



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de un tutorial interactivo para el aprendizaje de
Prolog

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: García Muñoz, Carlos

Tutor/a: Vidal Oriola, Germán Francisco

CURSO ACADÉMICO: 2023/2024

Resumen

Este proyecto pretende ayudar a aquellas personas que se adentran por primera vez en la programación lógica, ofreciéndoles una introducción paso a paso, ejemplos claros y prácticos, y un entorno amigable para aprender los conceptos fundamentales de Prolog. Para ello, se pretende desarrollar una aplicación web con la finalidad de que los usuarios puedan encontrar un tutorial interactivo que explique los distintos aspectos del lenguaje Prolog acompañados de ejemplos y ejercicios. Además, se hace uso de un intérprete integrado en la aplicación web que permite la ejecución de código en tiempo real.

El proceso de desarrollo de este proyecto se ha llevado a cabo siguiendo metodologías ágiles, lo que ha permitido añadir funcionalidades a la aplicación al finalizar cada sprint. Con el objetivo de lograr una interfaz moderna y fácil de usar, se utilizaron herramientas como React, empleando principalmente HTML, CSS y JavaScript. También se utilizaron otro tipo de tecnologías durante el proyecto, como puede ser el sistema de control de versiones Git o Visual Studio Code como entorno de desarrollo.

Palabras clave: Prolog, metodologías, aplicación web, programación lógica.

Abstract

This project aims to assist those who are delving into logic programming for the first time, offering them a step-by-step introduction, clear and practical examples, and a user-friendly environment to learn the fundamental concepts of Prolog. To achieve this, a web application is being developed to provide users with an interactive tutorial explaining various aspects of the Prolog language accompanied by examples and exercises. Additionally, an interpreter integrated into the web application allows real-time code execution.

The development process of this project has been carried out following agile methodologies, enabling the addition of functionalities to the application at the end of each sprint. In order to achieve a modern and easy-to-use interface, tools like React were utilized, primarily employing HTML, CSS, and JavaScript. Other technologies such as the version control system Git and Visual Studio Code were also employed during the project development.

Keywords: Prolog, methodologies, web application, logic programming.



Índice de contenidos

1.	Introducción	8
1.1.	Motivación	8
1.2.	Objetivos.....	9
1.3.	Estructura	9
2.	Estado del arte	11
2.1.	Aplicaciones similares.....	11
2.2.	Propuesta	12
3.	Metodología	14
4.	Análisis del problema	16
4.1.	Casos de uso	16
4.2.	Análisis DAFO	19
4.3.	Prototipado	21
5.	Lenguaje Prolog	23
5.1.	Introducción a Prolog	23
5.2.	Términos.....	23
5.3.	Cláusulas.....	24
5.4.	Hechos	25
5.5.	Reglas	25
5.6.	Consultas.....	27
5.7.	Negación	27
5.8.	Definición de predicado.....	28
5.9.	Programa.....	28
5.10.	Inferencia.....	29
5.11.	Listas	30
5.12.	Operadores Aritméticos	30
5.13.	Operadores Relacionales	31
5.14.	Operadores de listas	31
5.15.	Otros operadores	32
6.	Diseño de la solución	33
6.1.	Arquitectura del Sistema.....	33
6.2.	Estructura del proyecto	34
6.2.1.	Components	35

6.2.2.	Assets.....	35
6.2.3.	Utils.....	35
6.2.4.	Interfaces.....	35
7.	Desarrollo de la solución propuesta.....	36
7.1.	Organización en sprints	36
7.1.1.	<i>Sprint 0</i>	37
7.1.2.	<i>Sprint 1</i>	37
7.1.3.	<i>Sprint 2</i>	38
7.1.4.	<i>Sprint 3</i>	40
7.1.5.	<i>Sprint 4</i>	41
7.2.	Tecnologías utilizadas.....	44
7.2.1.	HTML.....	44
7.2.2.	CSS	44
7.2.3.	JavaScript.....	45
7.2.4.	GitHub	45
7.2.5.	Git.....	46
7.2.6.	React	46
7.2.7.	Visual Studio Code.....	47
7.2.8.	Figma.....	47
8.	Pruebas.....	51
8.1	Cuestionario de usabilidad.....	51
9.	Conclusiones	53
9.1.	Conclusiones.....	53
9.2.	Relación del trabajo desarrollado con los estudios cursados	54
9.3.	Trabajo futuro	54
10.	Referencias.....	55
Anexo A.	Objetivos de Desarrollo Sostenible	56

Índice de figuras

Figura 1. Sitio web de SWI-Prolog	13
Figura 2. Sitio web de Learn Prolog Now.....	13
Figura 3. Sitio web de W3schools	13
Figura 4. Diagrama de casos de uso	16
Figura 5. Esquema DAFO	19
Figura 6. Wireframe de una sección	21
Figura 7. Wireframe de un cuestionario	22
Figura 8. Visualización de las secciones de la aplicación	38
Figura 9. Interfaz del intérprete del lenguaje Prolog en la aplicación.....	39
Figura 10. Página de registro de usuario	40
Figura 11. Barra superior con la funcionalidad de inicio de sesión y registro	41
Figura 12. Barra superior cuando un usuario ha iniciado sesión	41
Figura 13. Cuestionario de cada sección de la aplicación.....	42
Figura 14. Ejercicio de completar código sin respuesta	43
Figura 15. Ejercicio de completar código con respuesta.....	43
Figura 16. Página de registro de usuario	48
Figura 17. Página del contenido de una sección.....	49
Figura 18. Página de un ejercicio de una sección contestado correctamente.....	49
Figura 19. Página de un cuestionario de una sección contestado erróneamente.....	50
Figura 20. Distintos umbrales en la System Usability Scale.....	52

Índice de tablas

Tabla 1. US01, registro de usuario	17
Tabla 2. US02, inicio de sesión	17
Tabla 3. US03, cerrar sesión actual	17
Tabla 4. US04, visualizar secciones	18
Tabla 5. US05, uso del intérprete del lenguaje	18
Tabla 6. US06, realizar los cuestionarios	18
Tabla 7. US07, interfaz adaptable según el tamaño de ventana	18
Tabla 8. Operadores aritméticos más utilizados en Prolog.....	30
Tabla 9. Operadores relacionales más utilizados en Prolog.....	31
Tabla 10. Operadores de listas más utilizados en Prolog.....	31
Tabla 11. Operadores extralógicos en Prolog.....	32
Tabla 12. Sprints del desarrollo de la aplicación.....	36
Tabla 13. User Stories del Sprint 1	37
Tabla 14. User Stories del Sprint 2.....	38
Tabla 15. User Stories del Sprint 3.....	40
Tabla 16. User Stories del Sprint 4.....	41
Tabla 17. Puntuación SUS de cada usuario	52



1. Introducción

Prolog, abreviatura de "**Programming in Logic**", es un lenguaje de programación lógico que, en contraste con los lenguajes convencionales como C++ o Java, sigue un paradigma lógico-declarativo, donde los programas se estructuran alrededor de reglas lógicas y relaciones.

Prolog se basa en investigaciones realizadas por científicos de la computación en Europa en los años 60 y 70, especialmente en las Universidades de Marsella, Londres y Edimburgo. Su primer desarrollo se realizó en la Universidad de Marsella durante los años 70. El desarrollo posterior en la Universidad de Edimburgo llevó a una versión estándar de facto, ahora conocida como Edinburgh Prolog.

La singularidad de Prolog radica en su capacidad para resolver problemas a partir de la lógica y la inferencia. Al escribir programas en Prolog, los desarrolladores describen las relaciones entre entidades y establecen reglas lógicas, permitiendo que el sistema derive soluciones a partir de consultas o preguntas planteadas.

Prolog tiene aplicaciones en una variedad de campos, destacando su relevancia en la Inteligencia Artificial y el procesamiento de lenguaje natural. Aunque es un lenguaje de propósito general, su enfoque principal es la resolución de problemas simbólicos más que la realización de cálculos numéricos.

1.1. Motivación

La motivación de este proyecto surge de intentar paliar la dificultad de aprender a programar en un lenguaje de programación lógico como Prolog, el cual pertenece a un paradigma de programación distinto al de los lenguajes de programación convencionales. En este lenguaje, en lugar de instrucciones secuenciales, tenemos relaciones lógicas definidas mediante hechos y reglas, permitiendo inferencias a partir de estas relaciones.

Aunque existe una oferta abundante de aplicaciones web de calidad destinadas al aprendizaje de diversos lenguajes de programación, se observa una ausencia de recursos especializados en Prolog, lo que destaca como una oportunidad de desarrollo en el campo de la educación y la formación en programación.

Por otro lado, la creación de una aplicación web desde el principio, pasando por todas las etapas de su proceso de desarrollo, supone un gran ejercicio de aprendizaje.

1.2. Objetivos

El principal objetivo del proyecto se basa en desarrollar una aplicación web que facilite el aprendizaje del lenguaje Prolog. Esta herramienta busca proporcionar una experiencia de aprendizaje práctica, accesible y atractiva para usuarios principiantes que fortalezca la comprensión de este lenguaje.

Este objetivo puede desglosarse en los siguientes objetivos más concretos:

- **Lecciones estructuradas:** En la aplicación, el contenido estará dividido en bloques o secciones. Cada parte se centrará en conceptos específicos de Prolog, comenzando con los fundamentos, como la sintaxis y la semántica, para luego ir avanzando a temas más complejos.
- **Ejercicios:** Cada bloque debe incluir ejemplos prácticos que ilustren los conceptos en cuestión. Estos ejemplos pueden ser problemas resueltos paso a paso o ejercicios que los usuarios puedan realizar para aplicar lo aprendido.
- **Interactividad:** La aplicación web proporcionará un entorno interactivo donde los usuarios puedan practicar directamente con el lenguaje Prolog, realizando consultas, experimentando con ejemplos y recibiendo retroalimentación en tiempo real gracias al uso de un intérprete del lenguaje.
- **Interfaz fácil de usar:** Se desarrollará una interfaz de usuario atractiva e intuitiva para proporcionar una buena experiencia de usuario y facilite la utilización de la aplicación.

1.3. Estructura

A continuación, se establece la organización que se seguirá en este documento, exponiendo los puntos principales junto con una concisa explicación de cada uno de ellos.

En el segundo capítulo, se hará una revisión sobre el estado del arte, explorando detalladamente las alternativas más relevantes que compiten con nuestro proyecto actualmente. A partir de este análisis, podremos elaborar una propuesta sólida para las funcionalidades que planeamos implementar en nuestro proyecto, asegurándonos de que estén alineadas con las necesidades y las expectativas de nuestros usuarios potenciales.

En el tercer capítulo se detalla la metodología Scrum empleada en el proyecto, explicando los roles clave, las herramientas y los eventos que se llevan a cabo en esta metodología para organizar y priorizar el trabajo.

Seguidamente, en el cuarto capítulo, se presentarán las actividades realizadas durante la fase inicial de definición de requisitos de la aplicación. Por tanto, este capítulo sentará las bases para el desarrollo posterior del proyecto, asegurando una comprensión clara y una planificación sólida para abordar eficazmente el proyecto planteado.

En el quinto capítulo, se llevará a cabo una exposición detallada y didáctica sobre los fundamentos y conceptos esenciales del lenguaje de programación Prolog, esto resulta fundamental en el contexto de la aplicación ya que el propósito principal de la misma es el aprendizaje de Prolog. Por tanto, este capítulo servirá como una guía introductoria para aquellos lectores que no están familiarizados con Prolog, proporcionando una mejor comprensión del proyecto.

En el capítulo sexto, se mostrará el diseño, aspecto y arquitectura de la aplicación, así como los patrones aplicados y el porqué de ellos. Además, se comentarán las tecnologías utilizadas durante el desarrollo del proyecto.

Pasando al séptimo capítulo, se comentará la forma en que se estructuró el proyecto así como el trabajo realizado durante cada sprint.

En el capítulo dedicado a las pruebas, se llevará a cabo una evaluación de la aplicación mediante el uso de un cuestionario de usabilidad. Los participantes en las pruebas utilizarán la aplicación en condiciones simuladas o reales, y completarán el cuestionario proporcionando respuestas a preguntas específicas sobre su experiencia.

En el último capítulo, se presentarán las conclusiones del proyecto. También se analizarán las implicaciones de estos resultados en relación con los objetivos iniciales del proyecto y se ofrecerá una visión prospectiva del posible trabajo futuro que podría realizarse en el mismo, señalando áreas de mejora o posibles extensiones del proyecto que podrían explorarse. Por último, se discutirá la relación entre el proyecto y los estudios cursados, destacando cómo los conocimientos adquiridos durante los estudios han influido en el desarrollo del proyecto, y cómo este trabajo se alinea con los objetivos y competencias del plan de estudios.

2. Estado del arte

En este capítulo, analizaremos las alternativas a nuestra aplicación, evaluando tanto sus ventajas como sus desventajas. En primer lugar, si bien ya existen aplicaciones similares, nuestra aplicación no busca simplemente replicar lo existente, sino innovar en la forma en que se presenta y se enseña el lenguaje a través de una estructura didáctica cuidadosamente diseñada y mediante el uso de ejemplos y ejercicios prácticos, proporcionando una herramienta que resulte interesante para el público objetivo, tanto estudiantes como profesionales.

2.1. Aplicaciones similares

Tutorial de SWI-Prolog: SWI-Prolog nos ofrece un tutorial¹ (Figura 1) en línea mediante una herramienta llamada SWISH. El tutorial está dividido por apartados que explican distintos conceptos del lenguaje con ejemplos predefinidos que se pueden ejecutar para ver el resultado.

La parte derecha de la página web es un intérprete del lenguaje Prolog donde se puede escribir hechos, reglas y consultas. Además, el tutorial tiene un apartado de ejercicios, aunque actualmente no tiene contenido, lo que es ciertamente un posible aspecto de mejora de la aplicación.

