Universidad de Valladolid

Escuela de Ingeniería Informática de Valladolid

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención Computación

Librería para el desarrollo de juegos de rol distribuidos en Elixir

Autor:
Alonso Sayalero Blázquez

Tutor:
Dr. César Llamas Bello

Resumen

Con el paso de los años el mundo docente se ha dado cuenta que el estudio de diversas materias cala mejor en el estudiante si se presenta de una manera amena e interactiva. En un mundo constantemente cambiante y cada vez más rápido, esto se ha visto reflejado en el aumento de la gamificación dentro del aula como una forma de combatir la constante perdida de atención y como una forma de que los conceptos se entiendan mejor y de forma clara.

Por otro lado el cómputo distribuido cada vez toma más fuerza en esta nueva era en la que nos adentramos, dominada por los grandes modelos de inteligencia artificial y cada vez más conectada a la web. En este nuevo paradigma web impera la disponibilidad inmediata de los recursos, dando lugar al aumento inequívoco de los sistemas distribuidos.

Con el fin de utilizar el juego como forma de aprendizaje de los sistemas distribuidos nace este proyecto, una librería en Elixir para la creación de un juego con varios escenarios para el aprendizaje de los alumnos.

Palabras clave: Sistema Distribuido, Elixir, juego, aprendizaje.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Introducción

La educación está en constante evolución y la informática es ya parte fundamental de la misma, desde la proyección de diapositivas en aplicaciones como Microsoft PowerPoint o Canvas, hasta el empleo de aplicaciones que fomentan la competición sana y hacen más ameno el estudio y el repaso de los conceptos aprendidos como Kahoot. En algunos centros los libros comienzan a quedarse obsoletos y son sustituidos por ordenadores o tablets.

Los sistemas distribuidos adquieren mayor importancia en este momento en el que la inteligencia artificial avanza a pasos agigantados en un mundo cada vez más conectado mediante páginas, servicios y aplicaciones web. Todos estos elementos requieren de una alta disponibilidad en todo momento y en casi todos los puntos geográficos del planeta. Es decir, requieren de sistemas distribuidos robustos y tolerantes a los fallos.

En este panorama, la educación superior también empieza a cambiar, siempre con cautela pero sin pararse. Dentro de las numerosas formas de evolución educativa en la asignatura de Sistemas Distribuidos se inclinan hacia la gamificación de la educación. Con gamifiación nos referimos al empleo de cualquier tipo de juegos para enseñar al alumno conceptos que, sin visualizarse, pueden llegar a ser complicados de imaginar o comprender.

Este proyecto surge de la oportunidad para mejorar el juego empleado en la asignatura en este momento. Dicho juego es una aplicación *Java* que corre de forma local en el ordenador del alumno. Con este proyecto se pretende crear un motor que permita la creación de todo tipo de videojuegos, no solo educativos, que corran de manera distribuida, ya sea en la propia máquina del jugador, en una serie de servidores online o de forma híbrida entre un servidor y la máquina del jugador. Además, también forma parte del proyecto la creación de un juego educativo empleando el motor y la propuesta de algunos escenarios educativos haciendo uso del juego para mostrar algunas características de los sistemas distribuidos.

El motor pasa a ser la pieza principal del proyecto y sobre el que se basa el análisis y diseño del mismo. Algunas de las características que se buscan en este nuevo motor es que se puede instalar fácilmente en las máquinas de los alumnos, sea fácilmente ampliable en el futuro y emplee una

arquitectura de sistema distribuido basada en actores. Por lo tanto el lenguaje de programación que surge como candidato a utilizarse en el proyecto es *Elixir*. *Elixir* [1] es un lenguaje de programación funcional modular y entre sus características podemos encontrar, las siguientes:

- **Basado en *Erlang***: *Erlang* es un lenguaje de programación funcional más antiguo que *Elixir* sobre el que este se basa. Al estar basado en este lenguaje *Elixir* hereda tanto la máquina virtual de *Erlang* como sus librerías.
- **Distribuido y Escalable**: Elixir se ejecuta en procesos ligeros que están aislados y se comunican mediante mensajes, lo que permite añadir nuevos procesos sin afectar el funcionamiento del resto. Estos procesos se ejecutan en una máquina virtual de *Erlang* y representan actores en el sistema.
- **Tolerante a los fallos**: Si algún componente del sistema falla, es posible recuperar la parte del sistema en la que se encuentra este fallo sin poner en riesgo el funcionamiento del resto del sistema.
- **Hot Swapping** (Actualización de código en caliente): Aplicaciones que pueden ser modificadas sin detener el sistema, lo que permite realizar modificaciones en el código o en la arquitectura sin necesidad de detener por completo el sistema.

