

MEMORIA ESCRITA DEL PROYECTO

CFGS Desarrollo de Aplicaciones Web

Blog Web – Harvest Moon Comunidad en Español

Autor: Mario Alfonso Medina Macías

Tutor: David Medrano

Fecha de entrega: 09/05/2022

Convocatoria: 2S – 2122

Documentos del proyecto:

https://drive.google.com/drive/folders/1810LrG2iAb3S75GbSRjiX-C7HR_5jsP?usp=sharing



Índice de contenidos

1. INTRODUCCIÓN	3
1.1. Motivación	3
1.2. Abstract.....	3
1.3. Objetivos propuestos (generales y específicos)	4
2. METODOLOGÍA USADA.....	5
2.1 Metodología Kanban.....	5
2.1.1 ¿Qué es Kanban?.....	5
2.1.2 Estructura Kanban y Ventajas y Desventajas	6
2.2 Ciclo de vida del proyecto.....	8
2.3 Fases del diseño del proyecto.....	9
3. TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS EN EL PROYECTO	10
4. ESTIMACIÓN DE RECURSOS Y PLANIFICACIÓN.....	13
5. ANÁLISIS DEL PROYECTO	17
5.1 Requisitos.....	17
5.2 Diagramas E/R, Casos de uso y Clases.....	19
5.2.1 Diagrama de Entidad-Relación.....	19
5.2.2 Diagrama de Casos de uso	20
5.2.3 Diagrama de Clases	25
6. DISEÑO DEL PROYECTO	27
6.1 Mockup y Bocetos de vistas	27
6.2 Realización del proyecto	28
6.2.1 Estructura de carpetas.....	28
6.2.2 SGBD, Tablas y Administración	29

6.2.3 Modelo Vista-Controlador	31
6.2.4 Helpers y funciones auxiliares	40
6.2.5 Archivos de configuración.....	42
6.2.6 Carpeta Public, Estilos, Imágenes y Maqueta.....	43
7. DESPLIEGUE Y PRUEBAS	44
8. CONCLUSIONES	47
9. VÍAS FUTURAS.....	48
10. GLOSARIO	50
11. BIBLIOGRAFÍA/WEBGRAFÍA.....	51

En la normativa de proyectos vigente encontrarás una breve descripción de cada uno de estos apartados para saber qué información debes incluir en ellos

1. Introducción

La idea de esta aplicación se basa en un sitio web a modo de blog en español donde usuarios de una comunidad de una saga de videojuegos, en concreto "*Harvest Moon*" (conocida ahora como "*Story Of Seasons*"), puedan tener un punto de encuentro donde: informarse sobre noticias y guías del juego en cuestión, abrir foros de discusión, elaborar posts de contenido *fandom*, comunicarse por chat privado y comprar *merchandising* no oficial entre otras posibles funcionalidades futuras.

Haciendo un preanálisis de las tecnologías que deberemos usar se encuentran, sin lugar a dudas, lenguajes de marcas como HTML y CSS en aspectos de *FrontEnd* del aplicativo y PHP como lenguaje de programación del lado *BackEnd*. Para almacenar los datos y realizar consultas también deberemos disponer de un sistema gestor de bases de datos como es MySQL. A todo esto, tendremos que añadirle un IDE (entorno de desarrollo integrado) en el cuál desarrollar todos los distintos códigos de nuestra aplicación, como así también depurar la ejecución y ayudarnos a formatear de una manera correcta todos estos archivos.

Como propuesta inicial al IDE que usar podría tratarse de NetBeans o Eclipse y un editor de código como Visual Studio Code aunque, este último comparte muchas de las funcionalidades de un IDE.

1.1. Motivación

Pienso que la temática de este proyecto, además de ilusionarme y constituir un interés personal por desarrollar un sitio web relacionado con mis intereses, me ayudará a trabajar de forma más entusiasta; los distintos apartados funcionales de la web harán que me desarrolle como profesional y aprenda a fondo varias tecnologías imprescindibles en el sector del desarrollo de aplicaciones web.

1.2. Abstract

We want to welcome you to the "*Harvest Moon & Story of Seasons Community in Spanish*"!

Warning:

*Currently this community is made up mostly of a Hispanic public but if you know Spanish or would like to practice it, do not hesitate to join and interact with other Spanish-speaking users.

However, if that is not your case, do not worry because we are currently working on support for the application in the English language, although there are no confirmed implementation dates.

This website is a meeting point for fans of the sagas of these beloved video games. Here you can find information about the latest news related to the saga, follow guides or tricks of the game you want, discuss topics related to each of these video games, participate in special events and even create your own posts for the community, in different ordered categories, such as: your own guides of your own elaboration, share your fanarts, notify the rest of the platform users of news that you have discovered, etc...

Although if you really have no interest in contributing content to the community you can always interact with the different posts already uploaded by other users like you or meet people with whom you can create bonds of friendship.

Our moderation team is constantly checking the content that is being implemented in the forum, so don't worry because we have zero tolerance for toxic, racist, homophobic and illegal attitudes.

We hope you have the best time possible during your stay in this little corner of the internet made up of people like you who are in love with farm management video games.

A warm welcome from the "*Harvest Moon & Story of Seasons Community in Spanish*" team!

1.3. Objetivos propuestos (generales y específicos)

El principal objetivo de este proyecto web es almacenar todo tipo de datos relacionado con el contenido de cada uno de los juegos de esta saga, tanto antiguos como futuros lanzamientos, y ofrecer otras posibilidades tales como: guías para el jugador, eventos dentro del videojuego, curiosidades (*easter egg's*), noticias... y que distintos tipos de agentes (usuarios y administradores) puedan aportar, modificar o reaccionar a la información mostrada en las distintas entradas de esta página *fandom* (creada y concebida para los fans) en relación con el tipo de permisos que les sea concedido para ello; es decir, dentro de su rol en este sitio web.

Para esto, todos los usuarios deberán de poder interactuar con la web a distintos niveles, lo que implica además una distinta asignación de privilegios y cuotas en cuanto a dicha interacción e, inherentemente, un método para reconocer y validar (autenticar) cada uno de estos navegantes.

Se proporcionará así una página totalmente dinámica e interactiva, en especial con la información de esta, personalizando aún más la experiencia de cada visitante frente al resto.

Indagando más en las funciones específicas que se espera para la web se nos presentan las siguientes ideas:

- **Clasificación** de cada uno de los juegos mostrando: guías, información de lanzamiento, reviews...
- **Apartado de noticias** donde mostrarán distintos comunicados de la desarrolladora y la comunidad.
- **Foros de discusión**, dudas, ayuda, búsqueda de jugadores activos y temas *offtopic* aprobados.
- **Mercado de la comunidad tipo Workshop** abierto a ofertas y demandas de todos los usuarios.

2. Metodología usada

Dada la gran envergadura que supone el desarrollo de productos software es un requisito fundamental el aplicar una o diversas metodologías para que el proceso previo a producción sea lo menos caótico posible, evitando así fallos, imprevistos, malas organizaciones y reparto desproporcionado de tareas entre los desarrolladores. Esto nos ayuda de gran manera a tener el proyecto controlado y responder rigurosamente a fechas de entrega límite prestando un buen servicio empresarial tanto a los demás programadores como al cliente en cuestión.

2.1 Metodología Kanban

Para el desarrollo de este proyecto fin de ciclo, aunque no cuenta con la participación de más colaboradores (entendiéndose como yo mismo, Mario Alfonso Medina Macías, la única persona a cargo de todos los apartados del proyecto) ha sido imprescindible la aplicación de una metodología ágil en el desarrollo de software, como: Kanban, la cuál ha sido la principal metodología que se ha seguido durante el avance de toda la aplicación web.

2.1.1 ¿Qué es Kanban?

Kanban palabra japonesa formada por dos ideogramas de origen chino (en japonés, Kanji): Kàn (看) con el significado de “seña visual” y Bǎn (板) “tarjeta o tablero”, respectivamente.

Fue inventado por Taiichi Ohno, un ingeniero japonés de Toyota, en 1940. Esta metodología fue adaptada para el desarrollo de software a principio de la década de los 2000 y este usa el mismo “proceso de extracción”. En esta versión del Kanban cada equipo comienza con un listado de tareas a realizar. El trabajo es extraído de la columna de “cosas por hacer” o más conocido por “To Do” según la asignación de tareas previa que se haya planificado. Gracias a esto es posible mostrar a todo el equipo una idea general en constante actualización de todas las tareas del proyecto, dividiéndose en tres columnas: “To do” (a realizar), “Done” (hecho) y “Doing” (en proceso), la intermedia. Una vez finalizado el proyecto será visible que todas las tarjetas de tareas estarán agrupadas bajo la columna “Done”.

2.1.2 Estructura Kanban y Ventajas y Desventajas



Ilustración 1: Representación Kanban. Fuente: Google Images.

Como bien vimos en el anterior punto, la estructura de la metodología Kanban se resumen en un tablero con diferentes tareas dispuestas en las tres siguientes columnas:

- **To do:** Tareas a realizar.
- **Doing:** Tareas que se están realizando.
- **Done:** Tareas ya completadas.

Esta metodología como cualquier otra conlleva una serie de ventajas y desventajas, veamos algunas de ellas:

Ventajas:

- **Vista general del estado del proyecto y rendimiento:** cada colaborador podrá tener una vista organizada de cómo va el desarrollo de la solución y ordenar su propio trabajo.
- **Organización del trabajo:** el *project manager* podrá crear cada una de estas tarjetas organizando el trabajo de forma equitativa y asumible para que cualquiera de los trabajadores pueda elegir por sí mismo de qué tarjeta (tarea) se hará responsable.
- **Cumplir fechas de entregas previstas:** gracias a que todo el trabajo está estructurado desde el principio es más fácil ajustarse a las fechas porque el equipo siempre es consciente de cómo avanza el proyecto.
- **Evitar situaciones de “Burnout” entre los colaboradores:** puesto a que tenemos una buena organización y asignación de tareas asumibles y previstas desde el principio, será más difícil que los trabajadores presenten el conocido “Burnout” (desgaste profesional) y a su vez puedan trabajar más motivados al ver cómo han podido realizar cada hito y ser más productivos.

Desventajas:

- **Limitado al flujo de tareas:** si se cuenta con un equipo muy grande es más difícil asignar a cada uno una tarea en concreto, llegando al punto de tener empleados desocupados o sin tareas a realizar.
- **No se integra bien en todos los tipos de proyectos:** la metodología Kanban muchas veces implica sistemas de producción repetitivos, por ello no es recomendable trabajar de esta manera cuando los objetivos no están totalmente definidos por el cliente, debido a que esto puede afectar negativamente al resultado final.
- **Es difícil anticiparse a nuevas demandas o cambios en el desarrollo:** el hecho de tener una tarea tan definida genera también un inconveniente y es que si, por el cliente o por circunstancias externas, es preciso cambiar grandes partes del producto, no existirá previsión alguna y será difícil adaptarse a la nueva situación.

2.2 Ciclo de vida del proyecto

De forma general, las organizaciones suelen definir un ciclo de vida en el proyecto que se esté realizando, aplicándole distintas técnicas y conocimientos dependiendo de en qué fase se encuentre dicho proyecto. Entonces, el ciclo de vida del proyecto está dividido en 5 diferentes fases que son:

- **Inicio:** primera fase del ciclo de vida, se comprueban y se detallan los requisitos con el cliente y con el director de proyectos, quienes identificarán los costes del desarrollo, requerimientos a mayor y a menor nivel y riesgos y problemas asociados al producto. También se empezará a hacer un boceto de un plan de proyecto y a hacer estimaciones del tiempo que se empleará durante el desarrollo.
- **Planificación:** después de haberse asignado un *project manager* y establecidos los equipos de trabajo es el momento para definir detalladamente los requisitos que deba tener el proyecto, la metodología que se va a aplicar, el tiempo que se dispone para el cierre y la asignación de tareas con fechas a cada uno de los diferentes equipos y miembros de este.

En esta fase el director del proyecto creará un plan de comunicación para la fase de ejecución y monitoreo, tales como el “*daily meeting*”, reuniones diarias de no más de 5 minutos donde cada uno de los miembros del grupo notifique del estado sus tareas asignadas.

- **Ejecución:** se lleva a cabo el desarrollo de la fase anterior, se harán constantes pruebas conforme se vayan implementando requisitos y funcionalidades, mientras tanto se comunicarán el estado de estas al *project manager* en las distintas reuniones de control para asegurar la calidad del producto.

Al mismo tiempo de que los desarrolladores y equipos trabajan en la solución o producto en cuestión, el analista de negocio define completamente un plan de monitoreo para la fase de control comprobando así los requerimientos generados durante la fase de planificación.

