Diseño y desarrollo de una aplicación web para rol storytelling

Josep Enric Esteve Colomer
Grado de Ingeniería Informática

Consultor: Gregorio Robles Martínez

09/06/2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	Diseño y desarrollo de una aplicación web para rol storytelling
Nombre del autor:	Josep Enric Esteve Colomer
Nombre del consultor:	Gregorio Robles Martínez
Fecha de entrega (mm/aaaa):	06/2022
Área del Trabajo Final:	Desarrollo web
Titulación:	*Grado de ingeniería informática*

Resumen del Trabajo (máximo 250 palabras):

El ser humano siempre ha tenido la necesidad de contar historias y dar rienda suelta a su creatividad. Desde los albores de la humanidad, la capacidad para contar las historias se ha adaptado a los distintos medios que han ido surgiendo. Este proyecto busca innovar en el proceso de contar historias y de ser creativos, añadiendo elementos de rol a un sistema de historias encadenadas, que conoceremos a partir de ahora como rol storytelling.

El rol storytelling, consiste en que los usuarios desarrollen historias escritas en base a una ambientación propuesta y un personaje creado por ellos. Los moderadores generan la ambientación general y eventos más específicos donde se apuntan los personajes, para redactar e interactuar entre ellos. En los eventos, un usuario redacta una parte de la historia, asumiendo el rol del personaje que ha creado, y otro usuario la continúa hasta que se da por finalizada.

Los personajes, creados por los usuarios, tienen una serie de atributos y habilidades designadas en el momento de su creación, que se pueden utilizar a la hora de redactar. El uso de estas habilidades genera una serie de 'puntos de escritura', unos contenidos que el usuario debe introducir obligatoriamente en el momento de su redacción.

El proyecto desarrolla una aplicación web con una estructura tecnológica de MEAN *stack* (MongoDB, ExpressJS, AngularJS, NodeJS), consiguiendo crear un entorno para el desarrollo de historias y poner a prueba la creatividad de los usuarios.

Abstract (in English, 250 words or less):

The human race always had the necessity to tell stories and unleash their creativity. Since the dawn of humanity, the ability to tell stories has been adapted to the different media that have been appearing. This project seeks to innovate in the process of telling stories and being creative, adding role-playing elements to a system of linked stories, which we will know from now on as rol storytelling.

The rol storytelling consists in users developing written stories based on a general history and a character created by them. The moderators generate the general history and more specific events where the characters sign up, to write and interact with each other. In the events, a user writes a part of the story, assuming the role of the character that he had created, and another user continues it until it's finished.

Characters, created by users, have a set of attributes and abilities designated at the time of their creation, which can be used when the user writes in an event. The use of these skills generates 'writing points', content that the user must enter at the time of writing.

The project develops a web application with a MEAN stack technological structure (MongoDB, ExpressJS, AngularJS, NodeJS), managing to create an environment for the development of stories and test the creativity of users.

Palabras clave (entre 4 y 8):

historias, escritura, lectura, redacción, rol, creatividad, MEAN *stack*.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

El presente proyecto busca cubrir las necesidades narrativas y creativas de los usuarios que sientan la necesidad de escribir, contar historias y dar rienda suelta a su imaginación. De igual manera, este proyecto proporciona también relatos e historias creadas por los usuarios, para otro tipo de usuarios que solo busquen entretenimiento mediante la lectura.

En la actualidad, muchos usuarios se dedican a contar historias en base a una ambientación propuesta de antemano, lo que conoceremos con el término de *Storytelling*. Para ello, y con la necesidad de hacerlo de manera no presencial y asíncrona, hacen uso de foros. En estos foros, cada usuario genera un personaje, con un trasfondo acorde a la ambientación y asume el rol de este personaje cuando escribe algún mensaje. El uso de foros permite crear temas de conversación (cada tema de conversación correspondería a una historia), donde los personajes que participan en esa historia pueden escribir. De esta manera, se van creando temas de conversación y los usuarios generan historias. Existe también una ambientación general que es proporcionada, mediante temas de conversación, por usuarios más experimentados conocidos como Másters, para que las historias que desarrollan los usuarios sigan un contexto social, político y espacial concreto (es decir, proporcionan la ambientación). Estos Másters, hacen de moderadores del foro y velan por el cumplimiento de la ambientación.

Una gran mayoría de estos foros, intentan añadir contenido adicional aparte de únicamente servir para el *Storytelling*. Por ello, intentan añadir elementos de Rol como atributos, habilidades, dinero, etc. a los personajes, de manera que los personajes se puedan ir desarrollando, mejorando y de esta manera proporcionar nuevos elementos narrativos en el desarrollo de las historias. Este método de contar historias, lo conoceremos a partir de ahora como *Rol Storytelling*.

El uso de foros para el *Storytelling* proporciona una buena estructura para el desarrollo de historias en temas de conversación, la problemática puede venir dada por la manera de gestionar los usuarios que intervendrán en cada una de ellas, siendo un proceso un poco laborioso.

Si hablamos de *Rol Storytelling*, los foros no son una buena solución para llevarlos a cabo. El desarrollo de los personajes, consiste en mejorar sus atributos y las habilidades que hacen uso de estos atributos. Esta mejora, se realiza mediante la asignación de puntos de experiencia, que son otorgados por participar en historias, y son anotados de manera manual en el perfil del usuario. Manualmente también, se mejoran los atributos del personaje cuando la experiencia acumulada es suficientemente alta. A su vez, el uso de las habilidades, se hace de manera manual con la utilización de tablas y el azar, que el usuario resuelve y relata en base al resultado. Es decir, cuando el usuario utiliza una habilidad, tira un dado y en base a: el resultado, su nivel de su

habilidad y los atributos, consulta una tabla que le indica si el resultado ha sido un éxito o no y relata de acuerdo a ese resultado. No existe manera de contrastar si el usuario ha realizado el proceso correctamente, o si se ha inventado los resultados.

Por todo lo expuesto, y visto la necesidad de crear un sistema que permita realizar *Rol Storytelling* de una manera sencilla y amena, se plantea crear este proyecto. El proyecto busca estructurar la ambientación y las historias creadas de una manera clara, igual o superior a la que se consigue mediante el uso de los foros. Y añadir de manera automática diferentes elementos de Rol, creando así una aplicación de *Rol Storytelling*, que añada nuevos elementos narrativos a las historias a desarrollar.

1.2 Objetivos del Trabajo

1.2.1 Objetivo principal

Desarrollar una aplicación web, funcional y accesible, para el desarrollo de historias de *Rol Storytelling*. Una aplicación con esta finalidad puede tener infinidad de opciones o mejoras, y, por tanto, requerir mucho desarrollo; la aplicación se ceñirá a demostrar su potencial a través del cumplimiento de los subobjetivos. El logro de todos los subobjetivos conlleva la adquisición del objetivo principal del proyecto.

1.2.2 Subobjetivo 1: Servidor y dominio

Puesta en marcha de un servidor que pueda ejecutar y servir tanto el *frontend* como el *backend* de la aplicación. También, debe estar provisto del *software* necesario para realizar el desarrollo y el mantenimiento de todo el proyecto. La aplicación web ha de ser accesible a través de un dominio fácilmente reconocible y seguro.

1.2.3 Subobjetivo 2: Registro de usuarios

Los usuarios han de poder registrarse en la aplicación y acceder a ella. Existirán cuatro roles dentro de la aplicación: Visitante, Usuario, Máster y Administrador. Un usuario registrado puede tener permisos de Máster y/o Administrador, pero un Visitante (usuario no registrado), no puede tener ningún permiso asociado. Los usuarios Administradores, deben poder gestionar los usuarios registrados otorgando o retirando permisos e incluso bloquear el acceso a usuarios.

1.2.4 Subobjetivo 3: Informe de errores

Se establece un apartado de ‘Reportar Fallo’ en el menú, donde los usuarios pueden reportar posibles problemas detectados en la aplicación. Los Administradores pueden ver los informes, gestionarlos y tomar medidas para solucionar los posibles problemas.

1.2.5 Subobjetivo 4: Creación de personajes

Los usuarios registrados pueden crear un personaje siempre y cuando no tengan uno ya asociado. El personaje creado ha de pasar una validación a través de un usuario Máster, que verifica que el personaje está creado correctamente y se adecua a la ambientación general. Si el personaje no ha pasado la validación,

su creador puede modificarlo basándose en las notas que le ha dejado el Máster y volver a enviarlo a validación. Una vez validado, el personaje se puede utilizar. El usuario con permisos de Máster puede crear tantos personajes como desee y seleccionar en cada momento cual va a utilizar. Los personajes creados y validados, se pueden ver en el apartado 'Población' del menú. Los personajes pueden subir de nivel participando en eventos y mejorar así sus habilidades.

1.2.6 Subobjetivo 5: Habilidades de trasfondo

Durante la creación de personajes, este adquiere ciertas habilidades con un determinado valor. En base a esas habilidades y valores, se habilitan diferentes pestañas en el apartado 'Conocimientos' del menú. Según su habilidad, y el valor que tiene en esa habilidad, el usuario puede leer más o menos relatos, que aportan un mayor trasfondo a ambientación general y simulan el conocimiento del personaje sobre esa habilidad. Los relatos asociados a los conocimientos de estas habilidades son creados y gestionados por los usuarios con permisos de Máster.

1.2.7 Subobjetivo 6: Creación de eventos

Los usuarios con permisos de Máster pueden crear eventos con una pequeña historia de trasfondo, donde los personajes se pueden apuntar como posibles participantes. El Máster selecciona los personajes que acabarán participando en el evento de entre todos los posibles participantes y habilitará su inicio. Una vez termine, el Máster cerrará el evento y las recompensas serán distribuidas equitativamente entre sus participantes. La historia generada en el evento podrá ser leída en cualquier momento y por cualquier usuario.

1.2.8 Subobjetivo 7: Escritura y habilidades de escritura

Los usuarios pueden participar escribiendo en alguno de los eventos (conocidos como misiones) en los que participa su personaje. Entrando al evento, podrá acceder al panel de escritura si ningún personaje ha reservado el turno. La reserva de turno para poder escribir será de un tiempo determinado, que se liberará una vez se sobrepase el tiempo o el usuario envíe su parte del relato. El panel de escritura permitirá redactar con cierto formato y, además, hacer uso de ciertas habilidades que tenga el usuario a través de botones. El uso de estas habilidades durante la escritura, generan 'puntos de escritura' que el usuario ha de introducir en su relato; estos 'puntos de escritura' son generados en base a la habilidad, el nivel de la misma y las opciones seleccionadas en el momento de su uso. Al enviar el relato, el Máster que ha generado el evento lo recibe para su validación y, una vez comprobado que cumple con la ambientación, los 'puntos de escritura' y la continuidad con el anterior relato, lo valida y queda publicado.

1.3 Enfoque y método seguido

El producto a desarrollar será nuevo y estará basado los subobjetivos marcados. No habiéndose encontrado productos con el mismo enfoque que el que se pretende desarrollar. Los productos encontrados de características más similares han sido principalmente foros como: Wild Haunt [1], Harajuku Noir [2], Fall and Rise [3] y Endless Immortals [4]. Con el estudio de estos foros, preguntando necesidades de los jugadores, y la propia experiencia de juego, se han definido los subobjetivos.

En cuanto a las características y funcionalidades de la parte de rol, el proyecto se basa en el Reglamento de Rápido y Fácil 3.0 [5]. Las habilidades, atributos, sistemas de tiradas de dados y estadísticas, parten de este manual y se adaptan al sistema desarrollado.

La estrategia utilizada para el desarrollo del proyecto se basa en ir añadiendo nuevas características al proyecto de forma incremental, de manera que cada una de las fases en las que se divide el proyecto aporte nuevas funcionalidades al mismo. Después de cada una de las fases, el proyecto siempre ha de quedar operativo y cumpliendo con todas las funcionalidades que se detallan en los subobjetivos de cada fase.

La metodología más apropiada en este caso es la metodología *agile*. Donde cada una de las fases parte de unos requisitos, generados a partir de los subobjetivos. Posteriormente, se genera un diseño, que puede implicar cambios en diseños anteriores. Se desarrolla la parte de producto de esa fase, verificando lo desarrollado. Se preparan y ejecutan las baterías de pruebas unitarias y de integración de la fase. Si las pruebas han ido según lo esperado, se procede a desplegar la aplicación. Y, por último, se revisa que la aplicación desplegada funciona según lo esperado antes de pasar a la siguiente fase.

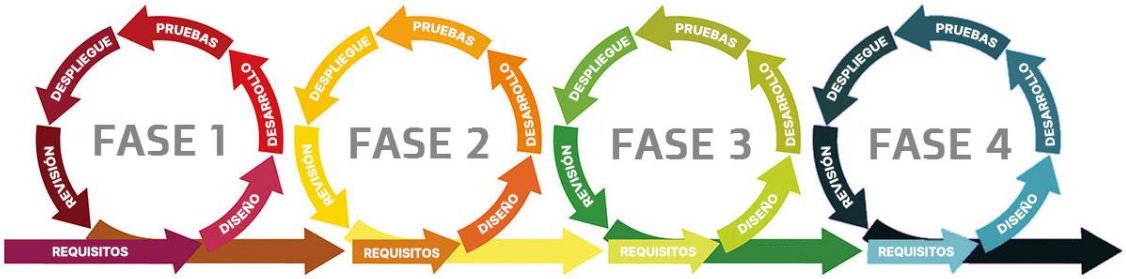

Figura 1: Metodología utilizada en el proyecto

Esta estrategia permite tener un producto funcional al final de cada fase. En caso de no conseguir terminar una fase, siempre queda un producto utilizable, aunque no se hayan conseguido las funcionalidades de fases posteriores. Como parte negativa, el adoptar este tipo de estrategia puede obligar a rediseñar partes ya desarrolladas en fases previas, pero esto es asumible frente al riesgo de no conseguir un producto final.

1.4 Planificación del Trabajo

El proyecto se divide en cuatro fases bien diferenciadas. Cada una de las fases del proyecto engloba diferentes subobjetivos, quedando la estructura de las distintas fases de la siguiente manera:

- Fase 1: ‘Subobjetivo 1: Servidor y dominio’.
- Fase 2: ‘Subobjetivo 2: Registro de usuarios’ y ‘Subobjetivo 3: Informe de errores’.
- Fase 3: ‘Subobjetivo 4: Creación de personajes’ y ‘Subobjetivo 5: Habilidades de trasfondo’.
- Fase 4: ‘Subobjetivo 6: Creación de eventos’ y ‘Subobjetivo 7: Escritura y habilidades de escritura’.

En la planificación se ha tenido en cuenta como único recurso humano a Josep Enric Esteve Colomer que ha sido asignado a todas las tareas del proyecto. Con una dedicación estimada de 3 horas diarias, se establece el proyecto en 340 horas totales.

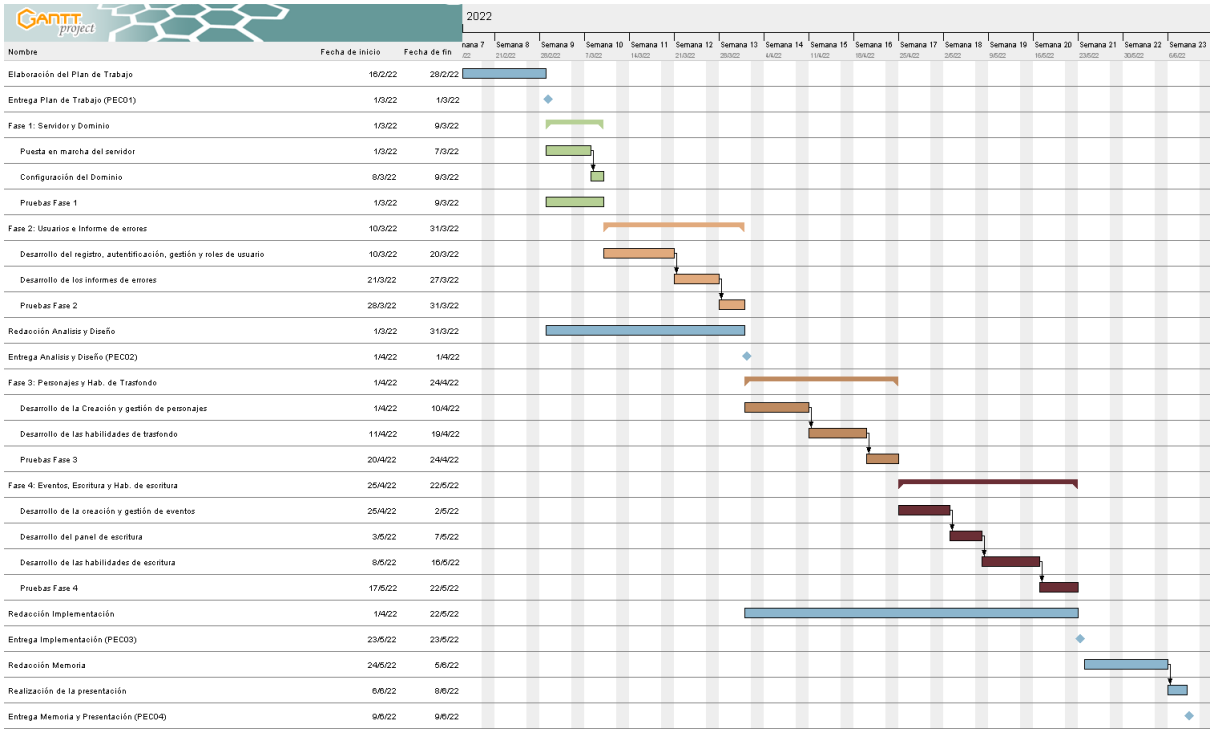

Figura 2: Planificación temporal

Como hitos principales del proyecto, tenemos los marcados por las PECS, que incluyen la finalización de fases del proyecto:

- Hito 1: Entrega del plan de trabajo (PEC01).
- Hito 2: Entrega análisis y diseño (PEC02). Implica finalización de las fases 1 y 2 del proyecto.
- Hito 3: Entrega implementación (PEC03). Implica finalización de las fases 3 y 4 del proyecto.
- Hito 4: Entrega memoria y presentación (PEC04).

La tabla detallada de las tareas y de sus cotes se puede consultar a continuación:

Tarea	Fecha Inicio	Fecha Fin	Duración (días)	Duración (Horas)	Porcentaje dedicación
Elaboración del Plan de Trabajo (PEC01)	16/02/2022	28/02/2022	13	34	10%
Puesta en marcha del servidor	01/03/2022	07/03/2022	7	14	4,1%
Configuración del Dominio	08/03/2022	09/03/2022	2	4	1,1%
Pruebas Fase 1	01/03/2022	09/03/2022	9	9	2,6%
Desarrollo del registro, autentificación, gestión y roles de usuario	10/03/2022	20/03/2022	11	22	6,4%
Desarrollo de los informes de errores	21/03/2022	27/03/2022	7	14	4,1%
Pruebas Fase 2	28/03/2022	31/03/2022	4	8	2,3%
Redacción Análisis y Diseño (PEC02)	01/03/2022	31/03/2022	31	31	9,1%
Desarrollo de la Creación y gestión de personajes	01/04/2022	10/04/2022	10	25	7,3%
Desarrollo de las habilidades de trasfondo	11/04/2022	19/04/2022	9	22	6,4%
Pruebas Fase 3	20/04/2022	24/04/2022	5	12	3,6%
Desarrollo de la creación y gestión de eventos	25/04/2022	02/05/2022	8	20	5,8%
Desarrollo del panel de escritura	03/05/2022	07/05/2022	5	12	3,6%
Desarrollo de las habilidades de escritura	08/05/2022	16/05/2022	9	22	6,4%
Pruebas Fase 4	17/05/2022	22/05/2022	6	12	3,6%
Redacción Implementación (PEC03)	01/04/2022	22/05/2022	52	31	9,1%
Redacción Memoria	24/05/2022	05/06/2022	13	39	11,4%
Realización de la presentación	06/06/2022	08/06/2022	3	9	2,6%

Tabla 1: Planificación detallada de tareas

1.5 Breve sumario de productos obtenidos

Al realizar el proyecto obtendremos los siguientes productos:

- Un servidor que permita alojar y mantener la aplicación web desarrollada.
- Un dominio para el acceso a la aplicación.
- Una aplicación web para el desarrollo de rol storytelling.
- Documentación, memoria y presentación de la aplicación.

1.6 Breve descripción de los otros capítulos de la memoria

- Capítulo 2: Arquitectura general de la aplicación. En este capítulo se describen las tecnologías que se utilizan en el proyecto y su interacción entre ellas. Además, se comenta brevemente las herramientas utilizadas durante el proyecto.
- Capítulo 3: Requisitos. Se definen las funcionalidades que ha de cumplir el sistema en cada una de las fases en las que está dividido el proyecto. Los requisitos están basados en los subobjetivos que debe cumplir el proyecto.
- Capítulo 4: Diseño. Este capítulo define de manera concreta cómo va a ser la aplicación desarrollada, haciendo uno de diagramas de casos de uso, diagramas de clases y mapas del sitio. El capítulo se estructura en las diferentes fases del proyecto, ya que en cada una de ellas se vuelve a plantear el diseño nuevamente.
- Capítulo 5: Desarrollo. Se describe el proceso de convertir el diseño previo en la aplicación desarrollada. El capítulo vuelve a estructurarse en las diferentes fases del proyecto.
- Capítulo 6: Pruebas. Se muestran en este capítulo las distintas pruebas realizadas para ver que la aplicación desarrollada cumple con los requisitos planteados. Se realizan pruebas del servidor, pruebas unitarias, pruebas de integración y pruebas de visualización.
- Capítulo 7: Valoración económica del trabajo. Se expone un breve estudio económico sobre los costes de desarrollo, los costes de mantenimiento, beneficios y viabilidad del producto.
- Capítulo 8: Conclusiones. Este capítulo comenta las lecciones aprendidas durante el proyecto, las dificultades que han surgido, la consecución de los objetivos y las posibles líneas de trabajo futuras.
- Capítulo 9: Glosario. Define los términos y acrónimos más utilizados durante la memoria.
- Capítulo 10: Bibliografía. Referencia las publicaciones y artículos utilizados durante el desarrollo del proyecto.
- Capítulo 11: Anexos. Este capítulo añade el manual de uso de la aplicación, las vistas desarrolladas, el informe de pruebas unitarias y el repositorio con el código fuente.

2. Arquitectura general de la aplicación

Las tecnologías utilizadas en el diseño y desarrollo, son las propias de un *MEAN stack* (MongoDB, ExpressJS, AngularJS, NodeJS). Este conjunto de tecnologías permite el desarrollo de todas las capas de una aplicación web con el lenguaje de programación Javascript.

En la parte del servidor, se utiliza el entorno de ejecución NodeJS que utiliza a su vez el motor Javascript de Google V8. NodeJS proporciona una arquitectura orientada a eventos y el desarrollo de API's asíncronas, que proporcionan un gran rendimiento y escalabilidad. Aquí, se incorpora el *framework* Express, que facilita el desarrollo de aplicaciones web. Con Express, se puede generar rápidamente una aplicación web, gestionar las peticiones, respuestas, rutas, cabeceras, etc.

Como base de datos se utiliza MongoDB que es un sistema de base de datos NoSQL, que utiliza Javascript para representar la información y almacena los datos en BJSON (*Binary JavaScript Object Notation*). Para la comunicación entre la aplicación y los datos, NodeJS incorpora la librería Mongoose. Con Mongoose se pueden realizar consultas a la base de datos, realizar validaciones, crear peticiones de datos, etc. Todo ello a partir de un esquema que detalla la configuración de los documentos de una colección de MongoDB.

Por otro lado, en la parte de cliente, se utiliza el *framework* Angular, que utiliza Typescript (extensión de la sintaxis de JavaScript que añade tipos estáticos y objetos basados en clases) como lenguaje de programación. Angular traslada al navegador el patrón Modelo-Vista-Controlador y dispone de multitud de bibliotecas, facilitando muchas tareas. El desarrollo consigue ser mucho más sencillo y potente que utilizando únicamente HTML, CSS y Javascript.

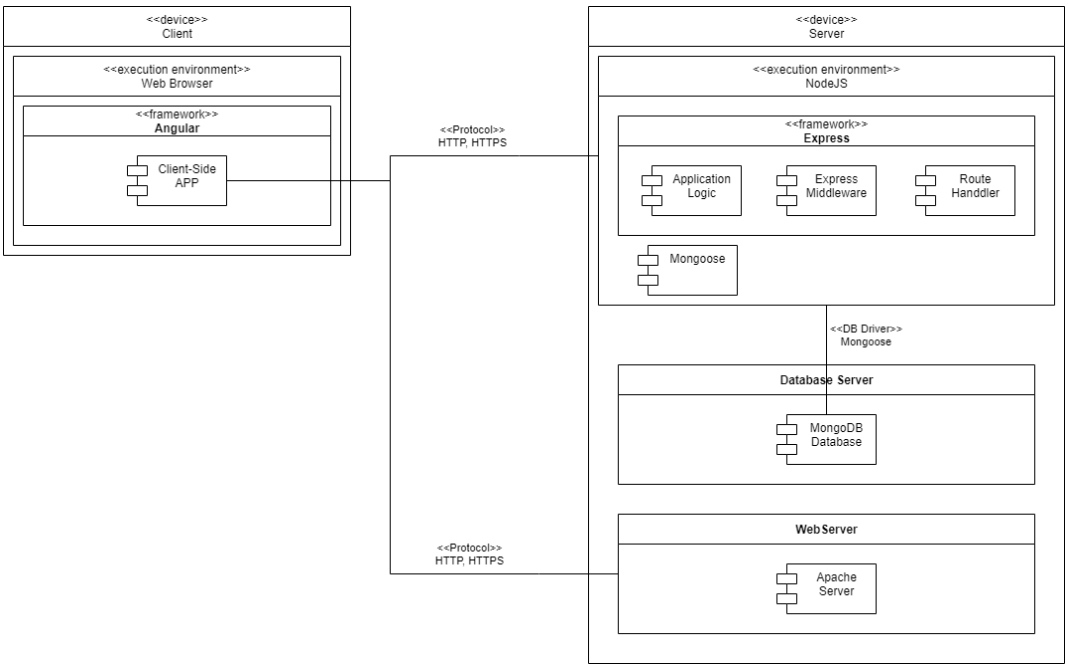

Figura 3: Arquitectura general de la aplicación

Durante el desarrollo, haremos uso de herramientas como Visual Studio Code para programar el código, junto a complementos del propio programa que nos ayudan con la corrección de errores del lenguaje Javascript y Typescript. Aparte, la gestión del control de versiones también se hace desde uno de los complementos del IDE, que se hará utilizando Git, y como repositorio remoto se utilizará GitHub.