Learn Prolog Now: Learn Prolog Now² (Figura 2) es un sitio web para el aprendizaje de Prolog que divide el contenido en distintos capítulos. Un aspecto positivo que destacar de esta aplicación es que al final de cada capítulo hay muchos ejercicios relacionados con el tema tratado en el capítulo.

Como aspectos negativos, se puede destacar la falta de ejemplos durante la explicación de los conceptos del lenguaje y la falta de interactividad al no disponer de un intérprete para los ejemplos o ejercicios.

Por último, la interfaz de usuario del sitio web no resulta especialmente atractiva visualmente (por aspectos del diseño como la tipografía o la paleta de colores utilizada), lo cual puede dificultar el aprendizaje, además de darle un aspecto obsoleto a la aplicación.

¹ <https://swish.swi-prolog.org>

² <https://www.let.rug.nl/bos/lpn//index.php>

W3schools: W3schools³ (Figura 3) es un sitio web que, aunque no incluye el aprendizaje del lenguaje Prolog, sí que es una aplicación web para el aprendizaje de muchos otros lenguajes de programación y, por tanto, es una aplicación interesante a tener en cuenta. De las alternativas que se mencionan en esta sección, es la más cercana a los objetivos de nuestro proyecto.

La aplicación presenta el contenido de una forma clara y organizada, con una interfaz de usuario que facilita el aprendizaje, contando además con numerosos ejemplos interactivos mediante el uso de un intérprete.

Por último, los ejercicios de cada apartado son simples y suelen consistir en rellenar trozos de código que faltan en un fragmento de código, con la posibilidad de consultar la solución.

2.2 Propuesta

Al analizar las opciones disponibles podemos deducir que, aunque hay aplicaciones con características parecidas, existen muy pocas aplicaciones dedicadas al aprendizaje del lenguaje Prolog. Además, ninguna ofrece todas las funcionalidades que perseguimos en esta aplicación, lo que brinda la oportunidad para que esta herramienta tenga cabida y se distinga del resto.

Por ejemplo, en el caso de W3schools, a pesar de las múltiples características positivas que hemos comentado anteriormente, como los ejemplos y ejercicios que acompañan las explicaciones o la inclusión de un intérprete del lenguaje, es una aplicación que no incluye el lenguaje Prolog.

Por el contrario, Learn Prolog Now [7], que es un sitio web que sí está dedicado al aprendizaje de Prolog, carece de interactividad al no disponer de un intérprete para los ejemplos o ejercicios.

Por último, el tutorial de SWI-Prolog no dispone de ejercicios o cuestionarios al final de cada apartado que puedan ayudar a la comprensión de los distintos conceptos del lenguaje.

Debido a estas ventajas y desventajas que presentan las aplicaciones mencionadas, se considera adecuado el desarrollo de una aplicación web destinada al aprendizaje de Prolog que tenga todas las características y funcionalidades indispensables para hacer una herramienta de aprendizaje que cumpla los objetivos definidos anteriormente.

³ <https://www.w3schools.com/>

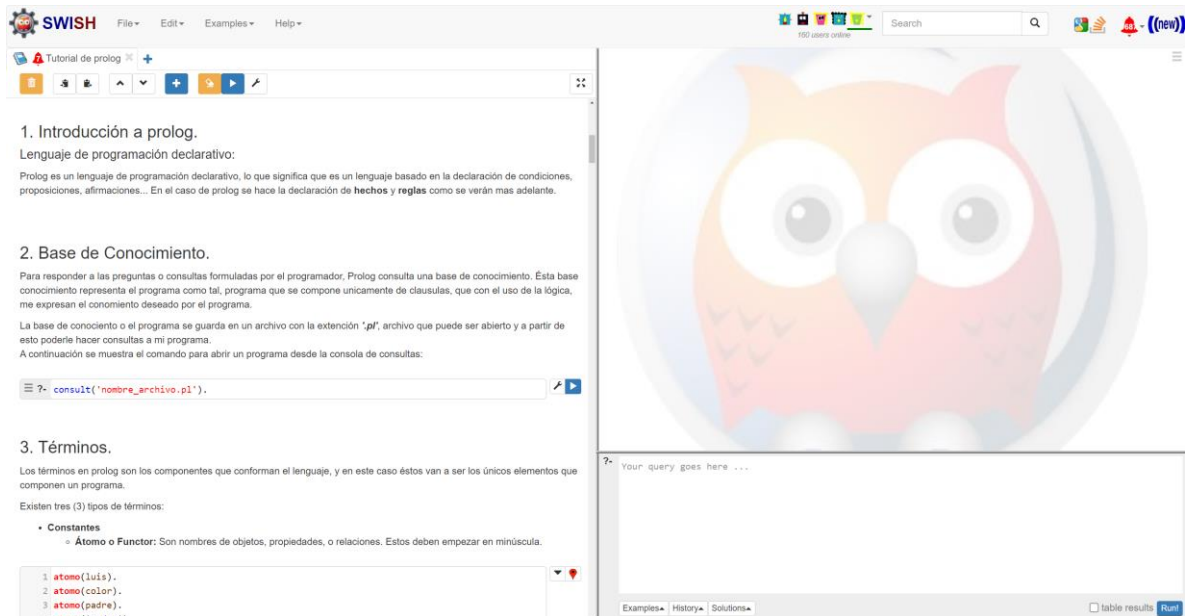


Figura 1. Sitio web de SWI-Prolog



Figura 2. Sitio web de Learn Prolog Now

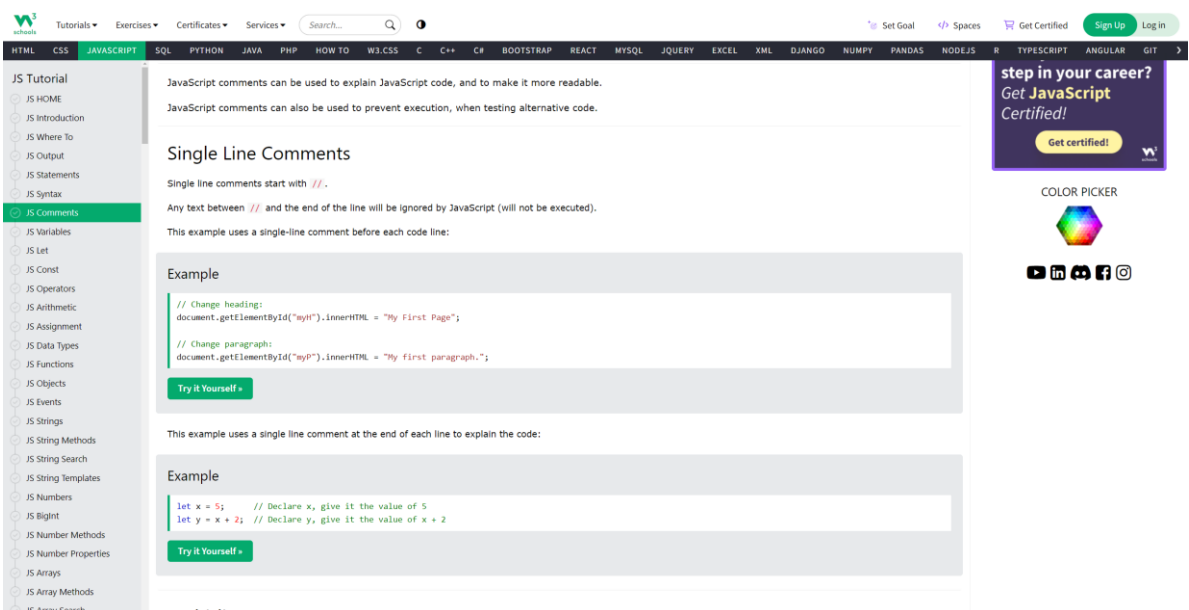


Figura 3. Sitio web de W3schools



3. Metodología

En el contexto del desarrollo de aplicaciones de software, la metodología Scrum emerge como un enfoque ágil muy eficiente. Las metodologías ágiles destacan por su capacidad para adaptarse a cambios en los requisitos en cualquier momento. Scrum, en particular, se distingue por sus principios fundamentales y roles bien definidos que promueven la colaboración efectiva y la optimización del proceso de desarrollo.

En contraste con otras metodologías, las metodologías ágiles destacan por su flexibilidad y su capacidad de resolución de problemas. En oposición a las metodologías tradicionales, el proceso en un proyecto que sigue una metodología ágil no está tan definido inicialmente, siendo más receptivo a posibles cambios en los requisitos en cualquier momento.

En un enfoque tradicional, la resolución de problemas implicaría informar al gerente del proyecto, mientras que, en las metodologías ágiles, un equipo autogestionado toma decisiones bajo la dirección del *Product Owner*.

Las metodologías ágiles se caracterizan por su contribución al final de cada sprint, añadiendo valor funcional al producto al término de cada iteración. Este concepto es inexistente en las metodologías tradicionales, las cuales se centran en un entregable final al concluir el proyecto. Trabajar con varios entregables resulta beneficioso para evaluar la conformidad del producto con las expectativas del cliente y facilita la flexibilidad mencionada anteriormente.

La base de la metodología Scrum se centra en los *sprints*, que son períodos de tiempo cortos y limitados, generalmente no excediendo las cuatro semanas. Durante cada *sprint*, el equipo trabaja para completar una cantidad determinada de trabajo, conocido como *User Stories*. Estas historias de usuario son encargos de trabajo solicitados por el *Product Owner*, que describen de manera informal y general una característica del software desde la perspectiva del usuario final. El propósito de las *User Stories* es articular cómo una característica del software proporcionará valor al cliente.

La planificación del *sprint* es un evento crucial en la metodología Scrum que inicia cada *sprint* y tiene como objetivo definir qué se puede entregar durante el período y cómo se llevará a cabo ese trabajo. Este proceso se realiza en colaboración con todo el equipo, garantizando una comprensión de los objetivos y tareas a abordar.

El *Backlog* es una lista ordenada de tareas que aún no han sido incluidas en un *sprint*, es gestionado constantemente por el *Product Owner* que decide las prioridades del *backlog*, establece criterios de aceptación y se comunica de manera continua con el equipo para asegurar que se estén desarrollando las funcionalidades correctas. Durante cada *sprint*, el equipo de desarrollo selecciona tareas del *backlog* según su capacidad, mediante iteraciones que no son impuestas por el *Product Owner*, permitiendo una flexibilidad y autonomía significativas.

Dentro de la metodología Scrum existen dos roles fundamentales que son el *Scrum Master* y el *Product Owner*. El *Scrum Master* actúa como facilitador y líder del equipo, asegurando que se sigan los principios y prácticas de Scrum de manera efectiva. Además, guía al equipo en la resolución de problemas y la mejora continua, buscando optimizar el proceso de desarrollo y la entrega de valor al cliente.

Por otro lado, el *Product Owner* representa las necesidades y expectativas del cliente o *stakeholders*⁴, tomando decisiones clave sobre el producto, definiendo prioridades y asegurando que las funcionalidades desarrolladas cumplan con las expectativas del cliente.

El tablero Kanban⁵ funciona como el centro de operaciones para el equipo, permitiendo la gestión y visualización del trabajo durante todo el proyecto. El desarrollador accede a este tablero para elegir en qué *User Story* trabajar después de finalizar una tarea o cuando esté disponible, facilitando la organización y la transparencia en el proceso de desarrollo.

Al tratarse de un proyecto individual, todas las responsabilidades recaen en una sola persona, quien asume los distintos roles mencionados anteriormente.

⁴ Los stakeholders de un proyecto son todas las partes interesadas en el resultado o el impacto del proyecto.

⁵ Un tablero Kanban es una herramienta visual para la gestión de tareas que organiza el trabajo en columnas y tarjetas, facilitando el seguimiento y la colaboración del equipo.

4. Análisis del problema

Durante el primer *sprint* del proyecto, se ha realizado la fase de identificación de requisitos de la aplicación, lo cual engloba la definición de casos de uso, un ejercicio de análisis sobre las debilidades, amenazas, fortalezas y oportunidades del proyecto, así como el prototipado de la aplicación.

4.1. Casos de uso

La creación de un diagrama de casos de uso es esencial en el proceso de desarrollo de un proyecto, ya que proporciona una representación visual de los distintos escenarios de interacción entre los actores y el sistema. Estos casos de uso describen las funcionalidades que el sistema ofrece a los usuarios y las acciones que estos pueden llevar a cabo. Además, ayudan a identificar los actores involucrados y los pasos necesarios para realizar cada funcionalidad.

Cada caso de uso representa una unidad de trabajo significativa, delineando claramente las acciones que se deben realizar y los resultados esperados. La representación gráfica de estos casos de uso, mediante el uso de elipses, proporciona una vista de alto nivel del comportamiento observable del sistema como se muestra en la Figura 4.