Una vez la fase de ejecución haya concluido, se dará paso al cierre del proyecto.

- **Monitoreo y control:** fase generalmente paralela a la ejecución donde el director del proyecto hace las comprobaciones que se especificaron en la fase de planificación y ejecución, asegurándose así que el desarrollo del producto avanza favorablemente con respecto a fechas, esfuerzo del equipo y presupuesto dado en el inicio del proyecto.

- **Cierre del proyecto:** fase final del ciclo de vida del proyecto cuando el jefe de proyecto concierta una cita con la entidad cliente y presenta el producto realizado asegurándose que el cliente está satisfecho con el trabajo puesto en producción y concretando facturas y pagos si correspondiesen. De lo contrario, deberá volverse a la fase de ejecución modificando requisitos de la fase de inicio y planificación y volviendo al cierre con la nueva propuesta terminada.

2.3 Fases del diseño del proyecto

Este ciclo de vida del proyecto debe ir acompañado de un modelo de fases que nos permita cómo actuar frente a la propuesta y, dependiendo de cómo sea esta, aportando más o menos flexibilidad.

Existen varios tipos de modelos de desarrollo y según las necesidades de mi proyecto de fin de ciclo he elegido usar el Modelo Iterativo Incremental.

El modelo iterativo incremental se basa en el modelo en cascada con retroalimentación con la diferencia de que estos ciclos se repiten hasta lograr el cierre del producto, de manera que se especifican unos requisitos pequeños y asumibles en la fase de análisis, si esta se cumple, se pasa al diseño y así sucesivamente hasta llegar a las pruebas y de nuevo se definen nuevos requisitos en la nueva iteración, acumulando una serie de iteraciones de este ciclo llegará la finalización del proyecto.

He elegido este modelo debido a que, como no tengo un cliente potencial más allá del tribunal de profesores en la defensa, debo asegurarme que los requerimientos y objetivos de mi aplicación web se llevan a cabo, por ello tengo que tomar el papel del cliente a la vez que el de desarrollador para que las propuestas se cumplan.

Además, este modelo es especialmente conveniente en mi caso debido a que puedo ir comprobando el funcionamiento y haciendo pequeños cambios a la vez que voy desarrollando el código y pasar a la siguiente iteración, añadiendo funcionalidades nuevas o puliendo las que ya estaban.

La mayor desventaja de este modelo y que he estado experimentando es que como realmente puedes hacer un número indeterminado de iteraciones puedes perder el foco de los requisitos funcionales que te marcaste como objetivo, implementando cada vez otros nuevos que no son tan relevantes o es más difícil su adaptación, haciendo así que el bucle nunca finalice y de que no se cumpla con la planificación ya hecha.

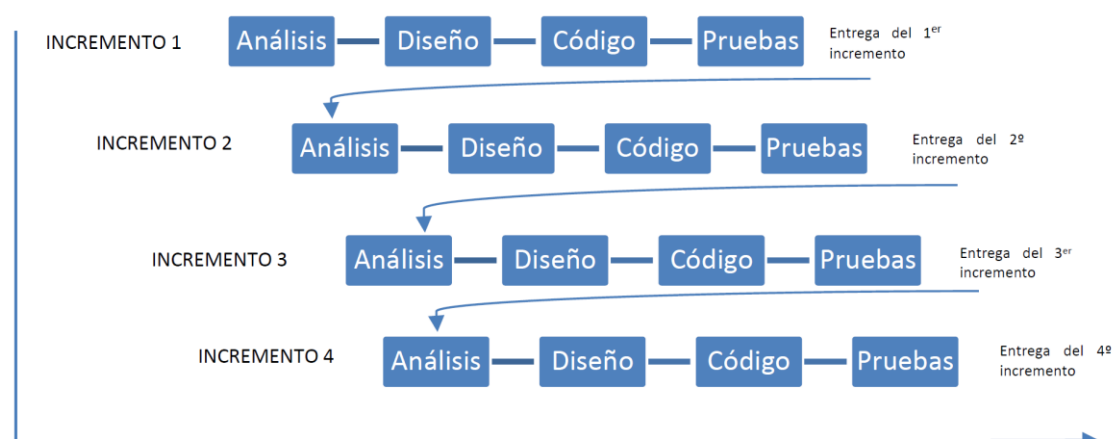


Ilustración 2: Modelo Iterativo Incremental. Fuente: Material Didáctico de Entornos de Desarrollo.

3. Tecnologías y herramientas utilizadas en el proyecto

Las tecnologías y herramientas que se han aplicado a este proyecto fin de ciclo relacionadas con los diferentes módulos técnicos del CFGS de Desarrollo de Aplicaciones Web han sido las siguientes:

❖ **Herramientas y Lenguajes de maquetado:**

- **HTML5:** Html5 (*HyperText Markup Language*) el número 5 indica la versión de este. Es un lenguaje de marcado utilizado globalmente para el desarrollo web que forma la estructura de un sitio o una aplicación, es decir, texto plano estático. Toda su sintaxis está formada por etiquetas “<html>” y soporta integración con css3 y JavaScript
- **CSS3:** Ccss3 (*Cascading Style Sheets*) el número 3 indica la versión de este. Es un lenguaje de diseño gráfico, sirve para dotar documentos html (entre otros) de estilos y presentación visual con animaciones, colores, fuentes de texto entre muchas otras opciones.

* El uso de la sintaxis HTML5 y CSS3 es imprescindible para realizar cualquier página web, en el caso de mi aplicación no es una excepción, el uso de estas dos tecnologías es obligatorio.

❖ **Lenguaje de programación:**

- **PHP8:** php es un lenguaje de programación de propósito general enfocado especialmente a la programación “*Back-end*” de aplicaciones web, ha sido muy criticado porque las primeras versiones contenían fallos de seguridad importantes, sin

embargo, a partir de la versión php4 y sus constantes actualizaciones esto ya no es así. PHP es hoy en día el lenguaje de *Backend* más utilizado hasta la fecha en toda la *World Wide Web*, por ello casi cualquier servidor local o hosting remoto soporta este lenguaje.

* PHP fue una buena elección como lenguaje de programación del lado del servidor debido a su larga pero asequible curva de aprendizaje, con una sintaxis simple, su gran comunidad de programadores y su gran compatibilidad con otras plataformas y tecnología, por esa razón decidí que era el lenguaje de programación más apropiado para mi aplicación.

❖ Entorno de desarrollo integrado (IDE) y Editor de código:

- **Visual Studio Code:** es un editor de código gratuito y de código abierto desarrollado por Microsoft a partir de *Electron*, un *framework* que se utiliza para implementar *Chromium* y *Node.js*. Al ser de código abierto y ampliamente utilizado, desarrolladores independientes crean funciones nuevas a partir de extensiones para la comunidad que usa este IDE.

* Justifico la utilización de VS Code debido a que, aunque al descargarlo no es más que un editor de código, su punto fuerte es la capacidad de instalar infinidad de extensiones, entre ellos el debugger de PHP, convirtiéndose así en un IDE bastante completo, además soporta multitud de lenguajes de marcado y de programación, lo que lo hace ideal para desarrollar todos los diferentes documentos en una misma aplicación de forma unificada.

❖ Sistema Gestor de Base de Datos:

- **MySQL:** Es el sistema de gestión de bases de datos relacional más popular en la web y basado en el lenguaje de consultas estructurado (SQL).
- **PhpMyAdmin:** Es un panel de administración web de bases de datos escrito en PHP con la intención de administrar tablas, ejecutar consultas y demás funcionalidades de MySQL (aunque también soporta otros gestores de bases de datos). Sigue teniendo soporte y una gran comunidad desde 1998.

* Al igual que en PHP MySQL y phpMyAdmin son herramientas del lado del servidor ampliamente utilizadas, en el caso de MySQL su sintaxis podría decirse que es estándar dada su popularidad, esto facilita mucho las cosas buscando posibles errores SQL, ya que como hemos mencionado hay una gran cantidad de usuarios familiarizados con este gestor de bases de datos.

❖ Sistemas operativos:

- **Windows 10 Pro 21H2:** Mundialmente conocido sistema operativo de Microsoft.
- **Ubuntu 20.04 LTS:** Distribución popular de Linux.

* La elección de estos dos sistemas operativos para realizar pruebas de despliegue ha sido debida a que representan los sistemas operativos más usados en ordenadores de escritorio y portátiles en todo el mundo.

❖ Navegadores Web:

- **Google Chrome:** Navegador Web más utilizado entre los usuarios por su gran compatibilidad con los propios servicios de Google. Basado en *Chromium*.
- **Microsoft Edge:** Proyecto que reemplazó a Internet Explorer dirigido por Microsoft y basado en el proyecto de código abierto *Chromium*. Gran compatibilidad con los servicios de Microsoft, implementado nativamente en Windows.
- **Mozilla Firefox (Developer Edition):** navegador web gratuito y de código abierto, está coordinado por la Corporación Mozilla y la Fundación Mozilla. Usa *Gecko* como motor para renderizar páginas web, el cual implementa actuales y futuros estándares web. Esta versión de Mozilla Firefox incluye mejoras para desarrolladores web.

* He necesitado estos tres navegadores diferentes para poder hacer pruebas de *caja negra* y comprobar que todo el aspecto visual de mi aplicación funcionaba correctamente, ya que no todos los navegadores interpretan siempre el código de la misma manera.

❖ Servidores Web (Local):

- **XAMPP:** servidor web centrado en despliegue de aplicaciones web en local, trae todo preparado para iniciar servicios como APACHE, MySQL, FileZilla o Tomcat, además soporta PHP y Pearl.

* La elección de XAMPP es bastante simple y es que al tratarse de un servidor local con soporte tanto para Mac como para Linux y también Windows hace que juegue un papel muy importante en la exportación de mi aplicación web en otros sistemas operativos. Además, viene con PHP, APACHE y MySQL instalado.

❖ Servicios de Google:

- **Herramientas de desarrollador de Google:** Chrome cuenta con diversas herramientas para desarrollo web tales como: inspeccionar elementos, rendimiento (recursos que está usando el navegador para mostrar la web), consola de depuración, vistas responsive...

- **Google Apps:** Google cuenta con una *suit* ofimática en la nube con posibilidad de sincronización en varias plataformas, algunos ejemplos son: Documentos, Hojas de cálculo, Google Calendar o Drive.
- **Google Fonts:** biblioteca de estilos de fuentes gratuito de google.
- **Motor de búsqueda de Google.**

* La mayoría de servicios de Google son imprescindibles a la hora de maquetar, depurar código y tener un entorno de pruebas controlado, además las GAPPS como “Drive” y “Gmail”, permitirán la subida de mi proyecto a la nube para que pueda ser enviado al tribunal del proyecto de fin de ciclo.

❖ Herramientas de control de versiones:

- **Git:** es un sistema de control de versiones local de código abierto el cual permite crear diferentes ramas de trabajo aplicándolo al trabajo simultáneo en equipo de varios desarrolladores y las metodologías ágiles como SCRUM dentro del desarrollo de software.
- **GitHub:** aplicación web desarrollada para git con funcionalidades de red social (seguir usuarios, ver sus últimos repositorios, comentar proyectos ajenos etc...) que te permite operar con repositorios remotos, es usado internacionalmente.

* Estas herramientas son un estándar en el desarrollo web tanto independiente como empresarial, ambas tienen una total compatibilidad y es fácil recuperar *commits*, (especialmente en GitHub ya que está respaldado por la nube de Microsoft) previos en caso de pérdida o una mala implementación de la función *push*.

❖ Librerías:

- **Font Awesome:** *framework* de iconos vectoriales y estilos css.

* La justificación del uso moderado de este framework/librería es poderle dar un estilo algo más intuitivo y profesional a la web cuidando a la vez en la medida de lo posible toda la usabilidad y la visibilidad de la aplicación para que ningún usuario se sienta confuso de qué tipo de formulario está rellenando.

4. Estimación de recursos y planificación

Existen diferentes métodos para ayudarnos a gestionar nuestros proyectos, esto es importante debido a que existen fechas de entrega y plazos límite a entregar el producto, pues es de vital importancia organizarnos para evitar situaciones inesperadas, agobios,

burnout o asignaciones de tareas desproporcionalmente extensas a pocos días del cierre del proyecto.

Para evitar estas situaciones he decidido implementar un diagrama de Gantt para seguir dicha programación a lo largo de las semanas.