Para la realización de pruebas, se utilizarán los programas Postman y Newman. Postman permite realizar las llamadas a la API con los métodos HTTP (GET, POST, PUT…), recibir los datos y generar conjuntos de pruebas, entre otras opciones. Newman, por su parte, permite ejecutar colecciones de pruebas, realizadas con Postman, desde la línea de comandos.

3. Requisitos

3.1 Fase 1: Servidor y dominio

Esta fase tiene como finalidad la puesta en marcha de un servidor que pueda ejecutar y servir tanto el *frontend* como el *backend* de la aplicación. También, debe estar provisto del *software* necesario para realizar el desarrollo y el mantenimiento de todo el proyecto. Al final de la fase, el proyecto ha de estar accesible a través de un dominio fácilmente reconocible y seguro.

3.2 Fase 2: Usuarios e informe de errores

En esta segunda fase tiene que quedar lista la base visual de la página. Se han de definir todos los estilos: fuentes, colores, menús, submenús, etc. La aplicación está pensada principalmente para trabajar en un ordenador de sobremesa, pero se ha de poder adaptar a los dispositivos móviles de la mejor manera posible. También, se han de crear dos entornos diferenciados, uno para labores de desarrollo y otro de producción.

Por otra parte, el principal requisito de esta fase es la parte de registro y acceso a la aplicación. Los usuarios que visiten la página, han de poder registrarse como usuarios, consultar los requisitos legales pertinentes y gestionar los datos de su cuenta una vez estén registrados. Por otra parte, los administradores han de poder gestionar los usuarios, pudiendo modificar los permisos que tienen e incluso bloquearlos. Todo esto ha de desarrollarse con unos requisitos de seguridad adecuados. En este punto, y pensando en futuras fases, se ha de poder hacer distinción entre los diferentes tipos de usuarios (Visitante, Usuario, Administrador y Máster).

Otro requisito de esta fase son los informes de errores. Se tiene que habilitar un sistema que permita a los usuarios registrados enviar informes de errores, que contengan los errores que puedan detectar al usar la aplicación. Estos informes serán recibidos por el administrador, quien los gestionará y tratará de corregirlos.

3.3 Fase 3: Personajes y habilidades de trasfondo

En esta tercera fase se añaden los personajes y las habilidades de trasfondo. Los usuarios registrados han de poder crear un personaje siempre y cuando no tengan ya uno asociado. Cada personaje creado ha de pasar una validación a través de un usuario Máster, que verifica que el personaje está creado correctamente y se adecua a la ambientación general. Si el personaje no ha pasado la validación, su creador puede modificarlo basándose en las notas que le ha dejado el Máster y volver a enviarlo a validación. Una vez validado, el personaje se puede utilizar. El usuario con permisos de Máster puede crear tantos personajes como desee y seleccionar en cada momento cual va a utilizar, pero la validación de su personaje tendrá que ser efectuada por otro Máster. Los personajes creados y validados, se pueden ver en el apartado ‘Población’ del menú. Los personajes adquieren experiencia y pueden subir de nivel participando en eventos mejorando así sus habilidades.

Por otra parte, durante la creación de personajes, este adquiere ciertas habilidades con un determinado valor o nivel. Unas de estas habilidades, son las llamadas habilidades de conocimientos, que son: 'Historia', 'Religión', 'Ocultismo' y 'Leyes'. Cuando el personaje adquiere alguna de estas habilidades de conocimientos, se habilitan diferentes pestañas en el apartado 'Conocimientos' del menú. Según su habilidad, y el nivel que tiene en esa habilidad, el usuario puede leer más o menos relatos, que aportan un mayor trasfondo o ambientación. Cada habilidad de conocimiento tiene asociados una serie de relatos que son gestionados por los usuarios con permisos de Máster, los cuales deben de poder crearlos, modificarlos, borrarlos y re-ordenarlos.

Adicionalmente, y fuera de los subobjetivos marcados para esta fase, se ha decidido implementar un editor para la página principal. Este editor será utilizado por el administrador y permitirá editar el contenido de la página principal de la aplicación con formato. También se añade una sección de ayuda con información sobre el funcionamiento de la aplicación.

3.4 Fase 4: Eventos y habilidades de escritura

En esta cuarta fase, se añaden los eventos (conocidos como misiones) y las habilidades de escritura. Los usuarios con permisos de Máster pueden crear eventos, con una pequeña historia de trasfondo a modo de introducción, donde los personajes se pueden apuntar como posibles participantes. El Máster selecciona los personajes que acabarán participando en el evento de entre todos los posibles participantes y sus propios personajes, las recompensas (dinero y experiencia) que se repartirán entre los participantes y, finalmente, habilitará su inicio. Una vez termine, el Máster cerrará el evento y las recompensas serán distribuidas equitativamente entre sus participantes. Las historias generadas en los eventos podrán ser leídas en cualquier momento y por cualquier usuario.

Los usuarios pueden participar escribiendo en alguno de los eventos en los que participa su personaje. Entrando al evento, podrá acceder al panel de escritura si ningún personaje ha reservado el turno. La reserva de turno para poder escribir será de dos días, que se liberará una vez se sobrepase el tiempo, el usuario rechace el turno o envíe su parte del relato. El panel de escritura permitirá redactar con formato y, además, hacer uso de habilidades de escritura del personaje. Al utilizar una habilidad, se muestra un panel con las distintas posibilidades que ofrece el uso de la habilidad, el usuario selecciona la acción entre las distintas opciones, se determina la dificultad del éxito y se ejecuta la prueba mostrando el resultado. El uso de estas habilidades durante la escritura, generan 'puntos de escritura' que el usuario ha de introducir en su relato; estos 'puntos de escritura' se han generado en base a la habilidad, el nivel de la misma y la dificultad de las opciones seleccionadas anteriormente. Al enviar el relato, el Máster que ha generado el evento lo recibe para su validación y, una vez comprobado que cumple con la ambientación, los 'puntos de escritura' y la continuidad con el anterior relato, lo valida y queda publicado.

4. Diseño

4.1 Fase 1: Servidor y dominio

El servidor ha de tener la capacidad suficiente para ejecutar un servidor web que pueda servir la parte del *frontend*, es decir, la aplicación que vamos a desarrollar en Angular. Para la parte del *backend*, se necesita ejecutar la aplicación desarrollada en NodeJS, aparte, esta ha de poder acceder a la base de datos MongoDB.

Para el desarrollo y mantenimiento, vamos a necesitar también un servicio de FTP, acceso SSH y SAMBA, y tener suficiente capacidad de procesamiento para atender las peticiones de los usuarios de la aplicación, así como del desarrollador.

En cuanto al número de peticiones de los usuarios, se estima un número de usuarios bajo. Esto es debido a que es una aplicación enfocada a un público bastante reducido. Se hace un estudio de proyectos similares basados en foros como: Wild Haunt [1], Harajuku Noir [2], Fall and Rise [3] y Endless Immortals [4] donde el número de usuarios registrados suele ser inferior a 150 y, activos en el mismo instante, de unos 10.

Por lo que se utilizará esta estimación de peticiones, junto con los requisitos mínimos del *software* a instalar, para determinar el *hardware* del Servidor.

4.1.1 Hardware

Entre las opciones estudiadas para el servidor, se ha barajado la posibilidad de un servidor en Amazon Web Services, a través de una cuenta AWS Educate o montar un servidor físico. Se ha optado por la segunda opción, ya que el proyecto se prevé que sea duradero y los créditos de AWS Educate son limitados, teniendo que ampliarlos con su coste asociado. De esta manera, se estima más conveniente el montaje de un servidor casero de bajo coste, ya que los requisitos del servidor son bajos, por motivos puramente económicos.

Entre las distintas opciones económicas para montar un servidor casero, se ha optado por utilizar una Raspberry Pi 4. Este *hardware* ha sido utilizado con anterioridad por el desarrollador y se posee experiencia con el mismo.

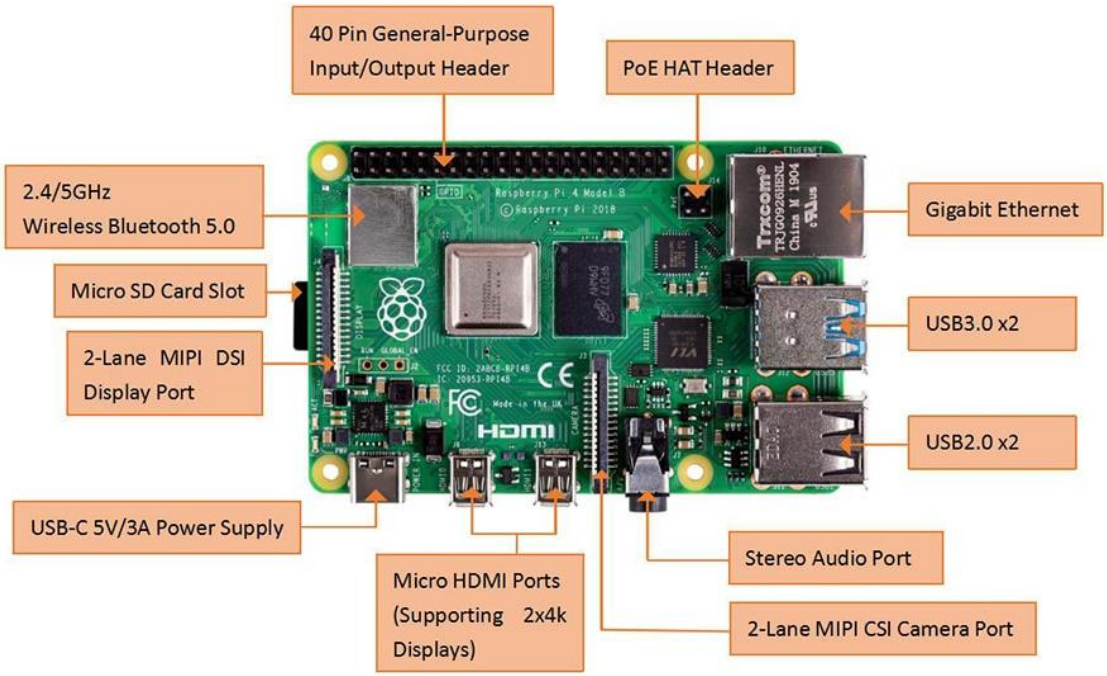

Figura 4: Hardware del Servidor Raspberry PI 4

Especificaciones del servidor: [6]

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 4GB LPDDR4-3200 SDRAM
- 2.4GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.1, Vulkan 1.0
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector
- 5V DC via GPIO header
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 ºC ambient

Adicionalmente, se han adquirido los siguientes elementos para su montaje: Adaptador de corriente de 5V-3A, Tarjeta MicroSD 128Gb y Carcasa Argon NEO.

4.1.2 Sistema operativo

Para el sistema operativo se han estudiado distintas posibilidades, ya que la Raspberry Pi admite diferentes sistemas operativos.

En un principio, se optó por utilizar el sistema operativo desarrollado por los fabricantes de la Raspberry Pi, conocido como Raspberry Pi OS. El Raspberry Pi OS en su versión de 64 Bits es muy reciente [7] y ha sido probado en diferentes versiones betas a lo largo del tiempo.

Otra alternativa sería utilizar el Raspberry Pi OS en su versión de 32 Bits, por tener un recorrido más amplio que su versión de 64 Bits. Pero puesto que algunos programas que vamos a utilizar, por ejemplo, MongoDB, ya no tienen soporte para 32 bits desde la versión 3.4 [8], tendríamos que utilizar versiones antiguas del software. Por cuestiones de Seguridad no es recomendable utilizar software desactualizado, más aún en un servidor. Vista esta problemática, se descarta el uso del sistema operativo Raspberry Pi OS en 32 Bits.

En cuanto a la opción de utilizar Windows como sistema operativo, queda descartada rápidamente. Se ha encontrado información sobre Windows 10 IoT Core (anteriormente Windows Embedded), que está orientado a desarrollar dispositivos para el Internet de las Cosas. Queda descartado, porque no es lo que buscamos.

La última alternativa valorada es utilizar Ubuntu. A través de su página web, podemos ver que ofrece el sistema operativo para la Raspberry Pi [9]. Las diferentes distribuciones vemos que se ofrecen tanto para 64 bits como para 32 bits. Encontramos por un lado la distribución Ubuntu Desktop, que convierte el dispositivo en un PC de sobremesa, con una interfaz gráfica y el software necesario para las operaciones básicas de navegar por internet, ver vídeos, escribir documentos, etc. Y, por otro lado, tenemos la distribución de Ubuntu Server, que es mucho más ligera y no contiene interfaz gráfica, así como programas de escritorio precargados. Esta distribución, se ofrece como LTS o no. La diferencia entra una distribución LTS y otra que no lo es, radica en que las distribuciones que no son LTS únicamente tienen soporte técnico durante los 18 meses después de su lanzamiento y su lanzamiento es frecuente, mientras que las LTS tienen soporte y actualizaciones durante 5 años desde su lanzamiento, en la versión servidor. Posteriormente, se pueden actualizar a versiones superiores, obteniendo el soporte y las actualizaciones de la nueva versión.

Se opta finalmente por instalar Ubuntu Server 20.04.4 LTS, por ser un sistema operativo más consolidado y conocido.

La instalación se realiza en la tarjeta *microSD* que va instalada dentro de la Raspberry Pi. Utilizando un PC de sobremesa y el software Raspberry Pi Imager 1.6.2 se selecciona la distribución a instalar a través de los menús que ofrece el software. Una vez finalizada la descarga e instalación por parte del programa, se extrae la tarjeta *microSD* y se inserta en la ranura correspondiente de la Raspberry Pi.

4.1.3 Servidor web

Se han valorado diferentes alternativas de servidor web, para ello, se han consultado los más utilizados en la actualidad:

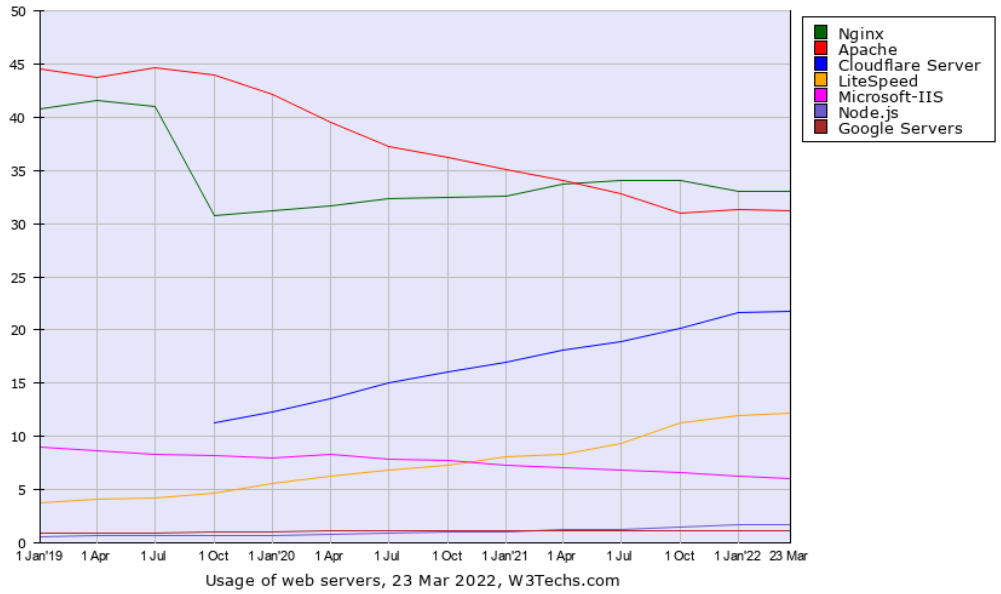

Figura 5: Porcentaje de uso de los diferentes servidores web [10]

Vemos como claros vencedores a Nginx, con una posible tendencia a la baja o a estabilizarse, y a Apache, que ha tenido una tendencia a la baja, pero parece que va a estabilizarse. Se valoran ambas soluciones.

Hemos visto que la popularidad entre ambos está muy igualada, y pese a no ser un criterio técnico sí que influye el soporte que ofrece la comunidad que hay detrás de cada uno. Las actualizaciones de seguridad son contantes en ambos. En cuanto a rendimiento, Nginx es más eficiente con los recursos informáticos, consumiendo menos RAM; su gestión de los subprocesos es más eficiente y responde mejor a las solicitudes de los clientes. Por el contrario, Apache es más flexible y personalizable, admite diferentes módulos que amplían sus funcionalidades y permite la personalización de conexiones. [11]

La aplicación a desarrollar no va a albergar a un gran número de usuarios, por tanto, las conexiones que va a recibir el servidor web van a ser pocas. Este hecho hace que la ventaja de Nginx respecto a Apache, no nos sea muy útil. En cambio, la flexibilidad de Apache con los módulos sí que nos ofrece ventajas. Pero lo más importante en este caso, es que la experiencia a la hora de configurar un servidor en Apache, es mucho mayor que la de Nginx, donde habría que partir de cero. Dada la estimación de horas de la planificación para este subobjetivo, nos decantamos por Apache, puesto con que Nginx se invertirían muchas horas de desarrollo en aprendizaje.

Finalmente, se decide optar por Apache como servidor web.

4.2 Fase 2: Usuarios e informe de errores

Para la segunda fase empezaremos creando un diagrama de casos de uso, un diagrama de clases y un mapa del sitio, basándonos en los requisitos establecidos. Estos diagramas nos servirán como base en el desarrollo posterior.

4.2.1 Diagrama de casos de uso

Para cumplir con los requisitos de esta segunda fase, se plantea el siguiente diagrama de casos de uso:

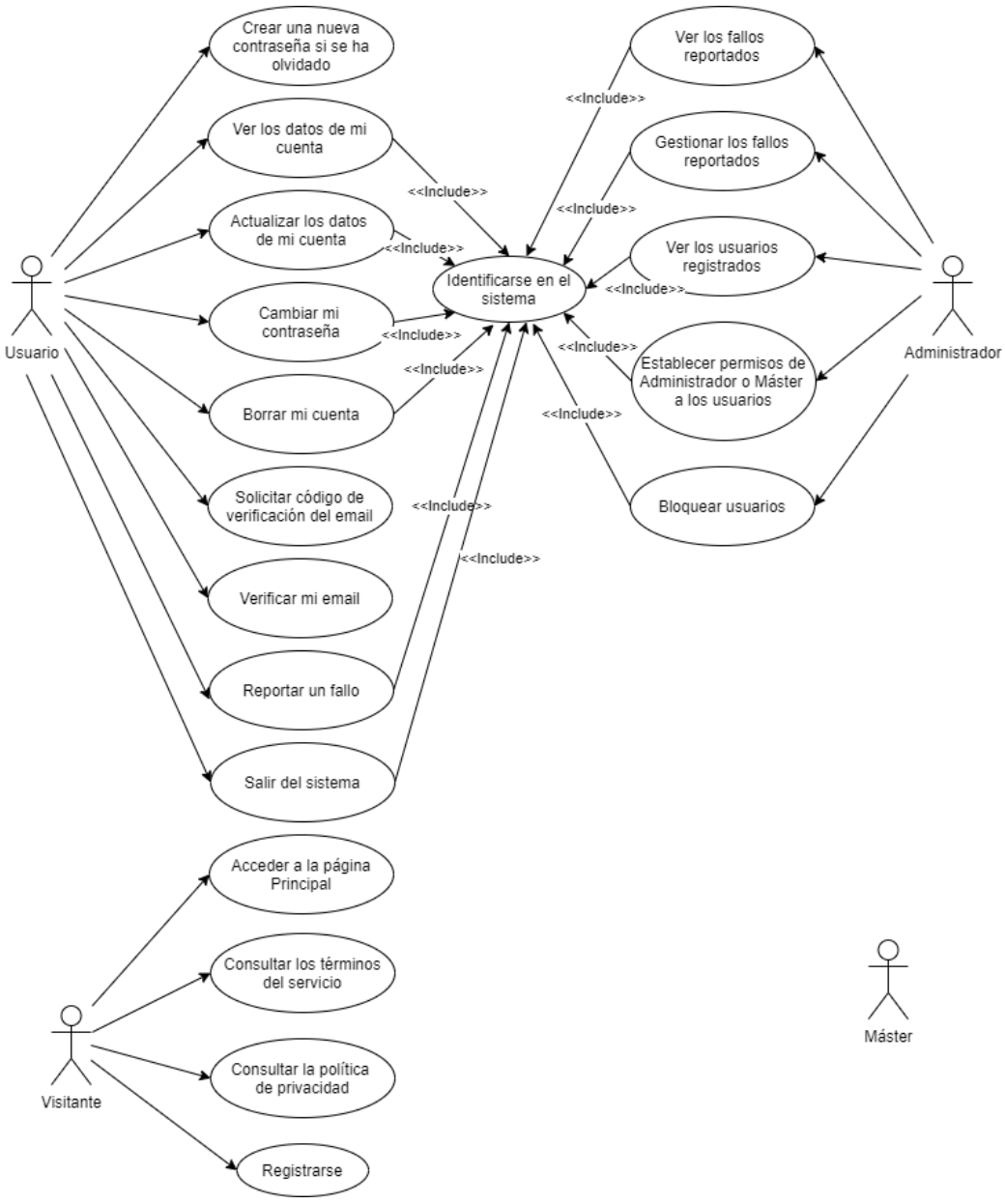

Figura 6: Diagrama casos de uso Fase 2

Se puede observar que los usuarios de tipo ‘Máster’ no tienen ninguna acción específica en esta fase.

4.2.2 Diagrama de clases

En base a los requisitos de esta fase, se plantea el siguiente diagrama de clases para ver cómo se relacionan las diferentes entidades que participan en el proyecto.

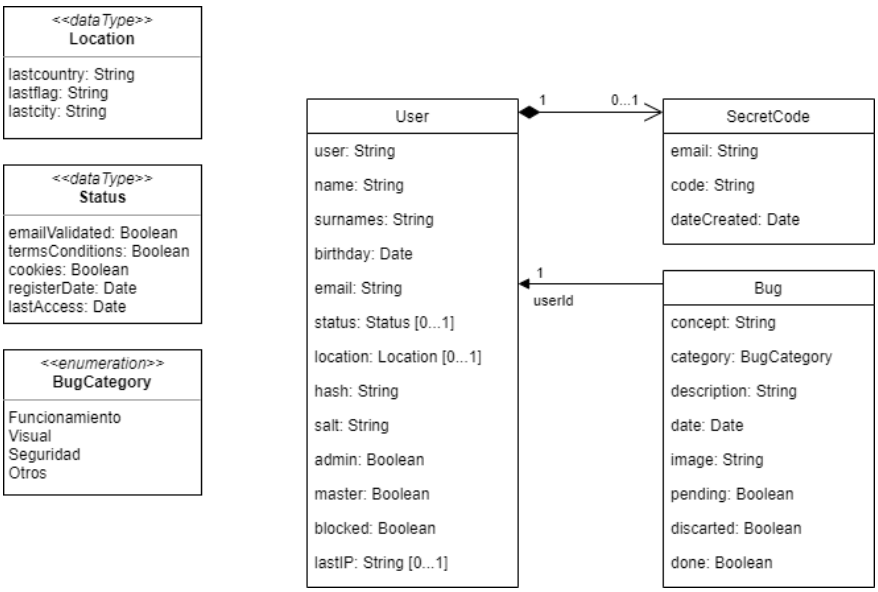

Figura 7: Diagrama de clases Fase 2

4.2.3 Mapa del sitio

El mapa del sitio muestra las vistas y los componentes a diseñar, así como la navegación, permisos e interacción que hay entre ellos. Para diseñarlo, nos basamos en los requisitos iniciales y en el desarrollo previsto del backend:

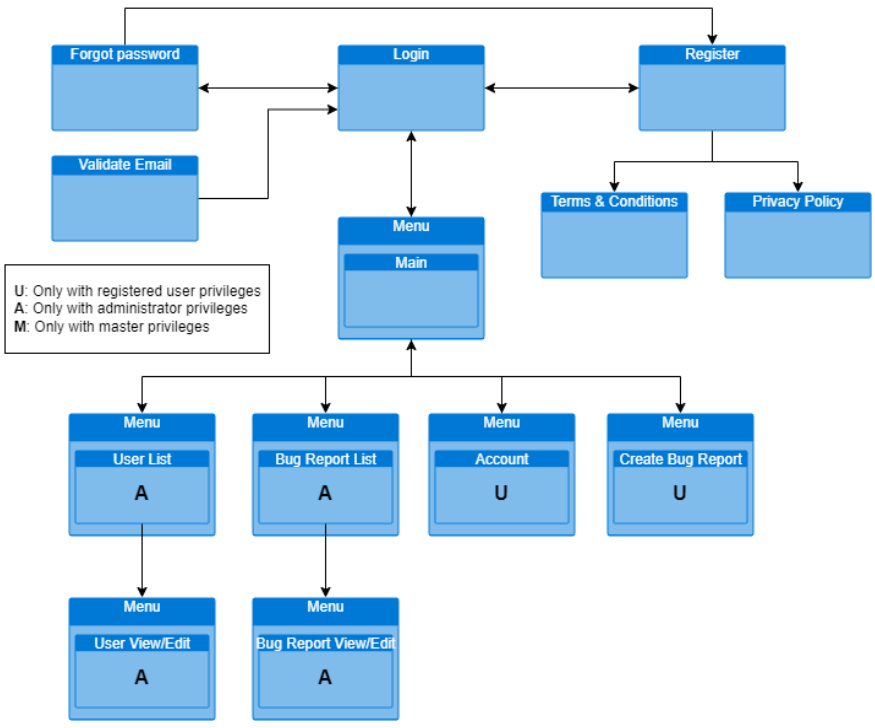

Figura 8: Mapa del sitio en Fase 2

4.3 Fase 3: Personajes y habilidades de trasfondo

Para la tercera fase añadimos nuevas funcionalidades basándonos en los requisitos establecidos, creando nuevamente un diagrama de casos de uso, un diagrama de clases y un mapa del sitio. Estos diagramas nos vuelven a servir como base en el desarrollo posterior.