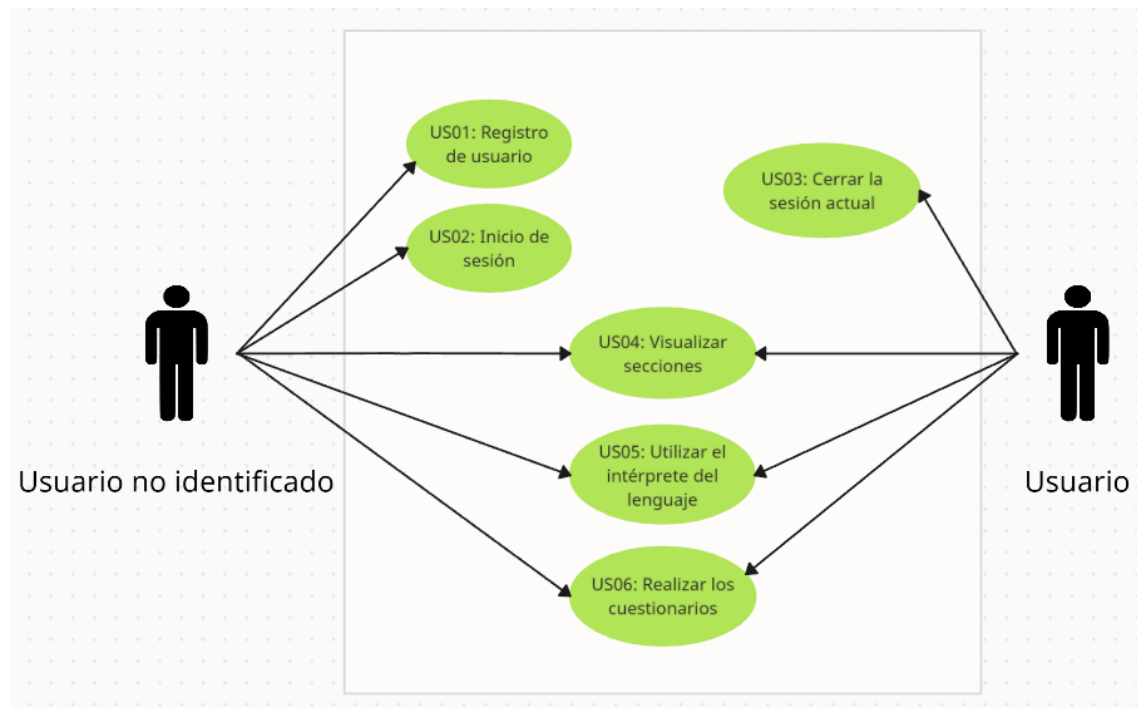


Figura 4. Diagrama de casos de uso

Este proceso de modelado no solo sirve como una herramienta de comunicación con los *stakeholders*, sino que también ayuda a tener claros los requisitos del sistema y a establecer una base sólida para el desarrollo.

Por todo esto, los casos de uso son una parte fundamental del proceso de desarrollo de software, ya que permiten definir y comprender las funcionalidades que el sistema debe ofrecer desde la perspectiva del usuario.

Durante la especificación de los requisitos funcionales de la aplicación se han definido los siguientes casos de uso mostrados anteriormente en el diagrama de casos de uso:

Caso de uso	US01: Registro de usuario
Actor	Usuario no identificado
Descripción	Registra en el sistema de un nuevo usuario
Tipo	Funcional
Entrada	El usuario no identificado rellena todos los campos necesarios para registrar al usuario
Proceso	El sistema registra el nuevo usuario o muestra un mensaje de error si existe algún usuario creado con la misma información
Salida	El sistema notifica al usuario del registro mediante una alerta en pantalla

Tabla 1. US01, registro de usuario

Caso de uso	US02: Inicio de sesión
Actor	Usuario no identificado
Descripción	Inicia sesión de usuario en el sistema
Tipo	Funcional
Entrada	El usuario no identificado escribe usuario y contraseña
Proceso	El sistema identifica con éxito al usuario o muestra un mensaje de error si no encuentra algún usuario creado con la misma información
Salida	El sistema actualiza la interfaz y muestra el nombre del usuario identificado

Tabla 2. US02, inicio de sesión

Caso de uso	US03: Cerrar la sesión actual
Actor	Usuario
Descripción	Cerrar la sesión de un usuario que se encuentra identificado
Tipo	Funcional
Entrada	El usuario identificado pulsa el botón de cierre de sesión
Proceso	El sistema cierra la sesión del usuario
Salida	El sistema actualiza la interfaz y vuelve a mostrar los botones para el inicio de sesión y registro.

Tabla 3. US03, cerrar sesión actual

Caso de uso	US04: Visualizar secciones
Actor	Usuario no identificado, Usuario
Descripción	Visualizar el contenido de las secciones
Tipo	Funcional
Entrada	El usuario accede a las distintas secciones desde la barra de navegación
Proceso	El sistema detecta el evento
Salida	El sistema muestra la sección correspondiente de la página

Tabla 4. US04, visualizar secciones

Caso de uso	US05: Utilizar el intérprete del lenguaje
Actor	Usuario no identificado, Usuario
Descripción	Utilizar el intérprete de Prolog de la web
Tipo	Funcional
Entrada	El usuario escribe en el primer campo los hechos o reglas y en el segundo campo la consulta.
Proceso	El intérprete procesa la entrada mediante el mecanismo de inferencia
Salida	El sistema muestra la respuesta por pantalla

Tabla 5. US05, uso del intérprete del lenguaje

Caso de uso	US06: Realizar los cuestionarios y ejercicios
Actor	Usuario no identificado, Usuario
Descripción	Realizar los cuestionarios y ejercicios de cada sección
Tipo	Funcional
Entrada	El usuario responde a los cuestionarios y ejercicios
Proceso	El sistema registra las respuestas a los cuestionarios y ejercicios
Salida	El sistema muestra la evaluación de las respuestas

Tabla 6. US06, realizar los cuestionarios

También se han especificado requisitos no funcionales para la aplicación, que describen otros aspectos del sistema en lugar de lo que el sistema debe hacer. Para estos requisitos no funcionales se han definido los siguientes casos de uso:

Caso de uso	US07: Interfaz adaptable según el tamaño de ventana
Actor	Usuario no identificado, Usuario
Descripción	La interfaz de la página web se adapta según el tamaño de la ventana
Tipo	No funcional
Entrada	El usuario cambia el tamaño de ventana de la página
Proceso	El sistema detecta el cambio de tamaño de ventana
Salida	El sistema adapta la interfaz de la página web

Tabla 7. US07, interfaz adaptable según el tamaño de ventana

4.2. Análisis DAFO

El análisis DAFO (Debilidades, Amenazas, Fortalezas y Oportunidades) es una forma de análisis muy útil para valorar el estado de un proyecto en términos de factores internos y externos.

Las fortalezas son aspectos internos positivos, como recursos relevantes y personal capacitado. Las debilidades son aspectos internos negativos, como procesos ineficientes o falta de recursos. Las oportunidades son factores externos que una entidad puede aprovechar, como cambios en el mercado o avances tecnológicos. Las amenazas son factores externos que tienen un impacto negativo, como la competencia intensa o las crisis económicas.

Este análisis brinda una visión completa para tomar decisiones de manera informada y poder desarrollar estrategias efectivas. Identificar y capitalizar las fortalezas y oportunidades, así como abordar debilidades y amenazas, son elementos clave en la planificación estratégica de un proyecto.

A continuación, se expone el análisis DAFO [9] de este proyecto, el cual también se muestra en la Figura 5.

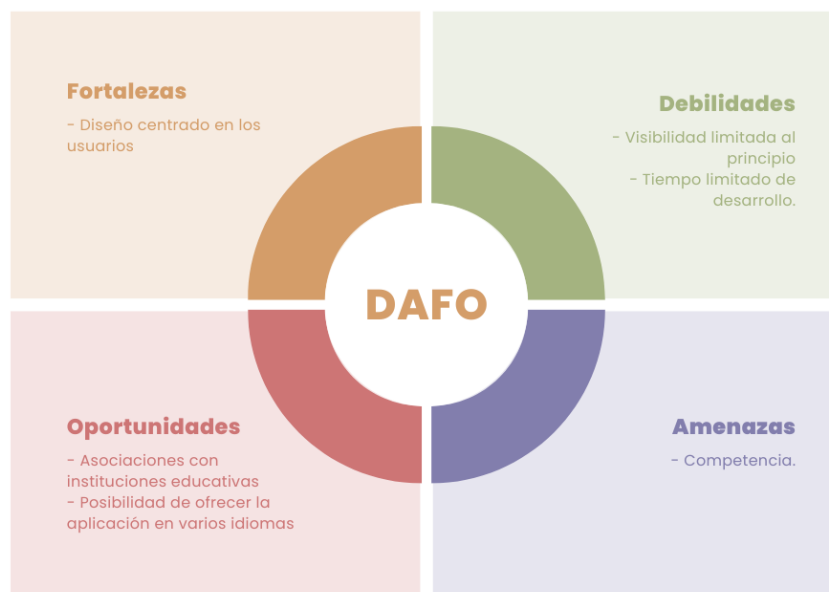


Figura 5. Esquema DAFO

- **Debilidades**
 - **Visibilidad limitada al principio:** Dado que es una plataforma reciente, podría encontrarse con un desafío al intentar aparecer en los resultados de los motores de búsqueda. Además, una visibilidad limitada también puede influir en la percepción del público sobre la credibilidad y utilidad de la aplicación, ya que los usuarios tienden a confiar más en plataformas que son conocidas y utilizadas por muchos usuarios.

- **Tiempo limitado de desarrollo:** Al tratarse de un trabajo individual con un plazo definido, se agrega una complejidad adicional, puesto que se cuenta con recursos y tiempo limitados. Esto implica la necesidad de gestionar eficientemente el tiempo y los medios disponibles para cumplir con la fecha de entrega establecida.

- **Amenazas**
 - **Competencia:** La existencia de otras aplicaciones web con el mismo propósito puede dispersar la atención de los usuarios, llevándolos a considerar diferentes opciones antes de comprometerse con una plataforma en particular. Para abordar esto, es crucial diferenciarnos mediante características únicas, como puede ser, por ejemplo, una interfaz amigable y eficiente.

- **Fortalezas**
 - **Diseño centrado en los usuarios:** La aplicación se concibe con la premisa de proporcionar una experiencia de uso sencilla y ajustada a las necesidades específicas de los estudiantes en su proceso de aprendizaje. La atención se centra en la optimización de la usabilidad, garantizando así que el proceso de aprendizaje sea fluido y eficiente, con la capacidad de adaptarse a diversas preferencias y estilos de aprendizaje.

- **Oportunidades**
 - **Asociaciones con instituciones educativas:** Una asociación con una institución educativa, como una universidad, puede desempeñar un papel fundamental en la promoción del uso de una aplicación de aprendizaje de Prolog. Al colaborar estrechamente con estas instituciones, la aplicación puede beneficiarse de una mayor visibilidad entre los estudiantes y profesores. Esta asociación puede traducirse en la creación de servicios exclusivos para los estudiantes de la universidad, como contenido educativo personalizado, acceso preferente a recursos y una integración más estrecha con el plan de estudios académico.

 - **Posibilidad de ofrecer la aplicación en varios idiomas:** Al brindar una interfaz multilingüe, no solo facilitamos el acceso a usuarios de diferentes regiones y culturas, sino que también mejoramos la accesibilidad global. Esto amplía nuestra audiencia y aumenta la inclusividad y la adaptabilidad de la aplicación, generando un mayor atractivo para usuarios de diversas nacionalidades. Además, también fortalece la experiencia de usuario, fomentando una mayor retención y satisfacción entre los usuarios existentes y potenciales.

4.3. Prototipado

El prototipado [10] es una técnica utilizada en el desarrollo de software que implica la creación de versiones preliminares o modelos parciales de un sistema o producto antes de su implementación completa. Estos prototipos son representaciones tangibles y visuales que permiten a los desarrolladores, diseñadores y usuarios obtener una vista previa y comprender mejor el aspecto y la funcionalidad del producto final.

El objetivo principal del prototipado es facilitar la comunicación con los *stakeholders* para identificar y abordar rápidamente posibles problemas, ajustar requisitos y mejorar la comprensión mutua de las expectativas del proyecto.

Existen diferentes enfoques de prototipado, como el prototipado de baja fidelidad, que se centra en representaciones simples y rápidas, y el prototipado de alta fidelidad, que busca crear modelos más detallados y cercanos al producto final.

En este proyecto se ha decidido utilizar un prototipado de baja fidelidad, también llamado *Wireframe* [11], que es una representación visual simplificada y estructurada de una interfaz de usuario que se utiliza en las etapas iniciales del diseño. Para crear estos *Wireframes*, se ha utilizado la herramienta Figma [12].

El *Wireframe* que se puede ver en la Figura 6 proporciona un esquema visual de cómo se presenta la información en las diferentes secciones de la aplicación y cómo se organiza esta sección en el contexto de la interfaz. Además, se observa que en la barra lateral izquierda se han integrado las diversas secciones disponibles para la navegación, lo que permite que los usuarios tengan acceso de forma fácil y rápida a diferentes partes del contenido.