Un diagrama de gantt es una herramienta de gestión de proyectos ideada por Henry Gantt a principios del siglo XX, por aquél entonces se realizaba en papel, pero con la consistente evolución de las tecnologías de la información cada vez los diagramas de Gantt se han vuelto cada vez más complejos y más usados no solo en el desarrollo de software sino en otras industrias muy dispares. Actualmente las herramientas de diagramas de Gantt se les conoce como "herramientas de hoja de ruta". Esta metodología de planificación se caracteriza por estar orientada a modelos de diseño en cascada iterativa y se le considera una metodología ágil, esto es especialmente importante ya que concuerda casi perfectamente con los procedimientos de análisis que estoy implementando en el proyecto.

Un diagrama Gantt consta de una estructura básica, a uno de los lados de la tabla tendremos las filas que indiquen la estimación y el tiempo real que se está consumiendo a lo largo de las semanas mientras que las columnas centrales se especifica el tramo temporal que vamos a medir (días, semanas, meses...).

Veamos un par de ejemplos con mis propios diagramas Gantt que he realizado para el desarrollo:

Diagrama de Gantt - Programación PFC

Mes de Abril

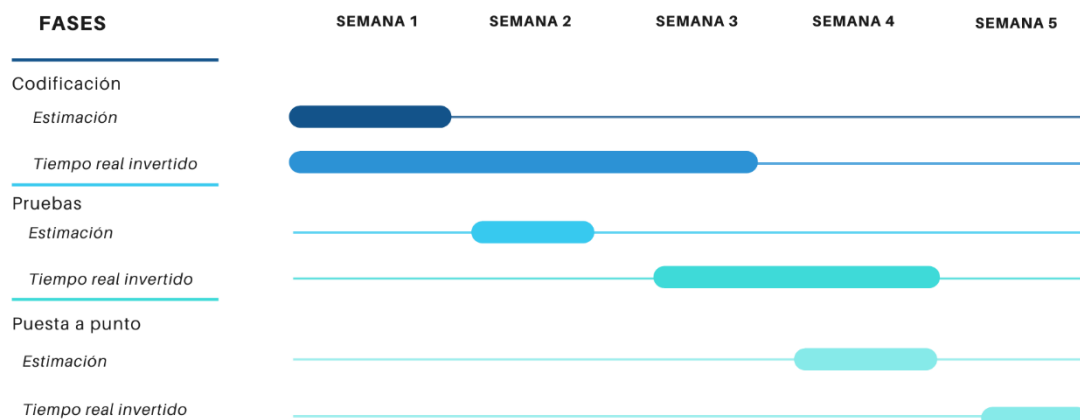


Diagrama de Gantt - Programación PFC

Mes de Marzo

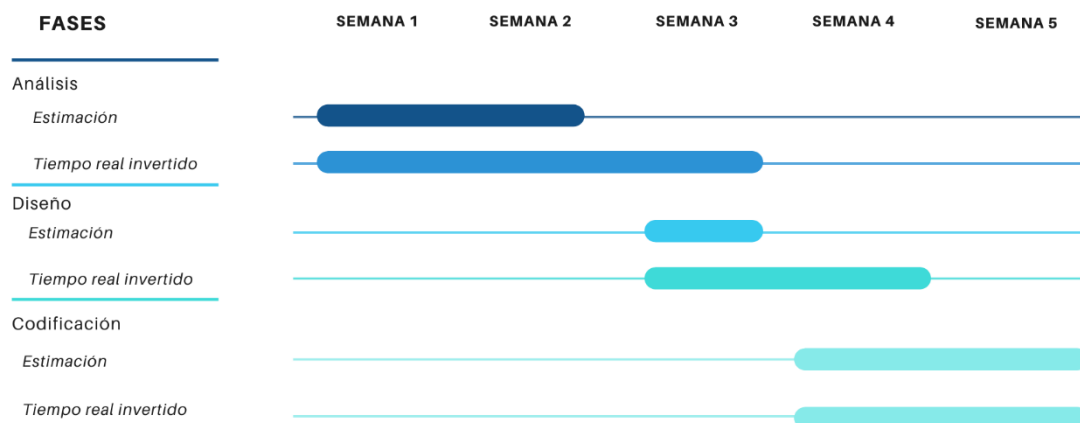


Ilustración 3, 4: Representación diagrama de Gantt. Fuente: elaboración propia en “<https://www.canva.com/>”.

Como se puede apreciar en el diagrama las fases e hitos del desarrollo han sido:

❖ Análisis:

- Estimación: Semana 1 de Marzo – Semana 2 de Marzo (2 semanas).
- Tiempo real invertido: Semana 1 de Marzo – Semana 3 de Marzo (3 semanas).

Aun habiendo planteado y presentado el tema con sus objetivos y herramientas además de la justificación, a la hora de plantearlo todo de la manera más estructural y detallada posible me di cuenta de que había hecho una estimación excesivamente optimista. Teniendo que redactar los primeros puntos de la memoria incluyendo los análisis y diagramas de casos de uso y sumándole unas *FCT* en las que había hecho una previsión de jornada bastante más pequeña de las que han sido en realidad (9 – 12h) por motivos de entregas en la empresa, lo cual conlleva también adquirir nuevos conocimientos y tecnologías antes no vistas. Esto causó un recorte importante de funcionalidades aún más complejas que se dieron en la primera propuesta del proyecto tomando como objetivos aquellos campos más importantes, que no por ello fáciles ni para nada sencillos de implementar y desarrollar y moviendo a vías futuras esas primeras funcionalidades escritas extremadamente ambiciosas. El terminar la fase de análisis conllevó un desfase en la planificación de una semana y unos días que se estaría constantemente acarreando durante las siguientes fases.

❖ **Diseño:**

- Estimación: Semana 3 de Marzo (1 semana).
- Tiempo real invertido: Semana 3 de Marzo – Mitad Semana 4 de Marzo (1 semana y media).

Aunque llevase un atraso de una semana durante esta fase no fue difícil intentar remontar algunos días realizando un poco más de esfuerzo y de tiempo a la elaboración de *mokups* y planear la estructura de carpetas y archivos con el modelo vista-controlador, pero seguía teniendo un retraso de unos 4 días.

❖ **Codificación:**

- Estimación: Semana 4 de Marzo – Semana 1 de abril (3 semanas).
- Tiempo real invertido: Semana 4 de Marzo – Semana 3 de abril (5 semanas).

Sin duda la fase del proyecto que más esfuerzo y tiempo ha conllevado. La planificación fue de 4 semanas ya que se esperaban imprevistos y errores que depurar en el código, sumando la implementación de funciones realmente complejas de desarrollar y modularizar para que se adapten al modelo vista-controlador. Gracias a incansables búsquedas en la documentación de PHP y de consultas por Google y diversos foros de desarrollo web fue posible terminar un código limpio, estructurado y funcional.

❖ **Pruebas:**

- Estimación: Semana 2 de Abril (1 semana).
- Tiempo real invertido: Semana 3 de Abril – Semana 4 de Abril (2 semanas).

Esta fase tuvo que tener un desarrollo parcialmente paralelo con el de codificación para hacer posible que la fase de codificación fuese plenamente satisfactoria. Se necesitaron muchas pruebas de caja negra para comprobar que el código escrito era totalmente funcional y exento de errores aplicándole diferentes restricciones y reglas de uso para los roles de la aplicación.

❖ **Puesta a punto:**

- Estimación: Semana 4 de Abril (1 semana).
- Tiempo real invertido: Semana 5 de Abril y primeros de Mayo (1 semana y media).

Mientras se estaban terminando de realizar todas las comprobaciones en la fase de pruebas se empezó también a arreglar y pulir ciertos aspectos estructurales y visuales tanto del código como la memoria y cierta parte de la presentación con diapositivas. En esta fase de cierre también se le dedicó a la grabación de la única toma que se usó para la videodefensa.

Se ha llegado al plazo de entrega en fechas ajustadas debido a un análisis muy exigente y una estimación realmente optimista, pero finalmente ha desembocado en un buen resultado de calidad.

5. Análisis del proyecto

5.1 Requisitos

Antes de proceder a crear los distintos diagramas que necesitaremos seguir para entender y aplicar los requisitos funcionales y no funcionales de la aplicación, tendremos precisamente que detallar estos de manera clínica para poder diseñar y aplicar estos diagramas de una manera fidedigna y segura para el desarrollo de la aplicación.

Entonces, pasamos a analizar los requisitos y sus tipos:

❖ **Requisitos funcionales:**

- **Roles de usuario:** existirán diferentes permisos asignados a 3 tipos de roles (Visitante, Usuario y Usuario Administrador)
- **Formulario de Registro:** Los visitantes con intención de crearse una cuenta para interactuar con el blog más a profundidad deberán de tener la opción habilitada y en funcionamiento de realizar un registro al sitio con un email, nombre de usuario y contraseña.
- **Formulario de Inicio de Sesión:** Al igual que con los registros se deberá disponer de un formulario para validar credenciales de los usuarios que se enviarán a la base de datos y permitirán o no la entrada de dicha solicitud.

- **Navegar entre Categorías:** para que los usuarios puedan navegar dentro del blog tendremos el requisito fundamental del proyecto, que será contar con una barra de navegación por el cuál moverse dentro de las distintas opciones que nos ofrece este sitio web.
 - **Ver la página de la Entrada:** además todos los usuarios deberán poder ver la página del post en cuestión que se solicite.
 - **Gestionar Entradas:** los usuarios registrados deben de poder gestionar (crear, eliminar o editar) sólo las propias entradas que ellos mismos realicen, sin embargo, los usuarios administradores del blog cumplirán una función de moderación pudiendo hacer esto con todos los posts de los usuarios, incluyéndose ellos mismos con el motivo de controlar las entradas nuevas que se generan por otros usuarios.
 - **Gestionar Categorías:** los usuarios administradores tendrán la responsabilidad de crear temas para las entradas, es decir, categorías, y gestionarlas (crear, eliminar o editar), limitando así a los usuarios a que deban de elegir un tema de la entrada que estén creando y organizando los posts que se suben evitando categorías no relevantes, inapropiadas o de bajo interés en la comunidad.
 - **Apartado de Modificación de Perfil:** los usuarios administradores y usuarios normales, aparte de todo esto, también tendrán la posibilidad de cambiar algunos de sus datos relacionados con el registro, como por ejemplo el email y el nombre de usuario y que esto se vea reflejado en la correspondiente tabla de la base de datos.
- ❖ Requisitos no funcionales:
- **Visualización:** toda la parte visual (*FrontEnd*) tendrá que ser soportada por la mayoría de navegadores y resoluciones de monitor de *1080p* a *1440p* en ventana de navegador web.
 - **Usabilidad:** con este apartado necesitamos que el proyecto a los ojos y manos de usuarios de a pie sea totalmente intuitivo y de fácil comprensión para que puedan disfrutar el producto de la mejor manera posible sin que haya momentos en los que no entiendan el funcionamiento del aplicativo o se sientan confusos.
 - **Eficiencia:** el blog deberá responder rápidamente, generando las consultas a la base de datos de manera síncrona y mostrando las vistas sin retardo para que el usuario no abandone el sitio o recargue la página.
 - **Mantenibilidad:** este apartado de requisito no funcional estará protagonizado por la implementación de una estructura *MVC* (Modelo Vista-Controlador) que nos permitirá hacer el proyecto mucho más modular y mantenible, puesto que todas las funciones clases, vistas o modelos serán independientes unas de otras.

- **Compatibilidad:** el despliegue y funcionamiento de esta aplicación web tendrá que requerir el uso de un servidor web como XAMPP para asegurar la compatibilidad entre sistemas operativos Windows, MacOS y distribuciones Linux.
- **Seguridad lógica y de datos:** La información contenida de carácter sensible de todos los usuarios, incluyendo los administradores y visitantes, deberá estar protegida frente a ataques externos, por ello uno de los requisitos con los que debe lidiar este apartado es el cifrado de contraseñas en la base de datos, así, ni si quiera los propios administradores de la base de datos podrán ver las contraseñas de los usuarios registrados. Para lograr esto también tendremos que protegernos frente a inyecciones SQL en todos nuestros formularios.

5.2 Diagramas E/R, Casos de uso y Clases

Después de haber quedado claros los requisitos que debemos implementar en el proyecto es hora de realizar distintos diagramas para comprender aún mejor las necesidades del proyecto y tener una referencia a la hora de implementar todos los requisitos, relaciones y casos de uso de nuestro aplicativo.