Otro de los aspectos fundamentales a la hora de enfocar la creación de un motor de videojuegos, de cualquier tipo, es el estilo de juego que se quiere crear. Para este proyecto se ha optado por la creación de un motor de juegos de rol o RPGs.

Los juegos de rol surgen originalmente como juegos de mesa en los que una serie de jugadores participan con sus personajes en el mundo, normalmente de fantasía, que el narrador principal ha creado. Este narrador principal actúa también como árbitro en las disputas que puedan ocurrir dentro del mundo de la partida, como puede ser un combate entre uno de los jugadores y un enemigo. Los jugadores van por turnos interactuando con el mundo siguiendo las indicaciones del narrador y eligiendo que hacer cuando se les presenta un evento. El máximo exponente de los juegos de rol de mesa es *Dungeons and Dragons* [2].

Con la llegada de los videojuegos, los juegos de rol pasaron a formar parte del elenco de géneros que podemos encontrar dentro de esta industria, junto con géneros como los videojuegos de plataformas, los de acción, los de aventura gráfica o los de simulación entre otros muchos. De la mezcla de los juegos de rol con el resto de géneros surgen muchos subgéneros como los ARPGs o los JRPGs, pero en este proyecto no ahondaremos en ellos y nos centraremos únicamente en crear un motor que permita la creación y gestión de elementos clásicos de los juegos de rol, como los personaje, los enemigos o los objetos entre otros.

1.2. Motivación

La motivación para adentrarme en este proyecto surge de la posibilidad de profundizar en el mundo de los videojuegos, pasión que tengo desde pequeño, mientras empleo tecnologías y para-

digmas de programación que no he utilizado o he empleado poco.

Este proyecto implica aprender a utilizar de manera medianamente fluida el paradigma de programación funcional al mismo tiempo que se utiliza un lenguaje de programación nunca antes visto ni utilizado, como es *Elixir*, el cual proporciona grandes ventajas a la hora de programar sistemas distribuidos.

A su vez el proyecto me permite utilizar mi experiencia como alumno de la asignatura para aportar la visión de alguien que aprende pero que no ha enseñado nunca a los escenarios por los que pasan los alumnos durante el estudio de la asignatura y de esta forma poder hacer que los compañeros que vienen detrás puedan aprender de manera entretenida los conceptos que más adelante, en su vida profesional, puedan llegar a necesitar.

Por último, la motivación que mueve a cualquier ingeniero informático de seguir aprendiendo nuevas cosas y actualizar las tecnologías que se emplean en el mundo real. Investigar piezas de software que pueden no tener utilidad real pero que funcionan como cimientos para lo que pueda llegar o que puedas crear tú mismo en el futuro. En resumen, el afán por el constante crecimiento como ingenieros.

1.3. Objetivos y etapas del proyecto

1.3.1. Objetivos

El principal objetivo del proyecto es crear un motor de videojuegos que funcione de manera distribuida. Dicho motor deberá funcionar mediante una arquitectura de actores para las partes distribuidas y tener la capacidad de modelar los elementos básicos de un juego de rol. Aparte del objetivo principal el proyecto tiene otras metas para cumplir:

- **Aprendizaje de nuevas tecnologías**: Para el proyecto se emplea el lenguaje de programación *Elixir*, lo que presenta una buena oportunidad para aprender un lenguaje que tiene su hueco en el diseño y programación web.

- **Creación de un juego**: Tal y como se ha descrito, otra parte del proyecto es el empleo del motor para crear un juego que actúe como un servidor local distribuido sobre el que poder ejecutar escenarios controlados para los alumnos.

- **Creación de escenarios**: Crear una serie de escenarios controlados sobre los que los alumnos de la asignatura de Sistemas Distribuidos puedan utilizar y modificar para poder aprender sobre el funcionamiento de diferentes conceptos de dichos sistemas.