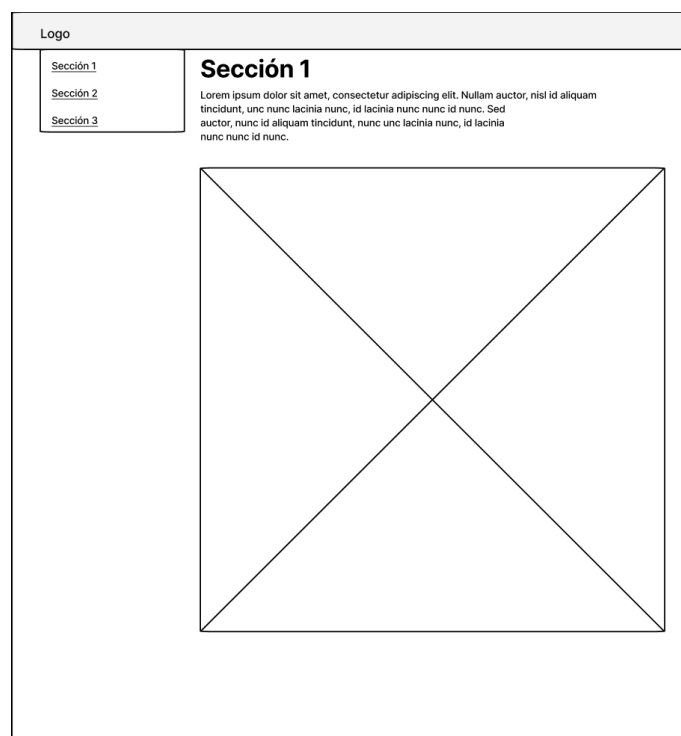
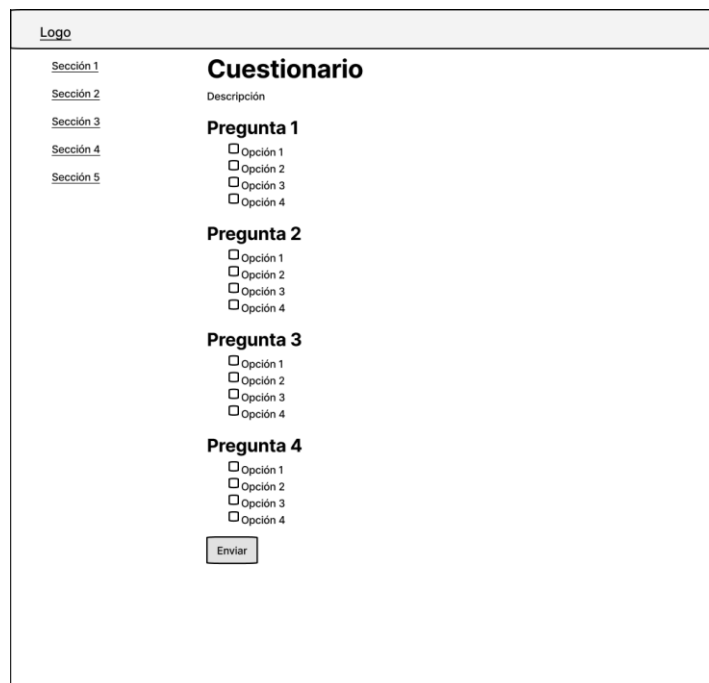


Figura 6. Wireframe de una sección

A continuación, el *Wireframe* que se puede ver en la Figura 7 muestra un cuestionario donde se observa en primer lugar el título del cuestionario, seguido de una breve descripción introductoria sobre el mismo. A continuación, vemos una serie de preguntas numeradas, cada una acompañada de un campo de respuesta, que podría ser un cuadro de texto, una lista desplegable o botones de opción, dependiendo del formato de la pregunta.

En la parte inferior, se incorpora un botón de "Enviar" para que los usuarios puedan finalizar y enviar sus respuestas. Además, se incluirá algún mensaje de confirmación después de enviar el cuestionario para proporcionar retroalimentación sobre el estado de la acción realizada, pudiendo aparecer también la puntuación lograda y qué preguntas fueron contestadas correctamente y cuáles incorrectamente.



Logo

Sección 1
Sección 2
Sección 3
Sección 4
Sección 5

Cuestionario

Descripción

Pregunta 1

- Opción 1
- Opción 2
- Opción 3
- Opción 4

Pregunta 2

- Opción 1
- Opción 2
- Opción 3
- Opción 4

Pregunta 3

- Opción 1
- Opción 2
- Opción 3
- Opción 4

Pregunta 4

- Opción 1
- Opción 2
- Opción 3
- Opción 4

Enviar

Figura 7. *Wireframe* de un cuestionario

5. Lenguaje Prolog

5.1. Introducción a Prolog

Prolog es un lenguaje de programación declarativo⁶, donde los programas se escriben en términos de relaciones y reglas lógicas, en lugar de instrucciones imperativas típicas de otros lenguajes de programación. Su sintaxis se basa en predicados, lo que permite realizar afirmaciones sobre objetos y las relaciones entre ellos.

La estructura básica de un programa Prolog consiste en la definición de una serie de hechos y reglas. Estos hechos y reglas conforman una base de conocimiento que se puede consultar y utilizar para inferir respuestas a consultas.

5.2. Términos

Los términos en Prolog son la unidad básica de construcción utilizada para representar datos y expresiones en el programa. A su vez, los términos pueden estar formados por átomos, variables y estructuras.

- **Átomo:** En Prolog, un átomo es un término que representa un nombre simbólico o constante. Los átomos son básicamente identificadores sin argumentos y pueden consistir en letras, dígitos y subrayados. Los átomos en Prolog deben empezar en minúscula. Por ejemplo, `cebra` o `verde` son átomos.
- **Variable:** Las variables en Prolog deben empezar en mayúscula o con `'_'`.

Una **variable anónima** se representa por el identificador `'_'` y se usa comúnmente cuando no necesitamos hacer referencia a esa variable específica.

- **Estructura:** Es la forma en que se definen términos más complejos. Su sintaxis es la siguiente:

átomo(t₁, t₂, ..., t_n).

⁶ Un lenguaje declarativo define qué se quiere lograr en lugar de cómo hacerlo, delegando la implementación de una solución al sistema.

Donde el átomo recibe el nombre de **functor** y t_1, t_2, \dots, t_n son a su vez términos.

Por ejemplo, para la estructura `persona(juan, 25, ingeniero)`, "persona" es el functor, y sus términos son "juan", "25", e "ingeniero". Esta estructura podría representar información sobre una persona, donde "juan" es el nombre, "25" es la edad, e "ingeniero" es su profesión.

5.3. Cláusulas

Como cláusula podemos identificar un átomo que se define como $p :- t_1, t_2, \dots, t_n$ donde p es un símbolo de predicado y t_1, t_2, \dots, t_n son términos. Además, una cláusula también es un literal, que puede ser un átomo o su negación (haciendo uso de "not").

Una cláusula es la unidad básica de construcción en un programa Prolog. Una cláusula puede ser un hecho, una regla o una consulta, y su estructura está determinada por la sintaxis del lenguaje.

$h :- q_1, q_2, \dots, q_n.$

donde h es la **cabeza** de la cláusula (un átomo) y q_1, q_2, \dots, q_n es el **cuerpo** (una secuencia de literales).

Átomo: $p :- t_1, t_2, \dots, t_n$ Donde p es un símbolo de predicado) y t_1, t_2, \dots, t_n son términos

Literal: un átomo o su negación (haciendo uso de "not")

Existen tres tipos distintos de cláusulas:

- Una cláusula que tiene cabeza y no tiene cuerpo se llama **hecho**.
- Una cláusula que tiene cabeza y tiene cuerpo se llama **regla**.
- Una cláusula que no tiene cabeza y tiene cuerpo se llama **consulta**.

Un ejemplo de cláusula es el siguiente:

`come(A,B) :- carnivoro(A), herbivoro(B), masFuerte(A, B).`

Esta cláusula reescrita en lenguaje natural se leería de la siguiente manera:

"A come a B si A es carnívoro y B es herbívoro y A es más fuerte que B."

5.4. Hechos

En Prolog, un hecho es una afirmación o una declaración que establece una relación que se considera verdadera sin ninguna condición adicional.

Los hechos son la forma más básica de representar información en Prolog. La sintaxis de un hecho es la siguiente:

predicado(t₁, t₂, ..., t_n).

donde el predicado es un átomo que describe una propiedad o relación y los argumentos t₁, t₂, ..., t_n son términos. Los hechos pueden emplearse de varias formas:

- **Propiedades:** En el contexto de la programación lógica, los objetos pueden tener propiedades que describen sus características. Por ejemplo:

```
color(azul).  
planeta(marte).
```

- **Relaciones:** Establecen conexiones lógicas o asociaciones entre elementos. Por ejemplo:

```
padre(antonio, luis).  
edad(jose, 30).
```

En conclusión, los hechos en Prolog son fundamentales para construir la base de conocimiento con la cual el programa puede realizar consultas y realizar inferencias lógicas. Además, estos hechos pueden combinarse con reglas y consultas para modelar situaciones más complejas y resolver problemas específicos.

5.5. Reglas

Una regla en Prolog es una construcción lógica que establece una relación entre diferentes predicados mediante condiciones o antecedentes.

Una regla representa la implicación:

q₁, q₂, ..., q_n -> h

Las reglas permiten definir relaciones más complejas y derivar nuevas afirmaciones a partir de información existente en la base de conocimiento.

Al ejecutar una consulta en Prolog, el motor de inferencia utiliza estas reglas para determinar si se cumplen las condiciones necesarias y, en su caso, deducir nuevos hechos para responder a la consulta realizada.

En Prolog, la secuencia q_1, q_2, \dots, q_n representa la **conjunción** $q_1 \wedge q_2 \wedge \dots \wedge q_n$. Por ejemplo:

```
% Una persona es propietaria de un gato si es humana y tiene un gato
es_propietario_de_gato(X) :- humano(X), tiene_gato(X).
```

Además, con el objeto de proporcionar cláusulas más compactas, Prolog también nos permite representar disyunciones empleando el operador `;`. Por ejemplo:

```
% Una persona tiene una mascota si es humana y tiene un gato o un perro.
tiene_mascota(X) :- humano(X), (tiene_gato(X) ; tiene_perro(X)).

% Este ejemplo se puede reescribir con dos cláusulas de la siguiente manera:
tiene_mascota(X) :- humano(X), tiene_gato(X).
tiene_mascota(X) :- humano(X), tiene_perro(X).
```

En Prolog, una regla puede ser **recursiva** si se llama a sí misma, ya sea de manera directa (**p** llama a **p**) o mediante predicados intermedios (**p** llama a **q** y **q** llama a **p**). Por tanto, la recursividad en Prolog se basa en la idea de dividir un problema en subproblemas más pequeños y resolverlos de manera incremental.

Aquí hay un ejemplo práctico de una regla recursiva en Prolog para el cálculo del factorial:

```
factorial(0, 1).
factorial(N, Solución1) :-
    N > 0, N1 is N - 1, factorial(N1, Solucion2), Solución1 is N * Solución2.
```

Al realizar consultas como **factorial(5, F)**, Prolog utilizará la definición recursiva y obtendrá el factorial de 5 de forma incremental.

5.6. Consultas

En Prolog, una consulta es una cláusula que se utiliza para hacer preguntas a la base de conocimiento definida en el programa. La sintaxis de una consulta es similar a la de una cláusula sin cabeza, pero se utiliza un símbolo de interrogación seguido de un guion en lugar de ':-'. Por ejemplo:

?- gusta(juan, libro).

En esta consulta, se está preguntando si es cierto que a Juan le gusta un libro. El sistema Prolog buscará en la base de conocimiento y responderá con "sí" o "no", o bien con los valores correspondientes si hubiera variables en la consulta empleando para ello un mecanismo de inferencia.

Las consultas permiten obtener información específica de la base de conocimiento, realizar verificaciones lógicas y explorar las relaciones definidas en el programa. Es mediante las consultas que se interactúa con el sistema para obtener respuestas lógicas basadas en las reglas y hechos previamente establecidos.

5.7. Negación

La negación en Prolog se implementa principalmente mediante la técnica conocida como "negación por fallo". Esta técnica se basa en el hecho de que si Prolog no puede probar una afirmación, entonces asume que la afirmación es falsa.

La negación por fallo funciona bajo el supuesto de que el conjunto de hechos y reglas en la base de conocimiento es completo, ya que cualquier cosa que no pueda probarse es asumida como falsa, lo cual puede no ser siempre correcto si la base de conocimiento está incompleta.

Por ejemplo, supongamos que queremos verificar si no hay un hecho `hijo(juan, _)` que esté presente en la base de conocimiento. Podríamos hacerlo de la siguiente manera:

```
come(juan, pizza).
```

```
come(pedro, ensalada).
```

```
?- not come(juan, ensalada).
```

En este ejemplo, Prolog intenta probar `come(juan, ensalada)`. Como no puede encontrar tal hecho, la negación por fallo hace que la consulta devuelva `true`.



5.8. Definición de predicado

La definición de un predicado en Prolog se refiere al conjunto de cláusulas que tienen el mismo predicado en el átomo de la cabeza. En otras palabras, un predicado en Prolog está compuesto por todas las reglas y hechos que comparten el mismo nombre de predicado en la parte izquierda (cabeza) de la cláusula.

Por ejemplo, dado el siguiente conjunto de cláusulas que definen el predicado `hijo`:

```
hijo(juan, maria).  
hijo(pedro, maria).  
hijo(luis, ana).
```

En este caso, las tres cláusulas tienen el mismo predicado `hijo` en el átomo de la cabeza, por lo que todas estas cláusulas forman parte de la definición del predicado `hijo`.

Las definiciones de predicados pueden incluir tanto hechos como reglas. Por ejemplo:

```
hijo(X, maria) :- padre(juan, X).
```

En esta regla, también se está definiendo el predicado `hijo`, pero utilizando una regla en lugar de un hecho. Todas las cláusulas que tienen `hijo` como el átomo de la cabeza formarían parte de la definición del predicado `hijo`.

La definición de un predicado en Prolog es fundamental para establecer las relaciones lógicas entre los objetos y para que el intérprete de Prolog pueda realizar inferencias lógicas al responder consultas.

5.9. Programa

Un programa en Prolog es un conjunto de definiciones de predicado que definen un conjunto de relaciones lógicas entre objetos en lugar de especificar instrucciones de control de flujo como en otros lenguajes de programación, y está compuesto por un conjunto de cláusulas, que pueden ser hechos o reglas, organizadas de manera que el intérprete de Prolog pueda utilizarlas para responder consultas.

La ejecución de un programa en Prolog implica realizar inferencias lógicas para responder consultas basadas en las reglas y hechos definidos en el programa.

5.10. Inferencia

La inferencia es el proceso mediante el cual el intérprete de Prolog deduce soluciones a partir de las reglas y hechos definidos en el programa. La inferencia en Prolog se basa en el mecanismo de resolución.

Cuando se realiza una consulta en Prolog, el intérprete busca en la base de conocimiento de hechos y reglas para encontrar soluciones que satisfagan la consulta. Utiliza el mecanismo de unificación y el algoritmo de resolución para buscar coincidencias entre la consulta y las reglas definidas, con el objetivo de encontrar valores para las variables presentes en la consulta.