5.2.1 Diagrama de Entidad-Relación

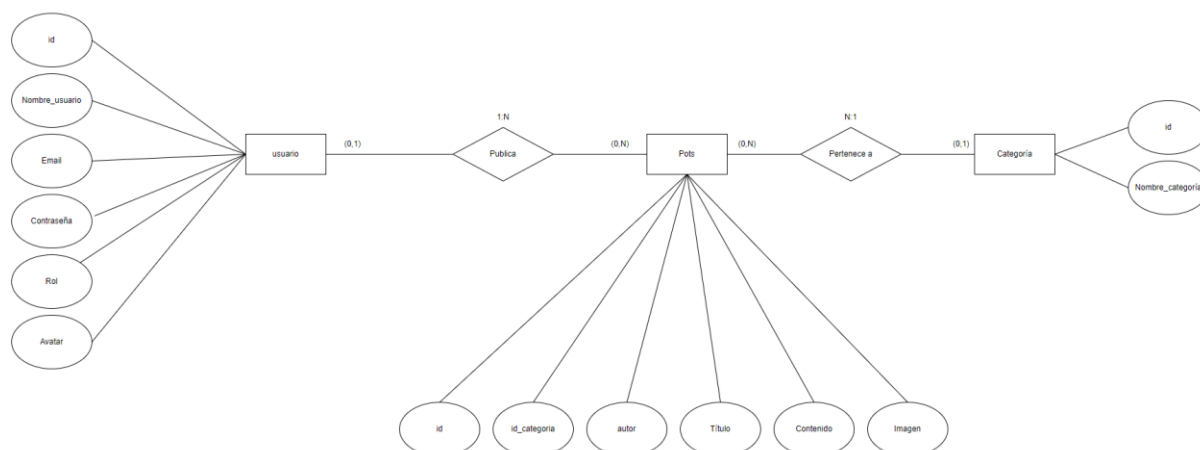


Ilustración 5: Representación diagrama de E/R. Fuente: elaboración propia en “<https://app.diagrams.net/>”

Como bien muestra el diagrama de Entidad-Relación tenemos como resultado tres entidades distintas relacionadas entre sí: Usuario, Posts y Categoría.

“Usuario” contiene los atributos necesarios para poder identificarse y firmar los Posts debido a que es necesario que un usuario los publique, mantiene una relación (“Publica”) entre Posts, la cual se define como de uno a muchos y el motivo es que un usuario puede publicar cuantos posts quiera; sin embargo, un post siempre será escrito por un único usuario a la vez. Por otro lado, tenemos la entidad “Post” que almacena los atributos que necesita para que “Categoría” los filtre, estos mantienen una relación N:1 debido a que un post sólo puede estar en una categoría a la vez, pero una categoría puede contener a varios posts dentro de ella.

Para finalizar con la explicación remarcaremos que “Categoría” solo necesita dos atributos (*id* y *nombre_categoria*) para gestionar los posts que les sean asignados a cada una de ellas.

5.2.2 Diagrama de Casos de uso

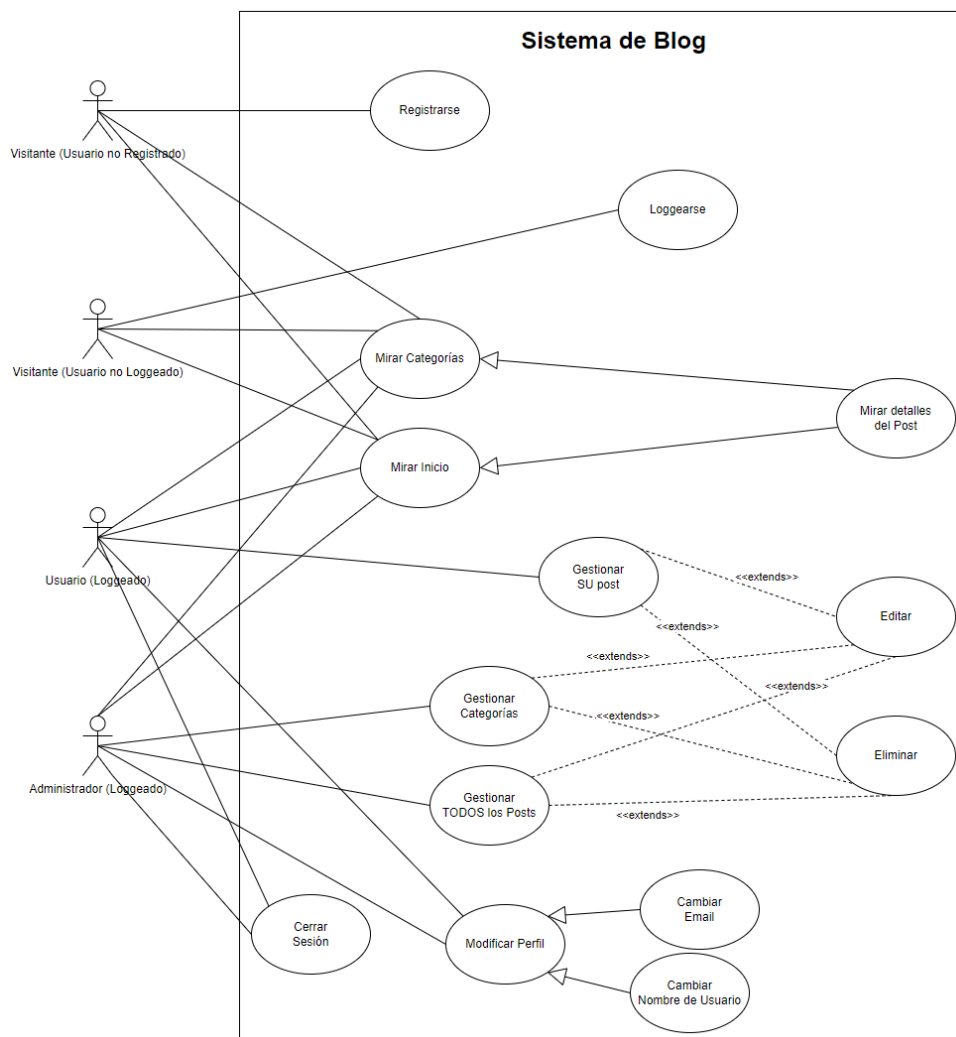


Ilustración 6: Representación diagrama de Casos de uso. Fuente: elaboración propia en “<https://app.diagrams.net/>”

Para entender mejor este diagrama deberemos de apoyarnos de algunas tablas inspiradas en el Modelo basado en el escenario:

Identificador de Caso de Uso	1
Nombre del caso de uso	Registrarse.
Actores Implicados	Visitante (Usuario no Registrado).
Precondición	El usuario no debe estar registrado, por tanto, tampoco loggeado.
Curso normal	El usuario rellena los campos para el registro.
Poscondiciones	El registro se realiza correctamente y se guarda la cuenta en la base de datos.
Alternativas	Realiza mal el registro con lo cual será redirigido al Index y deberá volver a rellenar el formulario.

Identificador de Caso de Uso	2
Nombre del caso de uso	Loggearse.
Actores Implicados	Visitante (Usuario no Loggeado).
Precondición	Debe ser un usuario con una cuenta creada y guardada en la base de datos del servidor.
Curso normal	El usuario rellena los campos para el inicio de sesión.
Poscondiciones	El usuario es capaz de visualizar en la barra lateral los paneles de gestión indicando que el inicio de sesión ha sido un éxito.
Alternativas	Realiza mal el inicio de sesión con lo cual será redirigido al Index y deberá volver a rellenar el formulario.

Identificador de Caso de Uso	3
Nombre del caso de uso	Cerrar Sesión.
Actores Implicados	Usuario (Loggeado) Administrador (Loggeado).
Precondición	Debe ser un usuario con una cuenta creada y guardada en la base de datos del servidor.
Curso normal	Hace click en el botón de cerrar la sesión.
Poscondiciones	Se le redirige al Index, su sesión se cierra y no puede ver los paneles de administración.

Alternativas	no hay alternativas posibles.
---------------------	-------------------------------

Identificador de Caso de Uso	4
Nombre del caso de uso	Mirar Inicio.
Actores Implicados	Visitante (Usuario no Registrado), Visitante (Usuario no Loggeado), Usuario (Loggeado), Administrador (Loggeado).
Precondición	Ingresa a la página Index.
Curso normal	El usuario accede a través de su ordenador a la URL.
Poscondiciones	Es redirigido automáticamente al Index de la comunidad.
Alternativas	no hay alternativas posibles.

Identificador de Caso de Uso	5
Nombre del caso de uso	Mirar Categorías.
Actores Implicados	Visitante (Usuario no Registrado), Visitante (Usuario no Loggeado), Usuario (Loggeado) Administrador (Loggeado).
Precondición	El usuario accede a través de su ordenador a la URL.
Curso normal	Interactúa con la barra de navegación superior.
Poscondiciones	Es capaz de mirar el listado de posts por la categoría seleccionada.
Alternativas	no hay alternativas posibles.

Identificador de Caso de Uso	6
Nombre del caso de uso	Mirar detalles del Post.
Actores Implicados	Visitante (Usuario no Registrado), Visitante (Usuario no Loggeado), Usuario (Loggeado) Administrador (Loggeado).
Precondición	Ha accedido a una sección por categorías.
Curso normal	Hace click en el botón “Leer Más” del post.
Poscondiciones	Es redirigido a la página del post que haya elegido.
Alternativas	no hay alternativas posibles.

Identificador de Caso de Uso	7
Nombre del caso de uso	Gestionar SU post.
Actores Implicados	Usuario (Loggeado).
Precondición	Estar registrado y acceder al Index.
Curso normal	El usuario clica en el panel de gestión de posts.
Poscondiciones	Es redirigido y se le muestra el panel de administración de sus posts.
Alternativas	No cumple con todos los datos cumplimentados o intenta subir una imagen de con un tipo de archivo no soportado.

Identificador de Caso de Uso	8
Nombre del caso de uso	Gestionar Categorías.
Actores Implicados	Administrador (Loggeado).
Precondición	Tener el rol de usuario administrador y acceder a la comunidad.
Curso normal	El administrador clica en el panel de gestión de categorías.
Poscondiciones	Es redirigido y se le muestra el panel de administración de categorías.
Alternativas	No cumple con todos los datos cumplimentados.

Identificador de Caso de Uso	9
Nombre del caso de uso	Gestionar TODOS los Posts.
Actores Implicados	Administrador (Loggeado).
Precondición	Tener el rol de usuario administrador y acceder a la comunidad.
Curso normal	El administrador clica en el panel de gestión de posts.
Poscondiciones	Es redirigido y se le muestra el panel de administración de todos los posts.
Alternativas	No cumple con todos los datos cumplimentados.

Identificador de Caso de Uso	10
Nombre del caso de uso	Editar.
Actores Implicados	Usuario (Loggeado), Administrador (Loggeado).
Precondición	Haber iniciado sesión y accedido a la web.
Curso normal	En el panel de Categorías o de Posts se hace clic en el botón de Edición.
Poscondiciones	Se actualiza la edición en la base de datos y es mostrada en la aplicación.
Alternativas	No cumple con todos los datos cumplimentados o intenta subir una imagen de con un tipo de archivo no soportado.

Identificador de Caso de Uso	11
Nombre del caso de uso	Eliminar.
Actores Implicados	Usuario (Loggeado), Administrador (Loggeado).
Precondición	Haber iniciado sesión y accedido a la web.
Curso normal	En el panel de Categorías o de Posts se hace clic en el botón de Eliminar.
Poscondiciones	Se elimina la fila en la base de datos y la información ya no es mostrada en la aplicación.
Alternativas	La categoría que se quiere eliminar aún contiene posts anidados y devuelve error.

Identificador de Caso de Uso	12
Nombre del caso de uso	Modificar Perfil.
Actores Implicados	Usuario (Loggeado), Administrador (Loggeado).
Precondición	Haber iniciado sesión y accedido a la web.
Curso normal	El usuario hace clic en el botón de “Mi Perfil”.
Poscondiciones	Es redirigido al formulario de cambio de nombre de usuario y correo electrónico.
Alternativas	No hay alternativas posibles.

Identificador de Caso de Uso	13
-------------------------------------	----

Nombre del caso de uso	Cambiar Email.
Actores Implicados	Usuario (Loggeado), Administrador (Loggeado).
Precondición	Haber iniciado sesión y accedido a la web.
Curso normal	Se rellena el formulario correspondiente al email.
Poscondiciones	Se cierra la sesión para actualizar los datos de la página y hace la modificación en la base de datos.
Alternativas	El email que se ha introducido ya está en uso, no se aplican cambios en la base de datos.