- **Empleo de *Docker***: Con el fin de simular un sistema distribuido en las máquinas de los alumnos, *Docker* surge como una tecnología que permite dicha simulación y, por lo tanto, el objetivo es crear un *Dockerfile* que permita el empaquetado en contenedores del motor y el juego, así como proporcionar un fichero *docker compose* que permita la inicialización de los suficiente contenedores para poder ejecutar los escenarios sin problemas.

1.3.2. Etapas

A continuación se detallan las etapas del proyecto que se siguieron.

Primera etapa: Investigación y recopilación de datos sobre motores y diseño de videojuegos

Antes de comenzar a analizar y diseñar el motor es necesario realizar una investigación previa sobre las diferentes metodologías que abarcan el diseño tanto de videojuegos como de motores. Entre los diferentes puntos de investigación destacan:

- Estudio del funcionamiento de un motor de videojuegos.
- Diseño de videojuegos de rol.
- Partes fundamentales de los juegos de rol tradicionales.
- Diseño de algoritmos de resolución de conflictos.

Segunda etapa: Análisis y diseño del motor

La segunda etapa del proyecto consiste en describir de forma analítica como se va a ver el motor de videojuegos. A partir de este análisis se hará el diseño del mismo. Para esta etapa se siguen las siguientes tareas:

- Análisis de la arquitectura del motor de videojuegos.
- Descripción de los elementos que forman parte del motor.
- Análisis del funcionamiento entre componentes del motor.
- Diseño del motor de videojuegos.

Tercera etapa: Aprendizaje de *Elixir*

La tercera etapa consiste en familiarizarse y aprender a utilizar el lenguaje de programación que se va a emplear durante el proyecto. Como ya se ha mencionado, dicho lenguaje es *Elixir* y por tanto hay que hacerse con los siguientes conceptos para poder presentar un motor que, aunque sea simple, funcione:

- Funcionamiento básico del lenguaje, ejecución y entorno de desarrollo.
- Creación de actores y su funcionamiento.
- Creación de servidores TCP/IP para comunicación.
- Funcionamiento de los métodos para actuar ante los fallos.

Cuarta etapa: Creación del motor

Tras tener todos los componentes necesarios para poder programar el motor la siguiente etapa consiste en programarlo, para ello se divide el trabajo en diferentes partes del motor que son necesarias para su funcionamiento. Estas partes son:

- Elementos básicos de un juego de rol que se ejecuten siguiendo una arquitectura de actores.
- Capacidad de manipulación de la base de datos utilizada como sistema de guardado del estado del juego.
- Utilidades para el correcto desarrollo de una partida de rol, como puede ser el lanzar un dado.
- Sistema de login para diferentes usuarios.

Quinta etapa: Creación del juego

Una vez el motor está en funcionamiento la siguiente etapa es crear un juego que haga uso del motor creado y demuestre las capacidades del mismo. Al tratarse de un juego sencillo no cuenta con muchas partes, pero se pueden destacar las siguientes:

- Servidor TCP/IP que escuche por el puerto elegido.
- Sistema de arranque del juego.
- Red de supervisores que actúen frente a los fallos de los diferentes actores que conforman el videojuego.

Sexta etapa: Creación de escenarios de estudio

Como en la etapa anterior se creó un juego que actúa como un servidor de la misma forma que se hace en la arquitectura cliente-servidor, los escenarios modificables por los alumnos tendrán que actuar como clientes del juego y realizar peticiones concretas. Para conseguir esto los escenarios tienen que cumplir las siguientes características:

- Estar completos. Los escenarios que se entregan con este proyecto están completos y actúan como soluciones.
- Demostrar una o varias características de los sistemas distribuidos.
- Estar escritos en *Java*. Es el lenguaje de programación empleado en la asignatura de Sistemas Distribuidos.

Capítulo 2

Planificación

2.1. Tecnologías

En todo proyecto de ingeniería informática se utilizan una gran variedad de tecnologías para diferentes partes del proyecto. Todas estas tecnologías tienen que ser definidas antes de dar comienzo al proyecto y forman parte de la planificación del mismo. En este proyecto, las tecnologías que han sido elegidas para su ejecución son las siguientes:

- **Elixir**: Lenguaje de programación utilizado para la construcción del motor y el juego.
- **Java**: Lenguaje de programación utilizado para la construcción de los escenarios de aprendizaje.
- **Docker**: Programa para la construcción y gestión de contenedores docker.
- **SQL**: Lenguaje de consulta sobre bases de datos relacionales.
- **JSON**: Formato de intercambio de datos.
- **Postgresql**: Base de datos relacional opensource.