Resolución: La resolución en Prolog se refiere al proceso mediante el cual se busca la solución a una consulta, y funciona de la siguiente manera:

- Prolog toma la consulta que se desea resolver.
- Intenta satisfacer esa consulta mediante la búsqueda de cláusulas en la base de conocimiento que puedan unificar con la consulta.
- Si encuentra una cláusula que unifica con la consulta, Prolog la utiliza y continúa resolviendo cualquier subconsulta que pueda surgir.
- Si no puede encontrar una cláusula que coincida con la consulta actual, intenta retroceder (*backtracking*) para buscar otras cláusulas que puedan coincidir.
- Este proceso continúa hasta que todas las consultas se satisfacen o no hay más cláusulas que coincidan.

Unificación: La unificación es el proceso que determina si dos términos pueden considerarse iguales y, en caso afirmativo, cómo pueden coincidir. En Prolog, la unificación se realiza mediante el intento de hacer coincidir las estructuras de los términos y las variables dentro de ellos.

- Dos términos se pueden unificar si son iguales, o si contienen variables que se pueden asignar de tal manera que los términos se vuelvan idénticos.
- Las variables se consideran "ligadas" cuando se les asigna un valor durante el proceso de unificación. Si dos términos contienen la misma variable, Prolog intentará encontrar una asignación que haga que esas variables se unifiquen.
- La unificación es el mecanismo que permite a Prolog sustituir variables por valores concretos y comparar términos eficientemente.

En resumen, la resolución implica la búsqueda de cláusulas que coincidan con una consulta dada utilizando el proceso de unificación para determinar si dos términos pueden considerarse iguales y, en caso afirmativo, cómo pueden coincidir. Este proceso se repite recursivamente hasta que todas las consultas se satisfacen o no hay más cláusulas que coincidan.



5.11. Listas

En Prolog, una lista es una estructura de datos fundamental que permite almacenar una secuencia ordenada de elementos. Se representa como una secuencia de elementos separados por comas y encerrados entre corchetes [].

Las listas están formadas recursivamente por una cabeza, que es el primer elemento de la lista y una cola, que es una lista del resto de elementos. Por ejemplo, para la lista [1, 2, 3], la cabeza es 1 y la cola es [2, 3]. Así, la lista [1,2,3] podría representarse también como [1 | [2,3]].

Los elementos de una lista pueden ser cualquier término válido en Prolog, incluyendo átomos, números, variables, otras listas, o incluso estructuras complejas.

Las listas son una herramienta muy útil en Prolog y se utilizan ampliamente en la manipulación de datos, la representación de estructuras de datos y la implementación de algoritmos.

5.12. Operadores Aritméticos

Los operadores aritméticos en Prolog son cruciales para realizar operaciones matemáticas y comparar expresiones numéricas dentro del código. Permiten realizar sumas, restas, multiplicaciones, divisiones y otras operaciones básicas, así como también comparar la igualdad o desigualdad entre expresiones numéricas. Estos operadores son esenciales para el desarrollo de algoritmos y la manipulación de datos en Prolog, ya que permiten realizar cálculos y tomar decisiones basadas en valores numéricos.

A continuación, la Tabla 8 muestra los operadores aritméticos más utilizados en Prolog:

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/ y //	División real y División entera
^ y **	Potencia
mod	Resto de la división (módulo)

Tabla 8. Operadores aritméticos más utilizados en Prolog

5.13. Operadores Relacionales

Los operadores relacionales en Prolog son esenciales para comparar términos y establecer condiciones en las reglas y consultas a la base de conocimiento. Permiten realizar acciones como verificar la igualdad o desigualdad de términos, así como establecer relaciones de orden entre ellos. Estos operadores son fundamentales para la lógica de programación en Prolog, ya que permiten definir restricciones y condiciones que guían el comportamiento del programa.

A continuación, la Tabla 12 muestra los operadores relacionales más comunes en Prolog:

Operador	Significado
= , is	Unificación
==	Igualdad aritmética
\==	Desigualdad aritmética
<	Menor que
=<	Menor o igual que
>	Mayor que
>=	Mayor o igual que

Tabla 9. Operadores relacionales más utilizados en Prolog

5.14. Operadores de listas

En Prolog, los operadores para manipular listas son fundamentales para realizar diversas operaciones sobre este tipo de estructuras de datos. Estos operadores permiten determinar la longitud de una lista, verificar la presencia de elementos, concatenar listas o invertir el orden de los elementos. Algunos de los operadores de listas están predefinidos aunque otros se suelen encontrar en algún módulo como *lists* en SWI-Prolog. A continuación, la Tabla 13 muestra los operadores de listas más comunes en Prolog:

Operador	Significado
length	Determina la longitud de una lista
member	Verifica si un elemento está presente en una lista
append	Concatena dos listas
sort	Ordena los elementos dentro de una lista
reverse	Invierte el orden de los elementos en una lista
is_list	Comprueba si es una lista

Tabla 10. Operadores de listas más utilizados en Prolog

5.15. Otros operadores

En Prolog existen otro tipo de predicados extralógicos que no están definidos dentro del marco lógico del lenguaje y pueden incluir, por ejemplo, operaciones de entrada/salida o manipulación de archivos. A continuación, la Tabla 14 muestra algunos de los operadores extralógicos que existen en Prolog:

Operador	Significado
!	Es el operador de corte que se utiliza para controlar el <i>backtracking</i> .
assert	Se utiliza para agregar hechos a la base de conocimiento de Prolog durante la ejecución del programa.
retract	Se utiliza para eliminar un hecho de la base de conocimiento.
findall	Se utiliza para recopilar todas las soluciones válidas de una consulta en una lista.
write	Se utiliza para imprimir un término en la salida estándar
writeln	Similar a <i>write</i> pero siempre agrega una nueva línea al final.
read	Se utiliza para leer una entrada desde la entrada estándar.
open, close	Se utilizan para abrir y cerrar archivos respectivamente.

Tabla 11. Operadores extralógicos en Prolog

6. Diseño de la solución

Habiendo explicado ya previamente los requisitos funcionales y no funcionales que debe cumplir la aplicación, pasamos a comentar su diseño.

El presente capítulo tiene como objetivo repasar la arquitectura en la que se basa la aplicación y la estructura detallada en la que se organizará el proyecto.

6.1. Arquitectura del Sistema

El patrón Modelo-Vista-Controlador (MVC) es una arquitectura de software que separa la aplicación en tres componentes principales: el Modelo, la Vista y el Controlador.

A continuación se explica cada uno de ellos:

1. Modelo:

- El modelo representa la estructura de datos de la aplicación y la lógica subyacente que manipula esos datos.
- La responsabilidad del modelo es gestionar la manipulación y la integridad de los datos, así como realizar cualquier operación relacionada con la lógica de negocio.
- Por ejemplo, en el caso de nuestra aplicación, el modelo puede incluir las clases y métodos para crear, leer o actualizar los datos de usuario, así como la lógica para validar la entrada del usuario.

2. Vista:

- La vista es la interfaz de usuario que presenta los datos al usuario y permite la interacción.
- Es responsabilidad de la vista mostrar los datos de una manera comprensible y atractiva al usuario final.
- La vista no realiza ninguna manipulación de datos ni contiene lógica de negocio, simplemente muestra la información proporcionada por el modelo.
- Por ejemplo, en el caso de nuestra aplicación, la vista incluye lo que se mostraría en la página web compuesto por archivos HTML y CSS (texto, formularios, botones, etc).



3. Controlador:

- El controlador actúa como intermediario entre el modelo y la vista.
- Responde a las acciones del usuario en la interfaz de usuario, como pulsar un botón o enviar un formulario.
- Luego, el controlador interpreta estas acciones y realiza las operaciones correspondientes en el modelo.
- Una vez que se actualizan los datos en el modelo, el controlador se asegura de que la vista se actualice adecuadamente para reflejar estos cambios.
- Por ejemplo, en el caso de nuestra aplicación, el controlador podría manejar la lógica detrás de responder a un formulario o navegar entre las distintas secciones de la aplicación.

En resumen, el patrón Modelo-Vista-Controlador (MVC) proporciona una forma organizada y modular de diseñar aplicaciones, separando claramente las capas de datos, presentación y control. Esto facilita el desarrollo, la sostenibilidad y la escalabilidad del software al dividir la aplicación en componentes independientes y reutilizables.

6.2. Estructura del proyecto

En este proyecto, se ha organizado el código de una manera lógica y fácil de seguir. A continuación vamos a mostrar la estructura de directorios del proyecto:

- **Components:** Un componente es una pieza fundamental de nuestra aplicación web; es un módulo independiente que encapsula tanto la lógica como la interfaz de usuario de una parte específica de la aplicación web. Dentro de esta carpeta encontramos:
 - La carpeta "**Elements**" contiene componentes pequeños y reutilizables que cumplen una función crucial a la hora de mantener la consistencia de diseño y funcionalidad en toda la aplicación. Estos componentes, como botones o iconos, se utilizan en múltiples áreas para agilizar el desarrollo y mejorar la experiencia del usuario.
 - Una carpeta para cada componente principal del proyecto, donde cada uno realiza una tarea específica.
- **Assets:** Aquí almacenamos todas las imágenes utilizadas en la aplicación y también algunos estilos CSS.
- **Utils:** Está reservado para almacenar funciones y utilidades compartidas en toda la aplicación.
- **Interfaces:** Se utiliza para almacenar todas las interfaces de nuestro proyecto.

6.2.1. Components

Dentro de esta carpeta **components**, podemos encontrar más carpetas, una para cada vista principal en el proyecto. Estas vistas se han separado para facilitar la navegación entre componentes y optimizar el código de manera más sencilla.

En primer lugar, encontramos la primera carpeta:

- **Elements**: Aquí se almacenan todos los componentes básicos y genéricos que se comparten en toda la aplicación. Estos componentes suelen ser elementos visuales simples que se utilizan en varias partes de la interfaz de usuario.
- Luego, están las carpetas con las vistas principales con el nombre descriptivo de la vista. Estas, a su vez, están compuestas por dos carpetas:
 - **Container**: Aquí reside la lógica principal de la página; separar esta lógica en contenedores facilita la modularización y el mantenimiento.
 - **Components**: Son todas las partes que forman el contenedor, donde se gestiona el estado local del componente. Los componentes son útiles porque se pueden reutilizar fácilmente en diferentes partes del sitio web, facilitando la construcción y comprensión.

6.2.2. Assets

Aquí podemos encontrar dos carpetas diferentes:

- **Images**: Contiene imágenes utilizadas en la interfaz de usuario.
- **Styles**: Contiene hojas de estilo CSS o archivos relacionados.

6.2.3. Utils

En la carpeta "**utils**" se encuentran todos los métodos que se utilizan múltiples veces. Estas utilidades suelen ser funciones auxiliares que no están directamente relacionadas con componentes específicos o con la gestión del estado global, pero que son necesarias en diversos lugares del código.

6.2.4. Interfaces

Aquí, como se describió anteriormente, se encuentran todas las interfaces generales utilizadas en el proyecto. Las interfaces son las que definen la forma de un objeto, proporcionando el conjunto de propiedades que tiene.



7. Desarrollo de la solución propuesta

En este capítulo se ofrece una visión completa de la metodología utilizada durante el desarrollo de la aplicación, junto con una explicación detallada de los cuatro sprints en los que se ha estructurado este trabajo.

7.1. Organización en sprints

Para llevar a cabo este proyecto, hemos decidido emplear la metodología ágil Scrum, dedicando un determinado tiempo a cada *User Story* (US), que nos permita mantener un adecuado desarrollo de la aplicación.

Durante este período, hemos desarrollado la aplicación en *sprints* de un mes de duración cada uno, incorporando las funcionalidades más importantes en el momento oportuno, pero asegurándonos siempre de tener una aplicación funcional.

Estos *sprints* están detallados en la tabla 12, junto con sus fechas de inicio y finalización:

<i>Sprint</i>	Fecha inicial	Fecha final
<i>Sprint 0</i>	08/02/2024	22/02/2024
<i>Sprint 1</i>	22/02/2024	14/03/2024
<i>Sprint 2</i>	14/03/2024	28/03/2024
<i>Sprint 3</i>	28/03/2024	11/04/2024
<i>Sprint 4</i>	11/04/2024	25/04/2024

Tabla 12. *Sprints* del desarrollo de la aplicación

A continuación, veremos detalladamente los progresos realizados en cada sprint, junto con una evaluación individual de los resultados obtenidos.

7.1.1. *Sprint 0*

Este *sprint*, que actúa como una etapa inicial del proyecto, estuvo principalmente enfocado en realizar un análisis exhaustivo de la viabilidad de la aplicación. Se abordaron puntos críticos que son fundamentales en las fases iniciales de cualquier proyecto, incluyendo el análisis DAFO para evaluar las fortalezas, debilidades, oportunidades y amenazas, así como el desarrollo de prototipos para visualizar conceptos y posibles soluciones.

Una vez completadas estas tareas, se procedió a definir detalladamente los requisitos funcionales y no funcionales de la aplicación. Esto implica identificar las características y capacidades esenciales que la aplicación debe poseer, así como cualquier restricción técnica o de rendimiento que deba cumplir.

Para concluir este *sprint*, se tomaron decisiones importantes sobre las tecnologías que se utilizarán durante el desarrollo de la aplicación. Además, se creó el repositorio en GitHub, que serviría como lugar centralizado para el almacenamiento y gestión del código fuente del proyecto. También se preparó el entorno de desarrollo, utilizando herramientas como Visual Studio Code, para garantizar un entorno que esté listo para comenzar a trabajar de manera eficiente en las siguientes etapas del proyecto.

7.1.2. *Sprint 1*

El *sprint 1* tuvo comienzo el 22 de febrero y finalizó el 14 de marzo y se incluyeron las *User Stories* indicadas en la tabla 13:

Identificador de US	Descripción
US04	Visualizar secciones
US07	Interfaz adaptable según el tamaño de ventana

Tabla 13. *User Stories* del *Sprint 1*

En este *sprint* el trabajo se centró en establecer la base de la página web de la aplicación, para lo que se utilizó React aprovechando especialmente las capacidades de enrutamiento proporcionadas por el módulo React Router para facilitar la navegación entre las distintas secciones del sitio.