Identificador de Caso de Uso	14
Nombre del caso de uso	Cambiar Nombre de Usuario.
Actores Implicados	Usuario (Loggeado), Administrador (Loggeado).
Precondición	Haber iniciado sesión y accedido a la web.
Curso normal	Se rellena el formulario correspondiente al nombre de usuario.
Poscondiciones	Se cierra la sesión para actualizar los datos de la página y hace la modificación en la base de datos.
Alternativas	El nombre de usuario que se ha introducido ya está en uso, no se aplican cambios en la base de datos.

5.2.3 Diagrama de Clases

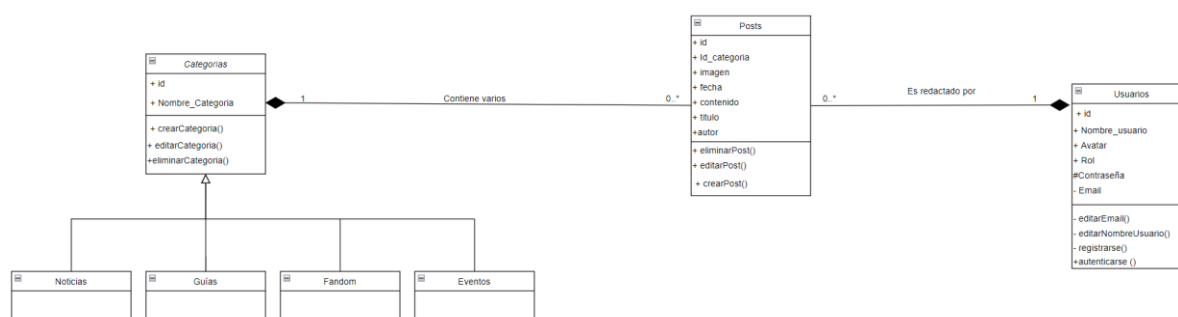


Ilustración 7: Representación diagrama de Casos de uso. Fuente: elaboración propia en “<https://app.diagrams.net/>”

Para ayudar a comprender este diagrama realizaremos una conveniente explicación.

Partimos de tres superclases: *Categorías*, *Posts* y *Usuarios*, igual que en el diagrama de entidad-relación solo que en este diagrama lo haremos desde un enfoque funcional de atributos y métodos y clases hipotéticas que deberían agrupar estos datos.

Definiendo primero la superclase *Categorías*, y como ya sabemos, esta es la encargada de organizar los diferentes posts de la aplicación. *Categorías* debería ser capaz de crear nuevas categorías (clases hijas) *Noticias*, *Guías*, *Fandom* y *Eventos*. Todas estas adquieren su información heredada de su superclase padre *Categorías* compartiendo atributos y métodos, por lo cual como *Categorías* es capaz de eliminar y editar cada una de ellas, estas también así mismas, pudiéndose transformar en otra categoría distinta o eliminándose. Además, *Categorías* es capaz de crear nuevas categorías hijas.

Posts vuelve a ser el nexo común de las relaciones con las demás clases, en ella como sabemos se encuentran todos los atributos necesarios para identificarse en clases ajenas a su paquete, y esta mantiene relaciones de composición tanto con categorías como con *Usuarios*, esto ocurre así porque *Posts* depende de estas dos clases para existir, si no existe una categoría que no la recoja para filtrarla el sistema no nos dejará crear la publicación, por otro lado, si no hay ningún usuario logeado que quiera escribir una nueva entrada, la clase *post* tampoco tendría cabida. De nuevo, el sistema no permitirá bajo ningún caso que un usuario no identificado pueda llegar a crear un *post*.

Aunque la clase *Posts* contenga grandes dependencias conserva atributos muy importantes para el funcionamiento básico de la aplicación agregando también que la clase *Posts* contiene los métodos necesarios para la gestión de estos (*eliminarPost()*, *crearPost()* y *editarPosts()*).

Por último, llegamos a la superclase *Usuarios* encargada de la gestión de estos y creando nuevas cuentas que hereden el funcionamiento de la clase padre. Esta clase es imprescindible para el funcionamiento del aplicativo debido a que gran mayoría de las operaciones que podamos hacer en el sitio deben estar validadas e identificadas por un único usuario, de otra manera sólo se podría leer información traída por consultas SQL.

La clase *usuario* es responsable de 3 métodos específicos privados (*editarEmail()*, *editarNombreUsuario()* y *registrarse()*), un atributo (*Email*) y otro protegido (*Contraseña*); los métodos deben de ser privados para garantizar la seguridad del sistema, ya que estos no son necesarios por ninguna otra clase en el diagrama UML, con respecto a los métodos es

prácticamente la misma premisa diferenciándose en que el atributo de Contraseña es realmente sensible a ataques y debe estar protegido en todo momento (#).

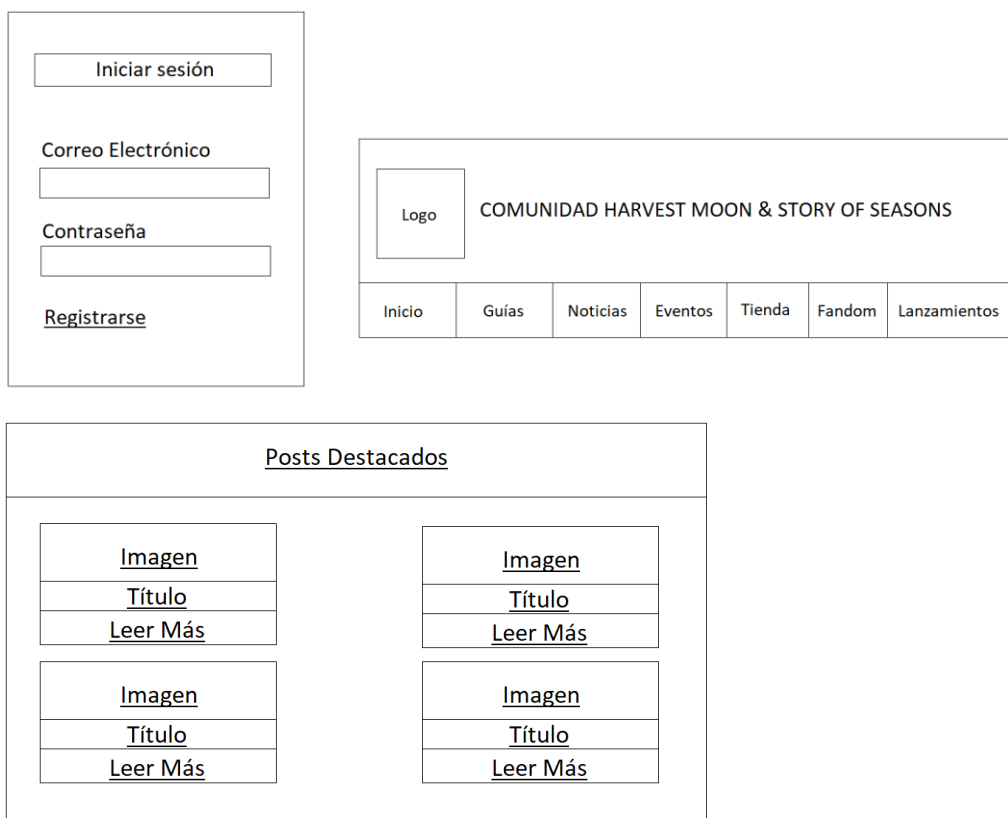
6. Diseño del proyecto

En este apartado veremos algunos bocetos realizados previamente a aplicarle estilos finales a las diferentes vistas de nuestra aplicación web, por otra parte, se explicará todo el funcionamiento de la aplicación además de cómo se ha implementado y las diferentes utilidades que han sido necesarias desarrollar.

6.1 Mockup y Bocetos de vistas

Para tener un diseño base al que seguir y por el que guiarme, he tenido que diseñar algunos esbozos de aquellas vistas más significativas del *Frontend* del aplicativo.

A continuación, se mostrarán algunos de estos *Mockup's* diseñados para este propósito:



Iniciar sesión

Correo Electrónico

Contraseña

Registrarse

Logo

COMUNIDAD HARVEST MOON & STORY OF SEASONS

Inicio

Guías

Noticias

Eventos

Tienda

Fandom

Lanzamientos

Posts Destacados

Imagen

Título

Leer Más

Imagen

Título

Leer Más

Imagen

Título

Leer Más

Imagen

Título

Leer Más

Ilustración 8,9,10: Representación Mockups. Fuente: elaboración propia con la aplicación de "Paint".

6.2 Realización del proyecto

Pasaremos ahora a explicar detalladamente aquellos aspectos técnicos que han jugado un papel fundamental en el desarrollo del proyecto.

6.2.1 Estructura de carpetas

Empezaremos con una introducción de la organización de archivos y carpetas que se ha seguido y que, además, ha sido necesaria para posteriormente aplicar el modelo vista-controlador de manera eficiente.

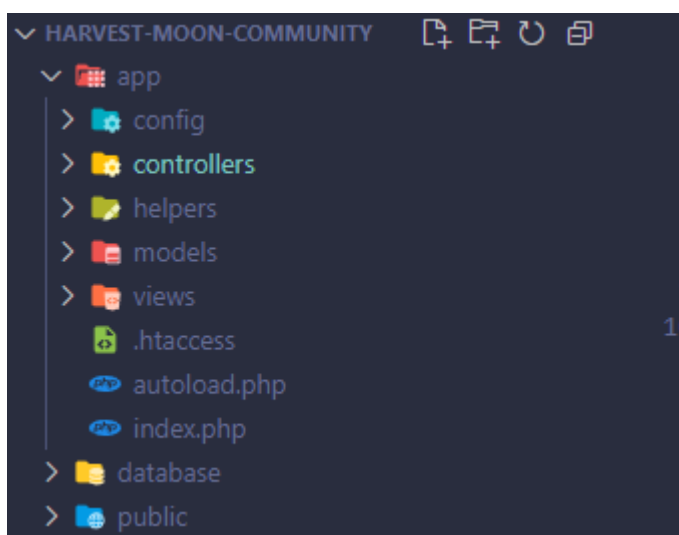


Ilustración 11: Estructura de carpetas. Fuente: elaboración propia.

Como podemos apreciar en la captura de pantalla proporcionada, la aplicación se divide en las siguientes carpetas y subcarpetas:

- **Config:** en ella encontraremos los ficheros .php relacionados a aspectos de configuración referentes a la base de datos y cómo esta interactúa y muestra información en la aplicación web.
- **Controllers:** esta carpeta almacena los distintos controladores (.php) necesarios para realizar todas las funcionalidades de la aplicación, en ella podremos ver los controladores de Categoría, Posts y Usuarios además de otro auxiliar para mostrar errores de navegación.
- **Helpers:** contiene un único fichero .php en el que se definen pequeñas funciones globales que van a ser necesarias en algunas de las otras estructuras de código.
- **Models:** en ella tenemos cada uno de los ficheros que representan una fila dentro de nuestra estructura de bases de datos tales como categoría, usuario y post.

- **Uploads:** carpeta con la funcionalidad de guardar imágenes subidas por los usuarios y así evitando congestionar y saturar a las correspondientes tablas en la base de datos.
- **Views:** aquí veremos todas las diferentes visualizaciones del blog en sintaxis HTML5 y PHP organizada en las siguientes sub-carpetas: category, posts, template y user.

Acabamos de dar un repaso a las carpetas contenidas dentro de “app” que se encuentra en el directorio raíz (*Harvest-Moon-Community*). Además de la carpeta app, en el directorio raíz se encuentran las carpetas database y public que serán explicadas al final del punto 6.2.

6.2.2 SGBD, Tablas y Administración

Antes de pasar a la explicación en profundidad de todos los archivos, carpetas y código es necesario tener una idea general de la composición de la base de datos, cómo se ha administrado, el número de tablas que tiene y cómo están relacionadas unas con otras para entender mejor todo este programa complejo.

