Código de la suma de una lista extrayendo el último elemento como descomposición.

```
    def add_list_fin(lista):
        if len(lista) == 0:
            return 0
        return lista[-1] + add_list_fin(lista[:-1])
```

una última vez.

A.2. Sumar los elementos de una lista

Es un ejercicio de listas, en el cual se sumarán los elementos de una lista.

A.2.1. Eliminar el primero

El código A.3 resuelve la suma en tiempo lineal, la descomposición extrae el primer elemento y lo suma con el resultado de la llamada.

Código A.5: Código de la suma de los elementos de la lista subdividiendo la lista en dos.

```
    def add_list_half(lista):
        if len(lista) == 0:
            return 0
        elif len(lista) == 1:
            return lista[0]
        return add_list_half(lista[:len(lista)//2]) +
           add_list_half(lista[len(lista)//2:])
```

Código A.6: Código de la comparación de cadenas extrayendo el primer elemento de ambas listas como descomposición.

```
    def same_string_ini(first, second):
        if first == "" or second == "":
            return first == second
        return first[0] == second[0] and same_string_ini(first
           [1:], second[1:])
```

A.2.2. Eliminar el último

El código A.4 obtiene la solución al ejercicio de sumar los elementos de una lista en tiempo lineal, la descomposición es extraer el último elemento y sumarlo con el resultado de la llamada recursiva.

A.2.3. Partir a la mitad

El código A.5 calcula el resultado de la suma de los elementos de una lista sumando la llamada de dividir en dos la lista, la profundidad del árbol es logarítmica, pero el número de hijos es lineal.

A.3. Misma cadena

Es un ejercicio en el que se comprobará si dos cadenas de caracteres son la misma.

A.3.1. Eliminar el primero

El código A.6 genera como solución un valor de verdad comparando el primer elemento de la cadena extraído en la descomposición y hace el *and* con el resto. El tiempo es lineal.

Código A.7: Código de la comparación de cadenas extrayendo el último elemento de ambas listas como descomposición.

```
    def same_string_fin(first, second):
        if first == "" or second == "":
            return first == second
        return first[-1] == second[-1] and same_string_fin(
           first[:-1], second[:-1])
```

Código A.8: Código de la comparación de cadenas subdividiéndolas a la mitad.

```
    def same_string_half(first, second):
        if first == "" or second == "":
            return first == second
        if len(first) == 1 or len(second) == 1:
            return first == second
        return same_string_half(first[:len(first)//2], second[:
           len(second)//2]) and same_string_half(first[len(
           first)//2:], second[len(second)//2:])
```

A.3.2. Eliminar el último

El código A.7 obtiene la solución de hacer el *and* de la descomposición comparando los últimos elementos con el valor de verdad de la llamada recursiva. Se calcula en tiempo lineal.

A.3.3. Partir a la mitad

El código A.8 calcula el resultado del *and* utilizando los valores de verdad de la comparación de ambas mitades. La profundidad es logarítmica pero el número de nodos es lineal.

A.4. Decimal a binario

En este ejercicio se obtiene la representación binaria del número en base diez.

A.4.1. Dividir entre dos

El código A.9 hace la representación binaria del número en base diez al descomponer realizando la división entera entre dos del número. El tiempo es logarítmico.

Código A.9: Código de la generación de la representación binaria de un número en base diez al descomponer haciendo la división entera del número.

```
    def decimal_to_binary(n):
        if n < 2:
            return n
        else:
            return decimal_to_binary(n//2) * 10 + (n % 2)
```

Código A.10: Código que indica si un dígito está dentro de un número descomponiéndolo de diez en diez.

```
    def digit_in_number(numero, digito):
        if numero < 10:
            return numero == digito
        else:
            return digit_in_number(numero//10, digito) or
                numero % 10 == digito
```

A.5. Dígito en un número

Este ejercicio busca encontrar si un dígito se encuentra dentro de un número.

A.5.1. Dividir entre diez

El código A.10 averigua si un dígito se encuentra dentro de un número al comparar si el resto de dividir entre 10 es ese número o ya estaba en la llamada recursiva. El tiempo es lineal.

A.6. Palíndromo

Este ejercicio busca averiguar si una palabra es palíndroma, es decir se lee igual de principio a fin que al revés.

A.6.1. Eliminar el primero y el último

El código A.11 averigua si una palabra es palíndroma comparando el primer y último elemento con el *and* de la llamada recursiva. El tiempo es lineal a pesar de que sean la mitad de llamadas, dado que los factores multiplicativos no se cuentan para la complejidad.

Código A.11: Código que indica si una palabra es palíndroma.

```
    def palindrome(palabra):
        if len(palabra) <= 1:
            return True
        else:
            return palindrome(palabra[1:-1]) and palabra[0] ==
                palabra[-1]
```

Código A.12: Código que calcula la representación en base n de un dígito en base diez.

```
    def decimal_to_n_ary(n, base):
        if n < base:
            return n
        else:
            return decimal_to_n_ary(n//base, base) * 10 + (n %
                base)
```

Código A.13: Código que calcula la operación del factorial restando uno.

```
    def factorial(n):
        if n == 0:
            return 1
        else:
            return factorial(n-1) * n
```

A.7. Decimal a base n

Este ejercicio genera la representación n-aria hasta nueve de un dígito en base diez.