Una de las características clave implementadas durante este *sprint* fue la integración de una barra lateral izquierda, que sirve como un menú de navegación intuitivo para los usuarios, como se puede ver en la Figura 8. Esta barra lateral permite acceder fácilmente a las diferentes secciones de la aplicación, lo que mejora significativamente la experiencia del usuario y facilita la exploración del contenido.

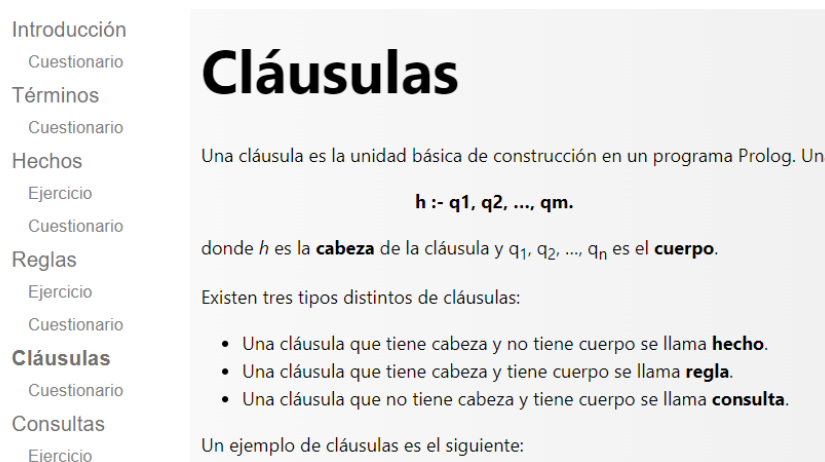


Figura 8. Visualización de las secciones de la aplicación

Además, se puso un énfasis considerable en garantizar que la interfaz de usuario fuera totalmente adaptable y *responsive*, lo que significa que se diseñó y desarrolló de manera que se adapte correctamente a diferentes tamaños de ventana. Esto implica que la aplicación se vea bien y sea fácil de usar tanto en pantallas grandes como en pequeñas, proporcionando una experiencia coherente y satisfactoria para todos los usuarios.

En resumen, el *sprint* 1 fue fundamental para establecer una sólida base técnica y de diseño para la aplicación, incorporando funcionalidades de enrutamiento, navegación y adaptabilidad.

7.1.3. *Sprint* 2

El *sprint* 2 tuvo comienzo el 14 de marzo y finalizó el 28 de marzo y se incluyeron las *User Stories* indicadas en la tabla 14:

Identificador de US	Descripción
US05	Utilizar el intérprete del lenguaje

Tabla 14. *User Stories* del *Sprint* 2

En este *sprint* el trabajo se centró en la implementación de un intérprete del lenguaje Prolog directamente en la aplicación web. Esto fue una fase crucial para la aplicación, ya que permitió integrar ejemplos con código predefinido a lo largo de las secciones para que los usuarios pudieran experimentar y practicar Prolog de manera interactiva sin necesidad de instalar software adicional en sus dispositivos.

Además de la integración del intérprete, se trabajó en el diseño de una interfaz de usuario intuitiva para interactuar con el intérprete de Prolog como se puede ver en la Figura 9. Se implementó un campo para la entrada del código de programa Prolog, otro campo para la consulta y un botón para ejecutar la consulta.

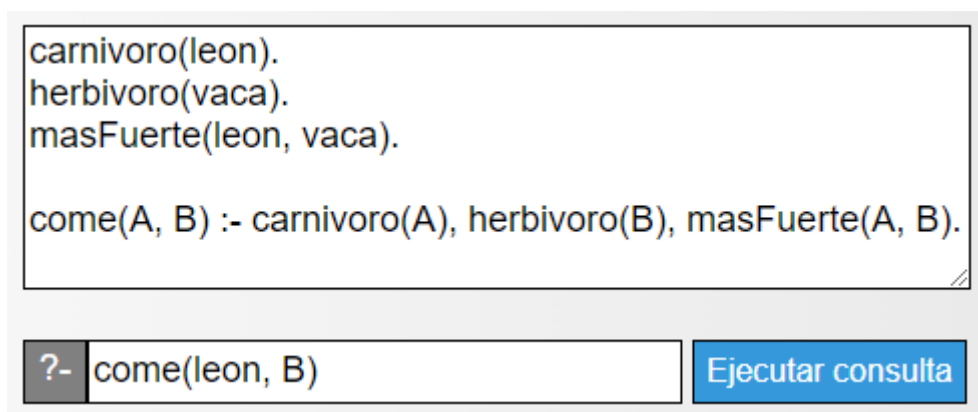


Figura 9. Interfaz del intérprete del lenguaje Prolog en la aplicación

En un principio, se exploró la posibilidad de utilizar *Pengines*⁷ para la implementación del intérprete de Prolog en la aplicación web, que es una tecnología que permite ejecutar código Prolog en un servidor remoto. La idea era aprovechar esta tecnología para ejecutar el código Prolog del usuario en un servidor remoto y luego enviar los resultados de vuelta a la interfaz de usuario en el navegador.

Sin embargo, después de evaluar esta opción, se decidió no seguir adelante con el uso de *Pengines* por varias razones. Una de las principales preocupaciones fue la complejidad y la dependencia de un servidor remoto para ejecutar el código. Esto podría haber introducido problemas de rendimiento, escalabilidad y seguridad, así como una mayor complejidad en el desarrollo y mantenimiento de la aplicación.

Además, el uso de *Pengines* podría haber limitado la capacidad de la aplicación para funcionar sin conexión a Internet, lo que habría afectado negativamente la experiencia del usuario en ciertos escenarios.

Finalmente se optó por utilizar un intérprete contenido en un archivo JavaScript obtenido de un repositorio [13], incluyéndolo en el mismo código fuente del proyecto.

⁷ Sistemas que proporcionan una interfaz para interactuar con programas Prolog a través de solicitudes HTTP, lo que facilita la integración de la lógica de Prolog en aplicaciones web y permite su ejecución remota.

Esto proporcionó una experiencia fluida y sin problemas, eliminando la necesidad de configuraciones complicadas o la instalación de software adicional.

En resumen, el *sprint* 2 fue un hito importante en el desarrollo de la aplicación, ya que permitió la inclusión de un intérprete de Prolog en la aplicación web, brindando a los usuarios la capacidad de practicar y experimentar con el lenguaje de manera interactiva.

7.1.4. *Sprint* 3

El *sprint* 3 tuvo comienzo el 28 de marzo y finalizó el 11 de abril y se incluyeron las *User Stories* indicadas en la tabla 15:

Identificador de US	Descripción
US01	Registro de usuario
US02	Inicio de sesión
US03	Cerrar la sesión actual

Tabla 15. *User Stories* del *Sprint* 3

En el *sprint* 3 del desarrollo de la aplicación el trabajo se enfocó en la implementación de funcionalidades relacionadas con la gestión de usuarios, incluyendo el registro, inicio de sesión y cierre de sesión.

Durante este *sprint*, se diseñó y desarrolló una página dedicada al registro de usuario, donde se pueden registrar nuevos usuarios proporcionando nombre de usuario y contraseña como se puede ver en la Figura 10. Además, se implementaron validaciones para garantizar que no se pudiera crear un nombre de usuario que ya existe durante el proceso de registro.

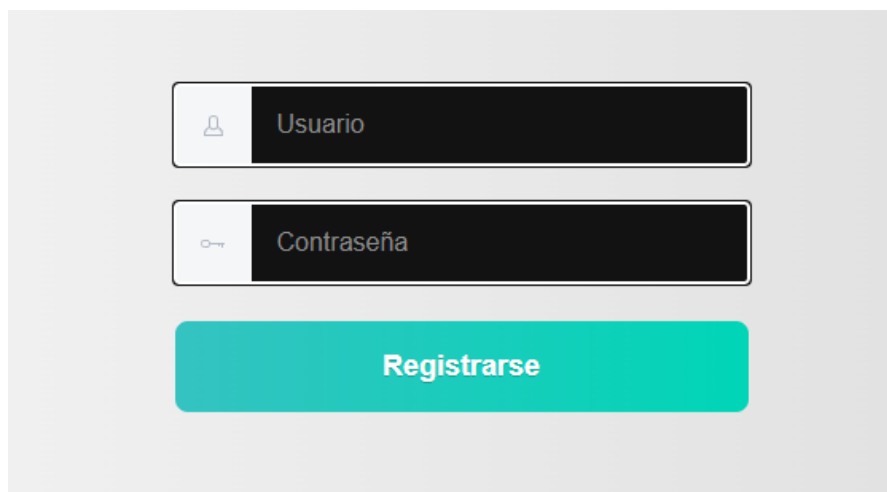


Figura 10. Página de registro de usuario

Además del registro de usuario, se trabajó en la implementación de un sistema de inicio de sesión y cierre de sesión. Se utilizó el almacenamiento local del navegador para guardar la información de inicio de sesión, como el nombre de usuario y la contraseña.

Para mejorar la experiencia de usuario, se añadió una barra superior en la interfaz de la aplicación. En esta barra, se ubicaron botones para el inicio de sesión y registro en la parte derecha, lo que facilita el acceso a estas funciones desde cualquier página de la aplicación como se puede ver en la Figura 11.



Figura 11. Barra superior con la funcionalidad de inicio de sesión y registro

Además, si un usuario ya estaba autenticado, se muestra su nombre de usuario junto con un botón de cierre de sesión para permitir al usuario cerrar sesión cuando fuera necesario como se puede ver en la Figura 12.

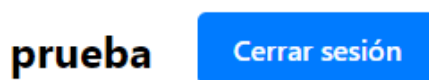


Figura 12. Barra superior cuando un usuario ha iniciado sesión

En resumen, el *sprint* 3 se centró en la implementación de funcionalidades de gestión de usuarios, incluyendo el registro, inicio de sesión y cierre de sesión, utilizando el almacenamiento local del navegador para proporcionar una experiencia conveniente para los usuarios de la aplicación. Además, se mejoró la accesibilidad a estas funciones mediante la adición de una barra superior en la interfaz de usuario.

7.1.5. *Sprint* 4

El *sprint* 4 tuvo comienzo el 11 de abril y finalizó el 25 de abril y se incluyeron las *User Stories* indicadas en la tabla 16:

Identificador de US	Descripción
US06	Realizar los cuestionarios

Tabla 16. *User Stories* del *Sprint* 4

En este *sprint* del desarrollo de la aplicación para el aprendizaje de Prolog, el enfoque principal estuvo en la implementación de cuestionarios y ejercicios para cada sección de la aplicación. Estos cuestionarios y ejercicios están diseñados para evaluar el conocimiento y comprensión de los usuarios sobre los conceptos presentados en cada sección, ofreciendo una forma interactiva y práctica de aprender Prolog.

Al final de cada cuestionario se integró un botón de "Enviar", que permite al usuario enviar sus respuestas una vez que ha contestado a todas las preguntas del cuestionario, y entonces se muestra un mensaje que indica cuántas preguntas ha acertado. El cuestionario se puede ver en la Figura 13.

Cuestionario

Responde a las siguientes preguntas:

1. ¿Cuál es la definición de un hecho en Prolog según la descripción dada?

- a) Una afirmación que establece una relación verdadera con condiciones adicionales.
- b) Una expresión lógica que puede ser verdadera o falsa.
- c) Una declaración que establece una relación verdadera sin condiciones adicionales.
- d) Una regla que establece una relación entre dos elementos.

2. ¿Cuál es la sintaxis de un hecho en Prolog?

- a) {t1, t2, ..., tn}.
- b) [t1, t2, ..., tn].
- c) predicado(t1, t2, ..., tn).
- d) predicado(t1, t2, ..., tn) :- condición.

3. ¿Cuál es el propósito principal de los hechos en Prolog?

- a) Definir relaciones lógicas entre objetos.
- b) Establecer condiciones adicionales para las consultas.
- c) Representar información básica verdadera en el programa.
- d) Resolver problemas específicos mediante inferencias complejas.

4. ¿Qué tipo de información pueden representar los hechos en Prolog?

- a) Solo propiedades de objetos.
- b) Solo relaciones entre elementos.
- c) Tanto propiedades como relaciones.
- d) Únicamente consultas complejas.

Obtuviste 3 de 4 puntos

Figura 13. Cuestionario de cada sección de la aplicación

Además, se añadieron ejercicios a algunas secciones que consisten en rellenar un fragmento de código que falta, con un botón de "Mostrar respuesta" (Figura 14) que cambia a otro estado de "Ocultar respuesta" (Figura 15). Cuando se pulsa el botón de "Enviar respuesta" aparece una alerta indicando si la respuesta proporcionada es correcta o no.

Ejercicio

Completa el código que falta:

```
tiene(cuervo, plumas).  
vuela(cuervo).  
ave(X) :- tiene(X, plumas),  (cuervo).
```

Figura 14. Ejercicio de completar código sin respuesta

Ejercicio

Completa el código que falta:

```
tiene(cuervo, plumas).  
vuela(cuervo).  
ave(X) :- tiene(X, plumas),  (cuervo).
```

Figura 15. Ejercicio de completar código con respuesta

En resumen, el *sprint* 4 se centró en la implementación de los cuestionarios y ejercicios en cada sección de la aplicación lo que proporcionó una forma estructurada y efectiva de evaluar el progreso del usuario y reforzar el aprendizaje de Prolog. Además, esta funcionalidad agregó un elemento interactivo y práctico a la experiencia de aprendizaje, haciendo que el proceso fuera más dinámico para los usuarios.