4.3.1 Diagrama de casos de uso

Este diagrama, para una mejor visualización, únicamente mostrará los casos de esta fase y no de cualquier otra fase anterior.

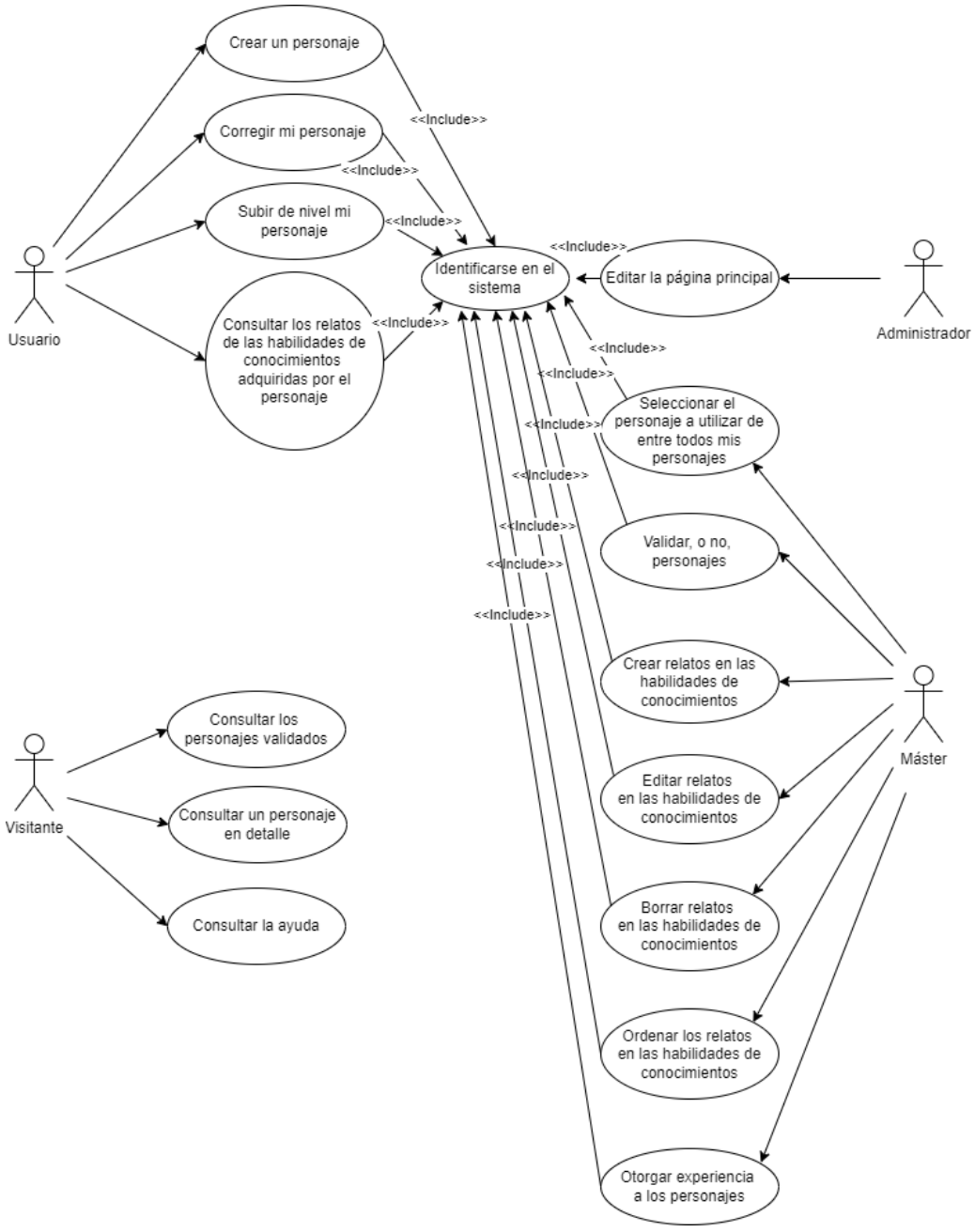

Figura 9: Diagrama casos de uso Fase 3

4.3.2 Diagrama de clases

En base a los requisitos de esta fase, se plantea el siguiente diagrama de clases para ver cómo se relacionan las diferentes entidades que participan.

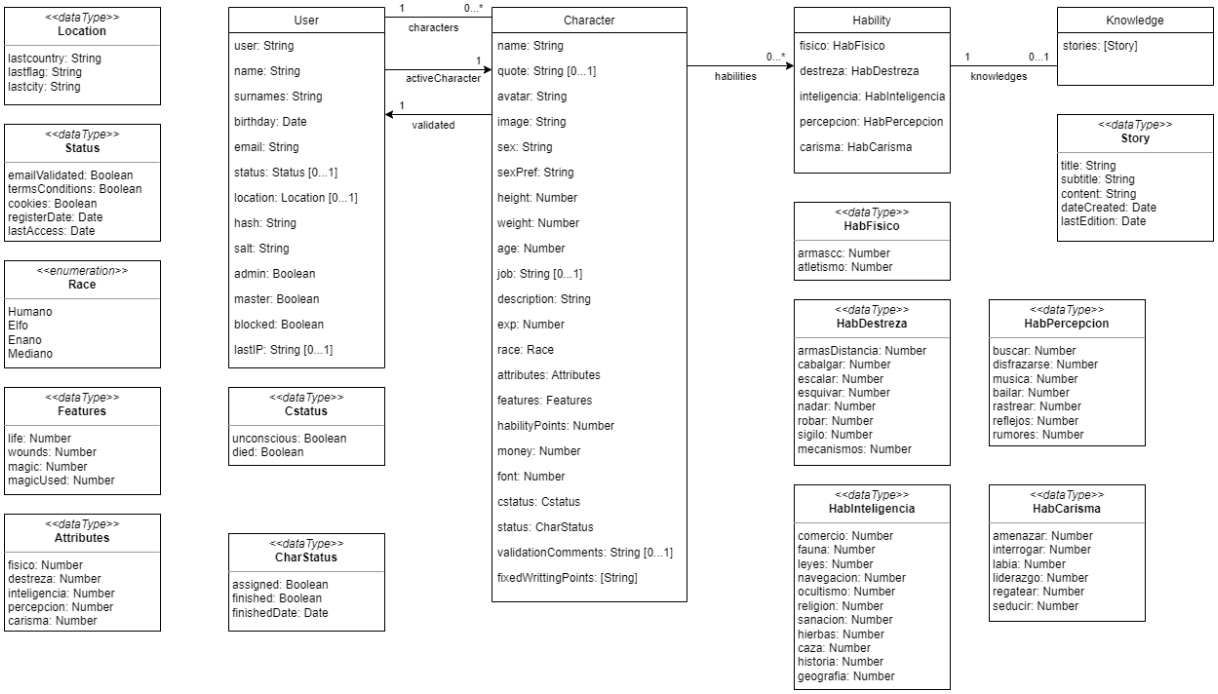

Figura 10: Diagrama de clases Fase 3

4.3.3 Mapa del sitio

Este mapa del sitio muestra las vistas y los componentes a diseñar, así como la navegación, permisos e interacciones que aparecen en esta fase.

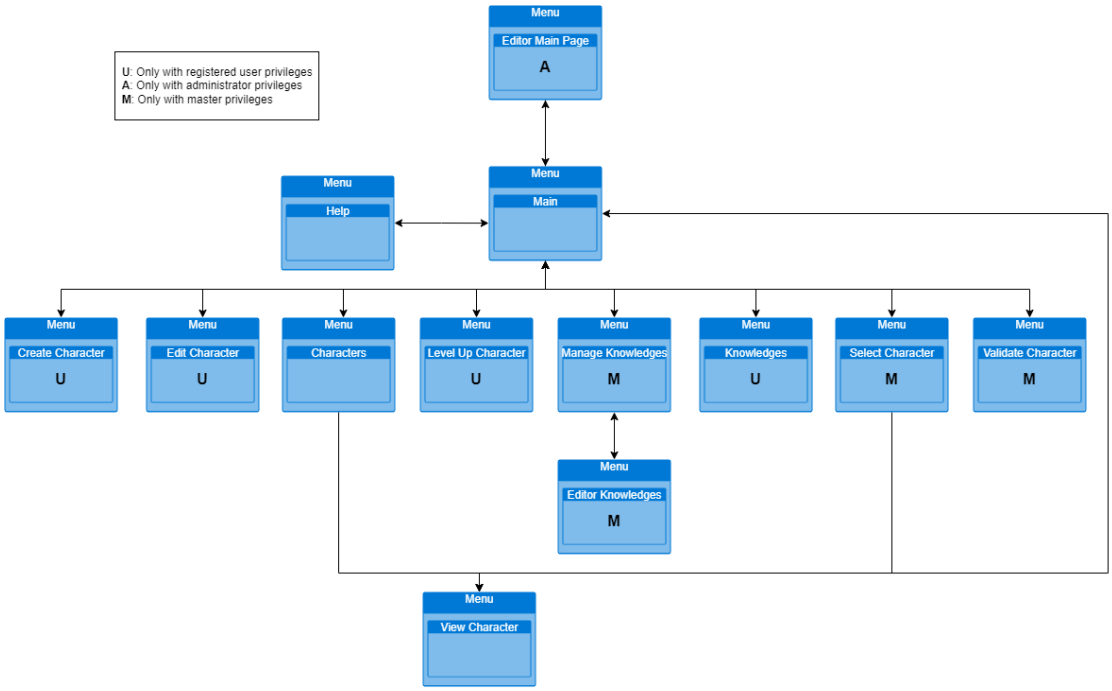

Figura 11: Mapa del sitio en Fase 3

4.4 Fase 4: Eventos y habilidades de escritura

Para la cuarta y última fase volvemos a generar un diagrama de casos de uso, un diagrama de clases y un mapa del sitio.

4.4.1 Diagrama de casos de uso

Este diagrama, para una mejor visualización, únicamente mostrará los casos de esta fase y no de cualquier otra fase anterior.

Figura 12: Diagrama casos de uso Fase 4

Se puede observar que los usuarios de tipo ‘Administrador’ no tienen ninguna acción específica en esta fase.

4.4.2 Diagrama de clases

En base a los requisitos de esta fase, se plantea el siguiente diagrama de clases para ver cómo se relacionan las diferentes entidades que participan.

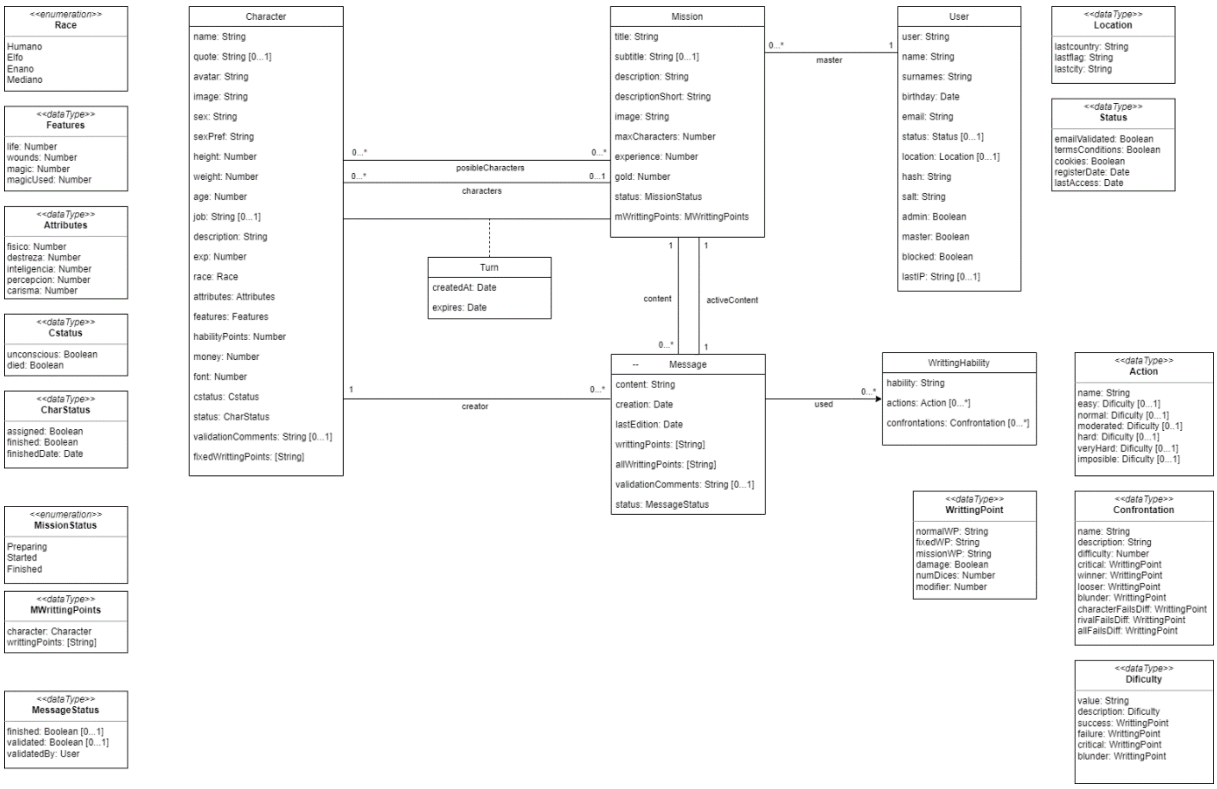

Figura 13: Diagrama de clases Fase 4

4.4.3 Mapa del sitio

Este mapa del sitio muestra las vistas y los componentes a diseñar, así como la navegación, permisos e interacciones que aparecen en esta fase.

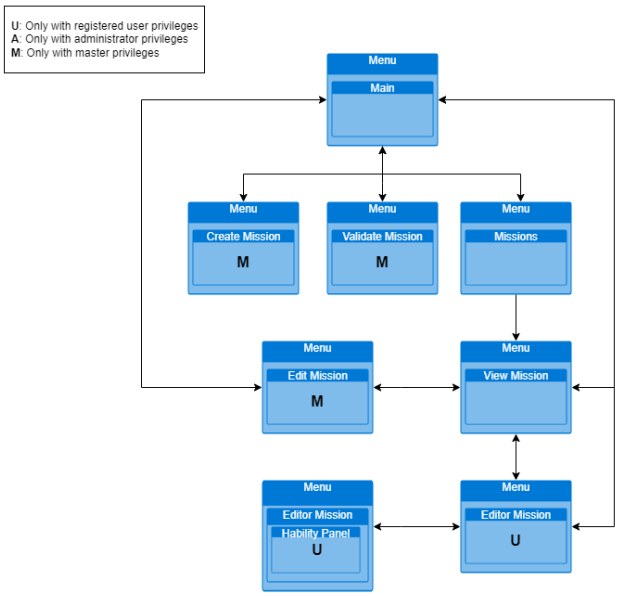

Figura 14: Mapa del sitio en Fase 4

5. Desarrollo

5.1 Fase 1: Servidor y dominio

5.1.1 Configuración de red en el Servidor

Después de instalar el sistema operativo y conectar el servidor, se accede a él a través de SSH utilizando el programa Putty. El servidor por defecto tiene habilitado por defecto el DHCP por lo que miramos en el router la dirección IP que se le ha asignado y poder acceder a el por SSH.

En el primer acceso, el usuario y contraseña por defecto serán ‘ubuntu’, el sistema solicita que se cambien, y se realiza el cambio.

Lo primero a configurar, será el establecer una IP estática, para que no tengamos que consultar en el router la dirección asignada por DHCP cada vez que queramos acceder a el servidor. Para establecer una IP estática, ejecutamos los siguientes comandos:

```
sudo nano /etc/netplan/00-installer-config.yaml
```

Aquí realizamos la configuración de red, indicando la dirección IP estática, la dirección del *Getaway* y los DNS.

Tal y como podemos observar en el fichero de configuración anterior, tenemos que deshabilitar la capacidad de configuración de la red de Cloud Init creando un fichero, ejecutamos el comando:

```
sudo nano /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg
```

Donde añadimos: ‘network: {config: disabled}‘ y guardamos.
Comprobamos que la configuración de la red es correcta:

```
sudo netplan try
```

Y la aplicamos:

```
sudo netplan apply
```

Queda establecida a partir de ahora la IP fija 192.168.1.40 para el servidor.

5.1.2 Instalación del servidor web

Comenzamos con la instalación del servidor web Apache [12]. Primero, actualizamos los paquetes locales:

```
sudo apt update
```

Instalamos el paquete apache2:

```
sudo apt install apache2
```

A partir de este punto, el servidor web Apache se encuentra instalado en el equipo. Para la administración del servidor utilizaremos los siguientes comandos a partir de ahora:

```
sudo systemctl status apache2   //Comprueba el estado de ejecución del servicio
sudo systemctl stop apache2   //Detiene el servidor web
sudo systemctl start apache2   //Inicia el servidor web
sudo systemctl start apache2   //Reinicia el servidor web
sudo systemctl reload apache2   //Aplica cambios de configuración sin cerrar conexiones
sudo systemctl disable apache2   //Deshabilita el inicio automático del servidor web
sudo systemctl enable apache2   //Habilita el inicio automático del servidor web
```

Realizamos la prueba de que el servidor está funcionando correctamente, para realizar esta prueba, abrimos el navegador y accedemos poniendo la dirección IP del servidor en la barra de direcciones. Podemos observar la pantalla de bienvenida de Apache, que nos indica que está funcionando correctamente.

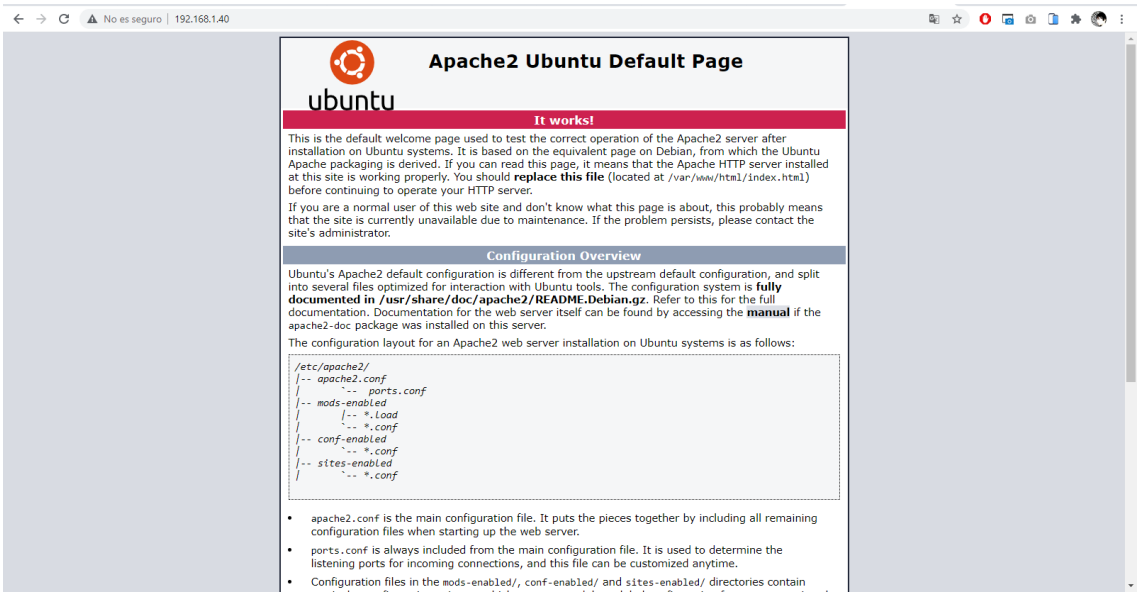

Figura 15: Comprobación del funcionamiento del servidor web Apache

5.1.3 Dominio

El dominio es necesario para que los usuarios que utilizan el proyecto, tengan un nombre memorizable y fácil de recordar a la hora de acceder. El nombre del dominio, traduce el nombre que elegimos a una dirección IP a través del sistema de nombres de dominio (DNS). El dominio elegido, apuntará a la dirección IP de nuestro router, y este redireccionará las peticiones a el Servidor.

Para el proyecto, se ha decidido optar por el nombre de ‘vinland.quest’ y se ha realizado la compra del dominio a través de Namecheap.

En la configuración del dominio, accesible a través del portal NacheCheap, se realizan dos configuraciones. Se registran dos Address Records (A name) a la dirección IP del router con el Host ‘@’ y ‘www’. En la configuración del router, se redireccionan las peticiones de los puertos 80 y 443 al Servidor.
Ahora, se puede acceder al dominio a través de las direcciones http://vinland.quest y http://www.vinland.quest.

5.1.4 DNS Dinámico

Como la IP externa del router no es fija, sino dinámica. Tenemos que hacer una serie de operaciones para que cuando se renueve la IP, por un reseteo del router o cualquier otro motivo, informe de la nueva dirección IP que se le haya otorgado. Namecheap ofrece un sistema de DNS Dinámico con el que hacer esto. Básicamente, consiste en hacer una petición HTTP con los datos de la IP actualizada para tu dominio, utilizando la contraseña que te proveen. Namecheap te ofrece un cliente para entorno Windows que realiza esta función periódicamente, pero no ofrece ninguno para Ubuntu. Por lo que buscamos alternativas ya hechas, antes de construir una propia.

En la tienda de aplicaciones Snapcraft para Linux, vemos que alguien tiene desarrollada ya una solución para Ubuntu y Namecheap [13]. Esta solución realiza la petición HTTP periódicamente con los datos que le facilitamos. Para realizar la instalación, debemos instalar primero la tienda de Snapcraft y después, el cliente. Lo realizamos con los siguientes comandos:

```
sudo apt update
sudo apt install snapd
sudo snap install namecheap-ddns-client --edge
```

Después de tener todo instalado, lo configuramos con el siguiente comando:

```
snap set namecheap-ddns-client domain=vinland.quest host=@ password=pass
```

Ahora, podemos ver que cada cinco minutos realiza una renovación de la dirección IP. Podemos comprobar su funcionamiento mirando los registros con el siguiente comando:

```
sudo journalctl -fu snap.namecheap-ddns-client.ncddns
```

5.1.5 Habilitar el protocolo SSL/TLS en el servidor web

La ‘seguridad de la capa de transporte’ (*Transport Layer Security* o TLS) y su protocolo anterior ‘capa de puertos seguros’ (*Secure Sockets Layer* o SSL) son unos protocolos criptográficos que ofrecen seguridad e integridad en las comunicaciones.

Estos protocolos, utilizan certificados X.509 y, por tanto, criptografía asimétrica. Hacen uso de claves de sesión para que cada sesión sea segura y única. Durante la conexión inicial (*handshake*), las claves pública y privada se utilizarán para crear una clave de sesión, que luego se utilizará para cifrar y descifrar los datos que se transfieren. Esta clave de sesión seguirá siendo válida por un tiempo limitado y solo se utilizará para esa sesión en particular. [14]

Primero se configura un *virtual host* en Apache. Los servidores virtuales (*virtual host*) permiten administrar más de un sitio y tener varias configuraciones en un mismo servidor Apache. En este caso, lo vamos a configurar para tener la configuración fácilmente identificable y para que el proceso posterior para conseguir habilitar el protocolo SSL/TLS sea más sencillo. [15]

Creamos un directorio con el nombre del dominio en la ruta '/var/www' donde actualmente está la carpeta 'html' que contiene el host actual. Ejecutamos el siguiente comando:

```
sudo mkdir /var/www/vinland
```

Otorgamos los permisos necesarios:

```
sudo chmod 775 /var/www/vinland
```

Creamos el fichero de configuración con las directivas necesarias:

```
sudo nano /etc/apache2/sites-available/vinland.conf
```

Que incluye el siguiente contenido:

```
<VirtualHost *:80>
    ServerAdmin admin@vinland.quest
    ServerName vinland.quest
    ServerAlias www.vinland.quest
    DocumentRoot /var/www/vinland
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Habilitamos el fichero de configuración haciendo uso de la herramienta a2ensite:

```
sudo a2ensite vinland.conf
```

Posteriormente, se deshabilita el host que está por defecto:

```
sudo a2dissite 000-default.conf
```

Se comprueba que no haya errores:

```
sudo apache2ctl configtest
```

Y, por último, se resetea el servicio de Apache:

```
sudo systemctl restart apache2
```

Ahora mismo, el host de Vinland está configurado en Apache. Se carga un fichero HMTL en la carpeta como index.html y se accede correctamente a través de la IP del servidor.

Pasamos ahora a utilizar Certbot para la obtención de un certificado SSL gratuito y cuya renovación se haga de manera automática. Realizamos la instalación de los paquetes 'certbot' y 'python3-certbot-apache':

```
sudo apt install certbot python3-certbot-apache
```

El paquete 'python3-certbot-apache' permite la integración de Certbot con Apache, obtiene los certificados y configura HTTPS en el servidor.

Ejecutamos el comando para obtener los certificados con Certbot, durante la obtención del certificado, se nos hacen una serie de preguntas y una aceptación

de los términos y condiciones de Let's Encrypt. Let's Encrypt es una entidad certificadora que otorga certificados TLS/SSL de forma gratuita.