2.1.1. Elixir

Sobre *Elixir* ya se ha hablado en la introducción. Es un lenguaje de programación que ofrece unas características que lo hacen especialmente apto para el diseño y programación de sistemas distribuidos. Sigue el paradigma de programación funcional y está diseñado para ser implementado en una arquitectura modular.

Ofrece tolerancia al fallo, la capacidad de distribuir los módulos en diferentes sistemas mediante la propia API del lenguaje, hot swaping y una alta escalabilidad. Todas estas características le han hecho posicionarse como un lenguaje muy interesante en el mundo de la programación web, gracias al framework de desarrollo de aplicaciones web Phoenix [3].

2.1.2. Java

Java [4] es un lenguaje de programación diseñado para seguir el paradigma de programación orientada a objetos y cuyas fortalezas residen en:

- **Independiente de la plataforma**: *Java* es un lenguaje de programación compilado. El resultado de los lenguaje compilados, tradicionalmente, está sujeto a la arquitectura del sistema operativo en el que se compila y, por ejemplo, un binario de un lenguaje compilado en Windows no se puede ejecutar en Linux. *Java* toma otro acercamiento y se ejecuta sobre una máquina virtual, la *JVM (Java Virtual Machine)*. Gracias a esto, el resultado de la compilación es un binario que se puede ejecutar sobre cualquier *JVM* independientemente del sistema operativo sobre el que se ejecute. El único requisito es que el sistema operativo sobre el que se ejecuta el programa tiene que tener instalada la máquina virtual de *Java*.

- **Fuertemente tipado**: Los lenguajes de programación fuertemente tipados evitan la asignación de valores a variables cuando el tipo de ambos no coincide y obligan a asignarle un tipo a cada variable, no pudiendo haber variables sin tipo hasta que se las asigna un valor. Esto evita errores durante la programación que puedan llevar al programa a fallar durante la ejecución.

- **Recolector de basura**: *Java* cuenta con un recolector de basura. El recolector de basura es un mecanismo que tienen algunos lenguajes de programación que permite liberar espacio de memoria de manera automática en tiempo de ejecución. Para ello el recolector de basura libera el espacio ocupado por variables que ya han sido utilizadas y no tienen uso en el futuro de la ejecución.

- **Amplia comunidad**: Es un lenguaje utilizado en multitud de sistemas que aún siguen en funcionamiento y, pese a su antigüedad, sigue siendo escogido por multitud de empresas y personas. Esto hace que *Java* cuente con una gran comunidad de programadores e ingenieros y por consiguiente se pueden encontrar una gran cantidad de recursos para cualquier necesidad que pueda surgir durante un proyecto.

2.1.3. Docker

Anteriormente se ha mencionado la portabilidad de *Java* y la escalabilidad de *Elixir*, ambas de estas características se pueden ver reforzadas mediante *Docker* [5]. *Docker* es un sistema de gestión y ejecución de contenedores. Un contenedor es una pieza de software que simula un sistema operativo reducido dentro del mismo y permite ejecutar programas dentro de este sistema operativo en miniatura empleando los recursos del sistema operativo donde se ejecuta el contenedor. Esto le permite a *Docker* contar con varias ventajas a la hora de distribuir programas a servidores o usuarios:

- **Control de versiones**: Al crear un contenedor este tiene la capacidad de otorgarle una etiqueta con la versión de la que se trata. Esto permite que el cambio de versiones consista únicamente en actualizar la etiqueta del contenedor dentro del fichero en el que se encuentra la definición del contenedor.

- **Portabilidad**: Los contenedores se pueden ejecutar en cualquier sistema con *Docker* instalado.
- **Escalabilidad**: Existen tecnologías que permiten la duplicación de contenedores, el balance de carga sobre contenedores duplicados y creación y destrucción automática de contenedores. Una de las tecnologías más populares que permite esto es *Kubernetes* [6].

En este proyecto *Docker* permite simular un sistema distribuido dentro de una sola máquina gracias a que ejecuta pequeños sistemas operativos.

2.1.4. SQL

SQL [7] es un lenguaje de programación de dominio específico diseñado para gestionar datos estructurados en sistemas de gestión de bases de datos o RDBMS por sus siglas en inglés.