7.2. Tecnologías utilizadas

En esta sección, exploraremos y analizaremos las diversas tecnologías utilizadas durante el desarrollo de la aplicación, como por ejemplo, lenguajes de programación, *frameworks*, librerías o herramientas específicas.

7.2.1. HTML

Hypertext Markup Language o HTML⁸ es un estándar de codificación utilizado para la creación y estructuración de contenidos en páginas web. Está compuesto por una serie de etiquetas y elementos que permiten definir la estructura y el formato de un documento web, incluyendo texto, enlaces, imágenes, formularios y otros elementos multimedia.

HTML es el lenguaje básico de una web ya que proporciona la base sobre la cual se construyen las páginas web, permitiendo a los desarrolladores especificar la presentación y organización del contenido para su visualización en navegadores.

Además, son necesarias otras tecnologías para completar la apariencia y funcionalidad de una página web, ya que su función es proporcionar una estructura, por lo que es necesario hacer uso de otras herramientas como CSS o JavaScript.

7.2.2. CSS

Cascading Style Sheets o CSS⁹ es un lenguaje utilizado para definir el estilo y el formato de documentos HTML en páginas web, por lo que es una herramienta complementaria al lenguaje HTML. Permite controlar la presentación visual de los elementos de una aplicación web en el navegador, como colores, fuentes, márgenes o tamaños, facilitando la separación entre la estructura del contenido y su apariencia.

En CSS los estilos se aplican en cascada, por lo que cada elemento de la aplicación web puede tener su propio estilo, o varios elementos compartir el mismo. Esto facilita la inclusión inadvertida de código redundante, lo que lleva a que estilos preexistentes afecten a otros nuevos, siendo éstos sobrescritos y resultando a veces complicado identificar el origen del error.

En conclusión, CSS ofrece flexibilidad, reutilización de estilos, y la capacidad de crear diseños responsivos, contribuyendo a la creación de páginas web visualmente atractivas y consistentes.

⁸ <https://developer.mozilla.org/es/docs/Web/HTML>

⁹ <https://developer.mozilla.org/es/docs/Web/CSS>

7.2.3. JavaScript

JavaScript¹⁰ es un lenguaje de programación interpretado y orientado a objetos ampliamente utilizado en el desarrollo web. Diseñado para mejorar la interactividad en las páginas, se ejecuta directamente en el navegador del usuario sin necesidad de compilación previa.

Es un lenguaje dinámicamente tipado y destaca por su modelo de ejecución asíncrona, lo que permite realizar operaciones sin bloquear la ejecución del código, lo que permite probar el código inmediatamente.

Este lenguaje se utiliza comúnmente en el desarrollo web *front-end*, aunque cada vez es más común utilizarlo en el *back-end* mediante el uso de *frameworks* como Node.js, facilitando el desarrollo de aplicaciones web escalables y eficientes.

Ampliamente extendido, JavaScript desempeña un papel crucial en el desarrollo moderno por lo que es una herramienta fundamental para la creación de aplicaciones web interactivas y dinámicas.

7.2.4. GitHub

GitHub¹¹ es una herramienta que utiliza el sistema de control de versiones Git para la colaboración en proyectos de software. Permite a los desarrolladores alojar y compartir repositorios de código, colaborar en equipo, realizar seguimiento de cambios, gestionar problemas y realizar contribuciones a proyectos de código abierto.

En GitHub, los usuarios pueden clonar repositorios, crear ramas para trabajar en nuevas características o correcciones, y luego solicitar la inclusión de sus cambios a través de *pull requests*.

Además de las funciones centradas en Git, GitHub también proporciona herramientas sociales y de colaboración, como *wikis*, seguimiento de problemas, y la posibilidad de realizar comentarios y revisiones en el código. Es una plataforma ampliamente utilizada en la comunidad de desarrollo de software para facilitar la colaboración efectiva y el control de versiones en proyectos.

¹⁰ <https://developer.mozilla.org/es/docs/Web/JavaScript>

¹¹ <https://github.com/>



7.2.5. Git

Git ¹² es un sistema de control de versiones distribuido que simplifica la colaboración en proyectos de software. Permite registrar y controlar las modificaciones realizadas en el código del proyecto a lo largo del tiempo.

Al utilizar Git, los desarrolladores pueden trabajar de manera colaborativa en un mismo proyecto, gestionar ramas para desarrollar nuevas características o corregir errores de forma independiente, y fusionar esos cambios de manera eficiente.

Además, Git proporciona herramientas para la gestión remota de repositorios, lo que facilita el trabajo en equipos distribuidos y la integración continua en el desarrollo de software.

7.2.6. React

React ¹³ es una librería de JavaScript para desarrollar interfaces interactivas y reutilizables. En la actualidad, se encuentra entre las herramientas más usadas para la creación de páginas web, junto con Angular.

React ha ganado popularidad en el desarrollo web gracias a su enfoque en la eficiencia, la modularidad y su capacidad para crear una interfaz de usuario dinámica y funcional. Algunas de las características claves de React que han sido utilizadas en este proyecto son:

- **Componentes:** React utiliza un modelo de programación basado en componentes, donde las distintas partes de la interfaz son desglosadas en componentes independientes y modulares. Estos componentes son "reactivos" ya que se actualizan cuando hay cambios en los datos de la aplicación.
- **Virtual DOM:** Una de las características clave de React es el uso del DOM virtual, que permite actualizaciones eficientes del DOM real al minimizar las manipulaciones directas. Además, React se integra fácilmente con otros *frameworks* y bibliotecas, lo que lo hace altamente flexible y compatible con diversos entornos de desarrollo.
- **JSX:** React hace uso de JSX, que es una extensión de JavaScript que unifica archivos HTML y JavaScript en uno solo. Por lo que JSX facilita la creación y manipulación de elementos de la interfaz de usuario.
- **React Router:** React Router es una biblioteca adicional que se utiliza para manejar la navegación en aplicaciones React de una sola página, permitiendo la transición entre diferentes vistas y componentes.

¹² <https://git-scm.com/>

¹³ <https://es.react.dev/>

7.2.7. Visual Studio Code

Visual Studio Code¹⁴ es un entorno de desarrollo muy popular que destaca por su interfaz ligera y eficiente, ofreciendo soporte para una gran variedad de lenguajes de programación.

Proporciona características como pueden ser herramientas para la escritura y depuración de código, integración con Git para control de versiones o una terminal incorporada.

Su interfaz minimalista y su capacidad de personalización mediante extensiones hacen que sea una elección popular entre los desarrolladores que buscan una experiencia de desarrollo ágil y productiva.

7.2.8. Figma

Figma¹⁵ es una herramienta que se usa para diseñar interfaces de usuario y crear prototipos basada en la web que ofrece una interfaz intuitiva y funciones avanzadas para la creación de diseños.

Permite a los diseñadores colaborar en tiempo real, crear prototipos interactivos, utilizar componentes reutilizables y acceder a una amplia gama de herramientas de diseño. Por todo esto, se ha vuelto una plataforma ampliamente usada por diseñadores y equipos de desarrollo en todo el mundo.

¹⁴ <https://code.visualstudio.com/>

¹⁵ <https://www.figma.com/>

7.3. Escenario de uso

En esta sección se mostrará un escenario de uso de la aplicación, ilustrando el proceso que seguiría un usuario en una sesión típica de la aplicación ya finalizada tras los sprints de desarrollo. Una vez iniciada la aplicación web en el almacenamiento local del dispositivo, se puede acceder a la aplicación a través de cualquier navegador web.

Primeramente, el usuario abre la aplicación y accede a la página de registro (Figura 16). Aquí, completa los campos de nombre y contraseña, y una vez registrado, recibe una alerta de confirmación del estado del registro.

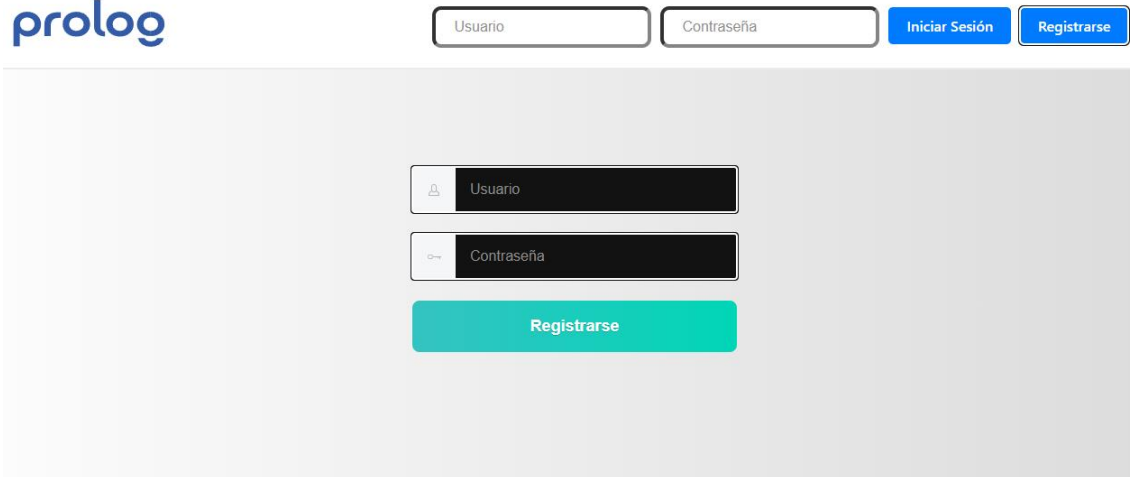


Figura 16. Página de registro de usuario

Una vez que se ha creado la cuenta, el usuario inicia sesión utilizando los campos de usuario y contraseña ubicados en la barra superior.

Posteriormente, el usuario puede explorar las diferentes secciones del tutorial utilizando la barra de navegación lateral izquierda. Esta barra ofrece acceso a distintos conceptos del lenguaje Prolog organizados de manera lógica.

Al seleccionar una sección, el usuario encuentra una combinación de contenido teórico y práctico (Figura 17). Cada sección comienza con una explicación del concepto en cuestión, complementada con ejemplos claros preconfigurados que el usuario puede ejecutar en un intérprete web integrado dentro de la aplicación. Estos ejemplos permiten al usuario ver los resultados del código en tiempo real.

- Introducción
 - Cuestionario
- Términos
 - Cuestionario
- Hechos
 - Ejercicio
 - Cuestionario
- Reglas**
 - Ejercicio
 - Cuestionario
- Cláusulas
 - Cuestionario
- Consultas
 - Ejercicio
 - Cuestionario
- Negación
 - Cuestionario
- Listas
 - Cuestionario
- Operadores
 - Cuestionario

Reglas

Una regla en Prolog es una construcción lógica que establece una relación entre diferentes predicados mediante condiciones o antecedentes.

La sintaxis de una regla es la siguiente:

$$h :- q_1, q_2, \dots, q_n$$

Una regla representa la implicación

$$q_1, q_2, \dots, q_n \rightarrow h$$

Las reglas permiten definir relaciones más complejas y derivar nuevas afirmaciones a partir de información existente en la base de conocimiento.

Al ejecutar una consulta en Prolog, el motor de inferencia utiliza estas reglas para determinar si se cumplen las condiciones necesarias y, en su caso, deducir nuevos hechos para responder a la consulta realizada.

En Prolog, la secuencia q_1, q_2, \dots, q_n representa la **conjunción** $q_1 \wedge q_2 \wedge \dots \wedge q_n$. Por ejemplo:

```
humano(pedro).
tiene_gato(pedro).

%Una persona es propietaria de un gato si es humana y tiene un gato
es_propietario_de_gato(X) :- humano(X), tiene_gato(X).
```

Figura 17. Página del contenido de una sección

Además, tras el contenido teórico de cada sección se puede presentar un ejercicio donde el usuario debe completar fragmentos de código faltantes (Figura 18). Estos ejercicios están diseñados para reforzar la comprensión de la sintaxis y la lógica de Prolog.

- Introducción
 - Cuestionario
- Términos
 - Cuestionario
- Hechos
 - Ejercicio
 - Cuestionario
- Reglas
 - Ejercicio
 - Cuestionario
- Cláusulas
 - Cuestionario
- Consultas
 - Ejercicio**
 - Cuestionario
- Negación
 - Cuestionario
- Listas

Ejercicio

Dado el siguiente programa lógico:

```
p(X):- s(X),r(X).
s(a).
s(x).
r(a).
r(b).
r(c).
```

¿Cuáles son **TODAS** las respuestas computadas para el objetivo **?- p(X)**, usando la estrategia de búsqueda predefinida de Prolog?

X =

Figura 18. Página de un ejercicio de una sección contestado correctamente



En los ejercicios, el usuario dispone de la opción de mostrar la respuesta correcta del mismo mediante un botón que cambia su estado para mostrar u ocultar la respuesta. Además, si el ejercicio es contestado correctamente por el usuario, se pondrá de color verde en la barra lateral izquierda, si por el contrario se contesta erróneamente se pondrá de color rojo.

Para evaluar el aprendizaje, cada sección termina con un cuestionario que contiene preguntas de opción múltiple (Figura 19). Este cuestionario ayuda a consolidar el conocimiento y proporciona retroalimentación instantánea sobre el desempeño del usuario. El cuestionario también cambia de color a verde si se responde correctamente a todas las preguntas y a rojo si hay alguna pregunta no contestada correctamente.

prolog prueba123 [Cerrar sesión](#)

Introducción
Cuestionario
Términos
Cuestionario
Hechos
Ejercicio
Cuestionario
Reglas
Ejercicio
Cuestionario
Cláusulas
Cuestionario
Consultas
Ejercicio
Cuestionario
Negación
Cuestionario
Listas
Cuestionario
Operadores
Cuestionario

Cuestionario

Responde a las siguientes preguntas:

1. ¿Cuál es la sintaxis de una regla en Prolog?

- h ?- q1, q2, ..., qn
- h :- q1, q2, ..., qn
- h | q1, q2, ..., qn
- h :: q1, q2, ..., qn

2. ¿Cuál es el propósito principal de las reglas en Prolog?