```
sudo certbot --apache
```

Ahora el servidor ya es accesible por HTTPS. Se hace una prueba con un fichero index.html accediendo a https://vinland.quest y la prueba resulta satisfactoria.

Los certificados de Let's Encrypt tienen una validez de noventa días, pero el paquete de Certbot se encarga de realizar la renovación de los certificados cuando corresponde.

Se puede comprobar el estado del servicio de Certbot a través del comando:

```
sudo systemctl status certbot.timer
```

O bien realizar un simulacro de renovación a través del comando:

```
sudo certbot renew --dry-run
```

5.1.6 Servidor FTP

El servicio de FTP se va a utilizar para subir los archivos de distribución del *frontend*, es decir, la página web. El protocolo funciona con una arquitectura cliente-servidor permitiendo establecer una comunicación y transferir archivos. Utiliza el puerto 21 TCP para llevar a cabo este cometido. El acceso únicamente será posible desde la red local y no se abrirá ningún puerto hacia el exterior.

Comentamos la instalación de Vsftpd, que es un servidor FTP para sistemas Unix: [16]

```
sudo apt update
sudo apt install vsftpd
sudo nano /etc/vsftpd.conf
```

Para configurarlo, editamos las siguientes líneas del código:

```
write_enable=YES   //Permite la escritura
chroot_local_user=YES   //Habilita el Chroot de los usuarios del sistema
chroot_list_enable=YES   //Permite crear una lista con los usuarios en Chroot
chroot_list_file=/etc/vsftpd.chroot_list   //Lista de usuarios y sus rutas
```

Editamos el fichero 'chroot_list' para añadir el usuario Ubuntu:

```
sudo nano /etc/vsftpd.chroot_list
```

Se añade Ubuntu. Este usuario tendrá acceso a todo y se configurará una conexión en Filezilla (el cliente FTP que utilizaremos) para que por defecto acceda al directorio del proyecto.
Se prueba a hacer una conexión y a transferir ficheros. El resultado es correcto.

5.1.7 Servidor Samba

Samba es un protocolo de archivos compartidos de Microsoft Windows. El servidor Samba se va a utilizar para poder programar la parte del *backend*. Estos ficheros se encuentran ubicados en el servidor y se programan desde una PC en la misma red local. Por lo que necesitamos acceso a los ficheros.

Comenzamos la instalación:

```
sudo apt update
sudo apt install samba
sudo nano /etc/samba/smb.conf
```

Para configurarlo, editamos las siguientes líneas del código:

```
read only = no   //Permite la escritura y modificación de los ficheros
 [backend]
    comment = Backend
    path = /var/www/backend
    read only = no
    browsable = yes
    force create mode = 0777
    force directory mode = 0777
```

Samba no utiliza por defecto la contraseña que tiene el usuario en el sistema, debemos otorgarle una:

```
sudo smbpasswd -a ubuntu
```

Por último, reiniciamos el servicio de Samba:

```
sudo service smbd restart
```

Comprobamos que podemos acceder correctamente. En el explorador de Windows, escribimos '//192.168.1.40' y nos muestra las carpetas del servidor que hemos compartido.

5.1.8 NodeJS y NPM

El Backend consiste en el desarrollo de la API que vamos a utilizar en el proyecto. Para desarrollar la API, se hace uso de NodeJS que es un entorno de ejecución multiplataforma basado en el lenguaje de programación Javascript que utiliza una arquitectura orientada a eventos [17]. También utilizaremos NPM, que es un sistema de gestión de paquetes para NodeJS que permite manejar las dependencias e instalar aplicaciones que se encuentren en el repositorio [18].

Realizamos la instalación:

```
sudo apt install nodejs
sudo apt install npm
```

5.1.9 MongoDB

MongoDB es una base de datos NoSQL orientada a objetos, donde los datos son almacenados en estructuras BSON con un esquema dinámico [19]. Con esta base de datos, conseguimos que la integración con la aplicación que estamos desarrollando en NodeJS sea fácil y rápida.

Se realiza la instalación: [20]

```
sudo apt update
sudo apt install mongodb
```

Tras confirmar que el servicio se está ejecutando como se espera, habilite el servicio MongoDB para que se inicie en el arranque:

```
sudo systemctl enable mongodb
```

Se crean dos usuarios nuevos, uno para el acceso desde la aplicación desarrollada en NodeJS y otro para el acceso desde otro PC de la red, para tareas de consulta y mantenimiento:

```
mongo
use admin
db.createUser(
{
user: "api",
pwd: "pass",
roles: [ { role: "root", db: "admin" } ]
})
db.createUser(
{
user: "ubuntu",
pwd: "pass",
roles: [ { role: "root", db: "admin" } ]
})
```

Editamos el archivo de configuración para añadir la IP fija del servidor y la del pc remoto de la red

```
sudo nano /etc/mongodb.conf
```

Realizamos los siguientes cambios, donde '192.168.1.30' es la dirección IP del PC de la red local que se encargará de la gestión:

```
bind_ip = [127.0.0.1,192.168.1.40,192.168.1.30]
auth=true
```

Reiniciamos el servicio:

```
sudo service mongod restart
```

Para acceder a la base de datos desde Windows se utilizará MongoDB Compass. Mientras que si queremos acceder desde el servidor, utilizaremos el siguiente comando:

```
mongo -u ubuntu -p pass 192.168.1.40 --authenticationDatabase "admin"  //Acceder
```

5.1.10 Gestión del proceso de la aplicación desarrollada en NodeJS

Al contrario de lo que ocurre con todo el software que hemos instalado de momento (Apache, MongoDB, Vsftpd, Samba, etc.) la aplicación que vamos a desarrollar no se iniciará automáticamente con el arranque del sistema operativo. Necesitamos que se inicie con el arranque, para que un corte de luz o un reinicio del servidor, no nos obligue a tener que iniciar una actuación para levantar el servicio. Para realizar este cometido, utilizaremos PM2. PM2 es un administrador

de procesos para aplicaciones que han sido desarrolladas con NodeJS. Esta aplicación permite que las aplicaciones que hemos desarrollado funcionen en segundo plano como servicios, además, se reiniciarán de forma automática en el caso que la aplicación se bloquease o se detuviese. [21]

```
sudo npm install pm2@latest -g
```

Añadimos el comando para iniciar la aplicación a la lista de procesos de PM2 con el siguiente comando. Añadimos de paso, un nombre para poder identificar el proceso fácilmente:

```
sudo pm2 start npm --name "vinland-quest" -- start
sudo pm2 startup centos
sudo pm2 save
```

Para la gestión de los procesos en PM2 disponemos de los siguientes comandos:

```
sudo pm2 stop vinland-quest   //Detiene el servicio
sudo pm2 restart vinland-quest   //Reinicia el servicio
sudo pm2 list   //Muestra todos los procesos de PM2
```

5.1.11 Redirección para la aplicación desarrollada en Angular

A la hora de desplegar la aplicación desarrollada en Angular, hay que redirigir el tráfico al fichero 'index.html' para que funcionen correctamente las rutas cuando accedemos desde el exterior. Se puede consultar una guía para hacerlo en el apartado de despliegue de la web de Angular [22].

Entramos a editar el fichero:

```
sudo nano /etc/apache2/sites-available/vinland-le-ssl.conf
```

Y agregamos las siguientes líneas dentro de la etiqueta 'Virtual-Host'. Estas etiquetas habilitan la redirección a el fichero index.html:

```
RewriteEngine On
# If an existing asset or directory is requested go to it as it is
RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -f [OR]
RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -d
RewriteRule ^ - [L]

# If the requested resource doesn't exist, use index.html
RewriteRule ^ /index.html
```

Cabe comentar, que la redirección la estamos haciendo únicamente en el acceso por HTTPS, ya que cuando accedemos por HTTP ya nos redigire a HTTPS.

Y para finalizar, reiniciamos apache para que utilice la nueva configuración:

```
sudo systemctl restart apache2
```

5.2 Fase 2: Usuarios e informe de errores

5.2.1 Entornos

En esta fase se van a crear los dos entornos de desarrollo que utilizaremos a lo largo de todo el proyecto.

Existen dos entornos, el de desarrollo, el cual se utilizará durante el desarrollo del proyecto y donde todas las pruebas se realizarán utilizando una base de datos diferente. Y, por otra parte, tenemos el entorno de producción, que ejecuta la última versión del proyecto utilizando la base de datos que alberga toda la información. Cada uno de estos procesos, hará uso de diferentes puertos para poder ejecutarse a la vez.

En el *backend* se hace la distinción entre un entorno de desarrollo u otro en la programación ayudándose de la librería Dotenv. En el *frontend*, hay definidos dos ficheros en la carpeta 'environments' que definen las rutas de acceso a la API en base al entorno utilizado.

El proceso de NodeJS que ejecuta la versión de producción se encontrará siempre en activo, para ello, en la anterior fase hemos hecho uso del administrador de procesos PM2. Para cargar una nueva versión en el proceso de producción, ejecutaremos el siguiente comando:

```
sudo pm2 restart vinland-quest
```

Mientras que, para ejecutar la versión de producción del *frontend*, será necesario ejecutar el siguiente comando, en la carpeta del proyecto del *frontend*, para generar los ficheros y luego subirlos por FTP al servidor web:

```
ng build
```

Por otro lado, el proceso de NodeJS que ejecuta la versión de desarrollo no se encuentra siempre activo, y tenemos que ejecutar el proceso manualmente con el siguiente comando:

```
cd /var/www/backend/
sudo npm test
```

Una vez se terminen las pruebas de desarrollo, se detiene el proceso.
Mientras que, para ejecutar la versión de desarrollo del frontend, será necesario ejecutar el siguiente comando, en la carpeta del proyecto del frontend:

```
cd vinland
ng serve
```

Ahora ya se puede acceder a través de http://localhost:4200
Sobre la base de datos, en desarrollo se utiliza 'API-tfg' y en producción 'api'.

5.2.2 Autentificación, seguridad y privacidad

En cuanto al sistema de autentificación de la aplicación, se han valorado diferentes métodos para llevarlo a cabo. Entre las opciones valoradas, están las que hacen uso de NodeJS, Express y Passport; las opciones de usar OAuth (donde un servicio externo como Google provee la identificación del usuario); o el uso de un sistema de autentificación propio, que queda descartado dada su complejidad y la necesidad de cumplir con los plazos de la fase. Podemos optar, por tanto, por una estrategia local (con estado, haciendo uso de 'passport-local', utilizando Cookies y sesiones de Express) u optar por una estrategia basada en JSON Web Tokens (sin estado, haciendo uso de 'passport-jwt' y utilizando tokens en formato JWT).

El uso de una estrategia local, con *cookies* y sesiones, permite almacenar en una *cookie* un identificador que permite al servidor hacer un seguimiento de los usuarios que han iniciado sesión, de esta manera no es necesario enviar las credenciales en cada petición, aunque sí la id de la cookie.

Por otro lado, el uso de *tokens* hace necesario enviar el *token* en cada petición y el servidor decide otorgar acceso o no. Nos evita tener que guardar el estado de las sesiones en el servidor, aunque el tamaño de los datos a transferir cada vez sea mayor.

Pese a que ambas opciones son igual de válidas, se ha optado por implementar una estrategia basada en JWT por tener una mayor experiencia en este sistema y dado que evita tener problemas con cookies, ya que algunos clientes pueden no a soportar cookies. [23] [24]

El formato de JWT permite almacenar diferentes datos en la parte del *payload*, y la firma del JWT evita que estos datos puedan ser cambiados. En el proyecto, añadiremos los permisos de Administrador o Máster en el *token* además del id del usuario. Podemos ver la información que contiene un JWT en la siguiente imagen:

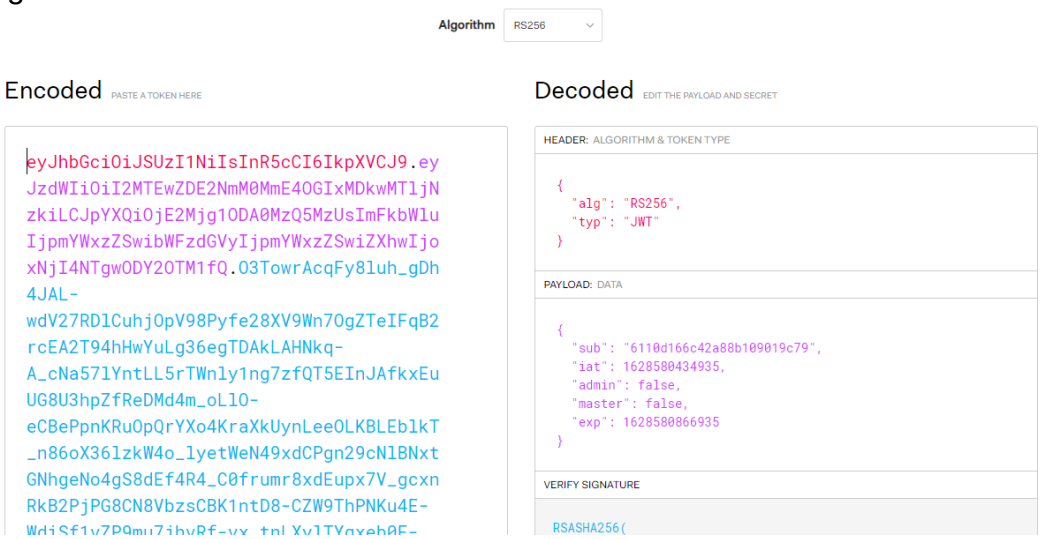

Figura 16: Estructura decodificada de un JWT [25]

El JWT lo almacenaremos en el almacenamiento local del navegador.

Sobre la seguridad y privacidad, se comentan algunas de las opciones de seguridad que se han implementado:

- Los JWT no contienen ningún tipo de información confidencial, ya que la parte del *payload* no se encuentra encriptada.
- En la base de datos no se guardan las contraseñas directamente, si no que se hacen a través de un sistema de *hash* y *salt*.
- Las peticiones a la API en producción únicamente se pueden hacer a través de los dominios registrados de Vinland.
- El acceso a la aplicación web y a la API se hace a través de HTTPS.

5.2.3 Backend

Librerías

Para el desarrollo, hemos utilizado diferentes librerías en nuestro proyecto que nos han ayudado a que el proyecto implementase soluciones ya desarrolladas y probadas. Para la instalación de estas librerías en el proyecto se ha utilizado NPM (*Node Package Manager*). Se muestra a continuación las librerías utilizadas y su función dentro del proyecto:

- **Connect-multiparty:** Es un *middleware* que se utiliza para poder realizar la subida de ficheros en las rutas que lo requieran.
- **Dotenv:** Permite la creación de variables de entorno, para poder gestionar correctamente el entorno el que cual estamos desarrollando o testeando la aplicación (producción o desarrollo). Estas variables se definen en un fichero '.env' en la raíz del proyecto. Cabe decir que muchas de estas variables contienen información de contraseñas y por tanto, este fichero '.env' está incluido en el fichero '.gitignore' y no aparece subido en el repositorio.
- **Express:** Es un *framework* para NodeJS que permite el desarrollo de aplicaciones web y API's. Es uno de los pilares fundamentales en el desarrollo del proyecto, permitiéndonos responder correctamente a las solicitudes hechas por el cliente.
- **Ipstack:** API que nos permite obtener la geolocalización de un usuario en base a su IP.
- **Mongoose:** Nos va a permitir conectarnos, realizar consultas y modelar objetos en nuestra base de datos MongoDB.
- **Nodemailer:** Con esta librería podemos realizar el envío de correos electrónicos desde nuestra aplicación.
- **Nodemon:** Es una interfaz de línea de comandos que permite reiniciar el proceso de NodeJS cuando se realizan cambios en el código. Se utilizará en el entorno de desarrollo.
- **Passport:** Es un *middleware*, compatible con Express, que nos permite implementar diferentes *plugins* con distintas estrategias de autentificación.
- **Passport-jwt:** *Plugin* para Passport que permite la autentificación para el acceso a las rutas, mediante un JSON Web Token.
- **Path:** Proporciona diferentes herramientas para trabajar con ficheros y directorios.
- **Public-ip:** Permite obtener la IP pública de acceso de los usuarios.

Claves y Autentificación

La aplicación hace uso de una clave pública y privada, que hemos generado con el fichero Javascript ‘generateKeyPair.js’. Estas claves las utilizaremos en la configuración de Passport, para la autentificación de los usuarios. Las llaves están guardadas en la carpeta ‘config’ junto con el fichero de configuración de Passport llamado ‘passport.js’. En este fichero de configuración implementamos la estrategia de autentificación de usuarios con JSON Web Token (JWT) utilizando las claves anteriormente mencionadas. Esta autentificación, se complementa haciendo uso de un *middleware* llamado ‘authentification.js’ en la carpeta ‘middlewares’, que permite saber el rol que tiene el usuario, o si está bloqueado, decodificando el JWT.

Configuración

En la carpeta ‘config’ podemos encontrar también el fichero ‘database.js’ que realiza la conexión a la base de datos MongoDB utilizando Mongoose. Dependiendo de si estamos en producción o en desarrollo, hará la conexión a una base de datos u otra. En esta carpeta también encontramos el fichero ‘emailService.js’ que configura el envío de correos para la librería ‘nodemailer’.

Controladores

En la carpeta ‘controllers’ nos encontramos los controladores que utilizaremos en la aplicación con sus respectivos métodos. Pasamos a comentar cada uno de estos controladores y una breve descripción de cada uno de los métodos que se han implementado:

Bug, este controlador se encarga de gestionar los informes de errores. Sus métodos son los siguientes:

- **createBug:** Permite crear un informe de errores.
- **getBugs:** Devuelve todos los informes de errores existentes.
- **getBug:** Obtiene un informe de errores en base al id proporcionado por parámetros.
- **updateBug:** Actualiza el informe de errores, cuyo id se ha pasado por parámetros, a un nuevo estado (*discarted*, *done* o *pending*).
- **uploadBugImage:** Sube una imagen al servidor (carpeta ‘uploads/bugs’) y devuelve su nombre de fichero. Se hace una comprobación previa de formato de imagen y tamaño adecuados.

Image, este controlador se encarga servir las imágenes que se han subido a la carpeta ‘uploads’, o a alguna de sus subcarpetas. Sus métodos son los siguientes:

- **getImageFile:** Permite obtener un fichero de imagen ubicado en la carpeta ‘uploads’.
- **getImageBug:** Permite obtener un fichero de imagen ubicado en la carpeta ‘uploads/bugs’.

Ip, este controlador se encarga de proporcionar la IP y la localización de un usuario cuando accede a la aplicación. Sus métodos son los siguientes:

- **getIP:** Proporciona la IP a partir del objeto *request* proporcionado en la aplicación Express.
- **getLocation:** Haciendo uso de la IP y de la API de 'ipstack', obtenemos la información de localización del usuario.

User, este controlador se encarga del registro de nuevos usuarios, proporcionar acceso a la aplicación, gestionar los usuarios y, en definitiva, de todo lo relacionado con los usuarios. Sus métodos son los siguientes:

- **login:** Comprueba que el usuario y su contraseña son correctos, que el usuario ha validado su email y que no es un usuario bloqueado. En caso que el acceso sea correcto, se genera y envía un JWT con una validez de 5 días. Posteriormente, se actualiza la información de la última IP de acceso y localización en el usuario.
- **register:** Realiza la comprobación de toda la información proporcionada: coincidencia de las contraseñas, formato correcto de los datos, aceptación de las condiciones, etc. Y en caso que no haya error, genera un nuevo usuario. Posteriormente, genera un código de verificación del usuario (válido durante 10 minutos) y se envía por correo al email que ha proporcionado el usuario.
- **verifyAccount:** Valida la cuenta de un usuario, haciendo uso del id de la cuenta y el código de verificación. Si todo ha ido correctamente, actualiza el estado del usuario, como que ya es un usuario validado. Posteriormente, elimina todos los códigos de verificación que tuviese asociados el usuario.
- **getActivationMail:** Crea un nuevo código de verificación para el usuario que lo solicita, elimina los que pudiese tener anteriormente, y se lo envía al correo con el que se ha registrado.
- **passwordResetCode:** Genera un código de restablecimiento de contraseña (válido durante 10 minutos) que se envía al correo del usuario que quiere cambiar la contraseña.
- **passwordReset:** El usuario puede cambiar su contraseña, proporcionando un código de restablecimiento de contraseña válido y la información necesaria. Se comprueba el formato y la coincidencia de las contraseñas, en caso que todo esté correcto, se cambia la contraseña.
- **updateAccount:** El usuario puede actualizar los datos de nombre, apellidos o fecha de cumpleaños de su cuenta.
- **deleteAccount:** Elimina la cuenta del usuario que lo solicita. Únicamente se borra el usuario, cualquier información que haya generado en la aplicación se guarda. Se envía información del usuario y su contraseña.
- **getUsers:** Proporciona la información de todos los usuarios.
- **getUser:** Proporciona la información del usuario que ha realizado la solicitud.
- **getUserById:** Proporciona la información del usuario cuya id se ha suministrado.
- **setMaster:** Otorga permiso de 'Máster' a un usuario.
- **unsetMaster:** Quita el permiso de 'Máster' a un usuario.
- **setAdmin:** Otorga permiso de 'Administrador' a un usuario.

- **unsetAdmin:** Quita el permiso de 'Administrador' a un usuario.
- **blockUser:** Bloquea a un usuario impidiéndole el acceso.
- **unblockUser:** Desbloquea a un usuario permitiéndole de nuevo el acceso.

Modelos

En cuanto a los modelos, los podemos encontrar en la carpeta 'models'. Aquí tenemos tres ficheros: 'bug.js', 'secretCode.js' y 'user.js' que se han definido acorde al diagrama de clases que hemos diseñado. Comentar únicamente que los objetos creados con el esquema de 'secretCode' expiran siempre al pasar 10 minutos.

Rutas

Por otra parte, tenemos las rutas de acceso a la API que se han implementado. Están divididas en tres ficheros distintos, por facilidad a la hora de otorgar permisos. Así pues, en la carpeta 'routes' encontramos el fichero 'admin.js' con una serie de rutas a las que únicamente tiene acceso el administrador, encontramos también el fichero 'user.js' que contiene las rutas a las que puede acceder un usuario registrado y, por último, tenemos el fichero 'visitor.js' que contiene las rutas accesibles por todo el mundo. También se tiene en cuenta que un usuario bloqueado no pueda acceder a las rutas. Se exponen y explican a continuación cada una de estas rutas clasificadas por permisos:

La ruta base de la API en producción es: https://vinland.quest:3443/api
La ruta base de la API en desarrollo es: https://vinland.quest:2443/api

- Rutas de Administrador:
  - **GET /users** → Llama al método 'getUsers' del controlador 'user'. Recibe todos los usuarios.
  - **GET /user/:id** → Llama al método 'getUserById' del controlador 'user'. Recibe el usuario cuya id se ha pasado por parámetros.
  - **PUT /setMaster** → Llama a el método 'setMaster' del controlador 'user'. Otorga permisos de máster al usuario pasado por el cuerpo de la solicitud.
    Los datos clave-valor a enviar son:
    - user: contiene el usuario.
  - **PUT /unsetMaster** → Llama a el método 'unsetMaster' del controlador 'user'. Quita el permiso de máster al usuario pasado por el cuerpo de la solicitud.
    Los datos clave-valor a enviar son:
    - user: contiene el usuario.
  - **PUT /setAdmin** → Llama a el método 'setAdmin' del controlador 'user'. Otorga permisos de administrador al usuario pasado por el cuerpo de la solicitud.
    Los datos clave-valor a enviar son:
    - user: contiene el usuario.
  - **PUT /unsetAdmin** → Llama a el método 'unsetAdmin' del controlador 'user'. Quita el permiso de administrador al usuario pasado por el cuerpo de la solicitud.
    Los datos clave-valor a enviar son:

    - user: contiene el nombre del usuario.
  - **PUT /blockUser** → Llama a el método ‘blockUser’ del controlador ‘user’. Bloquea al usuario pasado por el cuerpo de la solicitud.
    Los datos clave-valor a enviar son:
    - user: contiene el usuario.
  - **PUT /unblockUser** → Llama a el método ‘unblockUser’ del controlador ‘user’. Desbloquea al usuario pasado por el cuerpo de la solicitud.
    Los datos clave-valor a enviar son:
    - user: contiene el usuario.
  - **GET /getBugs** → Llama al método ‘getBugs’ del controlador ‘bug’. Recibe todos los informes de errores.
  - **GET /getBug/:id** → Llama al método ‘getBug’ del controlador ‘bug’. Recibe el informe de error cuya id se ha pasado por parámetros.
  - **PUT /updateBug/:id/:status** → Llama al método ‘updateBug’ del controlador ‘bug’. Actualiza el estado del informe de errores cuya id se ha pasado por parámetros, por el estado que se ha pasado en su parámetro correspondiente.
  - **GET /getImgBug/:image** → Llama al método ‘getImageBug’ del controlador ‘image’. Devuelve el fichero de imagen de la imagen solicitada por parámetros.
    Los datos clave-valor a enviar son:
    - image: nombre de la imagen.

- Rutas de Usuario:
  - **GET /user** → Llama al método ‘getUser’ del controlador ‘user’. Recibe la información del usuario que ha hecho la solicitud.
  - **PUT /updateAccount** → Llama al método ‘updateAccount’ del controlador ‘user’. Actualiza la información de la cuenta del usuario que ha hecho la solicitud con la información pasada por el cuerpo de la solicitud.
    Los datos clave-valor a enviar son:
    - name: contiene el nuevo nombre del usuario.
    - surname: contiene los nuevos apellidos del usuario.
    - birthday: contiene la nueva fecha de nacimiento del usuario.
  - **POST /deleteAccount** → Llama al método ‘deleteAccount’ del controlador ‘user’. Elimina la cuenta del usuario cuya información se ha pasado por el cuerpo de la solicitud.
    Los datos clave-valor a enviar son:
    - user: contiene el usuario.
    - password: contiene la contraseña del usuario.
  - **GET /password-reset/get-code** → Llama al método ‘passwordResetCode’ del controlador ‘user’. Envía un correo para restablecer la contraseña al email cuya información se ha pasado por la solicitud de consulta.
    Los datos clave-valor a enviar son:
    - email: contiene el email del usuario.
  - **PUT /password-reset/change-password** → Llama al método ‘passwordReset’ del controlador ‘user’. Cambia la contraseña de la cuenta del usuario cuya información se ha pasado por el cuerpo de la solicitud.

Los datos clave-valor a enviar son:
  - email: contiene el email del usuario.
  - password: contiene la contraseña del usuario.
  - password2: contiene la contraseña del usuario.
  - code: contiene el código para cambiar la contraseña recibido por email.
- **POST /createBug** → Llama al método 'createBug' del controlador 'bug'. Crea un nuevo informe de error con la información pasada por el cuerpo de la solicitud.
  Los datos clave-valor a enviar son:
  - concept: contiene el concepto del fallo.
  - category: contiene la categoría del fallo.
  - description: contiene la descripción del fallo.
- **POST /uploadImageBug/** → Llama al método 'uploadBugImage' del controlador 'bug'. Sube una imagen del bug al servidor y devuelve su nombre de fichero.

- Rutas de visitante:
  - **POST /login** → Llama al método 'login' del controlador 'user'. Identifica al usuario en la aplicación, devolviéndole un *token* válido por cinco días, con la información pasada por el cuerpo de la solicitud.
    Los datos clave-valor a enviar son:
    - user: contiene el usuario.
    - password: contiene la contraseña del usuario.
  - **POST /register** → Llama al método 'register' del controlador 'user'. Crea un nuevo usuario en la aplicación con la información pasada por el cuerpo de la solicitud.
    Los datos clave-valor a enviar son:
    - user: contiene el usuario.
    - name: contiene el nombre del usuario.
    - surnames: contiene los apellidos del usuario.
    - birthday: contiene la fecha de nacimiento del usuario.
    - email: contiene el email del usuario.
    - password: contiene la contraseña del usuario.
    - password2: contiene la contraseña del usuario.
    - terms: valor booleano que indica la aceptación de los términos y condiciones del servicio.
    - cookies: valor booleano que indica la aceptación de la política de privacidad.
  - **GET /verification/get-activation-email** → Llama al método 'getActivationMail' del controlador 'user'. Envía un nuevo email de activación de la cuenta al email del usuario cuya información se ha pasado por la solicitud de consulta.
    Los datos clave-valor a enviar son:
    - user: contiene el usuario.
  - **GET /verification/verify-account/:id/:secretCode** → Llama al método 'verifyAccount' del controlador 'user'. Activa la cuenta cuya id se ha pasado por parámetros con el código que se ha pasado por parámetros.

Fichero principal

Por último, tenemos el fichero principal de la aplicación 'app.js'. Que haciendo uso de todo lo descrito anteriormente, pone en marcha el proceso de la aplicación en el entorno correspondiente. La aplicación carga las variables de entorno, crea la aplicación de Express, utiliza las claves del dominio de Vinland para configurar HTTPS, carga la configuración de la base de datos, los modelos, la estrategia de autentificación, las rutas, el intercambio de recursos de origen cruzado y pone el servidor a la escucha en HTTP y HTTPS.

5.2.4 Frontend

Librerías

Para el desarrollo, se han utilizado diferentes librerías en el proyecto que han ayudado a que el proyecto implementase soluciones ya desarrolladas y probadas. Para la instalación de estas librerías en el proyecto se ha utilizado NPM (*Node Package Manager*). Se muestra a continuación las librerías añadidas, que no venían por defecto, y su función dentro del proyecto:

- **@auth0/angular-jwt:** Esta librería proporciona un interceptor de HTTP para añadir un JWT a las peticiones por HTTP del cliente.
- **@ng-bootstrap/ng-bootstrap:** Integración del *framework* Bootstrap.
- **bootstrap:** Necesario para la integración del *framework* Bootstrap.
- **rxjs:** Librería de que simplifica la composición de código asíncrono basado en eventos con el uso de secuencias observables. [26]

Recursos

La carpeta 'assets' contiene las imágenes, fuentes, *plugins* y ficheros Javascript adicionales que se necesitan en el proyecto. En la carpeta 'js' por ejemplo, podemos encontrar el fichero 'app.js' que permite mostrar el menú adaptado a dispositivos móviles.

Componentes

En la carpeta 'components' encontramos los diferentes componentes de nuestro sistema. Cada uno de ellos se encuentra en una carpeta separada, y contiene el fichero *.ts que corresponde al componente, el fichero de hoja de estilos *.css y la plantilla con la estructura *.html. Cada componente utiliza la hoja de estilos generales 'styles.css' que se complementa a veces con la propia del componente. Cualquier error producido, se muestra en el componente correspondiente. Se procede a comentar brevemente el funcionamiento de cada uno de los componentes:

- **account:** Este componente gestiona la cuenta del usuario, utilizando los servicios de 'user' y 'auth'. Para ello, muestra tres bloques bien diferenciados. El primer bloque muestra la información de la cuenta y permite modificar los datos del nombre, apellidos y fecha de nacimiento. El segundo bloque permite cambiar la contraseña, además, desde el mismo bloque se puede solicitar el código para poder cambiarla. Y el tercer y último bloque, permite borrar la cuenta.
- **bugs:** Este componente muestra en formato tabla todos los informes de errores, mostrando los campos más relevantes. Tiene opción de búsqueda

y de selección del estado del informe. Utiliza el servicio ‘bug’ para obtener los datos y la pipe ‘search’ para filtrar los datos. Se puede acceder a la información de un informe en concreto a través de la fila.

- **edit-bug:** Este componente muestra en detalle toda la información de un informe de error en un primer bloque, y la información del usuario que ha realizado el informe en un segundo bloque. Se hace uso de los servicios de ‘bug’ e ‘image’, este último para poder acceder a una imagen con permisos. El componente permite cambiar el estado del informe de errores y ver el usuario que ha creado el informe con más detalle.
- **edit-user:** Este componente muestra en detalle toda la información de un usuario en un primer bloque, y la información relacionada con su estado en un segundo bloque, además de permitir bloquear al usuario. Existe un tercer bloque que muestra los permisos y la posibilidad de cambiarlos, además de un cuarto bloque en previsión de la siguiente fase. El componente hace uso de los servicios de ‘user’.
- **login:** El componente de ‘login’ muestra el formulario de acceso a la aplicación a través del usuario y contraseña. Utiliza el servicio ‘user’ y ‘auth’. Y proporciona también la opción de reenviar el código de activación, si un usuario sin validar intenta acceder, le ofrece la opción de reenviarle el correo de validación.
- **main:** El componente de ‘main’, es una página provisional que únicamente muestra un mensaje de bienvenida en base a los permisos del usuario que ha accedido a la aplicación. Sirve únicamente como prueba de funcionamiento. Utiliza únicamente el servicio ‘auth’.
- **menu:** El componente de menú, muestra los diferentes menús y submenús de la aplicación. Muchos de ellos aún no están implementados, pero se muestran para ver el acabado visual que ofrecerán. El menú se muestra normalmente acompañado de otro componente en la parte central. Utiliza el servicio ‘user’ y ‘auth’.
- **privacy:** Este componente únicamente muestra los términos y condiciones del servicio. No utiliza ningún servicio.
- **register:** El componente de ‘register’ proporciona el formulario para que un usuario se registre en el sistema, utiliza el servicio ‘user’.
- **report-bug:** Este componente proporciona el formulario para que un usuario envíe un informe de fallo. Utiliza el servicio ‘bug’ y ‘upload’.
- **reset-password:** Este componente permite obtener un código de cambio de contraseña en el correo, y posteriormente, con ese código, cambiar la contraseña mediante un formulario. Utiliza el servicio ‘user’.
- **terms:** Este componente únicamente muestra la política de privacidad. No utiliza ningún servicio.
- **users:** Este componente muestra en formato tabla todos los usuarios, mostrando los campos más relevantes. Tiene opción de búsqueda y de selección del estado del informe. Utiliza el servicio ‘user’ para obtener los datos y la pipe ‘search’ para filtrar los datos. Se puede acceder a la información de un usuario en concreto a través de la fila.
- **validate-mail:** Este componente (el cual se accederá a través de un email recibido normalmente), valida la cuenta del usuario haciendo uso del id y el código proporcionado como parámetros en la ruta. Utiliza únicamente el servicio ‘user’.

Modelos

En cuanto a los modelos, los podemos encontrar en la carpeta 'models'. Aquí tenemos dos ficheros: 'bug.js' y 'user.js' que se han definido acorde al diagrama de clases que hemos diseñado.

Servicios

Por otra parte, tenemos los servicios en la carpeta 'services', estos servicios implementan las rutas de acceso a la API, obtienen los datos que necesitamos y operan con ellos. Estos servicios son consumidos por los componentes.
Se exponen y explican a continuación cada una de los servicios:

- **auth:** Este servicio proporciona los métodos necesarios para la autentificación. Lo complementa el fichero 'auth-guard.ts' de la misma carpeta, que implementa un método utilizado en las rutas para ver si se tiene acceso a la ruta, tanto por permisos del usuario como por estar identificado en el sistema. Otro fichero que complementa la parte la autentificación es el de 'auth-Interceptor.ts', que intercepta todas las peticiones HTTP y añade el *token* en la cabecera de la petición.
- **bug:** Este servicio implementa todos los métodos para acceder a las rutas de la API relacionadas con el informe de errores, que permiten obtener los informes de fallos, crearlos, borrarlos, actualizar su estado, etc.
- **image:** El servicio de 'image' tiene implementado el método getImageBug permite obtener las imágenes de los informes de errores. Este método solamente lo puede utilizar el administrador que es quien puede acceder a esas imágenes.
- **uploads:** implementa el método 'makefilerequest' que permite enviar ficheros al servidor. Pese a que la petición se hace por XML HTTP, se adjunta el *token* en la petición aquí.
- **bug:** Este servicio implementa todos los métodos para acceder a las rutas de la API relacionadas con los usuarios, que permiten el acceso, registro, obtener los usuarios, establecer permisos, etc.

Aparte, en la carpeta de los servicios, podemos encontrar el fichero 'Global.ts' que se utiliza para poder cambiar la URL de la API fácilmente si hiciese falta.

Pipes

Tenemos también una carpeta llamada 'pipes' con dos *pipes* que nos permiten transformar la información que tenemos. Pasamos a detallar el funcionamiento de estas *pipes*:

- **age:** La *pipe* de edad, recibe una fecha en formato Date y devuelve la edad en años en un formato String. Está pensado para calcular fácilmente la edad de un usuario con su fecha de nacimiento.
- **search:** Este pipe se ha diseñado para filtrar objetos en las búsquedas de forma genérica. Se genera un objeto 'searching', que contiene el valor a buscar, la cantidad de objetos encontrados (para luego poder paginar), un *array* con los atributos a omitir en la búsqueda y un último *array* con los atributos que hay que profundizar en la búsqueda (es decir, si el atributo indicado es un objeto, se buscará también en esos atributos). También se

le puede añadir un atributo que verifica si uno de los atributos está en *true*, esta parte se usa para filtrar por seleccionados.

Rutas

El fichero de rutas, llamado 'app-routing.module.ts' se ha creado siguiendo el mapa del sitio y otorgando los permisos que corresponden a cada una de las rutas. Las rutas quedan de la siguiente manera:

- login → Página de acceso.
- register → Página de registro.
- validate-mail/:id/:code → Página de validación del usuario, cuya dirección se manda en el email de validación. Valida el usuario con los parámetros enviados en la ruta.
- reset-password → Página que permite establecer una nueva contraseña en caso de olvidarse.
- terms → Página que muestra los términos y condiciones del servicio.
- privacy → Página que muestra la política de privacidad.
- menu → Muestra el componente de menú con los componentes hijos siguientes:
    - main → Página principal de bienvenida.
    - account → Página para ver y gestionar la cuenta. Accesible únicamente por usuarios registrados.
    - users → Lista de los usuarios registrados. Accesible únicamente por el administrador