En este proyecto se emplea para hacer uso de la base de datos desde el propio motor, en la cual se persiste el estado del juego y sus jugadores.

2.1.5. JSON

JSON (JavaScript Object Notation) [8] es un formato de intercambio de datos diseñado para poder ser legible para los humanos. Basado en un subconjunto de *JavaScript* [9], es completamente independiente a este y puede ser empleado para transmitir datos entre diferentes lenguajes de programación.

Dentro del proyecto es el lenguaje empleado para comunicar los escenarios con el juego y también es empleado para almacenar algunos campos en la base de datos.

2.1.6. Postgresql

Postgresql [10] se trata de una base de datos relacional de código abierto que utiliza y extiende *SQL* para ofrecer una gran variedad de tipos de datos para el almacenamiento, robustez, integridad de los datos, seguridad y rendimiento.

Para el proyecto, el empleo de *Postgresql* como base de datos permite el almacenamiento de datos tipo *JSON* gracias al tipo proporcionado JSONB, que a su vez permite realizar operaciones sobre los campos de los datos tipo *JSON* y un almacenaje de los mismos empleando menos espacio en el disco duro.

2.2. Planificación

2.2.1. Metodología ágil: Scrum

Las metodologías ágiles de desarrollo software se basan en el desarrollo iterativo del mismo. Por lo general, las metodologías ágiles promueven la constante revisión del software creado, trabajo en equipo y la intención de crear productos funcionales de manera rápida, adaptándose a las

necesidades del cliente [11] [12] [13].

Scrum es una de las metodologías ágiles más en uso en la actualidad. Adopta un enfoque incremental e iterativo dividiendo los proyectos en pequeñas partes las cuales se desarrollan y entregan a lo largo de periodos de tiempo definidos, los cuales se denominan Sprints.

Esta metodología se caracteriza por su capacidad de adaptación a los cambios y es muy adecuada para proyectos que presentan gran incertidumbre a la hora de ser desarrollados, es decir, que pueden estar sujetos a muchos cambios durante el desarrollo y necesitan de retroalimentación constante para mitigar la complejidad del proyecto.

Para la implementación de Scrum este cuenta con tres componentes claves: roles, eventos y artefactos.

Roles

Scrum define tres roles dentro del equipo de trabajo:

- **Product Owner**: Es el miembro del equipo encargado de mantener la comunicación entre el equipo de desarrollo y las partes interesadas que quieran cambiar o añadir al producto. Es el responsable de mantener el seguimiento del producto y de gestionar la prioridad que tienen las nuevas características y la solución de errores del mismo. Por ello es importante el hacer caso a las indicaciones del Product Owner, ya que es el único miembro del equipo con una visión global del producto y el cual recibe retroalimentación de diversas fuentes externas al equipo de desarrollo.

- **Scrum Master**: Es el encargado de que el equipo siga la metodología Scrum sin complicaciones eliminando complicaciones que puedan suponer un retraso o impactar en la productividad de todo el equipo.

- **Equipo de desarrollo**: El resto de miembros del equipo. Se encargan de crear los artefactos necesarios para completar un Sprint. Pueden tener multitud de habilidad diferentes pero tienen que ser capaces de repartir el trabajo entre los miembros y como partir el artefacto en otros más pequeños.

Eventos

Scrum emplea eventos para inspeccionar y adaptar los artefactos, crear regularidad y minimizar gasto de tiempo en reuniones innecesarias. Los eventos definidos por Scrum son los siguientes:

- **Sprint**: Es el evento más importante ya que contiene al resto de eventos dentro del mismo. No puede durar más de un mes y está diseñado para alcanzar el objetivo de incremento para el producto. Durante la ejecución del mismo no se puede modificar el objetivo final del evento, por lo tanto este tiene que ser razonable para el periodo de tiempo que abarca el Sprint.

- **Sprint Planning**: Es el primer evento dentro del propio evento de Sprint. En él se planea todo lo relacionado con el producto durante el tiempo que va a durar el Sprint. Está diseñado para evitar la sobreplanificación, limitándose a un máximo de 8 horas invertidas en este evento.
- **Daily Scrum**: Es el evento más corto, con un máximo de 15 minutos recomendados. Se realiza todos los días con el objetivo de revisar el progreso y ajustar el plan para alcanzar el objetivo del Sprint. No es la única oportunidad diaria para reunirse entre los miembros del equipo, estos siempre se podrán reunir para discutir temas más detallados.
- **Sprint Review**: Evento realizado al finalizar el Sprint con el fin de presentar los resultados del mismo y discutir el progreso hacía el objetivo del producto. También permite ajustar el producto en base a oportunidades que hayan podido presentarse durante el Sprint.
- **Sprint Retrospective**: Con un máximo de 3 horas de duración, el objetivo de este evento es reflexionar sobre las dinámicas de equipo y planificar formas de mejorar estas así como la calidad y la eficacia durante el próximo Sprint.