El Sistema Gestor de Bases de Datos que se ha utilizado es MySQL, por ello la sintaxis SQL ha jugado un papel imprescindible a la hora de crear y programar consultas a la BBDD.

```
CREATE DATABASE harvestmoon;
USE harvestmoon;

CREATE TABLE users(
  id          int(255) auto_increment not null,
  username    varchar(100) not null,
  email       varchar(255) not null,
  password    varchar(255) not null,
  role        varchar(20),
  avatar      varchar(255),
  CONSTRAINT pk_users PRIMARY KEY(id),
  CONSTRAINT uq_email UNIQUE(email),
  CONSTRAINT uq_username UNIQUE(username)
)ENGINE=InnoDB;

INSERT INTO users VALUES(NULL, 'Admin', 'admin@admin.com', '$2y$04$qcEAcK8mA96mShZxsUPm6.T5i4UhkIkNWS2/GSv9vg5q277jqdq6.', 'admin', null);

CREATE TABLE categories(
  id          int(255) auto_increment not null,
  category_name varchar(100) not null,
  CONSTRAINT pk_categories PRIMARY KEY(id),
  CONSTRAINT uq_category_name UNIQUE(category_name)
)ENGINE=InnoDB;

INSERT INTO categories VALUES(null, 'Noticias');
INSERT INTO categories VALUES(null, 'Guías');
INSERT INTO categories VALUES(null, 'Fandom');
INSERT INTO categories VALUES(null, 'Eventos');

CREATE TABLE posts(
  id          int(255) auto_increment not null,
  category_id int(255) not null,
  author      int(255) not null,
  title       varchar(100) not null,
  content     text,
  date        date not null,
  picture     varchar(255),
  CONSTRAINT pk_posts PRIMARY KEY(id),
  CONSTRAINT fk_post_category FOREIGN KEY(category_id) REFERENCES categories(id),
  CONSTRAINT fk_post_user FOREIGN KEY(author) REFERENCES users(id)
)ENGINE=InnoDB;
```

Ilustración 12: Código SQL. Fuente: elaboración propia.

Para explicar mejor las tablas y sus relaciones vamos a apoyarnos de un diagrama de modelo relacional

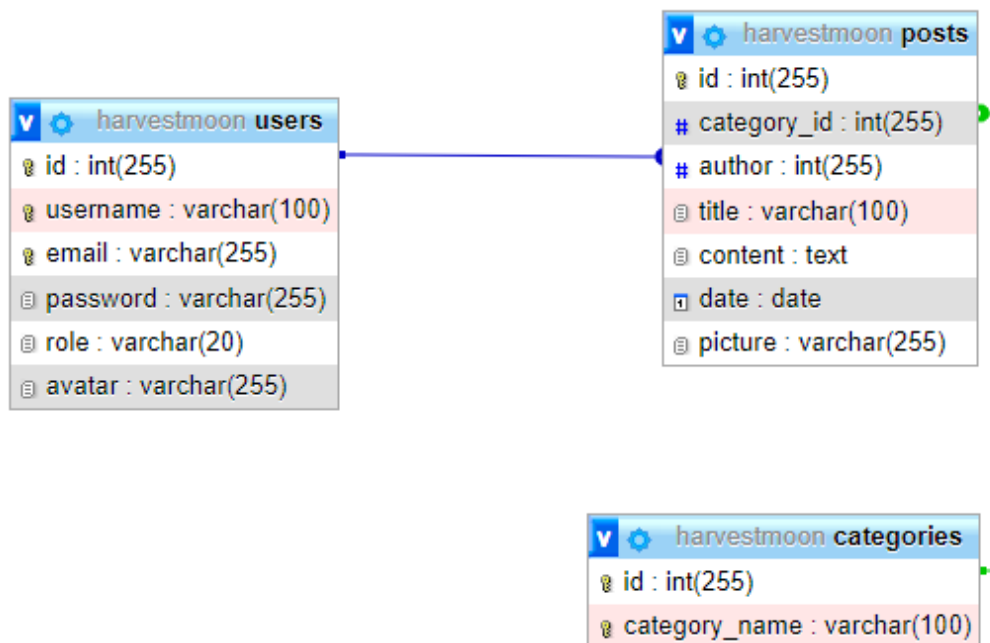


Ilustración 13: Estructura de carpetas. Fuente: elaboración propia en phpMyAdmin.

Como se puede observar nuestra base de datos consta de tres tablas conectadas entre sí:

- ❖ **Users:** relacionada con posts y dispone de las siguientes columnas:
 - (PK) Id: int (255)
 - (FK) Username: varchar(100)
 - (FK) Email: varchar(255)
 - Password: varchar(255)
 - Role: varchar(20)
 - Avatar: varchar(255)
- ❖ **Posts:** relacionada con users y dispone de las siguientes columnas:
 - (PK) Id: int (255)
 - (FK) Category_id: int (255)

- (FK) Author: int (255)
- Title: varchar(100)
- Content: text
- Date: date
- Picture: varchar(255)
- ❖ **Categories:** relacionada con posts y dispone de las siguientes columnas:
 - (PK) Id: int(255)
 - (FK) Category_name: varchar(100) (FK)

Por último, y acabando con este subapartado, simplemente mencionar que se ha utilizado un panel de administración web de bases de datos *phpMyAdmin*, esto ha sido especialmente conveniente ya que nos permite hacer multitud de operaciones bajo una interfaz gráfica, tales como: insertar datos, modificar tablas, cambiar relaciones, o incluso exportar la base de datos en archivo .sql.

6.2.3 Modelo Vista-Controlador

Una vez explicada la estructura de carpetas enfocada a aplicar el MVC en el punto anterior, en este indagaremos y explicaremos los ficheros y los fragmentos de código más relevantes en este proyecto.

6.2.3.1 Modelos

Como ya se explicó en el apartado de estructura de carpetas, los modelos representan filas de las bases de datos y cómo se comunican y hacen consultas organizadas a partir de estas.

Veremos los siguientes archivos y alguna de sus funciones más relevantes.

- ❖ **Category.php:** representa las consultas que debe hacer una categoría frente a la base de datos para que su controlador correspondiente pueda operar con él.


```

public function getAll()
{
    $categorias = $this->db->query("SELECT * FROM categories ORDER BY id ASC;");
    return $categorias;
}

public function getOne()
{
    $categoria = $this->db->query("SELECT * FROM categories WHERE id={$this->getId()}");
    return $categoria->fetch_object();
}

public function save()
{
    $sql = "INSERT INTO categories VALUES(NULL, '{$this->getCategory_name()}')";
    $save = $this->db->query($sql);

    $result = false;
    if ($save) {
        $result = true;
    }
    return $result;
}

```

Ilustración 14: Funciones category.php. Fuente: elaboración propia.

- Crea la clase Category con sus variables para más tarde ser usada como objeto en el controlador CategoryController.php.
 - Getters y Setters: métodos de acceso que se usarán para asignar (setters) y recuperar (Getters) valores de los atributos o campos de la tabla “categories” para poder usarlos en los correspondientes controladores.
 - getAll(): recoge todos los campos de la tabla “categories” y los devuelve a la variable \$categorias.
 - getOneById(): selecciona todos los campos de la tabla “categories” donde el campo id será atributo de este método y el resultado será almacenado en la variable \$post.
 - save(): guarda en la tabla categories los dos tipos de datos (id y category_name). Si la operación se ha completado correctamente guarda el resultado.
 - update(): actualiza la tabla “categories” e inserta en la columna category_name el resultado de su correspondiente getter relacionado al id de la categoría.
 - delete(): elimina de la tabla “categories” todos los campos del atributo id asociado y devuelve el resultado.
- ❖ **Post.php:** representa las consultas que debe hacer un post frente a la base de datos para que su controlador correspondiente pueda operar con él.

```
public function getPostsById($uid)
{
    $sql = "SELECT * FROM posts WHERE `author` = {$uid} ORDER BY id DESC";
    $posts = $this->db->query($sql);
    return $posts;
}

public function getIndex($limit)
{
    $post = $this->db->query("SELECT * FROM posts ORDER BY RAND() LIMIT $limit");
    return $post;
}

public function getOneById($id)
{
    $sql = "SELECT * FROM posts WHERE id = {$id}";
    $post = $this->db->query($sql);
    return $post;
}
```

Ilustración 15: Funciones post.php. Fuente: elaboración propia.

- Crea la clase Post con sus variables para más tarde ser usada como objeto en el controlador PostsController.php.
- Getters y Setters: métodos de acceso que se usarán para asignar (setters) y recuperar (Getters) valores de los atributos o campos de la tabla “posts” para poder usarlos en los correspondientes controladores.
- getPostsById(): selecciona todos los campos de la tabla “posts” y establece la relación de atributos con el que el autor estará relacionado con el campo id de la tabla “users” en la variable \$uid ordenados de forma descendente para después mostrarse en el panel de gestión de posts.
- getIndex(): recupera todos los campos de la tabla “posts” pero añadiéndole un atributo \$limit y generando un resultado aleatorio de esa consulta, para después mostrarse en la sección “Inicio”.
- getOneById(): recupera todos los valores de la tabla “posts” asignados al correspondiente id de post elegido por la variable (\$id) guardando la consulta en la variable \$post.
- save(): guarda en la tabla posts los todos los tipos de datos. Si la operación se ha completado correctamente guarda el resultado.
- update(): actualiza la tabla “posts” e inserta en todas las columnas de los campos editados en el formulario el resultado de su correspondiente setter relacionado al id de la categoría.

- `delete()`: elimina de la tabla “posts” todos los campos de las columnas de los atributos `$id` y `$authorId` asociados y devuelve el resultado de la operación.
- `queryByCategory()`: genera una nueva variable `$offset` que creará una compensación entre el número de páginas de la vista de la categoría y el número de posts que cabrán en ella, para esto se inicializará la variable `$offset` a 0 y se ejecutará la operación de cálculo entre `$page` y `$limit`, el resultado de esta operación desembocará en que la variable `$offset` sea integrada junto a `$page` y `$limit` como pseudo valores de la tabla y el método correspondiente en el controlador de Categorías.
- ❖ **User.php**: representa las consultas que debe hacer un usuario frente a la base de datos para que su controlador correspondiente pueda operar con él.

```
public function login()
{
    $result = false;
    $email = $this->email;
    $password = $this->password;

    // Comprobar si existe el usuario
    $sql = "SELECT * FROM users WHERE email = '$email'";
    $login = $this->db->query($sql);

    if ($login && $login->num_rows == 1) {
        $user = $login->fetch_object();

        // Verificar la contraseña
        $verify = password_verify($password, $user->password);

        if ($verify) {
            $result = $user;
        }
    }

    return $result;
}
```

Ilustración 16: Funciones user.php. Fuente: elaboración propia.

- Crea la clase User con sus variables para más tarde ser usada como objeto en el controlador UsersController.php.

- **Getters y Setters:** métodos de acceso que se usarán para asignar (setters) y recuperar (Getters) valores de los atributos o campos de la tabla “users” para poder usarlos en los correspondientes controladores.
- **save():** guarda en la tabla users los todos los tipos de datos introducidos en el formulario de registro. Si la operación se ha completado correctamente guarda el resultado.
- **login():** Comprueba con la base de datos si las columnas “email” y “password” coinciden con sólo únicamente una fila de la tabla con los valores que se han introducido en el formulario de inicio de sesión. Si coinciden devuelve \$result.
- **getUserById():** recoge el id del usuario en cuestión de la tabla “users” y lo devuelve a la variable \$result y este al método fetch_array() para más tarde mostrarse en el panel de posts relacionándose con el nombre del autor.
- **update():** actualiza la tabla “users” e inserta en todas las columnas de los campos editados en el formulario de edición de perfil el resultado de su correspondiente setter relacionado al id con el nombre de usuario y su email.
- **getAll():** recoge todos los campos de la tabla “users” y los devuelve a la variable \$result.

6.2.3.1 Vistas

Como pudimos ver en el punto 5 de Mockup’s algunos bocetos iniciales no definitivos, en este apartado de Vistas podremos ya contemplar el diseño final de la aplicación y ver un poco el resumen de los diferentes archivos de vistas con aquellos pequeños scripts incrustados entre la sintaxis HTML.

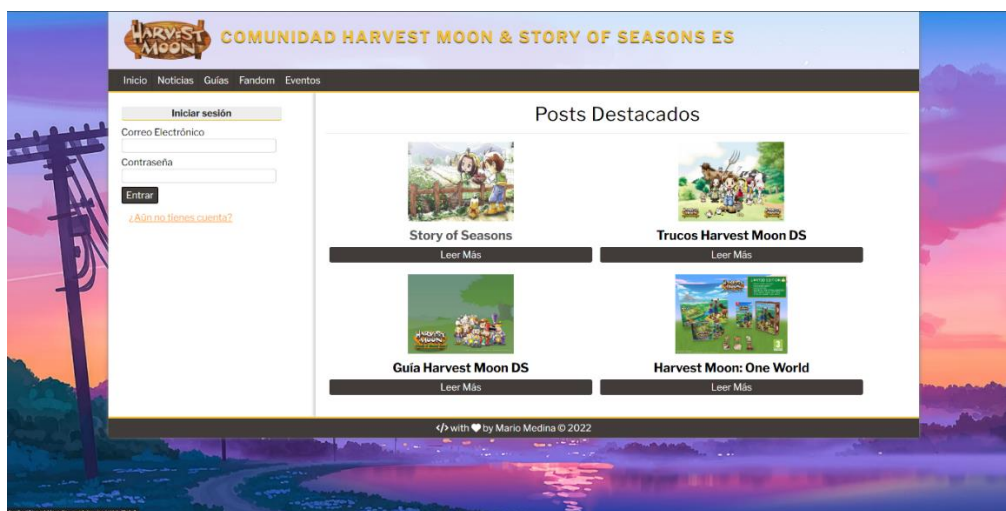


Ilustración 17: Vista final. Fuente: elaboración propia.