A.7.1. Dividir entre diez

El código A.12 utiliza el resto de la división entre la base y lo añade al final multiplicando la llamada por diez para crear la representación del número.

A.8. Factorial

Este ejercicio calcula la operación matemática del factorial de un número.

Código A.14: Código del mayor elemento de una lista extrayendo el primer elemento como descomposición

```
    def greater_in_list_ini(lista):
        if len(lista) == 1:
            return lista[0]
        return max(lista[0], greater_in_list_ini(lista[1:]))
```

Código A.15: Código del mayor elemento de una lista extrayendo el último elemento como descomposición.

```
    def greater_in_list_fin(lista):
        if len(lista) == 1:
            return lista[0]
        return max(lista[-1], greater_in_list_fin(lista[:-1]))
```

A.8.1. Restar uno

El código A.13 calcula el factorial multiplicando el numero por el factorial del número entero inmediatamente más pequeño. El tiempo es lineal.

A.9. Mayor de los elementos de una lista

Es un ejercicio de listas, en el cual se averigua el mayor de los elementos.

A.9.1. Eliminar el primero

El código A.14 resuelve el ejercicio en tiempo lineal, la descomposición extrae el primer elemento y lo compara con el resultado del mayor de la lista. Se resuelve en tiempo lineal.

A.9.2. Eliminar el último

El código A.15 obtiene la solución al ejercicio comparando el último con el mayor de la llamada. Resuelve el ejercicio en tiempo lineal. Se resuelve en tiempo lineal.

A.9.3. Partir a la mitad

El código A.16 calcula la solución comparando el resultado de la llamada de la primera mitad con el del la segunda. La profundidad del árbol es logarítmica pero el número de nodos es lineal.

Código A.16: Código del mayor de los elementos de una lista subdividiendo la lista en dos.

```
    def greater_in_list_half(lista):
        if len(lista) == 1:
            return lista[0]
        return max(greater_in_list_half(lista[:len(lista)//2]),
            greater_in_list_half(lista[len(lista)//2:]))
```

Código A.17: Código de invertir los caracteres de una cadena extrayendo el primer carácter como descomposición.

```
    def inverse_string_ini(word):
        if len(word) <= 1:
            return word
        return inverse_string_ini(word[1:]) + word[0]
```

Código A.18: Código de invertir los caracteres de una cadena extrayendo el último carácter como descomposición.

```
    def inverse_string_fin(word):
        if len(word) <= 1:
            return word
        return word[-1] + inverse_string_fin(word[:-1])
```

A.10. Invertir una cadena

El objetivo del ejercicio es invertir la cadena proporcionada.

A.10.1. Eliminar el primero

El código A.17 obtiene la solución al extraer el primer elemento y concatenarlo al final del resultado de la llamada recursiva. Se resuelve en tiempo lineal.

A.10.2. Eliminar el último

El código A.18 genera la cadena inversa al concatenar al principio el último elemento con el resultado de la llamada recursiva. Se resuelve en tiempo lineal.

Código A.19: Código del mayor de los elementos de una lista subdividiendo la lista en dos.

```
    def inverse_string_half(word):
        if len(word) <= 1:
            return word
        return inverse_string_half(word[len(word)//2:]) +
           inverse_string_half(word[:len(word)//2])
```

A.10.3. Partir a la mitad

El código A.19 realiza la concatenación del resultado de la llamada usando la mitad final con el obtenido utilizando la mitad inicial. La profundidad del árbol es logarítmica pero el número de nodos es lineal.

A.11. Ordenar una lista

Son los ejercicios de ordenar una lista, hay muchos métodos posibles.

A.11.1. merge sort

Es uno de los ejercicios de ordenar listas. Se utilizará el algoritmo del *merge sort*, el cual consiste en separar en dos mitades la lista, ordenarlas y recomponerlas en orden.

El código A.20 ordena la lista separándola a la mitad y luego mezcla ambas mitades escogiendo el menor elemento de las dos reiteradamente hasta ordenar la lista.

A.11.2. insert sort

Es uno de los ejercicios de ordenar listas. Se utilizará el algoritmo del *insert sort*, el cual consiste en quitar en colocar en el orden el último elemento.

El código A.21 ordena la lista extrayendo el último elemento e insertándolo en la posición que le corresponde en la lista ordenada de la llamada recursiva.

A.11.3. quick sort

Es uno de los ejercicios de ordenar listas. Se utilizará el algoritmo del *quick sort*, el cual consiste en elegir un pivote y colocar los elementos menores a la izquierda y los mayores a la derecha.

El código A.22 ordena la lista eligiendo un pivote mediante la partición hoare, tras lo cual solo tiene que llamar con la mitad inicial y final para ordenar los elementos en ambos lados.