    - edit-user/:id → Página para ver y poder editar el usuario pasado por los parámetros de la ruta. Accesible únicamente por el administrador.
    - bug → Lista de los informes de fallos. Accesible únicamente por el administrador
    - edit-bug/:id → Página para ver y poder editar el informe de fallos pasado por los parámetros de la ruta. Accesible únicamente por el administrador.
    - report-bug → Página que permite crear un informe de errores. Accesible únicamente por usuarios registrados.

Apartado visual

Sobre la parte visual, se parte de una plantilla base adquirida [27] para el desarrollo del HTML y CSS, con tal de poder cumplir con la planificación del proyecto. Partiendo de esta plantilla base, se edita el fichero SASS y posteriormente se genera el fichero de hoja de estilos principal 'styles.css' con el que se trabajará.

Se edita siguiendo la siguiente paleta de colores:

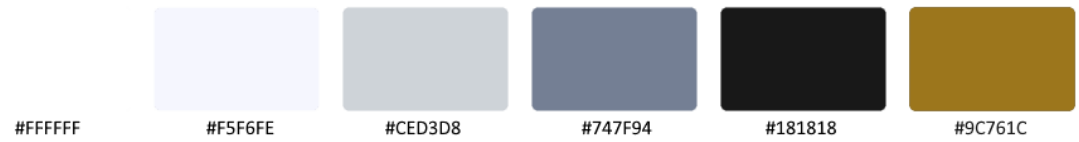

Figura 17: Paleta de colores de la aplicación

En cuanto a la tipografía, se mantiene intacta la de la plantilla, que es 'system-ui' una fuente que se muestra con una estética parecida a la del sistema operativo del usuario. Tiene como alternativas: "Segoe UI", "Roboto", "Helvetica Neue", Arial, "Noto Sans", sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI Symbol" y "Noto Color Emoji". Los diferentes encabezados y tamaños de los textos, se encuentran definidos en el fichero 'styles.css'.

Podemos ver los contrastes entre colores y textos de algunos de los elementos principales:

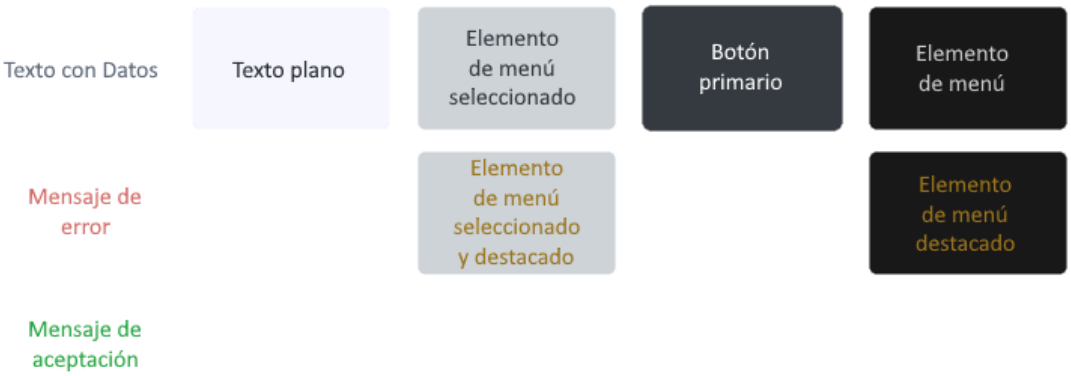

Figura 18: Colores de la aplicación

El logo de la aplicación se ha diseñado con gráficos vectoriales escalables (svg). Para el logo, se ha optado por diseñarlo como un escualo heráldico, dado que la ambientación de las historias que se van a relatar se basan en una época ficticia medieval. La heráldica de la forma del escudo recuerda a los escudos suizos, con esmalte azul y banda blanca (colores fríos que recuerdan al invierno), figura de sierpe marina y el nombre Vinland en la divisa o lema. [28]

Figura 19: Logo

Vistas

Las vistas desarrolladas en esta fase, y las siguientes, se pueden ver en el 'Anexo II: Vistas'.

Base de datos

El acceso a los datos desde la aplicación siempre se hace a través del Backend, utilizando Moongoose. Mongoose, crea los datos en la BBDD basándose en los esquemas que hemos definido. Por tanto, la estructura de los datos sigue el modelo que hemos definido en el diagrama de clases.

Despliegue

Para el despliegue, se reinicia el servicio de NodeJS con el gestor de servicios PM2. En el apartado visual, se genera compilan los ficheros y se suben al servidor web.

5.3 Fase 3: Personajes y habilidades de trasfondo

5.3.1 Backend

Librerías

Nuevamente, añadimos librerías en nuestro proyecto para implementar soluciones ya desarrolladas y probadas, instalándolas a través de NPM (*Node Package Manager*). Se muestra a continuación las librerías añadidas en esta fase:

- **image-size:** Módulo que permite obtener los valores de las dimensiones (ancho x alto) de una imagen subida. Se utiliza en las subidas de imágenes de los personajes, para determinar si tienen las dimensiones requeridas.

Controladores

En esta fase, se añaden los siguientes controladores con sus respectivos métodos. Si algún controlador anterior ha añadido algún método nuevo, se muestra aquí también:

Character, este controlador se encarga de gestionar los personajes. Sus métodos son los siguientes:

- **createCharacter:** Permite crear un nuevo personaje, que es asignado a su creador. Se comprueba que el nombre del personaje no esté en uso, y que, si su creador no es un usuario con permisos de máster, no tenga ya un personaje asignado.
- **updateCharacter:** Permite editar los valores y campos de un personaje, que pertenezca al usuario, y que no se encuentre finalizado o validado todavía. El método, comprueba que todos los valores introducidos sean correctos y estén dentro de sus límites. También, da la opción de marcar el personaje como finalizado.
- **levelUpCharacter:** Utilizando los puntos de habilidad que ha adquirido un personaje al subir de nivel, este método permite añadirlos a su personaje.

Se realizan comprobaciones de la autoría del personaje y de modificaciones erróneas de habilidades.

- **getUserCharacter:** Proporciona la información del personaje asignado al usuario que ha realizado la solicitud.
- **getUserCharacters:** Proporciona la información de los personajes pertenecientes al usuario que ha realizado la solicitud.
- **getUserCharactersById:** Proporciona la información de los personajes pertenecientes al usuario cuya id se ha suministrado.
- **getUserCharactersBasicInfo:** Proporciona la información básica de los personajes pertenecientes al usuario cuya id se ha suministrado.
- **getCharacters:** Proporciona la información básica de todos los personajes.
- **myCharacterById:** Proporciona la información del personaje cuya id se ha suministrado. En este método se comprueba la autoría del personaje respecto al usuario que realiza la petición y se devuelve la información privada del personaje.
- **setActiveCharacter:** Establece como personaje activo, o asignado, al personaje cuya Id se ha pasado por parámetros, siempre y cuando pertenezca al usuario que realiza la petición.
- **setExp:** Permite asignar experiencia a un personaje, con la correspondiente subida de nivel si ha alcanzado la experiencia suficiente.
- **getCharacterToValidate:** Obtiene el próximo personaje a validar, de la cola de personajes finalizados que aún no están validados.
- **validateCharacter:** Valida el personaje pendiente de validar si todo está correcto, asignándole un dinero inicial. O, por el contrario, queda sin validar, se le adjuntan las correcciones a realizar y se habilita de nuevo su edición.
- **uploadCharacterAvatar:** Sube una imagen al servidor (carpeta ‘uploads/characters) y devuelve su nombre de fichero. Se hace una comprobación previa de formato de imagen, tamaño y dimensiones adecuadas.
- **uploadCharacterImage:** Sube una imagen al servidor (carpeta ‘uploads/characters) y devuelve su nombre de fichero. Se hace una comprobación previa de formato de imagen, tamaño y dimensiones adecuadas.
- **updateCharacterImages:** Permite la actualización de las imágenes del personaje con los nombres de los ficheros.

Dice, este controlador es una utilidad para proporcionar números aleatorios a la hora de simular tiradas de dados. Sus métodos son los siguientes:

- **getNumber:** Obtiene un número aleatorio entre un valor máximo y mínimo pasado por parámetros.
- **getDecimalNumber:** Obtiene un número aleatorio, con dos decimales, entre un valor máximo y mínimo pasado por parámetros.
- **d10:** Obtiene un número aleatorio, como resultado de una tirada de un dado de diez caras (1-10).

Habilities, este controlador se encarga de gestionar todo lo relacionado con las habilidades de los personajes. Sus métodos son los siguientes:

- **updateKnowledge:** Permite añadir, actualizar o borrar los relatos que pertenecen a una habilidad de conocimiento.
- **getKnowledge:** Devuelve los relatos del conocimiento pasado por parámetros. El número de relatos devueltos es proporcional al nivel de habilidad del personaje activo perteneciente al usuario que realiza la petición.
- **getKnowledges:** Devuelve todos los relatos del conocimiento pasado por parámetros.

Image, este controlador se encarga servir las imágenes que se han subido a la carpeta 'uploads', o a alguna de sus subcarpetas. Sus métodos son los siguientes:

- **getImgCharacter:** Permite obtener un fichero de imagen ubicado en la carpeta 'uploads/characters'.

Page, este controlador se encarga de gestionar la información de la página principal. Sus métodos son los siguientes:

- **getMainPage:** Proporciona la información de la página principal junto con su formato.
- **updatePage:** Actualiza la información de la página principal junto con su formato.

Modelos

En cuanto a los modelos, en esta fase añadimos tres ficheros: 'character.js', 'knowledge.js' y 'page.js' que se han definido siguiendo el diagrama de clases diseñado.

Rutas

En las rutas de acceso a la API se han añadido las siguientes:

- Rutas de Administrador:
  - **GET /getUserCharactersById/:userId** → Llama al método 'getUserCharactersById' del controlador 'character'. Recibe todos los personajes del usuario cuya id se ha pasado por parámetros.
  - **PUT /updatePage** → Llama al método 'updatePage' del controlador 'page'. Recibe el usuario cuya id se ha pasado por parámetros.
    Los datos clave-valor a enviar son:
    - body: contiene el modelo de la página con el nuevo contenido y formato.

- Rutas de Máster:
  - **GET /getCharactersToValidate** → Llama al método 'getCharacterToValidate' del controlador 'character'. Recibe el próximo personaje a validar.

- **PUT /validateCharacter** → Llama al método 'validateCharacter' del controlador 'character'. Valida el personaje pendiente de validar si todo está correcto, asignándole un dinero inicial. O, por el contrario, queda sin validar, se le adjuntan las correcciones a realizar y se habilita de nuevo su edición.
  Los datos clave-valor a enviar son:
  - id: contiene el id del personaje a validar.
  - money: contiene la cantidad de dinero inicial asignada.
  - result: resultado de la validación (true/false).
  - comments: comentarios de la validación en caso de ser negativa.
- **PUT /setExp** → Llama al método 'setExp' del controlador 'character'. Asigna experiencia a un usuario.
  - id: contiene el id del personaje que recibirá los puntos de experiencia.
  - exp: cantidad de puntos de experiencia.
- **PUT /updateKnowledge** → Llama al método 'updateKnowledge' del controlador 'habilities'. Actualiza la habilidad de conocimiento con los nuevos datos.
  Los datos clave-valor a enviar son:
  - body: contiene el modelo de la habilidad de conocimientos, con las historias actualizadas.
- **GET /knowledges/:name** → Llama al método 'getKnowledges' del controlador 'habilities'. Recibe la habilidad de conocimiento que se ha pasado por parámetros, junto con las historias.
  Los datos clave-valor a enviar son:
  - name: contiene el nombre de la habilidad de conocimiento.
- **PUT /setActiveCharacter** → Llama al método 'setActiveCharacter' del controlador 'character'. Establece como personaje activo uno de los personajes del usuario.
  Los datos clave-valor a enviar son:
  - id: contiene el id del personaje que se establecerá como activo.

- Rutas de Usuario:
  - **GET /character** → Llama al método 'getUserCharacter' del controlador 'character'. Proporciona la información del personaje asignado al usuario que ha realizado la solicitud.
  - **GET /myCharacter/:id** → Llama al método 'myCharacterById' del controlador 'character'. Proporciona la información, incluida la privada, del personaje cuya id se ha suministrado por parámetros.
    Los datos clave-valor a enviar son:
    - id: contiene el id del personaje.
  - **GET /userCharacters** → Llama al método 'getUserCharacters' del controlador 'character'. Proporciona la información de los personajes pertenecientes al usuario que ha realizado la solicitud.
  - **GET /userCharactersBasicInfo** → Llama al método 'getUserCharactersBasicInfo' del controlador 'character'. Proporciona la información básica de los personajes pertenecientes al usuario que ha realizado la solicitud.

- **POST /createCharacter** → Llama al método ‘createCharacter’ del controlador ‘character’. Crea un nuevo personaje para el usuario, este personaje necesitará posteriormente de su edición.
  Los datos clave-valor a enviar son:
  - name: contiene el nombre del nuevo personaje.
- **PUT /updateCharacter** → Llama al método ‘updateCharacter’ del controlador ‘character’. Permite editar un personaje que aún no esté finalizado, y finalizarlo si se desea.
  Los datos clave-valor a enviar son:
  - body: contiene el modelo del personaje con todos sus datos.
- **PUT /levelUpCharacter** → Llama al método ‘levelUpCharacter’ del controlador ‘character’. Mejora las habilidades de un personaje, gastando los puntos de habilidad que posee.
  Los datos clave-valor a enviar son:
  - body: contiene el modelo de las habilidades del personaje con los valores nuevos.
- **POST /uploadAvatarCharacter** → Llama al método ‘uploadCharacterAvatar’ del controlador ‘character’. Sube la imagen de avatar del personaje y devuelve su nombre.
- **POST /uploadImageCharacter** → Llama al método ‘uploadCharacterImage’ del controlador ‘character’. Sube la imagen general del personaje y devuelve su nombre.
- **PUT / updateCharacterImages** → Llama al método ‘updateCharacterImages’ del controlador ‘character’. Actualiza el personaje con los nombres de los ficheros de imagen.
- **GET /knowledge/:name** → Llama al método ‘getKnowledge’ del controlador ‘habilities’. Recibe las historias de la habilidad de conocimiento que se ha pasado por parámetros, el número de historias será proporcional al nivel de la habilidad.
  Los datos clave-valor a enviar son:
  - name: contiene el nombre de la habilidad de conocimiento.

- Rutas de visitante:
  - **GET /getCharacter/:id** → Llama al método ‘getCharacterById’ del controlador ‘character’. Proporciona la información del personaje cuya id se ha suministrado por parámetros.
    Los datos clave-valor a enviar son:
    - id: contiene el id del personaje.
  - **GET /getCharacters** → Llama al método ‘getCharacters’ del controlador ‘character’. Proporciona la información básica de todos los personajes.
  - **GET /getImgCharacter/:image** → Llama al método ‘getCharacterBug’ del controlador ‘image’. Devuelve el fichero de imagen de la imagen solicitada por parámetros.
    Los datos clave-valor a enviar son:
    - image: nombre de la imagen.
  - **GET /getMainPage** → Llama al método ‘getMainPage’ del controlador ‘page’. Devuelve la información y formato de la página principal.

5.3.2 Frontend

Librerías

A continuación, se muestran las librerías añadidas en esta fase:

- **@angular/cdk:** El kit de desarrollo de componentes (CDK) es un conjunto de utilidades de comportamiento para crear componentes de interfaz de usuario.
- **@types/quill, ngx-quill y quill:** Editor de texto con formato.
- **@types/chart.js, apexcharts y ng-apexcharts:** Librerías que facilitan la creación de tablas y otros gráficos en Angular.
- **jquery:** Permite la manipulación del documento HTML.
- **ngx-image-cropper:** Permite el recorte de una imagen a unas dimensiones y píxeles específicos.

Componentes

Se procede a comentar brevemente el funcionamiento de cada uno de los componentes añadidos en esta fase:

- **characters:** Este componente muestra en tarjetas todos los personajes creados, mostrando los campos más relevantes. Tiene opción de búsqueda y selección de raza. Utiliza el servicio 'character' para obtener los datos y la pipe 'search-character' para filtrar los datos. Se puede acceder a visualizar un personaje presionando la tarjeta.
- **create-character:** Este componente permite comenzar la creación de un personaje. Muestra un campo para la introducción del nombre y botones para su creación, así como su posterior edición. Utiliza el servicio 'character' para la creación del personaje.
- **edit-character:** Este componente incorpora multitud de comprobaciones en el proceso de edición del personaje, para que se finalice sin errores. Aparecen varias tarjetas que agrupan los distintos datos a rellenar: información, atributos, habilidades e imágenes. Se hace uso de los servicios de 'character' y 'upload', este último para poder subir imágenes. El componente permite editar el personaje, guardarlo o dejarlo listo para la validación.
- **edit-knowledge:** Este componente muestra todas las historias creadas de la habilidad de conocimiento seleccionada. Las historias, que aparecen en tarjetas, se pueden ordenar arrastrando y soltando. Mediante el selector desplegable, podemos seleccionar otra habilidad de conocimiento. Utilizando el botón 'nuevo', se puede crear una historia nueva. El componente hace uso de los servicios de 'hability' y diversas librerías como 'apexcharts'.
- **editor-knowledge:** Este componente muestra el editor de las habilidades de conocimiento, permitiendo editar o crear una historia. En el componente se puede ver el editor y una previsualización del resultado. Utiliza el servicio 'hability' y 'auth', así como la librería Quill.
- **editor-main:** Este componente muestra el editor de la página principal, permitiendo modificar su contenido y formato. En el componente se puede ver el editor y una previsualización del resultado.

- **editor-images:** Este componente muestra el editor de imágenes, permitiendo subir una imagen y recortarla a unas dimensiones y píxeles concretos. En el componente se puede ver el editor y una previsualización del resultado.
- **knowledges:** Este componente muestra los diferentes relatos, asociados a la habilidad de conocimiento consultada, a los que tiene acceso el personaje. Se muestra una columna con tarjetas que representan los relatos y, al presionar una de ellas, se abre el relato para su lectura. Utiliza el servicio 'hability'.
- **lvl-up-character:** Este componente permite realizar la subida de nivel de un personaje. Se muestra una tarjeta con los puntos de habilidad restantes y el listado de habilidades. Utiliza el servicio 'character'.
- **select-character:** Este componente muestra en tarjetas todos los personajes del usuario Máster, mostrando los campos más relevantes. Tiene opción de visualizarlos en detalle presionando sobre su tarjeta o ponerlo como personaje activo mediante el botón de la misma. Utiliza el servicio 'character' para obtener los datos y asignar el personaje.
- **validate-character:** Este componente muestra un pequeño panel de información en la parte superior, indicando el número de personajes por validar. En la parte del medio, se puede ver el personaje a validar utilizando el componente 'view-character' y, en la parte inferior, dos tarjetas que conforman el panel de validación. En estas tarjetas, se le asignan los fondos iniciales y se valida, o no, el personaje (indicando los comentarios a la revisión en caso negativo). Utiliza el servicio 'character'.
- **view-character:** Este componente muestra todos los detalles de un personaje distribuyendo la información en distintas pestañas. Utiliza el servicio 'character'.

Modelos

En cuanto a los modelos, añadimos tres ficheros nuevos: 'character.js', 'page.js' y 'knowledge.js' que se han definido acorde al diagrama de clases que hemos diseñado.

Servicios

Los nuevos servicios que implementan las rutas de acceso a la API en esta fase, son los siguientes:

- **character:** Este servicio implementa todos los métodos para acceder a las rutas de la API relacionadas con los personajes, que permiten obtener sus datos, crearlos, editarlos, subir de nivel, etc.
- **hability:** Este servicio implementa todos los métodos para acceder a las rutas de la API relacionadas con las habilidades, que permiten obtener los relatos de las habilidades de conocimientos, crearlos, editarlos, borrarlos, etc.
- **page:** Este servicio implementa todos los métodos para acceder a las rutas de la API relacionadas con la página principal, que permiten obtener datos de la misma o editarlos.
- **image:** El servicio de 'image' se actualiza con el método 'getImageCharacter', que permite obtener las imágenes de los personajes.

Pipes

Tenemos también una carpeta llamada 'pipes' con una nueva *pipe* que nos permite transformar la información que tenemos. Pasamos a detallar el funcionamiento de esta *pipe*:

- **search-character:** Este *pipe* se ha diseñado para filtrar personajes en las búsquedas. Se genera un objeto 'searching', que contiene el valor a buscar, la cantidad de personajes encontrados (para luego poder paginar) y un *array* con los atributos a omitir en la búsqueda. También se le añade un atributo que verifica si uno de los atributos está en true, esta parte se usa para filtrar por razas.

Rutas

Las rutas añadidas en esta fase, quedan de la siguiente manera:

- menu → Muestra el componente de menú con los componentes hijos siguientes:
  - create-character → Página para la creación de personajes. Accesible únicamente por usuarios registrados.
  - edit-character → Página para la edición del personaje. Accesible únicamente por usuarios registrados.
  - lvlup-character/:id → Página para realizar la subida de nivel de un personaje. Accesible únicamente por usuarios registrados.
  - select-character → Página para seleccionar uno de los personajes del usuario y ponerlo como personaje activo. Accesible únicamente por másters.
  - validate-character → Página que permite la validación de personajes que han finalizado su edición. Accesible únicamente por Másters
  - view-character/:id → Página para ver la información de un personaje.
  - characters → Página que permite ver todos los personajes.
  - help → Página que muestra la información de ayuda para aprender a jugar.
  - editor-main → Página que contiene el editor de la página principal. Accesible únicamente por el administrador.
  - editor-images/:type/:id → Página que contiene el editor de imágenes. Accesible únicamente por usuarios registrados.
  - edit-knowledge → Página para la gestión de las habilidades de conocimiento. Accesible únicamente por Másters.
  - knowledges/:name → Página que muestra la información, en relatos, de las habilidades de conocimiento. Accesible únicamente por usuarios registrados.
  - editor-knowledge/:name/:pos → Página que contiene el editor de las habilidades de conocimientos, permitiendo editar un relato ya creado. Accesible únicamente por Másters.
  - editor-knowledge/:name → Página que contiene el editor de las habilidades de conocimientos, permitiendo crear un relato. Accesible únicamente por Másters.

Vistas

Las vistas desarrolladas en esta fase, y las siguientes, se pueden ver en el 'Anexo II: Vistas'.

Despliegue

Para el despliegue de la fase, se reinicia el servicio de NodeJS con el gestor de servicios PM2. En el apartado visual, se genera compilan los ficheros y se suben al servidor web.

5.4 Fase 4: Eventos y habilidades de escritura

5.4.1 Backend

Librerías

En esta última fase no se requieren librerías nuevas en el proyecto.

Controladores

En esta fase, se añaden los siguientes controladores con sus respectivos métodos. Si algún controlador anterior ha añadido o modificado algún método, se muestra aquí también:

Mission, este controlador se encarga de gestionar las misiones o eventos. Sus métodos son los siguientes:

- **createMission:** Permite crear una nueva misión, que será creada con el estado de 'En preparación'. Se comprueba que los parámetros recibidos sean correctos.
- **updateMission:** Permite editar los valores y campos de una misión, que pertenezca al máster que realiza la solicitud, y que no se encuentre como finalizada o empezada. El método, comprueba que todos los valores introducidos sean correctos, que estén el número de personajes requeridos para la misión y que ninguno de ellos esté en otra misión. También, da la opción de marcar la misión con el estado 'Empezada'.
- **finishMission:** Este método marca la misión con el estado 'Finalizado' y otorga las recompensas a partes iguales entre sus participantes.
- **eraseMision:** Permite eliminar una misión que pertenezca al máster que realiza la solicitud, y que no se encuentre como finalizada o empezada.
- **getMissionById:** Proporciona la información de la misión cuya id se ha suministrado.
- **getMissions:** Proporciona la información básica de todas las misiones.
- **getMyMissions:** Proporciona la información básica de todas las misiones en las que el usuario que realiza la solicitud está, o ha estado, como participante.
- **getMessagesToValidate:** Obtiene el próximo mensaje a validar, de entre todos los mensajes finalizados que aún no están validados y que pertenezcan al máster que realiza la solicitud.
- **validateMessage:** Valida el mensaje pendiente de validar si todo está correcto, el cual quedará publicado en la misión. O, por el contrario, queda

sin validar, se le adjuntan las correcciones a realizar y se habilita de nuevo su edición.
- **takeTurn:** Permite obtener el turno de una misión durante 2 días. Para obtener el turno, el personaje ha de pertenecer a la misión, encontrarse la misión en preparación y no haber otro personaje con el turno o estar revisándose algún mensaje.
- **leaveTurn:** Descarta el turno que ha obtenido el personaje.
- **joinMission:** Permite apuntar al personaje como participante en la misión, siempre y cuando se encuentre todavía en preparación.
- **getWrittingPoints:** Obtiene los puntos de escritura fijos, de misión y de mensaje.
- **updateMessage:** Permite editar el mensaje de una misión, que pertenezca al usuario que realiza la solicitud, y que no se encuentre como finalizado. El método, comprueba que todos los valores introducidos sean correctos y da la opción de marcar el mensaje como finalizado.
- **uploadMissionImage:** Sube una imagen al servidor (carpeta 'uploads/missions) y devuelve su nombre de fichero. Se hace una comprobación previa de formato de imagen, tamaño y dimensiones adecuadas.

Dice, este controlador es una utilidad para proporcionar números aleatorios a la hora de simular tiradas de dados. Sus métodos son los siguientes:

- **d3:** Obtiene un número aleatorio, como resultado de una tirada de un dado de tres caras (1-3).
- **nd6:** Obtiene un número aleatorio, como resultado de 'n' número de dados de seis caras (1-6) y luego aplicar un modificador a la tirada.
- **d10o3:** Realiza una tirada de tres dados de diez caras y obtiene el valor medio por defecto, lo que se conoce como dado objetivo. Tiene un parámetro para poder elegir el dado objetivo. Si la raza del personaje tiene suerte, el dado objetivo puede cambiar el dado objetivo al siguiente superior, con un 33% de probabilidades. Mientras el dado objetivo saque el valor máximo del dado, se repiten las tiradas y se suman los resultados. En el caso de obtener en la primera tirada tres unos, o un uno en el dado objetivo y cinco o menos en el siguiente, se informa de pifia.

Habilities, este controlador se encarga de gestionar todo lo relacionado con las habilidades de los personajes. Sus métodos son los siguientes:

- **getWrittingHabilities:** Obtiene el nombre y el icono de las habilidades de escritura implementadas.
- **getWrittingHability:** Obtiene toda la información para el uso de la habilidad de escritura.
- **useWrittingHability:** Recibe la información de la habilidad de escritura a utilizar, comprueba todos los valores recibidos y procede a su activación. En la activación, se realizan las tiradas de dados, se devuelven los resultados, se generan las habilidades de escritura y se actualizan los datos.

Image, este controlador se encarga servir las imágenes que se han subido a la carpeta ‘uploads’, o a alguna de sus subcarpetas. Sus métodos son los siguientes:

- **getImgMission:** Permite obtener un fichero de imagen ubicado en la carpeta ‘uploads/missions’.

Modelos

En cuanto a los modelos, en esta fase añadimos cinco ficheros: ‘message.js’, ‘mission.js’, ‘turn.js’, ‘hability.js’ y ‘writtingPoint.js’ que se han definido siguiendo el diagrama de clases diseñado.

Rutas

En las rutas de acceso a la API se han añadido las siguientes:

- Rutas de Máster:
  - **POST /createMission** → Llama al método ‘createMission’ del controlador ‘mission’. Crea una nueva misión en estado ‘Prepración’.
    Los datos clave-valor a enviar son:
    - title: contiene el nombre de la misión.
    - subTitle (Opcional): contiene una segunda línea apara el título.
    - description: contiene la descripción inicial de la misión.