- Representar información básica verdadera en el programa.
- Establecer relaciones entre diferentes listas mediante condiciones.
- Derivar nuevas afirmaciones a partir de la información existente en la base de conocimiento.
- Resolver consultas complejas mediante inferencias lógicas.

3. ¿Qué tipo de regla se llama recursiva en Prolog?

- Aquella que no tiene cuerpo.
- Aquella que contiene solo variables.
- Aquella que se llama a sí misma, ya sea directa o indirectamente.
- Aquella que establece una relación entre diferentes predicados.

Obtuviste 2 de 3 puntos

[Enviar](#)

Figura 19. Página de un cuestionario de una sección contestado erróneamente

8. Pruebas

Una vez concluido el ciclo de desarrollo, resulta crucial evaluar la usabilidad de la aplicación para identificar posibles inconvenientes antes de su lanzamiento. Para llevar a cabo la evaluación de usabilidad se ha decidido hacer uso de un cuestionario de usabilidad.

8.1 Cuestionario de usabilidad

Un cuestionario de usabilidad es una herramienta diseñada para recopilar información sobre la experiencia del usuario al interactuar con un producto, como puede ser una aplicación web. El cuestionario generalmente contiene una serie de preguntas cuidadosamente elaboradas que abordan diferentes dimensiones de la usabilidad.

Estas preguntas pueden variar desde la facilidad de navegación y comprensión de la interfaz hasta la satisfacción general del usuario con la experiencia. Los participantes responden a estas preguntas proporcionando datos que permiten identificar áreas de mejora y comprender mejor la percepción y experiencia que tiene el usuario.

En este proyecto se ha adoptado la *System Usability Scale (SUS)*, reconocida como una de las herramientas más eficaces y confiables utilizadas en proyectos actuales para evaluar la usabilidad. A partir de los resultados obtenidos mediante esta escala, es posible inferir la presencia de posibles problemas en el sistema si la puntuación resultante es baja. No obstante, para garantizar resultados fiables, se aconseja realizar la evaluación con las respuestas de al menos 5 usuarios.

El cuestionario del System Usability Scale (SUS) consta de 10 afirmaciones, diseñadas para evaluar la facilidad de uso percibida de un sistema. Aquí están las afirmaciones:

1. Creo que me gustaría utilizar este sistema con frecuencia.
2. Encuentro el sistema innecesariamente complejo.
3. Creo que el sistema es fácil de usar.
4. Creo que necesitaría la ayuda de una persona experta para poder utilizar este sistema.
5. Creo que las distintas funciones de este sistema están bien integradas.
6. Creo que el sistema es fácil de usar.
7. Creo que hay demasiada inconsistencia en este sistema.
8. Imagino que la mayoría de las personas aprenderían a utilizar este sistema muy rápidamente.
9. Encuentro que el sistema es muy complicado.
10. Me siento muy seguro/a utilizando este sistema.

Las respuestas a estas afirmaciones se registran en una escala de 5 puntos, que va desde "Totalmente en desacuerdo" hasta "Totalmente de acuerdo". Luego, se calcula el puntaje SUS para cada participante y se obtiene un promedio general de usabilidad para el sistema. Este puntaje oscila entre 0 y 100, donde puntajes más altos indican una mejor percepción de usabilidad. Para el cálculo de la puntuación SUS, se deben repetir una serie de pasos para las respuestas de cada usuario:

1. Convertir las respuestas del usuario en puntuaciones:
 - Si la pregunta es impar, se subtrae un punto de la valoración dada.
 - Si la pregunta es par, la puntuación obtenida es 5 menos la valoración dada.
2. Sumar las puntuaciones obtenidas para todas las preguntas del cuestionario.
3. Multiplicar la puntuación total por 2.5 para obtener la puntuación individual del usuario.
4. Repetir los pasos 1 al 3 para todos los usuarios.
5. Calcular el promedio de todos los puntajes de los usuarios para obtener un puntaje SUS general para la aplicación.

En el contexto de nuestro proyecto, llevamos a cabo encuestas con un grupo de cinco usuarios, quienes presentaban una diversidad de edades y niveles de experiencia en tecnología. Con el fin de que los usuarios se familiarizaran con la aplicación, se les solicitó que realizaran una serie de tareas antes de cumplimentar el formulario. Las puntuaciones obtenidas por cada usuario se muestran en la tabla 17.

Usuario	Puntuación SUS
Usuario 1	82.5
Usuario 2	70
Usuario 3	92.5
Usuario 4	85
Usuario 5	77.5

Tabla 17. Puntuación SUS de cada usuario

De estos resultados se obtiene un promedio de 81.5, lo que indica una percepción generalmente positiva de la usabilidad del sistema entre los usuarios evaluados. Estos resultados pueden interpretarse según la Figura 14.

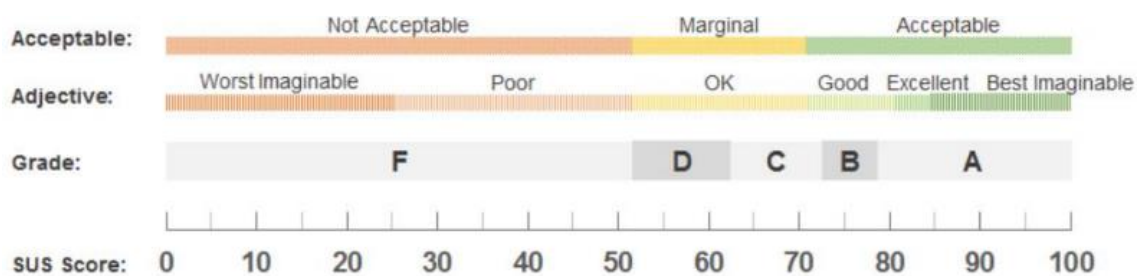


Figura 20. Distintos umbrales en la System Usability Scale

9. Conclusiones

Este capítulo finaliza el proyecto, presentando las conclusiones sobre el mismo y estableciendo cómo su desarrollo se relaciona con los estudios cursados. Además, se analizan las posibilidades de trabajo futuro para la aplicación y se reflexiona sobre los pasos que se podrían seguir para seguir mejorándola.

9.1. Conclusiones

El progreso de la aplicación web dedicada al aprendizaje de Prolog ha concluido con éxito, dando lugar a una herramienta valiosa para estudiantes, profesionales y entusiastas de la programación que buscan mejorar sus habilidades y expandir sus conocimientos en este área específica.

Uno de los principales desafíos que enfrentamos fue la implementación de un intérprete del lenguaje Prolog integrado en la aplicación. Dada nuestra inexperiencia previa en este ámbito, nos encontramos con dificultades para lograr que el intérprete devolviera respuestas a las consultas realizadas. Esto nos llevó a probar muchas implementaciones distintas, hasta que finalmente logramos incorporarlo con éxito a la aplicación.

A pesar de ello, hemos alcanzado todos los objetivos que conforman nuestra meta principal que consiste en la creación de una aplicación web que facilite el aprendizaje de los principales conceptos del lenguaje de programación Prolog acompañado por ejemplos y cuestionarios. También se ha tenido en cuenta el diseño de una interfaz de usuario atractiva y fácil de usar.

Además, el uso de la metodología ágil Scrum ha permitido organizar el proceso de desarrollo del proyecto de manera eficiente. Al adoptar este enfoque ágil, hemos logrado una mayor flexibilidad y eficiencia en la gestión del proyecto. A través de iteraciones cortas y regulares, hemos sido capaces de adaptarnos rápidamente a los cambios en los requisitos del proyecto.

En resumen, a lo largo de estos meses de trabajo han surgido distintos retos durante el desarrollo de la aplicación, pero hemos logrado superarlos y cumplir con los objetivos que nos habíamos propuesto inicialmente.



9.2. Relación del trabajo desarrollado con los estudios cursados

El desarrollo de este proyecto es el resultado de la aplicación práctica y el aprendizaje adquirido a lo largo de los estudios de Ingeniería Informática, ya que habilidades adquiridas como la programación han sido fundamentales para el desarrollo de esta aplicación.

Durante el recorrido académico, entre las asignaturas cursadas que más vitales han sido para enfrentar los desafíos del desarrollo de este proyecto se encuentran por ejemplo la asignatura de Ingeniería del Software (ISW) donde se vieron los principios del desarrollo de software, o la asignatura de Lenguajes, Teorías y Paradigmas de la Programación (LTP), donde se vio el paradigma de programación lógico mediante el uso de Prolog.

Además de los conocimientos técnicos, el grado en Ingeniería Informática también ha fomentado la capacidad para encontrar soluciones por cuenta propia, la gestión de la carga de trabajo, así como el uso eficiente del tiempo.

En conclusión, este proyecto es el resultado de la combinación entre la aplicación práctica de los conocimientos técnicos adquiridos durante los estudios y de la gestión eficiente del mismo.

9.3. Trabajo futuro

A pesar de lo obtenido hasta ahora, el proyecto tiene margen para futuras mejoras y desarrollos. A continuación se presentan algunas de las funcionalidades y aspectos considerados importantes para seguir mejorando la plataforma y la experiencia del usuario:

- Añadir problemas donde el usuario tenga que hacer programas cortos en Prolog para resolverlos.
- Extensión de los conceptos explicados sobre la programación en Prolog.
- Abordar las respuestas obtenidas a través del cuestionario de usabilidad para mejorar varios aspectos de la aplicación.
- Almacenar los datos de usuario de forma más segura mediante el uso de un servidor.

Estas mejoras de la aplicación contribuirán a que esta herramienta sea más valiosa para los estudiantes o profesionales que la puedan utilizar.

10. Referencias

- [1] Lloyd, J.W. (John Wylie). Foundations of logic programming (Symbolic computation. Artificial intelligence), Editorial Springer, 1ª edición, 1984, DOI: 10.1007/978-3-642-96826-6.
- [2] Clocksin, W.F. (William F.). Mellish, C.S. (Christopher S.). Programming in Prolog, Editorial Springer, 5ª edición, 2003.
- [3] Sterling, Leon. Shapiro, Ehud. The Art of Prolog: advanced programming techniques (with a foreword by David H. D. Warren), Editorial The MIT Press, 2ª edición, 1994.
- [4] Bramer, Max. Logic Programming with Prolog, Editorial Springer, 2ª edición, 2013.
- [5] Kowalski, Robert A. The early years of logic programming, Imperial College London, 1988. Disponible en: <https://www.doc.ic.ac.uk/~rak/papers/the%20early%20years.pdf> (accedido 22 Marzo, 2024).
- [6] Clocksin, W.F. (William F.). Clause and Effect: Prolog Programming for the Working Programmer, Editorial Springer, 1ª edición, 1997.
- [7] Blackburn, Patrick. Bos, Johan. Striegnitz, Kristina. Learn Prolog Now!, Editorial College Publications, 1ª edición, 2006.
- [8] Merritt, Dennis. Adventure in Prolog, Amzi, 2017. Disponible en: <https://www.amzi.com/AdventureInProlog/> (accedido 22 Marzo, 2024).
- [9] Sarsby, Alan. SWOT Analysis: A guide to SWOT for business studies students, Editorial Spectaris, 1ª edición, 2016.
- [10] Coleman, Ben. Goodwin, Dan. Designing UX: Prototyping, Editorial SitePoint, 1ª edición, 2017.
- [11] Angeles, Michael. Barnard, Leon. Carlson, Billy. Wireframing for Everyone (foreword by Ellen Chisa), Editorial Book Apart, 1ª edición, 2023.
- [12] Staiano, Fabio. Designing and Prototyping Interfaces with Figma: Learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop, Editorial Packt Publishing, 1ª edición, 2022.
- [13] <https://github.com/cubiwan/jsProlog> (accedido 22 Marzo, 2024).



Anexo A. Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS):

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.				X
ODS 4. Educación de calidad.	X			
ODS 5. Igualdad de género.			X	
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.				X
ODS 9. Industria, innovación e infraestructuras.				X
ODS 10. Reducción de las desigualdades.		X		
ODS 11. Ciudades y comunidades sostenibles.				X
ODS 12. Producción y consumo responsables.				X
ODS 13. Acción por el clima.			X	
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

Las Naciones Unidas han establecido una serie de Objetivos de Desarrollo Sostenible (ODS), una agenda global con 17 objetivos interconectados que buscan abordar los desafíos más urgentes que enfrenta el mundo actual. Estos ODS ofrecen un marco amplio para la cooperación internacional y la colaboración en diferentes áreas con el objetivo de crear un futuro más justo, igualitario y sostenible para todos.

Teniendo en cuenta los objetivos de desarrollo sostenibles mencionados anteriormente, se relaciona el proyecto con los mismos debido a los siguientes motivos:

- **Educación de calidad:** El uso de una aplicación web para el aprendizaje puede incorporar metodologías educativas innovadoras, como aprendizaje interactivo, ejemplos prácticos y evaluaciones en línea que complementen y faciliten el aprendizaje.
- **Igualdad de género:** La aplicación web puede ser una herramienta para proporcionar igualdad de acceso a la educación en programación, eliminando barreras basadas en el género. Al ofrecer recursos y oportunidades de aprendizaje en línea, se promueve la participación equitativa de mujeres y hombres en el ámbito de la tecnología.
- **Reducción de desigualdades:** Al proporcionar una plataforma en línea, la aplicación web puede ofrecer acceso a la educación en programación a personas de diversos orígenes y ubicaciones geográficas. Esto reduce las barreras de acceso y amplía las oportunidades de aprendizaje para aquellos que, de otra manera, podrían tener dificultades para acceder a recursos educativos.
- **Acción por el clima:** La implementación de una aplicación web para el aprendizaje evita la necesidad de materiales físicos, como libros de texto impresos, reduciendo así el impacto ambiental asociado con la producción y distribución de recursos educativos tradicionales.