Código A.20: Código que genera la ordenación de una lista mediante el algoritmo merge sort.

```
    def mezclar(lista1, lista2):
        lista_final = []
        while len(lista1) > 0 and len(lista2) > 0:
            if lista1[0] > lista2[0]:
                lista_final.append(lista2.pop(0))
            else:
                lista_final.append(lista1.pop(0))
    lista_final = lista_final + lista1 + lista2
    return lista_final


    def merge_sort(lista):
        if len(lista) <= 1:
            return lista
        else:
            return mezclar(merge_sort(lista[:len(lista)//2]),
               merge_sort(lista[len(lista)//2:]))
```

Código A.21: Código que genera la ordenación de una lista mediante el algoritmo insert sort.

```
    def insert_sort(lista):
        if len(lista) <= 1:
            return lista
        else:
            listado = insert_sort(lista[:-1])
            return colocar(lista[-1], listado)


    def colocar(numero, lista):
        i = len(lista)
        lista.append(numero)
        while i > 0 and numero < lista[i-1]:
            lista[i] = lista[i-1]
            lista[i-1] = numero
            i -= 1
        return lista
```

A.11.4. select sort

Es uno de los ejercicios de ordenar listas. Se utilizará el algoritmo del *select sort*, el cual consiste en extraer el menor elemento de la lista e insertarlo en la posición deseada.

Código A.22: Código que genera la ordenación de una lista mediante el algoritmo quick sort.

```
    def quick_sort(a):
        quick_sort_index(a, 0, len(a) - 1)

    def particion_Hoare(a, ini, fin):
        if fin >= 0:
            mitad = (ini + fin) // 2
            pivote = a[mitad]
            a[mitad] = a[ini]
            a[ini] = pivote
            izqa = ini + 1
            dcha = fin
            ha_terminado = False
            while not ha_terminado:
                while izqa <= fin and a[izqa] <= pivote:
                    izqa = izqa + 1
                while a[dcha] > pivote:
                    dcha = dcha - 1
                if izqa < dcha:
                    aux = a[izqa]
                    a[izqa] = a[dcha]
                    a[dcha] = aux
                ha_terminado = izqa > dcha

            a[ini] = a[dcha]
            a[dcha] = pivote
            return dcha

    def quick_sort_index(a, ini, fin):
        if ini < fin:
            indice_pivote = particion_Hoare(a, ini, fin)
            quick_sort_index(a, ini, indice_pivote - 1)
            quick_sort_index(a, indice_pivote + 1, fin)
```

Código A.23: Código que genera la ordenación de una lista mediante el algoritmo select sort.

```
    def select_sort(int_list):
        if len(int_list) <= 1:
            return int_list
        int_list.remove(min(int_list))
        return [min(int_list)] + select_sort(int_list)
```

El código A.23 ordena la lista extrayendo el menor elemento y concatenándolo al principio de la lista ordenada por la llamada recursiva.

B

Apéndice B: Ficheros de configuración

B.1. Fichero de configuración Apache

El fichero B.1 es el archivo de configuración del servidor **Apache**, se pone el fichero de ejemplo en el que se modifican los campos con los de la máquina que estamos utilizando como servidor, también se añadirán las rutas de las carpetas de los archivos estáticos y la del proyecto. También se tiene que configurar la ruta del archivo WSGI el cual es el encargado de correr la aplicación y la del entorno de **Python**.

B.2. Fichero de configuración MySql

El fichero B.2 es el archivo de configuración del servidor **MySql**, se necesita la dirección de destino, el nombre del usuario, el nombre de la base de datos a utilizar y el puerto en el que escucha.

El fichero B.3 es el archivo de configuración del servidor **MySql**, se añadiran el destino, puerto, nombre de la base de datos, usuario y la contraseña **Python**.

Código B.1: Archivo de configuración de Apache.

```
<VirtualHost *:80>

    ServerAdmin admin@your-domain.com
    ServerName your-domain.com
    ServerAlias www.your-domain.com

    DocumentRoot /var/www/django_project/

    ErrorLog ${APACHE_LOG_DIR}/your-domain.com_error.log
    CustomLog ${APACHE_LOG_DIR}/your-domain.com_access.log
       combined

    Alias /static /var/www/django_project/static
    <Directory /var/www/django_project/static>
        Require all granted
    </Directory>

    Alias /media /var/www/django_project/media
    <Directory /var/www/django_project/media>
        Require all granted
    </Directory>

    <Directory /var/www/django_project/django_app>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>

    WSGIDaemonProcess django_app python-path=/var/www/
       django_project python-home=/var/www/django_project/
       django_env
    WSGIProcessGroup django_app
    WSGIScriptAlias / /var/www/django_project/django_app/
       wsgi.py

</VirtualHost>
```

Código B.2: Archivo de configuración de Mysql datos.

```
[my_service]
host=localhost
user=USER
dbname=NAME
port=5432
```

Código B.3: Archivo de configuración de Mysql contraseña.

```
1    localhost:5432:NAME:USER:PASSWORD
```