Artefactos

En la metodología Scrum un artefacto se refiere a todo aquello que defina el producto que se está desarrollando, las acciones que producen el producto y las acciones que ya han sido tomadas para la creación del producto. Scrum define tres artefactos principales:

- **Product Backlog**: Es una lista de nuevas características, mejoras y arreglos del producto. También incluye las tareas y los requisitos para construir el producto. Se alimenta de muchas fuentes relacionadas con el producto como pueden ser los reportes de bugs, los análisis de mercado o las demandas del mercado. El Product Owner se encarga de que este artefacto se encuentre actualizado y sea correcto.
- **Sprint Backlog**: Es un conjunto de Product Backlogs escogidos para formar parte del siguiente incremento del producto. Está conformado por tareas que surgen de dividir tareas más grandes de los diferentes Product Backlogs que lo conforman con el fin de poder planificar el reparto de las mismas por el equipo de desarrollo.
- **Product Increment**: Estos artefactos se forman completando tareas definidas mediante los otros artefactos. Incluye los incrementos del resto de Sprints y se crean al decidir realizar un despliegue de producto para los usuarios. Son muy útiles para seguir la versión del producto.

2.2.2. Adaptación al proyecto

Se elige Scrum para el proyecto debido a la capacidad de generar artefactos rápidamente adaptándose a los cambios que surjan. Esto es especialmente útil ya que es un proyecto que cuenta con tres piezas de software diferentes dependientes entre sí y las tecnologías empleadas son nuevas para mí por lo tanto una rápida iteración permite la refactorización del código a menudo.

Scrum es una metodología pensada para equipos de trabajo sobre proyectos grandes, pero eso no significa que no se pueda adaptar para que funcione con proyectos más pequeños y una sola

persona.

Debido a que solo se encuentra una persona trabajando en el proyecto toda reunión referente a comunicación entre miembros del equipo no se realiza y todos los roles quedan asignados a la misma persona. A partir de aquí la definición de Sprints funciona de la misma forma que funcionaría en un equipo más grande, se define que incremento que corresponde al Sprint, las fechas en las que está activo el Sprint y la retroalimentación proporcionada por el cliente, en este caso el tutor del proyecto.

2.2.3. Planificación inicial

Con el fin de realizar el proyecto en un plazo lógico para las circunstancias en las que se encuentra el ejecutor del mismo se realiza una planificación inicial de 9 Sprints con 2 semanas de duración aproximadas para cada uno. Cada Sprint cuenta con una cantidad de trabajo variable pero que siempre se ajusta a un trabajo semanal de entre 15 y 20 horas, lo que da entre 30 y 40 horas de trabajo por Sprint.

El primer Sprint es un poco mas largo que el resto debido a que se caracteriza por que los artefactos que genera son todos relacionados con la investigación del proyecto. Este Sprint está enfocado en la búsqueda de todas las piezas necesarias para la ejecución del proyecto, desde diseño de juegos hasta documentación sobre los lenguajes de programación así como librerías que puedan ser de utilidad para facilitar el desarrollo del software planeado.

La previsión inicial del proyecto cuenta con 9 Sprints de los cuales a 8 de ellos se les puede estimar un promedio de 35 horas de trabajo mientras que el primero de todos, al ser un poco más largo se le puede estimar 40 horas de trabajo, dando una estimación total de 320 horas de trabajo.

En caso de que ocurran imprevistos se han añadido dos Sprints extra sumando un promedio de 70 horas extra más al proyecto que, de ser utilizadas, aumentarían la estimación de tiempo del proyecto hasta un total de 390 horas.

En la tabla 2.1 se muestra la previsión de las fechas de ejecución del Sprints planificados y las fechas de los eventos relacionados con cada Sprint.