- ❖ **Category:** Contiene todas las vistas relacionadas con las categorías.
 - Create.php: vista del formulario para crear nuevas categorías.
 - Editar.php: vista del formulario para editar categorías.
 - Filter.php: vista de todos los posts filtrados por esa categoría.
- ❖ **Posts:** Contiene todas las vistas relacionadas con las categorías.
 - Create.php: vista del formulario para crear posts.
 - Detail.php: vista específica del post en el que se muestra el contenido y la fecha.
 - Editar.php: vista del formulario para editar posts.
 - Latest.php: vista de inicio donde se muestran los posts más destacados.
 - Manage.php: vista del panel de control para gestionar los posts.
- ❖ **Template:** Contiene todas las vistas relacionadas con el *layout* principal de la página.
 - Footer.php: vista del pie de página.
 - Header.php: incluye la cabecera y un menú de navegación formado por las diferentes categorías que se muestran con un script incrustado de php.
 - Sidebar.php: barra lateral o aside que sirve para logearnos dentro de la aplicación, dependiendo de si tenemos una sesión abierta nos mostrará unas funciones u otras.
- ❖ **User:** Contiene todas las vistas relacionadas con las acciones de usuario.
 - Profile.php: vista del formulario para editar sus datos de registro.
 - Signup.php: vista del formulario para registrarse en la comunidad.

6.2.3.1 Controladores

Una vez las vistas y los modelos están totalmente definidos podremos jugar con ellos por medio de los controladores los cuales se estarán comunicando constantemente con las vistas y los modelos, modificando, llamando y aplicando lógica de programación para que las diferentes capas de la aplicación muestren una sinergia coherente.

Para finalizar con la explicación de mi estructura MVC pasaremos a detallar un poco más los métodos y funciones de los que depende estos controladores.

❖ **CategoryController:** controlador de categorías.

```
public function save()
{
    Utils::isAdmin();
    if (isset($_POST) && isset($_POST['category_name'])) {
        // Guardar la categoria en bd
        $category = new Category();
        $category->setCategory_name($_POST['category_name']);
        $save = $category->save();
    }
    header("Location:" . base_url . "category/index");
}
```

Ilustración 18: Función de categoryController. Fuente: elaboración propia.

- create(): comprueba si es administrador y si es así da acceso a la vista de creación de categoría.
- save(): comprueba si es administrador y si es así guarda los datos recibidos por `_POST` en una variable `$save`. Tanto si el proceso ha sido un éxito o no hace una redirección al panel de gestión de categorías.
- editar(): comprueba si es administrador y si es así recoge los datos necesarios en forma de array, identifica qué id de categorías se quiere editar y si el id de los datos enviados por `_POST` coincide con la correspondiente tabla, guarda el resultado y nos redirige al panel de administración de categorías
- eliminar(): comprueba si es administrador y si es así recoge los datos necesarios en forma de array para crear la consulta relacionada al método `delete()` de la parte de modelos, almacenando la información en una variable `$queries`. Si la consulta ha sido completada nos devolverá al panel de gestión de categorías, si no nos dará una excepción dependiendo del error en datos.
- filter(): comprueba todas las categorías en las tablas e implementa las variables `$page` y `$category` y se las pasa al modelo `offset` mostrando y redirigiendo así todos los posts organizados a su categoría correspondiente.

❖ **ErrorController:** controlador de errores.

```
<?php

class errorController{

    public function Index() {
        echo "<h1>La página que buscas no existe</h1>";
    }
}
```

Ilustración 19: Función de errorController. Fuente: elaboración propia.

- Index(): Muestra un mensaje de error si introduces una URL incorrecta.

❖ **PostsController:** controlador de posts.

```
public function editar()
{
    Utils::isAdmin();

    $queries = array();
    parse_str($_SERVER['REQUEST_URI'], $queries);

    $post_id = isset($queries['id']) ? $queries['id'] : 0;

    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $post = $this->fillPost();
        $post_id = $post->getId();
        if (!$post) {
        }
        $succ = $post->updatePostContent();
        if (!$succ) {
        }
        header('Location: ' . base_url . 'posts/manage');
    }

    $result = (new Post())->getOneById($post_id);
    $pos = $result->fetch_array();

    require_once 'views/posts/editar.php';
}
```

Ilustración 20: Función de postsController. Fuente: elaboración propia.

- Manage(): comprueba si la sesión corresponde a un administrador o un usuario, si es administrador recoge todos los posts de la tabla con el método getAll(), pero si por el contrario es usuario, ejecuta el método getPostsByUid() del modelo con la variable "\$id" de dicho usuario. Se crea un array de la información recogida de las tablas y se muestra en el panel de administrar posts.

- Create(): comprueba si es administrador y si es así da acceso a la vista de creación de posts.
- Save(): comprueba si es administrador o usuario y acto seguido se asegura lo que los campos del formulario pasados por _POST están bien cumplimentados y si es así los guarda en una variable \$save y se lo devuelve al modelo correspondiente.
- Editar(): comprueba si es administrador o usuario y acto seguido se asegura que los campos del formulario pasados por _POST están bien cumplimentados y si es así los guarda en una variable \$result y se lo devuelve el array en el modelo correspondiente.
- Eliminar(): comprueba si es administrador o usuario y le dice al modelo que recoja una sentencia query de los campos que identifican al post elegido, si no logra identificarlos saltará una excepción diciendo que ha habido un error en la petición. Redirige de nuevo al panel de gestión de posts.
- handleUploadPicture(): esta función realiza comprobaciones de que el archivo enviado por el método _FILES cumple con los requisitos, de ser así crea una carpeta en el proyecto en la ruta "uploads/images" (si es que no existe todavía) y mueve la imagen enviada a dicha ruta.
- detail(): le dice al modelo que recoja una sentencia query de los campos que identifican al post elegido, los valida y después redirige automáticamente a la vista de detalles de dicho post.

❖ **UsersController:** controlador de usuarios.

- Signup(): muestra la vista con el formulario de registro a la comunidad.
- login(): si hay datos introducidos por el método POST se empieza a comprobar los datos con la función login del apartado de métodos, una vez devuelta la consulta con confirma si el objeto devuelto lo es, en cuyo caso se hará una nueva condición comprobando si esas credenciales coinciden o no con un usuario administrador. Se es redirigido al Index con la sesión iniciada.
- logout(): si hay alguna sesión iniciada simplemente se destruye, ya sea un usuario normal o un administrador. Te redirige al Index.
- updateprofile(): primero se comprueban si están llegando los valores del formulario por el método POST, una vez hecho esto, se comprueba de nuevo si el usuario que tiene la sesión iniciada tiene un id, un email y una contraseña. Se le aplican nuevos valores a los setters del modelo de usuario y se cierra la sesión para actualizar los datos de la página.


```
public function save()
{
    if (isset($_POST)) {
        $username = isset($_POST['username']) ? $_POST['username'] : false;
        $email = isset($_POST['email']) ? $_POST['email'] : false;
        $password = isset($_POST['password']) ? $_POST['password'] : false;

        if ($username && $email && $password) {
            $user = new User();
            $user->setUsername($username);
            $user->setEmail($email);
            $user->setPassword($password);

            $save = $user->save();
            if ($save) {
                $_SESSION['signup'] = "complete";
            } else {
                $_SESSION['signup'] = "failed";
            }
        } else {
            $_SESSION['singup'] = "failed";
        }
    } else {
        $_SESSION['signup'] = "failed";
    }
    header("Location:" . base_url . 'users/signup');
}
```

Ilustración 21: Función de usersController. Fuente: elaboración propia.

6.2.4 Helpers y funciones auxiliares

Para apoyar el código de nuestra aplicación especialmente en las vistas y los controladores, hemos decidido crear una nueva carpeta llamada “helpers” la cual contiene un fichero “utils.php” en el que se ejecutan pequeños métodos. Veamos brevemente para qué sirven cada uno de ellos:

deleteSession(): es necesaria para el método logout() consiguiendo destruir la sesión y cerrarla.

isAdmin(): método de comprobación para saber si el usuario loggeado es administrador o no, toma un papel fundamental en los requisitos no funcionales, ya que su función es tener una aplicación íntegramente segura.

showCategories(): función auxiliar que sirve para mostrar todas las categorías de la comunidad, es usada por la barra de navegación o navbar.

showCategory(): función auxiliar que sirve para mostrar la categoría a la que se quiera hacer referencia, es usada por el panel de gestión de categorías.

getUserId(): consigue el id del usuario a través de la función “identity”, es necesaria al hacer modificaciones en el perfil e identificaciones.

```
class Utils{

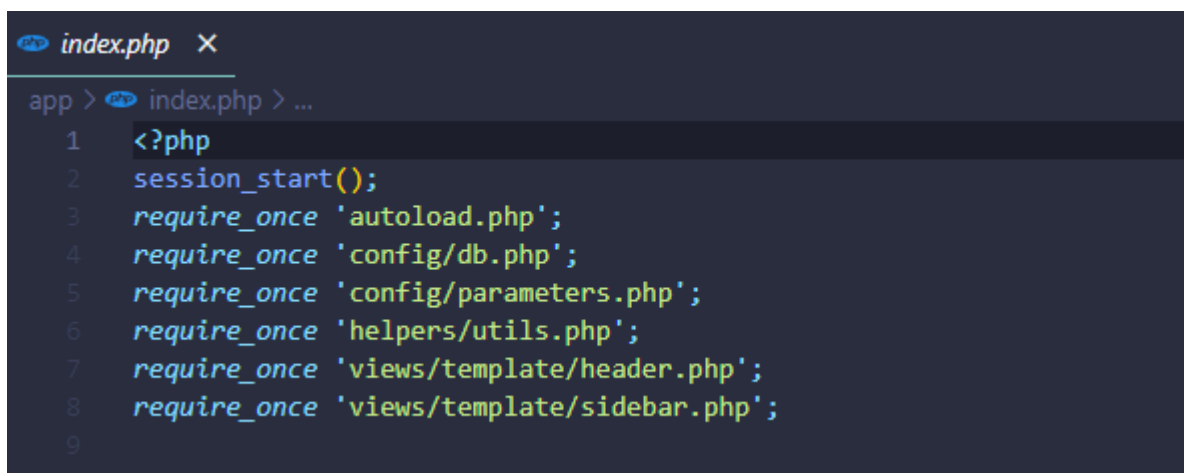
    public static function deleteSession($name){
        if(isset($_SESSION[$name])){
            $_SESSION[$name] = null;
            unset($_SESSION[$name]);
        }

        return $name;
    }
}
```

Ilustración 22: Función de utils.php Fuente: elaboración propia.

Nos queda también por hablar de dos ficheros realmente importantes a la hora de que PHP procese y monte la aplicación entera, estos son los ficheros “autoload.php” y “index.php”.

Index.php: se encarga de pedir requisitos de llamadas a todos los archivos y vistas que tengan que estar cargados cuando la aplicación inicie además de iniciar una sesión, llamar al método de error y todos los controladores necesarios.



```
index.php X
app > index.php > ...
1 <?php
2 session_start();
3 require_once 'autoload.php';
4 require_once 'config/db.php';
5 require_once 'config/parameters.php';
6 require_once 'helpers/utils.php';
7 require_once 'views/template/header.php';
8 require_once 'views/template/sidebar.php';
9
```

Ilustración 23: Estructura de index.php Fuente: elaboración propia.

Autoload.php: pequeña función en php encargada de crear rutas amigables y a la vez cargarlas en todos los controladores en el archivo index.php.

```

app > autoload.php > ...
1  <?php
2
3  function controllers_autoload($classname) {
4      include 'controllers/'. $classname . '.php';
5
6  }
7
8
9  spl_autoload_register('controllers_autoload');

```

Ilustración 24: Función de autoload.php Fuente: elaboración propia.