    - descriptionShort: contiene la descripción de la misión en un máximo de 250 caracteres.
    - maxCharacters: número de participantes en la misión.
    - experience: cantidad de puntos de experiencia a repartir entre los personajes participantes cuando termine la misión.
    - gold: cantidad de dinero a repartir entre los personajes participantes cuando termine la misión.
  - **PUT /updateMission** → Llama al método ‘updateMission’ del controlador ‘mission’. Actualiza la misión con los nuevos datos.
    Los datos clave-valor a enviar son:
    - body: contiene el modelo la misión, con el contenido actualizado.
  - **DELETE /eraseMission/:id** → Llama al método ‘eraseMission’ del controlador ‘mission’. Elimina la misión.
    Los datos clave-valor a enviar son:
    - id: contiene el id de la misión a eliminar.
  - **PUT /finishMission/:id** → Llama al método ‘finishMission’ del controlador ‘mission’. Finaliza la misión.
    Los datos clave-valor a enviar son:
    - id: contiene el id de la misión a finalizar.
  - **POST /uploadImageMission** → Llama al método ‘uploadMissionImage’ del controlador ‘mission’. Sube la imagen de la tarjeta de la misión y devuelve su nombre.
  - **GET /getMessagesToValidate** → Llama al método ‘getMessagesToValidate’ del controlador ‘mission’. Recibe el próximo mensaje a validar de las misiones del máster que realiza la solicitud.
  - **PUT /validateMessage** → Llama al método ‘validateMessage’ del controlador ‘mission’. Valida el mensaje pendiente de validar si todo

está correcto y pasa a formar parte del contenido de la misión. O, por el contrario, queda sin validar, se le adjuntan las correcciones a realizar y se habilita de nuevo su edición.
Los datos clave-valor a enviar son:
  - id: contiene el id del mensaje a validar.
  - result: resultado de la validación (true/false).
  - comments: comentarios de la validación en caso de ser negativa.

- Rutas de Usuario:
  - **GET /getMyMissions/private** → Llama al método 'getMyMissions' del controlador 'mission'. Proporciona la información básica de las misiones en las que el personaje ha sido asignado como participante.
  - **PUT /missionTakeTurn** → Llama al método 'takeTurn' del controlador 'mission'. Coge turno en la misión cuya id se ha suministrado por parámetros.
    Los datos clave-valor a enviar son:
    - id: contiene el id de la misión.
  - **PUT /missionLeaveTurn** → Llama al método 'leaveTurn' del controlador 'mission'. Cancela el turno en la misión cuya id se ha suministrado por parámetros.
    Los datos clave-valor a enviar son:
    - id: contiene el id de la misión.
  - **PUT /joinMission** → Llama al método 'joinMission' del controlador 'mission'. Apunta al personaje activo como posible participante en la misión cuya id se ha suministrado por parámetros.
    Los datos clave-valor a enviar son:
    - id: contiene el id de la misión.
  - **GET /getWrittingPoints/:id** → Llama al método 'getWrittingPoints' del controlador 'mission'. Recibe los puntos de escritura fijos, de mensaje y de la misión cuya id se ha suministrado por parámetros.
    Los datos clave-valor a enviar son:
    - id: contiene el id de la misión.
  - **GET /updateMessage** → Llama al método 'updateMessage' del controlador 'mission'. Actualiza el mensaje con los nuevos datos.
    Los datos clave-valor a enviar son:
    - body: contiene el modelo del mensaje, con el contenido actualizado.
  - **GET /getWrittingHabilities** → Llama al método 'getWrittingHabilities' del controlador 'habilities'. Proporciona el nombre y el icono de las habilidades de escritura.
  - **GET /getWrittingHability/:name** → Llama al método 'getWrittingHability' del controlador 'habilities'. Proporciona toda la información necesaria de la habilidad de escritura para su posterior uso.
  - **PUT /useWrittingHability** → Llama al método 'useWrittingHability' del controlador 'habilities'. Utiliza la habilidad de escritura pasada por parámetros.
    Los datos clave-valor a enviar son:
    - missionId: contiene el id de la misión.
    - characterId: contiene el id del personaje.
    - messageId: contiene el id del mensaje.
    - hability: contiene la habilidad a utilizar.

    - action: contiene la acción de la habilidad.
    - difficulty: contiene la dificultad seleccionada.
    - versus: contiene la acción de confrontamiento seleccionada.
    - rival: contiene el rival en una acción de confrontamiento.

- Rutas de visitante:
  - **GET /getMission/:id** → Llama al método 'getMissionById' del controlador 'mission'. Proporciona la información de la misión cuya id se ha suministrado por parámetros.
    Los datos clave-valor a enviar son:
    - id: contiene el id de la misión.
  - **GET /getMissions** → Llama al método 'getMissions' del controlador 'mission'. Proporciona la información básica de todas las misiones.
  - **GET /getImgMission/:image** → Llama al método 'getImgMission' del controlador 'image'. Devuelve el fichero de imagen de la imagen solicitada por parámetros.
    Los datos clave-valor a enviar son:
    - image: nombre de la imagen.

5.4.2 Frontend

Librerías

A continuación, se muestra la librería añadida en esta fase:

- **ngx-countdown:** Proporciona un módulo de cuenta atrás configurable.

Componentes

Se procede a comentar brevemente el funcionamiento de cada uno de los componentes añadidos en esta fase:

- **missions:** Este componente muestra en tarjetas todas las misiones creadas, mostrando los campos básicos. Tiene opción de búsqueda y selección del estado. Utiliza el servicio 'mission' para obtener los datos y la pipe 'search-by-attr' para filtrar los datos. Se puede acceder a visualizar una misión presionando la tarjeta.
- **create-mission:** Este componente permite la creación de una misión. Aparecen varias tarjetas que agrupan los distintos datos a rellenar: datos de la misión, recompensas, subida del archivo de imagen y el editor de texto con formato. Se hace uso de los servicios de 'mission' y 'upload', este último para poder subir imágenes. También utiliza la librería Quill.
- **edit-mission:** Este componente permite la modificación de los datos de la misión, seleccionar los personajes, cambiar su estado o borrar la misión. Aparecen varias tarjetas que agrupan los distintos datos a rellenar: datos de la misión, recompensas, subida del archivo de imagen, selección de personajes, el editor de texto con formato y el panel de acciones. Se hace uso de los servicios de 'mission' y 'upload', este último para poder subir imágenes, así como la librería Quill.
- **editor-mission:** Este componente permite coger turno y escribir mensajes en las misiones. En el componente se puede ver la descripción de la misión,

participantes, el mensaje anterior, el panel de habilidades, el botón de coger turno y el editor para escribir el mensaje. Utiliza el servicio 'mission' y 'auth', así como la librería Quill.
- **validate-mission:** Este componente muestra un pequeño panel de información en la parte superior, indicando el número de mensajes por validar. Se puede toda la información relativa a la misión y el mensaje a validar y, en la parte inferior, el panel de validación donde se valida, o no, el mensaje (indicando los comentarios a la revisión en caso negativo). Utiliza el servicio 'mission'.
- **view-mission:** Este componente muestra toda la información y el contenido de una misión distribuyendo la información en tarjetas. Utiliza el servicio 'mission'.
- **panel-hability:** Este componente muestra toda la información para la utilización de una habilidad, y la obtención de su resultado, en una ventana modal. Utiliza el servicio 'hability'.

Modelos

En cuanto a los modelos, añadimos tres ficheros nuevos: 'message.js', 'mission.js' y 'turn.js' que se han definido acorde al diagrama de clases que hemos diseñado.

Servicios

Los nuevos servicios que implementan las rutas de acceso a la API en esta fase, son los siguientes:

- **mission:** Este servicio implementa todos los métodos para acceder a las rutas de la API relacionadas con las misiones, que permiten obtener sus datos, crearlas, editarlas, borrarlas, etc.
- **hability:** Este servicio se actualiza con los métodos 'getWrittingHabilities', 'getWrittingHability' y 'useWrittingHability'. Estos métodos se utilizan para acceder a las rutas de la API relacionadas con las habilidades, permitiendo obtener sus datos y usarlas.
- **image:** El servicio de 'image' se actualiza con el método 'getImageMission', que permite obtener las imágenes de las misiones.

Pipes

Tenemos también una carpeta llamada 'pipes' con dos *pipes* que nos permiten transformar la información que tenemos. Pasamos a detallar el funcionamiento de estas *pipes*:

- **search-by-attr:** Este *pipe* se ha diseñado para filtrar en las búsquedas teniendo un atributo con un valor fijo. Se genera un objeto 'searching', que contiene el valor a buscar, la cantidad de resultados encontrados (para luego poder paginar), un *array* con los atributos a omitir en la búsqueda y un atributo con valor para tenerlo en cuenta en el filtrado.
- **difficulty-name:** Este *pipe* se ha diseñado para obtener un valor de texto (fácil, normal, moderado, difícil, muy difícil, casi imposible) en base al valor numérico de dificultad.

Rutas

Las rutas añadidas en esta fase, quedan de la siguiente manera:

- menu → Muestra el componente de menú con los componentes hijos siguientes:
    - create-mission → Página para la creación de misiones. Accesible únicamente por Másters.
    - edit-mission/:id → Página para la edición o gestión de las misiones. Accesible únicamente por Másters.
    - view-mission/:id → Página para ver la información y contenido de una misión.
    - validate-mission → Página que permite la validación de los mensajes de una misión que han finalizado su edición. Accesible únicamente por Másters.
    - missions/:name → Página que permite ver las misiones, o las misiones propias de un personaje.
    - editor-mission/:id → Página que contiene el editor de contenido de los mensajes de las misiones. Accesible únicamente por usuarios registrados.

Vistas

Las vistas desarrolladas en esta fase, y las siguientes, se pueden ver en el ‘Anexo II: Vistas’.

Despliegue

Para el despliegue de la fase, se reinicia el servicio de NodeJS con el gestor de servicios PM2. En el apartado visual, se genera compilan los ficheros y se suben al servidor web.

6. Pruebas

6.1 Pruebas de servidor y dominio

Las pruebas del servidor y dominio, correspondientes a la fase 1, se realizaban durante la instalación de los diferentes servicios. Al haberlo realizado de esta manera, tenemos realizadas ya las siguientes pruebas:

- Comprobado el acceso local a el servidor web Apache por HTTP
- Comprobado el acceso a través del dominio http://vinland.quest y http://www.vinland.quest a el servidor Apache
- Comprobado el correcto funcionamiento de las DNS dinámicas
- Comprobado el acceso local a el servidor web Apache por HTTPS
- Comprobado el acceso a través del dominio https://vinland.quest y https://www.vinland.quest a el servidor Apache
- Comprobación de la validez de los certificados para HTTPS
- Comprobación del correcto funcionamiento de Certbot para la renovación de los certificados con la entidad certificadora Let's Encrypt
- Comprobación del correcto funcionamiento del servicio FTP y de su acceso desde un PC con un entorno Windows en la misma red local y utilizando el software Filezilla.
- Comprobación del correcto funcionamiento de Samba, mediante el acceso a las carpetas compartidas del Backend desde un entorno Windows.
- Comprobación del correcto funcionamiento de la base de datos MongoDB y de su acceso a través de la aplicación MongoDB Compass en un entorno Windows.
- Comprobación de la correcta gestión del proceso en segundo plano para la API desarrollada con NodeJS utilizando PM2.
- Comprobación de la correcta redirección hacia index.html en la aplicación desarrollada con Angular.

Para muchas de las comprobaciones ha sido necesario hacer una aplicación de prueba del tipo 'hello world' para comprobar su funcionamiento.

6.2 Pruebas unitarias

Durante el desarrollo del resto de fases se han ejecutado continuamente pruebas con Postman para ver que los resultados eran los esperados. Se probaban continuamente las funciones desarrolladas y que todos los posibles fallos estaban cubiertos.

Cada vez que terminaba una fase, se realizaba un paquete de pruebas completo para la API, el cual se ejecuta utilizando Newman. En Postman se generan diferentes pruebas y se definen variables para que las pruebas se ejecuten, mostrando si el resultado recibido es el esperado. Estas pruebas, se pasan a Newman, que las ejecuta y genera un informe.

Los resultados de estas pruebas se pueden consultar en el 'Anexo III: Pruebas unitarias'.

6.3 Pruebas visuales

En cuanto a la parte gráfica, en esta fase se comprueba que la visualización de la página y su adaptación de distintos dispositivos y resoluciones es correcta. De esta manera, vemos que la plantilla se encuentra adaptada correctamente. Se puede ver, en las siguientes páginas, cómo se adapta la aplicación a diferentes dispositivos y resoluciones.

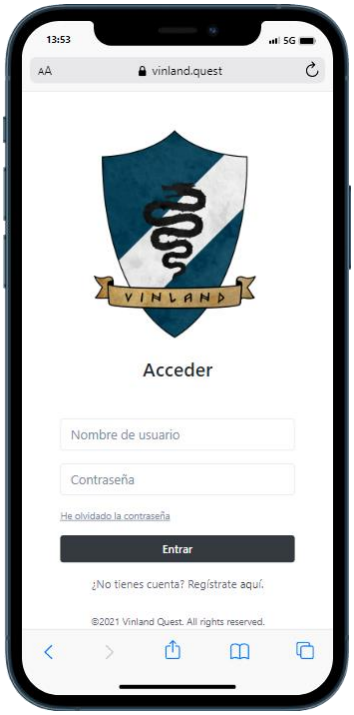

Figura 20: Vista página de acceso en iPhone 12 PRO

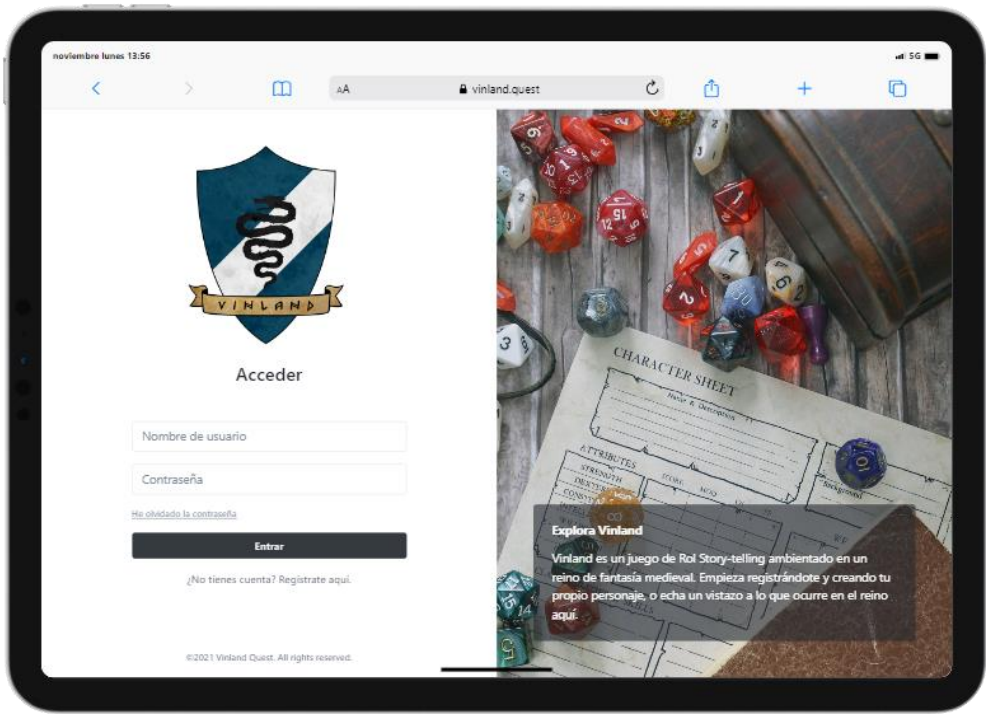

Figura 21: Vista página de acceso en iPad PRO 11

Figura 22: Vista página principal en iPhone 12 PRO

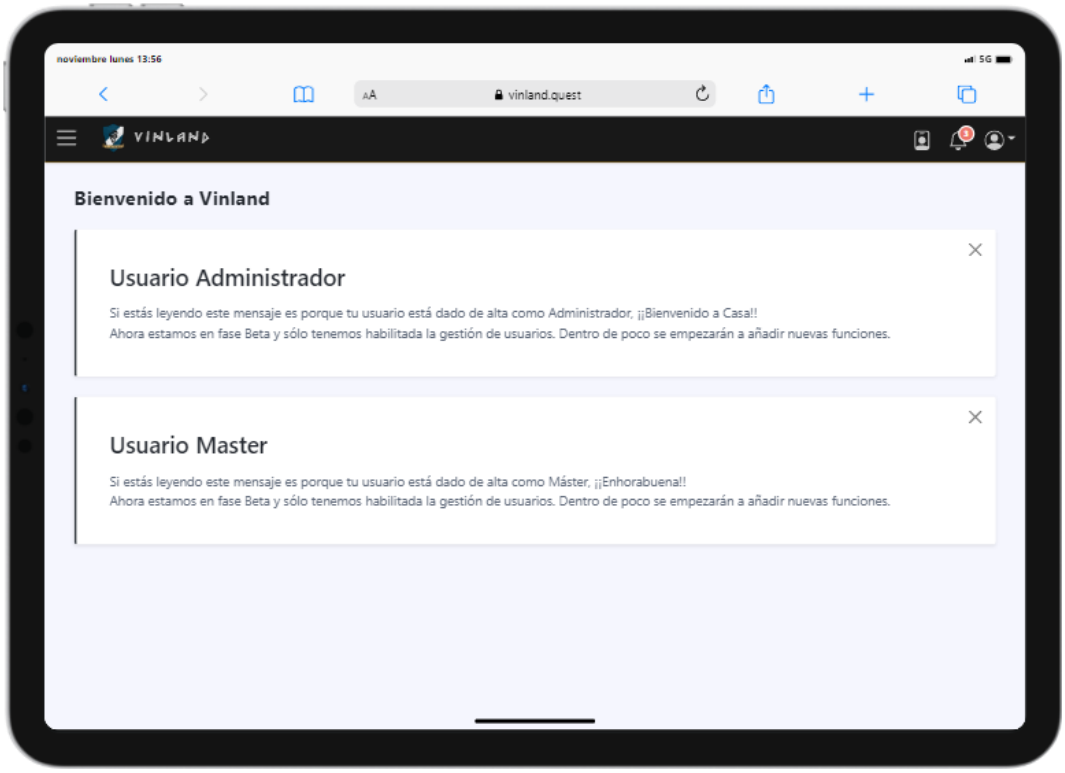

Figura 23: Vista página principal en iPad PRO 11

6.4 Pruebas de integración

Por último, se realizan las pruebas de integración siguiendo las siguientes tablas de pruebas. La columna V (Verificación) indica si la prueba ha sido exitosa. La columna P (Permisos) indica los permisos que ha de tener el usuario: A administrador, M máster, U usuario registrado. Si no se indican permisos, se considera que el usuario es un visitante.

F02-01	**Subsistema: Registro usuario** **Propósito: Registro en el sistema**			
Nº	Entrada	Salida esperada	P	V
01	El usuario se intenta registrar sin rellenar algún campo.	El registro no es efectivo. Se informa del error.		☑
02	El usuario se intenta registrar pero las contraseñas no coinciden.	El registro no es efectivo. Se informa del error.		☑
03	El usuario se intenta registrar pero la contraseña no cumple el patrón.	El registro no es efectivo. Se informa del error.		☑
04	El usuario se intenta registrar pero el email no cumple el patrón.	El registro no es efectivo. Se informa del error.		☑
05	El usuario se intenta registrar pero no ha aceptado los términos del servicio y/o la política de privacidad.	El registro no es efectivo. Se informa del error.		☑
06	El usuario se intenta registrar pero el email y/o el usuario ya existen.	El registro no es efectivo. Se informa del error.		☑
07	El usuario se intenta registrar con los datos correctamente.	El registro es efectivo.		☑
08	Se intenta validar la cuenta con un email y/o id de usuario incorrecto.	La verificación no es llevada a cabo. Se informa del error.		☑
09	Se intenta validar la cuenta con una cuenta ya validada.	La verificación ya existe. Se informa del error.		☑
10	Se valida la cuenta con los datos apropiados	Se lleva a cabo la validación.		☑

Tabla 2: Pruebas integración Fase 2 – Registro usuario

F02-02	**Subsistema: Acceso usuario** **Propósito: Acceso al sistema**			
Nº	Entrada	Salida esperada	P	V
01	Un usuario que no existe, intenta acceder al sistema.	No es posible el acceso. Se informa del error.		☑
02	Un usuario bloqueado intenta acceder al sistema.	No es posible el acceso. Se informa del error.		☑
03	Un usuario con el email por validar intenta acceder al sistema.	No es posible el acceso. Se informa del error.		☑
04	Un usuario intenta acceder al sistema con una contraseña que no es la del usuario.	No es posible el acceso. Se informa del error.		☑
05	El usuario intenta acceder con un usuario y contraseña correctos.	El usuario puede acceder y recibe su *token*.		☑
06	Se solicita un email de validación de un usuario que no existe.	No se envía ningún email de validación. Se informa del error.		☑
07	Un usuario ya validado, solicita un email de validación.	No se envía ningún email de validación. Se informa del error.		☑
08	Un usuario no validado, solicita un email de validación con los datos correctos.	Se envía un email de validación.		☑

Tabla 3: Pruebas integración Fase 2 – Acceso usuario

F02-03	**Subsistema: Gestión de cuentas de usuario Propósito: Gestionar la cuenta de los usuarios**			
Nº	Entrada	Salida esperada	P	V
01	Un usuario solicita un código para restablecer la contraseña de un email que no existe.	No se envía ningún email con el código. Se informa del error.	U	☑
02	Un usuario solicita un código para restablecer la contraseña con su email.	Se envía un email con el código.	U	☑
03	El usuario intenta cambiar su contraseña sin rellenar algún campo.	El cambio de contraseña no se produce. Se informa del error.	U	☑
04	El usuario intenta cambiar su contraseña pero las contraseñas no coinciden.	El cambio de contraseña no se produce. Se informa del error.	U	☑
05	El usuario intenta cambiar su contraseña pero la contraseña no cumple el patrón.	El cambio de contraseña no se produce. Se informa del error.	U	☑
06	El usuario intenta cambiar su contraseña pero el código no es correcto o ha expirado.	El cambio de contraseña no se produce. Se informa del error.	U	☑
07	El usuario intenta cambiar su contraseña con el código recibido.	Se cambia la contraseña del usuario.	U	☑
08	El usuario intenta cambiar los datos de nombre, apellidos y/o año nacimiento.	Se actualizan los datos que han sido pasados.	U	☑
09	El usuario intenta borrar su cuenta con una contraseña incorrecta.	La cuenta no es borrada. Se informa del error.	U	☑
10	El usuario intenta borrar su cuenta con su contraseña.	Se borra la cuenta del usuario.	U	☑
11	El usuario quiere acceder a sus datos	Se muestran los datos del usuario.	U	☑
12	El administrador quiere obtener los datos de todos los usuarios	Se proporciona un listado con los datos del usuario.	A	☑
13	El administrador quiere obtener los datos de un usuario a través de su Id, pero su id no existe.	No se muestra ningún dato. Se informa del error.	A	☑
14	El administrador quiere obtener los datos de un usuario a través de su Id.	Se muestran los datos del usuario que coincide con el Id.	A	☑
15	El administrador otorga permisos de administrador a un usuario que no existe.	No se otorga ningún permiso. Se informa del error.	A	☑
16	El administrador otorga permisos de administrador a un usuario que ya los tiene.	No se otorga ningún permiso. Se informa del error.	A	☑
17	El administrador otorga permisos de administrador a un usuario.	Se conceden los permisos al usuario.	A	☑
15	El administrador quita permisos de administrador a un usuario que no existe.	No se quita ningún permiso. Se informa del error.	A	☑
16	El administrador quita permisos de administrador a un usuario que no los tiene.	No se quita ningún permiso. Se informa del error.	A	☑
17	El administrador quita permisos de administrador a un usuario.	Se quita los permisos al usuario.	A	☑

18	El administrador otorga permisos de máster a un usuario que no existe.	No se otorga ningún permiso. Se informa del error.	A	☑
19	El administrador otorga permisos de máster a un usuario que ya los tiene.	No se otorga ningún permiso. Se informa del error.	A	☑
20	El administrador otorga permisos de máster a un usuario.	Se conceden los permisos al usuario.	A	☑
21	El administrador quita permisos de máster a un usuario que no existe.	No se quita ningún permiso. Se informa del error.	A	☑
22	El administrador quita permisos de máster a un usuario que no los tiene.	No se quita ningún permiso. Se informa del error.	A	☑
23	El administrador quita permisos de máster a un usuario.	Se quitan los permisos al usuario.	A	☑
24	El administrador bloquea un usuario que no existe.	No se bloquea ningún usuario. Se informa del error.	A	☑
25	El administrador bloquea un usuario ya bloqueado.	No se bloquea ningún usuario. Se informa del error.	A	☑
26	El administrador bloquea un usuario.	Se bloquea al usuario.	A	☑
27	El administrador desbloquea un usuario que no existe.	No se desbloquea ningún usuario. Se informa del error.	A	☑
28	El administrador desbloquea un usuario que no los tiene.	No se desbloquea ningún usuario. Se informa del error.	A	☑
29	El administrador desbloquea un usuario.	Se desbloquea al usuario. Se informa del error.	A	☑

Tabla 4: Pruebas integración Fase 2 – Gestión de cuentas de usuario

F02-04	**Subsistema: Informes de errores** **Propósito: Creación y gestión de los informes de errores**			
Nº	Entrada	Salida esperada	P	V
01	Un usuario intenta crear un informe de error, pero falta uno o más campos.	No se genera ningún informe de error. Se informa del error.	U	☑
02	Un usuario intenta crear un informe de error.	Se genera un nuevo informe de error.	U	☑
03	El administrador quiere ver todos los informes de errores.	Se muestran todos los informes de errores generados.		☑
04	El administrador quiere ver un informe de error en base a su Id, pero la Id no existe.	No se muestra ningún informe de error. Se informa del error.		☑
05	El administrador quiere ver un informe de error en base a su Id.	Se muestra el informe de error que corresponde con la Id facilitada.		☑
06	El administrador quiere actualizar el estado de un informe de error, a un informe cuyo Id no existe.	No se actualiza el estado de ningún informe de error. Se informa del error.		☑
07	El administrador quiere actualizar el estado de un informe de error, a un estado no válido.	No se actualiza el estado del informe de error. Se informa del error.		☑
08	El administrador quiere actualizar el estado de un informe de error, a un estado válido.	Se actualiza el informe de error al nuevo estado.		☑

Tabla 5: Pruebas integración Fase 2 – Informes de errores

F03-01	Subsistema: Personajes Propósito: Creación, edición y uso de los personajes			
Nº	Entrada	Salida esperada	P	V
01	El usuario intenta crear un personaje sin indicar el nombre.	El personaje no es creado. Se informa del error.	U	☑
02	El usuario intenta crear un personaje indicando un nombre repetido.	El personaje no es creado. Se informa del error.	U	☑
03	El usuario intenta crear un personaje indicando un nombre no repetido.	El personaje es creado.	U	☑
04	El usuario intenta crear un personaje indicando un nombre no repetido, teniendo ya un personaje.	El personaje no es creado. Se informa del error.	U	☑
05	El usuario intenta crear un personaje indicando un nombre no repetido, teniendo ya un personaje.	El personaje es creado.	M	☑
06	El usuario intenta editar un personaje que no está como personaje activo.	El personaje no se puede editar. Se informa del error.	M	☑
07	El usuario intenta editar un personaje que no es suyo.	El personaje no se puede editar. Se informa del error.	U	☑
08	El usuario intenta editar un personaje que ya ha sido finalizado.	El personaje no se puede editar. Se informa del error.	U	☑
09	El usuario intenta editar un personaje que ya ha sido validado.	El personaje no se puede editar. Se informa del error.	U	☑
10	El usuario intenta guardar o finalizar un personaje sin rellenar algún campo.	El personaje no se puede guardar o finalizar. Se informa del error.	U	☑
11	El usuario intenta guardar o finalizar un personaje sin adjuntar las imágenes.	El personaje no se puede guardar o finalizar. Se informa del error.	U	☑
12	El usuario intenta guardar o finalizar un personaje con valores de atributos inferiores a 4 o superiores a 10.	No es posible realizar la acción. Se informa del error.	U	☑
13	El usuario intenta guardar o finalizar un personaje con un valor total de atributos distinto a 27.	El personaje no se puede guardar o finalizar. Se informa del error.	U	☑
14	El usuario intenta guardar o finalizar un personaje con valores de habilidades inferiores a 0 o superiores a 10.	No es posible realizar la acción. Se informa del error.	U	☑
15	El usuario intenta guardar o finalizar un personaje con un valor total de habilidades distinto a 70.	El personaje no se puede guardar o finalizar. Se informa del error.	U	☑
16	El usuario intenta guardar o finalizar un personaje con un reparto de niveles de habilidad incorrectos.	El personaje no se puede guardar o finalizar. Se informa del error.	U	☑
17	El usuario intenta guardar o finalizar un personaje con un valor de altura, peso o edad fuera del rango de su raza.	No es posible realizar la acción. Se informa del error.	U	☑
18	El usuario intenta subir una imagen de avatar que supera las dimensiones o el tamaño establecido.	No es posible realizar la acción. Se informa del error.	U	☑

19	El usuario intenta subir una imagen de avatar acorde a las dimensiones o el tamaño establecido.	La imagen es subida y asignada al personaje.	U	☑
20	El usuario intenta subir una imagen de personaje que supera las dimensiones o el tamaño establecido.	No es posible realizar la acción. Se informa del error.	U	☑
21	El usuario intenta subir una imagen de personaje acorde a las dimensiones o el tamaño establecido.	La imagen es subida y asignada al personaje.	U	☑
22	El usuario intenta guardar un personaje con los valores y parámetros correctos.	Los cambios en el personaje quedan guardados.	U	☑
23	El usuario intenta finalizar un personaje con los valores y parámetros correctos.	Los cambios en el personaje quedan guardados y el personaje se marca como finalizado. El usuario ya no puede editar el personaje. Un Máster puede ya validarlo.	U	☑
24	El visitante intenta visualizar todos los personajes en la pestaña de población	Se muestran las tarjetas de todos los personajes validados.		☑
25	El visitante intenta visualizar un personaje haciendo clic sobre su tarjeta.	Se muestra la información detallada del personaje.		☑
26	El usuario intenta visualizar un personaje de su propiedad haciendo clic sobre su tarjeta.	Se muestra la información detallada del personaje, que incluye información sensible.	U	☑
27	El Máster intenta visualizar todos sus personajes en el panel de selección de personajes.	Se muestran todos los personajes del usuario y la opción de activar todos ellos, excepto el que ya está activo.	M	☑
28	El Máster selecciona otro personaje suyo para que sea el personaje activo.	El personaje activo es ahora el seleccionado.	M	☑
29	El Máster intenta validar un personaje de su propiedad.	No es posible. Se informa del error.	M	☑
30	El Máster intenta validar un personaje que no es de su propiedad.	Se muestra el personaje y el panel de validación.	M	☑
31	El Máster valida un personaje sin asignar fondos iniciales.	No es posible. Se muestra el error.	M	☑
32	El Máster valida un personaje asignando fondos iniciales.	El personaje queda validado.	M	☑
33	El Máster rechaza la validación de un personaje sin indicar el motivo.	No es posible. Se muestra el error.		☑
34	El Máster rechaza la validación de un personaje indicando los motivos.	El personaje pierde el estado de finalizado, pudiendo volverse a editar. Se le adjuntan los comentarios del máster, que se podrán visualizar a la hora de su edición.	M	☑

Tabla 6: Pruebas integración Fase 3 – Personajes

F03-02	Subsistema: Conocimientos Propósito: Gestionar y usar las habilidades de conocimientos			
Nº	Entrada	Salida esperada	P	V
01	El usuario intenta acceder a una habilidad de conocimiento que no tiene adquirida su personaje.	No es posible acceder. Se informa del error.	U	☑
02	El usuario intenta acceder a una habilidad de conocimiento que tiene adquirida su personaje.	Se muestra la página de conocimientos de la habilidad seleccionada con una cantidad de relatos acorde al nivel de la habilidad.	U	☑
03	El Máster accede a gestionar una habilidad de conocimientos y reordena las tarjetas de las historias.	Las historias quedan ordenadas según el nuevo orden.	M	☑
04	El Máster accede a gestionar una habilidad de conocimientos y edita el contenido de una historia mediante el editor.	La historia muestra los cambios de contenido o formato efectuados.	M	☑