Nº Sprint/Eventos	**Fecha Inicio**	**Fecha Fin**
Sprint 0	13/02/2025	27/02/2025
Sprint Planning	13/02/2025	
Sprint Weekly	20/02/2025	
Sprint Weekly, Sprint Planning y Sprint Restrospective	27/02/2025	
Sprint 1	27/02/2025	13/03/2025
Sprint Weekly	06/03/2025	
Sprint Weekly, Sprint Planning y Sprint Restrospective	13/03/2025	

Sprint 2	14/03/2025	27/03/2025
Sprint Weekly	20/03/2025	
Sprint Weekly, Sprint Planning y Sprint Restrospective	27/03/2025	
Sprint 3	28/03/2025	10/04/2025
Sprint Weekly	03/04/2025	
Sprint Weekly, Sprint Planning y Sprint Restrospective	10/04/2025	
Sprint 4	11/04/2025	24/04/2025
Sprint Weekly, Sprint Planning y Sprint Restrospective	24/04/2025	
Sprint 5	25/04/2025	08/05/2025
Sprint Weekly, Sprint Planning y Sprint Restrospective	08/05/2025	
Sprint 6	09/05/2025	22/05/2025
Sprint Weekly	15/05/2025	
Sprint Weekly, Sprint Planning y Sprint Restrospective	22/05/2025	
Sprint 7	23/05/2025	05/06/2025
Sprint Weekly	29/05/2025	
Sprint Weekly, Sprint Planning y Sprint Restrospective	05/06/2025	
Sprint 8	06/06/2025	19/06/2025
Sprint Weekly	12/06/2025	
Sprint Weekly, Sprint Planning y Sprint Restrospective	19/06/2025	
Sprint 9	20/06/2025	03/07/2025
Sprint Weekly	26/06/2025	
Sprint Weekly, Sprint Planning y Sprint Restrospective	03/07/2025	
Sprint Extra 1	04/07/2025	17/07/2025
Sprint Extra 2	18/07/2025	31/07/2025

Tabla 2.1: Planificación inicial de Sprints

2.3. Plan de riesgos

Un riesgo [14] es una situación potencial que puede acarrear un retraso en el proyecto o lo puede acelerar. Todo riesgo tiene un posibilidad de ocurrir y un impacto sobre el proyecto. Es natural que cualquier proyecto que sea ejecutado esté expuesto a una serie de riesgos que modifiquen los tiempos de ejecución del mismo ya que toda planificación de un proyecto está basada en estimaciones y sujeta a incertidumbres.

Una vez establecidos los tiempos de ejecución de los Sprints para el proyecto es necesario analizar los riesgos que pueden ocurrir durante el transcurso del mismo. Debido al tiempo, solo se analizan los riesgos que tengan un desenlace negativo. De esta forma se pueden establecer planes de contingencia para mitigar los efectos de estos riesgos y no retrasar la ejecución del proyecto.

Todo riesgo identificado en este plan de riesgos está formado por los siguientes cuatro elementos:

- **Probabilidad**: Nivel de posibilidad de que el riesgo suceda. Puede ser muy baja, baja, media, alta o muy alta.
- **Impacto**: Nivel del impacto que tiene el daño producido en caso de que el riesgo suceda sobre la ejecución del proyecto. Al igual que la probabilidad puede ser muy bajo, bajo, medio, alto o muy alto.
- **Plan de mitigación**: Lista de acciones que se pueden tomar para reducir la probabilidad de que le riesgo suceda.
- **Plan de contingencia**: Lista de acciones que se pueden llevar a cabo una vez el riesgo ha sucedido para reducir el impacto sobre los tiempos de ejecución del proyecto.

El plan de riesgos es fundamental en la planificación de un proyecto debido a que permiten identificar situaciones evitables que podrían poner en riesgo la correcta ejecución del proyecto. A su vez, permite establecer protocolos para mitigar los efectos adversos en caso de que se manifieste alguno de los riesgos.

Las tablas desde la 2.2 hasta la muestran los riesgos identificados que pueden afectar al proyecto.

Riesgo R01	**Falta de disponibilidad**
Descripción	Durante la duración del proyecto tanto el alumno que actúa como equipo Scrum como el tutor que actúa como usuario pueden tener variaciones en la disponibilidad haciendo que se tenga que volver a planificar los Sprints para ajustarse a las nuevas circunstancias.
Probabilidad	Media
Impacto	Medio
Plan de mitigación	▪ Evitar situaciones que puedan poner en riesgo la salud o la integridad física.
Plan de contingencia	▪ Distribuir las tareas no finalizadas en otros Sprints estableciendo prioridades para las tareas más importantes. ▪ Considerar tomar vacaciones en el trabajo para dedicarle más tiempo al proyecto.

Tabla 2.2: Riesgo R01

Riesgo R02	**Fallos en el hardware de trabajo**
Descripción	Todo equipo electrónico puede llegar a fallar, lo que puede suponer retrasos y perdida de información.
Probabilidad	Baja
Impacto	Muy alto
Plan de mitigación	▪ Disponer de un dispositivo de repuesto. ▪ Realizar copias de seguridad de todo lo relacionado con el proyecto de forma habitual. ▪ Instalar sistemas SAI para evitar perder datos ante un corte de luz.
Plan de contingencia	▪ Recuperar las copias de seguridad, de existir, en otro dispositivo. ▪ En caso de archivos corruptos intentar usar programas de recuperación para este tipo de archivos.

Tabla 2.3: Riesgo R02

Riesgo R03	**Error en la estimación de tiempo para las actividades a desarrollar**
Descripción	Las estimaciones de tiempo para alguna tarea resultan ser incorrectas y esta requiere de mayor cantidad de tiempo para su correcta realización provocando un retraso en la planificación del proyecto.
Probabilidad	Alta
Impacto	Alto
Plan de mitigación	▪ Realizar análisis en profundidad de las tareas para obtener estimaciones más precisas. ▪ Dar prioridad a las partes del proyecto que resulten críticas. ▪ Prever tiempo extra que pueda llevar la ejecución del proyecto teniendo Sprints extra.
Plan de contingencia	▪ Utilizar los Sprints extra previstos para este riesgo. ▪ Reorganizar el proyecto eliminando partes no importantes de manera que su objetivo principal no se vea afectado.

Tabla 2.4: Riesgo R03

Riesgo R04	**Mala utilización de Scrum**
Descripción	Debido a que el equipo de desarrollo no tiene experiencia previa con la metodología Scrum se puede hacer un mal uso de la misma llevando a que se produzcan artefactos innecesarios y se prioricen las tareas de manera incorrecta.
Probabilidad	Media
Impacto	Medio
Plan de mitigación	▪ Estudio intensivo de la metodología.
Plan de contingencia	▪ Compensar con los Sprints extra retrasos que puedan suceder.

Tabla 2.5: Riesgo R04

Bibliografía

[1] The Elixir team. Elixir. https://elixir-lang.org/, 2025. [Online; accessed 21-June-2025].

[2] Wizards of the Coast. Dungeons and dragons. https://dnd.wizards.com/es, 2025. [Online; accessed 22-June-2025].

[3] The Phoenix framework team. Phoenix framework. https://www.phoenixframework.org/, 2025. [Online; accessed 01-July-2025].

[4] Oracle. Java. https://www.java.com/es/, 2025. [Online; accessed 01-July-2025].

[5] The docker team. Docker. https://www.docker.com/, 2025. [Online; accessed 01-July-2025].

[6] The linux foundation. Kubernetes. https://kubernetes.io/es/, 2025. [Online; accessed 01-July-2025].

[7] Wikipedia. Sql. https://en.wikipedia.org/wiki/SQL, 2025. [Online; accessed 01-July-2025].

[8] Douglas Crockford. Json. https://www.json.org/json-en.html, 2001. [Online; accessed 01-July-2025].

[9] Mozilla foundation. Javascript. https://developer.mozilla.org/es/docs/Web/JavaScript, 2025. [Online; accessed 01-July-2025].

[10] The postgresql team. Postgresql. https://www.postgresql.org/, 2025. [Online; accessed 01-July-2025].

[11] cprime. What is scrum. https://www.cprime.com/resources/what-is-agile-what-is-scrum/, 2025. [Online; accessed 01-July-2025].

[12] Mary Iqbal. Which scrum events are time boxed? https://www.scrum.org/resources/blog/which-scrum-events-are-time-boxed, 2024. [Online; accessed 01-July-2025].

[13] Chandler Harris. Agile scrum artifacts. https://www.atlassian.com/agile/scrum/artifacts, 2025. [Online; accessed 01-July-2025].

[14] Bob Hughes. *Software Project Management 5e.* McGraw Hill, 2009.