05	El Máster accede a gestionar una habilidad de conocimientos y crea una nueva historia mediante el editor.	La nueva historia es creada y aparece en la última posición.	M	☑

Tabla 7: Pruebas integración Fase 3 – Conocimientos

F03-03	Subsistema: Página principal Propósito: Gestionar y visualizar la página principal			
Nº	Entrada	Salida esperada	P	V
01	El visitante accede a la página principal	Se muestran los datos de la página principal en el formato adecuado.		☑
02	El administrador modifica los datos o el formato de la página principal mediante su editor.	Se muestran los cambios efectuados en la página principal.	A	☑

Tabla 8: Pruebas integración Fase 3 – Página principal

F04-01	Subsistema: Eventos Propósito: Creación, Visualización y gestión de las misiones			
Nº	Entrada	Salida esperada	P	V
01	Un usuario sin permisos de máster intenta crear una misión.	No es posible realizar la acción.		☑
02	El Máster intenta crear una misión sin rellenar algún campo obligatorio.	La misión no es creada. Se informa del error.	M	☑
03	El Máster intenta crear una misión sin adjuntar una imagen.	La misión no es creada. Se informa del error.	M	☑
04	El Máster intenta gestionar una misión que no ha sido creada por él.	No se permite gestionar. Se informa del error.	M	☑
05	El Máster intenta gestionar una misión que ya no está en preparación.	No se permite gestionar. Se informa del error.	M	☑
06	El Máster intenta añadir un personaje a la misión, siendo que ya se encuentra en otra misión, y guardar los cambios.	No se permite añadir. Se informa del error.	M	☑

07	El Máster intenta añadir un personaje a la misión, que no está como posible participante, y guardar los cambios.	No es posible realizar la acción.	M	☑
08	El Máster intenta añadir un personaje a la misión, apuntado como posible participante, y guardar los cambios.	Se permite añadir. El personaje aparece como participante y el número de participantes aumenta.	M	☑
09	El Máster intenta añadir un personaje propio a la misión y guardar los cambios.	Se permite añadir. El personaje aparece como participante y el número de participantes aumenta.	M	☑
10	El Máster intenta comenzar una misión donde aún no están todos los personajes.	No se permite comenzarla. Se informa del error.	M	☑
11	El Máster intenta comenzar una misión estando todos los personajes.	La misión cambia su estado a empezada.	M	☑
12	El Máster intenta finalizar una misión que no ha sido creada por él.	No se permite finalizar. Se informa del error.	M	☑
13	El Máster intenta finalizar una misión que no está empezada.	No se permite finalizar. Se informa del error.	M	☑
14	El Máster intenta finalizar una misión empezada de su propiedad.	La misión cambia su estado a finalizada. Los personajes participantes ven incrementado su dinero y experiencia a partes iguales.	M	☑
15	El Máster intenta borrar una misión que no ha sido creada por él.	No se permite borrar. Se informa del error.	M	☑
16	El Máster intenta borrar una misión que ya no está en preparación.	No se permite borrar. Se informa del error.	M	☑
17	El Máster intenta borrar una misión de su propiedad, que está en preparación.	La misión se elimina, no aparece en el listado de misiones.	M	☑
18	El visitante intenta visualizar todas las misiones en la pestaña de ver misiones	Se muestran las tarjetas de todas las misiones creadas.		☑
19	El usuario intenta visualizar todas las misiones en las que participa haciendo clic en la pestaña de mis misiones	Se muestran las tarjetas de todas las misiones creadas en las que participa el personaje.	U	☑
20	El visitante intenta visualizar una misión haciendo clic sobre su tarjeta.	Se muestra la información detallada y el contenido de la misión.		☑
21	Un personaje no validado intenta unirse a una misión	No se permite unir. Se informa del error.	U	☑
22	Un personaje intenta unirse a una misión que no está en preparación.	No se permite unir. Se informa del error.	U	☑
23	Un personaje del máster creador de la misión, intenta unirse a la misión.	No se permite unir. Se informa del error.	M	☑
24	Un personaje ya unido a la misión, intenta unirse a la misión.	No se permite unir. Se informa del error.	U	☑
25	Un personaje seleccionado como participante de la misión, intenta unirse a la misión.	No se permite unir. Se informa del error.	U	☑
26	Un personaje intenta unirse a la misión.	El personaje pasa a formar parte de los posibles participantes de la misión.	U	☑

Tabla 9: Pruebas integración Fase 4 – Eventos

F04-02	Subsistema: Turnos y Mensajes Propósito: Obtención de turno y escritura			
Nº	Entrada	Salida esperada	P	V
01	Un personaje sin validar intenta coger turno en una misión.	No se permite coger turno. Se informa del error.	U	☑
02	Un personaje intenta coger turno en una misión que no está empezada.	No se permite coger turno. Se informa del error.	U	☑
03	Un personaje que no pertenece a la misión intenta coger turno en la misión.	No se permite coger turno. Se informa del error.	U	☑
04	Un personaje que pertenece a la misión intenta coger turno en la misión mientras otro personaje tiene el turno.	No se permite coger turno. Se informa del error.	U	☑
05	Un personaje que pertenece a la misión intenta coger turno en la misión mientras hay un mensaje en revisión.	No se permite coger turno. Se informa del error.	U	☑
06	Un personaje que pertenece a la misión intenta coger turno en la misión.	Se crea un turno para el personaje. Se habilita la opción de escribir en la misión.	U	☑
07	Un personaje que no tiene el turno actualmente intenta descartar su turno en una misión.	No se permite descartar el turno. Se informa del error.	U	☑
08	Un personaje que tiene el turno actualmente intenta descartar su turno en una misión.	Su turno queda descartado. Se deshabilita la opción de escribir en la misión.	U	☑
09	Un personaje intenta editar un mensaje que no es suyo.	No se permite editar el mensaje. Se informa del error.	U	☑
10	Un personaje intenta escribir un mensaje teniendo el turno en la misión, apretando el botón 'Escribir'.	Se muestra el editor de mensajes.	U	☑
11	Un personaje intenta editar un mensaje de la misión que no está empezada.	No se permite. Se informa del error.	U	☑
12	Un personaje intenta editar un mensaje de la misión que ha sido marcado como finalizado.	No se permite. Se informa del error.	U	☑
13	Un personaje marca su mensaje como finalizado	En mensaje cambia su estado a finalizado, y se le muestra como pendiente de finalizar al máster.	U	☑
14	Un personaje intenta editar un mensaje de la misión que ha sido validado.	No se permite. Se informa del error.	U	☑
15	El Máster accede a Validar mensajes	Se muestran los mensajes pendientes de validar pertenecientes a las misiones del máster	M	☑
16	El Máster intenta validar un mensaje ya validado.	No es posible. Se informa del error.	M	☑
17	El Máster intenta validar un mensaje sin validar.	Se valida el mensaje y se añade al contenido de la misión.	M	☑
18	El Máster rechaza la validación de un mensaje sin indicar el motivo.	No es posible. Se muestra el error.	M	☑
19	El Máster rechaza la validación de un mensaje indicando los motivos.	El mensaje puede volverse a editar. Se le adjuntan los comentarios del Máster, que se podrán visualizar a la hora de su edición.	M	☑

Tabla 10: Pruebas integración Fase 4 – Turnos y mensajes

F04-03	Subsistema: Habilidades de escritura y puntos de escritura Propósito: Uso de habilidades de escritura y obtención de puntos de escritura.			
Nº	Entrada	Salida esperada	P	V
01	Un personaje, accede al panel de escritura de la misión.	Se muestran los puntos de escritura fijos, de misión y de mensaje.	U	☑
02	Un personaje intenta utilizar una habilidad no implementada.	No es posible realizar la acción.	U	☑
03	Un personaje intenta utilizar una habilidad cuando no es su turno.	No es posible realizar la acción.	U	☑
04	Un personaje intenta utilizar una habilidad en una misión que no es la suya.	No es posible realizar la acción.	U	☑
05	Un personaje intenta utilizar una habilidad en una misión que no está empezada.	No es posible realizar la acción.	U	☑
06	Un personaje intenta utilizar una acción de enfrentamiento consigo mismo.	No es posible realizar la acción.	U	☑
07	Un personaje intenta utilizar una habilidad sin seleccionar la acción o la dificultad.	No es posible. Se muestra el error.	U	☑
08	Un personaje intenta realizar un enfrentamiento de la habilidad sin seleccionar la acción o especificar el rival.	No es posible. Se muestra el error.	U	☑
09	Un personaje utiliza una habilidad seleccionando la acción y la dificultad.	Se muestra el resultado de la utilización de la habilidad y el resultado de los dados. En el listado de puntos de escritura del editor de mensajes de misión, aparece reflejado el nuevo punto de escritura.	U	☑
10	Un personaje realiza un enfrentamiento de habilidad seleccionando la acción y especificando el rival.	Se muestra el resultado del enfrentamiento de habilidad y el resultado de los dados. En el listado de puntos de escritura del editor de mensajes de misión, aparece reflejado el nuevo punto de escritura.	U	☑
11	Un personaje realiza una acción o un enfrentamiento de habilidad seleccionando y especificando correctamente los datos y, como resultado, recibe daño.	Se muestra el daño recibido como resultado de uso de la habilidad. El personaje recibe los puntos de daño y se ven reflejados.	U	☑

Tabla 11: Pruebas integración Fase 4 – Hab. y puntos de escritura

7. Valoración económica del trabajo

7.1 Costes de desarrollo

En la valoración económica de los costes de desarrollo se han omitido las horas dedicadas la redacción de los entregables de la universidad, a excepción de la elaboración del plan de trabajo, que se ha considerado económicamente.

En el apartado de recursos humanos, para el coste del precio hora, se ha establecido el coste medio de un programador junior en España de 25€/hora.

En la parte de recursos tecnológicos, se han añadido los costes de hardware del servidor. En cuanto a *software*, se ha utilizado siempre *software* con licencias gratuitas, y librerías de código abierto y gratuito. El hospedaje de la aplicación se hace en el servidor, pero los costes asociados al dominio se incluyen en el presupuesto.

Tarea	Horas	Precio hora	Total
Recursos humanos			5750€
Fase 1			1525€
Elaboración del Plan de Trabajo	34	25€	850€
Puesta en marcha del servidor	14	25€	350€
Configuración del Dominio	4	25€	100€
Pruebas Fase 1	9	25€	225€
Fase 2			1100€
Desarrollo del registro, autentificación, gestión y roles de usuario	22	25€	550€
Desarrollo de los informes de errores	14	25€	350€
Pruebas Fase 2	8	25€	200€
Fase 3			1475€
Desarrollo de la Creación y gestión de personajes	25	25€	625€
Desarrollo de las habilidades de trasfondo	22	25€	550€
Pruebas Fase 3	12	25€	300€
Fase 4			1650€
Desarrollo de la creación y gestión de eventos	20	25€	500€
Desarrollo del panel de escritura	12	25€	300€
Desarrollo de las habilidades de escritura	22	25€	550€
Pruebas Fase 4	12	25€	300€

Tabla 12: Costes de desarrollo en recursos humanos

Tarea	Cantidad	Precio/Ud.	Total
Recursos tecnológicos			173,84€
Servidor Raspberry Pi 4	1	86,07€	86,07€
Registro de dominio vinland.quest	1 año	1.70€	1,70€
Licencia comercial plantilla html/css	1	44,83€	44,83€

Tabla 13: Costes de desarrollo en recursos tecnológicos

Tenemos un coste total de desarrollo de 5923,84€. Siendo la mayor parte del presupuesto las horas de desarrollo con un coste de 5750€, y los recursos tecnológicos con un coste de 173,84€.

Quedan excluidos del presupuesto los gastos asociados a costes eléctricos, de conexión de datos o de alquiler del puesto de trabajo.

7.2 Costes de mantenimiento

Los costes de mantenimiento de la aplicación incluyen el mantenimiento de la aplicación, el desarrollo de nuevas funcionalidades, renovación del dominio y los costes energéticos del servidor. Quedan excluidos de los costes de mantenimiento los gastos asociados a costes de conexión de datos o de alquiler del puesto de trabajo.

El mantenimiento de la aplicación consiste en mantener actualizado el software del servidor para evitar vulnerabilidades, crear copias de seguridad y corregir errores que puedan surgir. Por otra parte, en el desarrollo de nuevas funcionalidades, se añadirán nuevas funciones a la aplicación según lo detallado en el punto '8.4 Líneas de trabajo futuras '. Para ambas, se establece una cantidad estimada de horas, en función de las horas de desarrollo obtenidas.

Para el consumo eléctrico del servidor, se han obtenido los datos de consumo de la Raspberry Pi 4, siendo su consumo medio de 3W. El coste del consumo anual se ha tenido en cuenta con un precio KW/h de 0,2€, estando encendido 24 horas al día, 7 días a la semana. Quedando un total del consumo eléctrico del servidor de 5,16€ anuales.

Tarea	Horas anuales	Precio hora	Total
Recursos humanos			2850€
Mantenimiento de la aplicación	24	25€	600€
Desarrollo de nuevas funcionalidades	90	25€	2250€

Tabla 14: Costes de mantenimiento en recursos humanos

Tarea	Cantidad	Precio/Ud.	Total
Recursos tecnológicos			18,43€
Renovación del dominio vinland.quest	1 año	13,27€	13,27€
Consumo eléctrico del servidor	1 año	5,16€	5,16€

Tabla 15: Costes de mantenimiento en recursos tecnológicos

Tenemos un coste anual de mantenimiento de 2868,23€.

7.3 Beneficios

El proyecto no busca obtener beneficios, aunque sí intentar cubrir los costes de mantenimiento anuales.

Los posibles ingresos que puede obtener el proyecto únicamente pueden provenir a partir de patrocinadores que estén interesados en la aplicación. Para ello, se puede hacer uso de Patreon y ofrecer contenido exclusivo a los patrocinadores, o el desarrollo de las nuevas funcionalidades.

Si la aplicación obtuviera una buena acogida, y tuviese una cantidad de usuarios activos moderada, se podrían explorar otras posibilidades como el obtener

ingresos por mostrar publicidad de juegos de rol, de mesa, etc. cuyo público objetivo es el que suele utilizar la aplicación desarrollada.

Por tanto, para poder cubrir los costes de mantenimiento, deberíamos tener un total de 60 patrocinadores en Patreon con una suscripción mensual de 4€.

7.4 Viabilidad del producto

Según lo que hemos podido ver en los anteriores apartados, la obtención de beneficios con el producto no es factible. Aunque sí que es viable llegar a mitigar los costes de mantenimiento.

Pese a que un número de 60 suscriptores en este tipo de aplicación es muy elevado y poco probable, con 13 suscriptores cubriríamos los costes de mantenimiento de la aplicación sin el desarrollo de nuevas funcionalidades.

Concluyendo, el producto no va a buscar beneficio económico y únicamente intentará cubrir los costes de mantenimiento.

8. Conclusiones

8.1 Lecciones aprendidas

A lo largo de todo el proyecto he aprendido a plantear, de la mejor manera posible, los trabajos a realizar en su fase de diseño. He podido poner a prueba mis conocimientos en Angular y NodeJS, sin tener muchos percances. También he aprendido a buscar soluciones conocidas a problemas que me iban surgiendo, ya que la mayoría de veces existían librerías para integrar en el proyecto que me ofrecían una solución ya probada. Mis conocimientos de CSS han mejorado hasta un punto que no creía posible, llegando incluso a gustarme.

Por otro lado, he aprendido a trabajar con diferentes entornos de desarrollo, cosa que aún no había hecho. Al igual que configurar por primera vez un dominio, habilitar el protocolo SSL/TLS en el servidor y aprender a configurar Apache. La realización de automatizaciones de pruebas unitarias en Postman ha sido nueva para mí y la he encontrado de mucha utilidad.

Aunque el camino hasta lograr finalizar el trabajo haya sido muy duro, no lo cambiaría. Me ha servido para poner en práctica todas las competencias que he aprendido durante la carrera y fuera de ella.

8.2 Consecución de los objetivos planteados

El objetivo principal, y sus subobjetivos asociados, se han logrado plenamente. La aplicación está accesible a través de https://vinland.quest y cumple con las funcionalidades de registro de usuarios, informes de errores, creación de personajes, habilidades de trasfondo, creación de eventos, panel de escritura y habilidades de escritura.

Además de los objetivos marcados, se ha desarrollado uno más que consiste en gestionar el mensaje de la página principal del sitio.

8.3 Seguimiento de la planificación y metodología

La planificación se ha seguido en todo momento, con una dedicación aproximada de 3 horas diarias. Aunque en algunos días concretos, como festivos, puede no haberse dedicado horas y haberlas realizado otro día. Pero la media de horas diarias ha estado en torno a las 3 horas.

El uso de una metodología Agile, dividiendo el proyecto en 4 fases me ha parecido muy acertada para este proyecto, ya que permitía dividir todo el trabajo en bloques y no añadía mucho trabajo adicional. Aparte, de no haberse conseguido alguna de las fases, quedaría igualmente un producto usable, aunque con menos funcionalidades.

Durante todo el desarrollo, la parte que más se ha tenido que modificar para conseguir que funcionase bien ha sido la de las habilidades de escritura. El planteamiento inicial cambiaba conforme a las necesidades de las demás partes

y hubo que detenerse, verlo todo en conjunto y rediseñarlo. El resto de funcionalidades no han requerido mucho rediseño.

8.4 Líneas de trabajo futuro

En cuanto a las líneas de trabajo futuro, quedan muchas funcionalidades por implementar. Se detalla brevemente a continuación las funcionalidades que se pretender implementar:

- Desarrollar el resto de habilidades, que otorgarían nuevas funciones a la hora de redactar y generarían nuevos puntos de escritura.
- Añadir objetos, que servirían para reponer estados, comercio e intercambio entre usuarios y su uso como recompensas en misiones.
- Los combates entre usuarios pueden ser otra buena alternativa a explorar. Estos generarían un resultado del enfrentamiento que los usuarios deberían redactar.
- Implementar un sistema de logros para motivar a los usuarios a conseguir ciertos objetivos redactando.

Estas líneas de trabajo futuro, se seguirán desarrollando una vez finalizado el TFG.

9. Glosario

Storytelling: Arte o capacidad de contar una historia.
Máster: Persona encargada de la supervisión narrativa.
Rol: Juego en el que los jugadores interpretan un determinado papel o personalidad a través de su personaje.
Rol storytelling: Capacidad de contar historias interpretando un papel o personalidad a través de un personaje.
Software: Conjunto de instrucciones que realizan una tarea en una computadora.
Hardware: Conjunto de elementos físicos que componen una computadora.
Sistema operativo: Conjunto de programas que gestionan los elementos de *hardware* ofreciendo servicios a los programas de aplicación.
Frontend: Parte del *software* que interactúa con los usuarios.
Backend: Parte del *software* que procesa en el servidor la llamada desde el *frontend*.
Agile: Método de ingeniería del software basado en el desarrollo incremental e iterativo.
Servidor: Ordenador capaz de atender una petición por parte de un cliente y devolverle una respuesta.
Dominio: Nombre que identifica un área concreta de internet.
MEAN stack: *Framework* o conjunto de utilidades de software formado por MongoDB, Express.js, AngularJS y Node.js, para el desarrollo de aplicaciones web en lenguaje Javascript.
MongoDB: Sistema de base de datos NoSQL, que utiliza Javascript para representar la información y almacena los datos en BJSON (*Binary JavaScript Object Notation*).
Express o ExpressJS: Infraestructura de aplicaciones web para el entorno NodeJS que facilita la creación de aplicaciones web y API's.
AngularJS o Angular: *Framework* de Javascript utilizado para la creación de aplicaciones web que utiliza un patrón Modelo-Vista-Controlador.
NodeJS o Node: Entorno multiplataforma y en tiempo de ejecución que utiliza Javascript como lenguaje de programación.
Javascript: Lenguaje de programación orientado a objetos, dinámico, débilmente tipado y basado en prototipos.
API (Application Programming Interface): Conjunto de métodos que ofrece una biblioteca a otro *software* creando una capa de abstracción.
Framework: Conjunto de elementos de software utilizados para resolver un problema concreto.
NoSQL: Sistema de base de datos no relacional.
Typescript: Extensión de la sintaxis de JavaScript que añade tipos estáticos y objetos basados en clases
Patrón Modelo-Vista-Controlador (MVC): Patrón de arquitectura del software que separa los datos de la lógica de negocio y su representación.
IDE (Integrated Development Environment): Aplicación informática que ofrece elementos que facilitan al programador el desarrollo de *software*.
Git: Software utilizado para el control de versiones.
Github: Plataforma de almacenamiento de código fuente a través del control de versiones de Git.

HTML (HyperText Markup Language): Lenguaje construido con etiquetas y marcas que es utilizado para la creación de páginas web.
CSS (Cascading Style Sheets): Lenguaje de diseño gráfica que define la representación en pantalla de un código HTML.
FTP (File Transfer Protocol): Protocolo de transferencia de archivos entre un cliente y un servidor.
SSH (Secure SHell): Protocolo de acceso remoto, seguro y cifrado, a un servidor.
SAMBA: Protocolo de compartición de archivos utilizado por Microsoft Windows.
Nginx: Servidor web multiplataforma, de *software* libre y código abierto.
Apache: Servidor web multiplataforma de código abierto.
DHCP (Dynamic Host Configuration Protocol): Asignación dinámica de IP y parámetros de configuración de red a un equipo, por parte de un servidor DHCP.
Router: Dispositivo que permite la interconexión entre diferentes redes.
DNS (Domain Name Server): Sistema que asigna nombres a direcciones IP.
Dirección IP: Numeración que identifica una interfaz en una red.
TLS/SSL:
Let's Encrypt: Entidad certificadora que otorga certificados TLS/SSL de forma gratuita.
Certbot: Herramienta de software que habilita HTTPS en un sitio web utilizando Let's Encrypt.
BSON: Formato de intercambio de datos utilizado en la base de datos MongoDB.
OAuth: Servicio externo que provee la identificación del usuario.
Cookies: Información que ha sido enviada por un sitio web y almacenada en el navegador del usuario.
JWT (JSON Web Token): Permite la identificación y la definición de privilegios utilizando el formato para el intercambio de objetos en Javascript
JSON (JavaScript Object Notation): Formato para el intercambio de objetos en Javascript.
Token: Permite la identificación y la definición de privilegios.
Middleware: Elemento de *software* que proporciona servicios y funciones en común para las aplicaciones.
Mongoose: Biblioteca en Javascript que permite la comunicación entre una aplicación Express y la base de datos MongoDB.
Pipe: Herramienta de *software* que permite transformar la información.
Patreon: Pagina web para el mecenazgo de proyectos creativos a base de suscripciones o donaciones.

10. Bibliografía

[1] «The Witcher Wild Haunt,» [En línea]. Available: https://thewh.foroactivo.com/. [Último acceso: 22 03 2022].

[2] «Harajuku Noir,» [En línea]. Available: https://harajukunoir.foroactivo.com/. [Último acceso: 23 03 2022].

[3] «Fall and Rise - A Star Wars Roleplay Forum,» [En línea]. Available: https://fallandrise-rpg.foroactivo.com/. [Último acceso: 23 03 2022].

[4] «Endless Immortals,» [En línea]. Available: https://endless-immortals.foroactivo.com/. [Último acceso: 23 03 2022].

[5] «Rápido y Fácil,» [En línea]. Available: http://www.rapidoyfacil.es/. [Último acceso: 10 05 2022].

[6] «Especificaciones Raspberry Pi,» 21 10 2021. [En línea]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/. [Último acceso: 23 03 2022].

[7] «Rasperry Pi OS (64 Bits),» 02 02 2022. [En línea]. Available: https://www.raspberrypi.com/news/raspberry-pi-os-64-bit/. [Último acceso: 23 03 2022].

[8] «Manual instalación MongoDB,» [En línea]. Available: https://docs.mongodb.com/manual/installation/. [Último acceso: 23 03 2022].

[9] «Install Ubuntu on a Raspberry Pi,» [En línea]. Available: https://ubuntu.com/download/raspberry-pi. [Último acceso: 23 03 2022].

[10] «Historical quarterly trends in the usage statistics of web servers,» [En línea]. Available: https://w3techs.com/technologies/history_overview/web_server/ms/q. [Último acceso: 23 03 2022].

[11] «Nginx vs Apache: Lucha entre Servidores Web,» 21 02 2022. [En línea]. Available: https://kinsta.com/es/blog/nginx-vs-apache/. [Último acceso: 23 02 2022].

[12] «Cómo instalar el servidor web Apache en Ubuntu 20.04,» [En línea]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-the-apache-web-server-on-ubuntu-20-04-es. [Último acceso: 23 03 2022].

[13] «Install namecheap-ddns-client on Ubuntu,» 16 09 2021. [En línea]. Available: https://snapcraft.io/install/namecheap-ddns-client/ubuntu#install. [Último acceso: 23 03 2022].

[14] J. E. E. Colomer, «Redes y Aplicaciones de Internet - PEC03 y PR01,» UOC, 2020.

[15] «Cómo proteger Apache con Let's Encrypt en Ubuntu 20.04,» 21 05 2020. [En línea]. Available: https://www.digitalocean.com/community/tutorials/how-to-secure-apache-with-let-s-encrypt-on-ubuntu-20-04-es. [Último acceso: 23 03 2022].

[16] «Vsftpd servidor FTP para Linux: Instalación y configuración,» 06 08 2021. [En línea]. Available:

https://www.redeszone.net/tutoriales/servidores/vsftpd-configuracion-servidor-ftp/. [Último acceso: 23 03 2022].

[17] «Node.js,» 25 12 2021. [En línea]. Available: https://es.wikipedia.org/wiki/Node.js. [Último acceso: 23 03 2022].

[18] «Npm,» 28 10 2021. [En línea]. Available: https://es.wikipedia.org/wiki/Npm. [Último acceso: 23 03 2022].

[19] «MongoDB,» 22 03 2022. [En línea]. Available: https://es.wikipedia.org/wiki/MongoDB. [Último acceso: 23 03 2022].

[20] «Cómo instalar MongoDB en Ubuntu 20.04,» 11 09 2020. [En línea]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-mongodb-on-ubuntu-20-04-es. [Último acceso: 23 03 2022].

[21] «Cómo configurar una aplicación de Node.js para producción en Ubuntu 18.04,» 05 12 2019. [En línea]. Available: https://www.digitalocean.com/community/tutorials/como-configurar-una-aplicacion-de-node-js-para-produccion-en-ubuntu-18-04-es. [Último acceso: 23 03 2022].

[22] «Angular Guide - Server configuration,» [En línea]. Available: https://angular.io/guide/deployment#server-configuration. [Último acceso: 23 03 2022].

[23] «User Authentication in Web Apps (Passport.js, Node, Express),» 16 03 2021. [En línea]. Available: https://www.youtube.com/watch?v=F-sFp_AvHc8. [Último acceso: 23 03 2022].

[24] «Cookies y Sessions VS JSON Web Tokens,» [En línea]. Available: https://programacionymas.com/blog/jwt-vs-cookies-y-sesiones. [Último acceso: 23 03 2022].

[25] «JWT.IO allows you to decode, verify and generate JWT,» [En línea]. Available: https://jwt.io/. [Último acceso: 23 03 2023].

[26] «Reactive Extensions Library for JavaScript,» [En línea]. Available: https://rxjs.dev/. [Último acceso: 23 03 2022].

[27] «UX-Centered Bootstrap Themes & Templates,» [En línea]. Available: https://themes.3rdwavemedia.com/. [Último acceso: 23 03 2022].

[28] J. A. V. d. Riego, Taller de heráldica. Cómo diseñar y describir un escudo..

11. Anexos

Anexo I: Manual de la aplicación
Fichero PDF que contiene un pequeño manual para aprender a usar la aplicación.

Anexo II: Vistas
Fichero PDF que contiene un listado de las vistas desarrolladas en la aplicación.

Anexo III: Pruebas unitarias
Fichero HTML que contiene la batería de pruebas unitarias finales realizada. Que incluye las pruebas unitarias realizadas en las cuatro fases.

Repositorio del código fuente del *frontend*
El código se encuentra subido en el siguiente repositorio de Github:
https://github.com/josepec/TFG-VinlandQuest-Frontend

Repositorio del código fuente del *backend*
El código se encuentra subido en el siguiente repositorio de Github:
https://github.com/josepec/TFG-VinlandQuest-Backend