6.2.5 Archivos de configuración

En esta carpeta nos encontraremos con los dos ficheros, db.php y parameters.php el primero es necesario para hacer la función para conectarnos a la base de datos (nombre del servidor y credenciales) y nombre de la base de datos, que será llamada en múltiples ocasiones y segundo consta los parámetros que definirán constantes en el aplicativo como el controlador principal, la base URL que se apoya del fichero “.htaccess”, y la acción por defecto de la aplicación web.

```

. htaccess x
app > .htaccess
1  <IfModule mod_rewrite.c>
2
3  # Activar rewrite
4
5  RewriteEngine on
6  ErrorDocument 404 http://localhost/Harvest-Moon-Community/app/error/
7
8  RewriteCond %{SCRIPT_FILENAME} !-d
9  RewriteCond %{SCRIPT_FILENAME} !-f
10
11 RewriteRule ^/?(\[a-zA-Z\]+)/(\[a-zA-Z\]+) index.php?controller=$1&action=$2
12
13 </IfModule>

```

Ilustración 25: Estructura de .htaccess Fuente: elaboración propia.

6.2.6 Carpeta Public, Estilos, Imágenes y Maqueta.

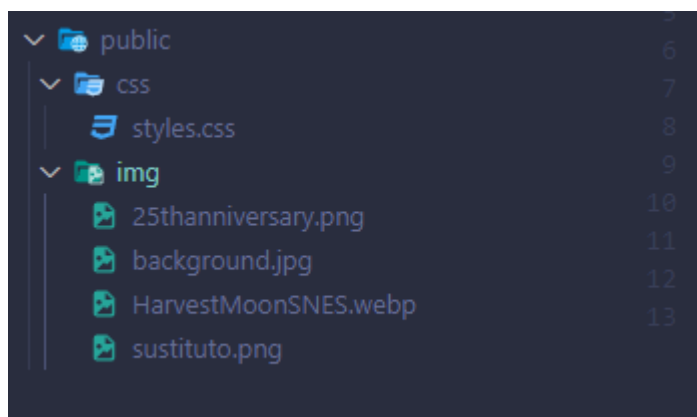


Ilustración 26: Estructura de carpetas public. Fuente: elaboración propia.

Para finalizar con nuestro recorrido de archivos necesarios para el funcionamiento de la web cabe mencionar la carpeta Public establecida en el directorio raíz que nos provee de aquella información de carácter público para otros desarrolladores, esta está formada a su vez por dos carpetas más: *Css* e *Img*, en *css* se aloja la hoja de estilos en cascada que modela la apariencia de toda la aplicación, mientras tanto *Img* contiene todas aquellas imágenes que necesite el diseño de la web para ser mostrada, como por ejemplo el logo.

Además de esto se añade un archivo “*maqueta.html*” en el interior de “*public*” que sirve como estructura de texto plano que se ha seguido para la elaboración de diferentes vistas del *layout*.

7. Despliegue y pruebas

Después de haber concretado y especificado más los detalles de la aplicación en los puntos anteriores (5. Análisis, 6. Diseño del proyecto), en este punto procederemos a someter nuestro código implementado a distintas pruebas para asegurarnos la estabilidad de todas las funciones que hemos realizado mostrándose en la siguiente tabla de pruebas:

Nº	ESPECIFICACIONES DE LA PRUEBA
1	<ul style="list-style-type: none"> Objetivo probado: registro de usuario. Requisitos probados: correcto funcionamiento del formulario con la llamada a la base de datos para poder crear una nueva cuenta. Pruebas a realizar: realizar un registro de un nuevo usuario.
2	<ul style="list-style-type: none"> Objetivo probado: registro de usuario con datos incorrectos. Requisitos probados: la base de datos deniega este nuevo usuario, no crea nuevas filas ni una nueva cuenta. Pruebas a realizar: realizar registro con un usuario nuevo.
3	<ul style="list-style-type: none"> Objetivo probado: inicio de sesión de usuario. Requisitos probados: correcto acceso a la sesión en cuestión. Pruebas a realizar: tratar de iniciar sesión.
4	<ul style="list-style-type: none"> Objetivo probado: inicio de sesión de usuario con credenciales incorrectas. Requisitos probados: redirección al Index de la página cuando se introducen credenciales incorrectas, no dejando iniciar sesión. Pruebas a realizar: tratar de iniciar sesión con credenciales incorrectas.
5	<ul style="list-style-type: none"> Objetivo probado: acceso al panel de manejo de categorías y operar (editar, eliminar o crear) sin ser administrador. Requisitos probados: redirección al Index de la página cuando se introducen URL's que no tengan acceso dicho rol, no permitiendo la interacción. Pruebas a realizar: tratar de acceder a las URL's de los paneles de gestión.
6	<ul style="list-style-type: none"> Objetivo probado: acceso al panel de manejo de categorías y operar (editar, eliminar o crear) siendo administrador. Requisitos probados: Requisitos probados: acceso al panel de gestión del blog cuando se introducen URL's que tengan acceso dicho rol y permitiendo su interacción. Pruebas a realizar: tratar de acceder a las URL's de los paneles de gestión.
7	<ul style="list-style-type: none"> Objetivo probado: acceso al panel de manejo de todos los posts y operar (editar, eliminar o crear) sin ser administrador. Requisitos probados: redirección al Index de la página cuando se introducen URL's que no tengan acceso dicho rol, no permitiendo la interacción. Pruebas a realizar: tratar de acceder a las URL's de los paneles de gestión.
8	<ul style="list-style-type: none"> Objetivo probado: acceso al panel de manejo de todos los posts y operar (editar, eliminar o crear) siendo administrador. Requisitos probados: Requisitos probados: acceso al panel de gestión del blog cuando se introducen URL's que tengan acceso dicho rol y permitiendo su interacción.

	Pruebas a realizar: tratar de acceder a las URL's de los paneles de gestión.
9	<ul style="list-style-type: none"> Objetivo probado: acceso al panel de manejo de los propios posts y operar (editar, eliminar o crear) del usuario en cuestión. Requisitos probados: acceso al panel de gestión del blog cuando se introducen URL's que tengan acceso dicho rol y permitiendo su interacción. Pruebas a realizar: tratar de acceder a las URL's de los paneles de gestión.
10	<ul style="list-style-type: none"> Objetivo probado: acceso al panel de manejo de posts y operar (editar, eliminar o crear) en usuarios ajenos. Requisitos probados: redirección al Index de la página cuando se introducen URL's que no tengan acceso dicho rol, no permitiendo la interacción. Pruebas a realizar: tratar de acceder a las URL's de los paneles de gestión.
11	<ul style="list-style-type: none"> Objetivo probado: filtrar entradas por categoría siendo visitante Requisitos probados: Es capaz de mostrar y filtrar las diferentes entradas por la categoría elegida. Pruebas a realizar: Intentar interactuar entre los distintos enlaces de la barra de navegación.
12	<ul style="list-style-type: none"> Objetivo probado: filtrar entradas por categoría siendo un usuario loggeado. Requisitos probados: Es capaz de mostrar y filtrar las diferentes entradas por la categoría elegida. Pruebas a realizar: Intentar interactuar entre los distintos enlaces de la barra de navegación.
13	<ul style="list-style-type: none"> Objetivo probado: filtrar entradas por categoría siendo un administrador. Requisitos probados: Es capaz de mostrar y filtrar las diferentes entradas por la categoría elegida. Pruebas a realizar: Intentar interactuar entre los distintos enlaces de la barra de navegación.
14	<ul style="list-style-type: none"> Objetivo probado: mirar en detalle una entrada y volver a la vista de categorías inicial siendo un visitante. Requisitos probados: Es posible entrar en la página de la publicación elegida y leer todos sus apartados. El botón de volver atrás funciona perfectamente. Pruebas a realizar: Leer una entrada elegida.
15	<ul style="list-style-type: none"> Objetivo probado: mirar en detalle una entrada y volver a la vista de categorías inicial siendo un usuario loggeado. Requisitos probados: Es posible entrar en la página de la publicación elegida y leer todos sus apartados. El botón de volver atrás funciona perfectamente. Pruebas a realizar: Leer una entrada elegida.
16	<ul style="list-style-type: none"> Objetivo probado: mirar en detalle una entrada y volver a la vista de categorías inicial siendo un usuario administrador. Requisitos probados: Es posible entrar en la página de la publicación elegida y leer todos sus apartados. El botón de volver atrás funciona perfectamente. Pruebas a realizar: Leer una entrada elegida.
17	<ul style="list-style-type: none"> Objetivo probado: Vista de los diferentes apartados de la web en distintas resoluciones de pantalla de escritorio. Requisitos probados: la vista en resoluciones de escritorio de 1920 x 1080p y de 2560 x 1440p es totalmente legible, usable y funcional.

	<ul style="list-style-type: none"> • Pruebas a realizar: Desplegar el proyecto en distintas configuraciones de pantalla del equipo.
18	<ul style="list-style-type: none"> • Objetivo probado: Despliegue y funcionamiento de la app en un entorno Windows. • Requisitos probados: Correcto funcionamiento de todas las demás pruebas bajo este sistema operativo. • Pruebas a realizar: Despliegue de la app en Windows.
19	<ul style="list-style-type: none"> • Objetivo probado: Despliegue y funcionamiento de la app en un entorno Linux. • Requisitos probados: Correcto funcionamiento de todas las demás pruebas bajo este sistema operativo. • Pruebas a realizar: Despliegue de la app en Linux.
20	<ul style="list-style-type: none"> • Objetivo probado: Modificación de cambios del perfil siendo un usuario registrado. • Requisitos probados: Correcto funcionamiento de esta utilidad, primero cumplimentando los campos a modificar y luego cerrando la sesión para que los datos se actualicen. • Pruebas a realizar: Modificar varios perfiles registrados.
21	<ul style="list-style-type: none"> • Objetivo probado: <i>Log out</i> de cualquier tipo de sesión activa. • Requisitos probados: Debe de poder eliminarse la sesión activa y no dejarte acceder de nuevo a no ser que vuelvas a hacer log in en el formulario de inicio de sesión. • Pruebas a realizar: Cerrar sesión.

8. Conclusiones

Objetivos alcanzados:

Haciendo un breve repaso a los objetivos que hemos conseguido implementar tenemos la siguiente estructura de funcionalidades del blog:

- Registro de nuevos usuarios.
- Inicio de sesión de usuarios previamente registrados.
- Inicio de sesión de administradores habilitados por el administrador de la base de datos.
- Denegar la entrada a paneles de administración de usuarios y administradores si esa sesión no corresponde con los privilegios de su propio rol (redirección a Index).
- Tanto los visitantes, como los usuarios registrados y los administradores podrán navegar por la página y leer cada una de las entradas en detalle (“Leer Más”).
- Filtrar cada una de las entradas por las distintas categorías.
- Exponer una sección de inicio donde se muestre una entrada al azar de las existentes en la base de datos actuando como “Posts Destacados”.
- Cada categoría muestra de forma ordenada, de más recientes a más antiguas, las entradas creadas por los distintos usuarios y administradores.
- Ver cada una de las entradas con más detalle.
- Volver a la vista de la categoría general asociada a esa entrada.
- Los usuarios pueden crear editar y eliminar solamente sus propias entradas.
- Los administradores del aplicativo web podrán crear y eliminar categorías además de moderar las entradas del blog de cualquier tipo de usuario (editar o eliminar).

Opinión del trabajo de fin de ciclo realizado:

Realmente ha sido una experiencia enriquecedora, a la par que en ciertos momentos estresantes. El desarrollo de una aplicación web a manos de una única persona no ha sido una tarea tan fácil como se había esperado. En las primeras semanas se tuvo que rehacer el análisis varias veces para cumplir unos objetivos y funcionalidades dignos del ciclo formativo a la par que se ajustasen a la programación temporal y fechas de entrega, esto resultó en el recorte de algunas funcionalidades posponiéndolas al apartado de esta memoria “*Vías futuras*” por falta de tiempo al desarrollarse e implementarse de manera óptima (pasando sus correspondientes pruebas de caja negra).

Además, la realización de un proyecto de estas proporciones me ha ayudado a generar seguridad como desarrollador web *FullStack*, olvidando un poco ese síndrome del impostor que me llevaba acompañado desde el inicio de las prácticas en empresa y por supuesto a mejorar y aprender muchas habilidades nuevas dentro de la industria del desarrollo de software.

Interesa destacar que mientras más investigaba para la producción de este trabajo más me daba cuenta de la inmensidad de opciones, herramientas, tecnologías y metodologías ágiles y de trabajo en equipo que pueden existir dentro de esta gran industria. Y que por ello el ser un “*Tech lead*” es una carrera de fondo de muchos años y no de un sprint único de estudios, *bootcamps* o titulaciones que, aunque, si bien te habilitan para las bases del mundo laboral es realmente en este último donde se pondrá en práctica y se seguirán estudiando todos los conceptos teóricos estudiados por otras vías en escenarios reales de la vida de un programador. Por todo esto y aunque me haya parecido todo un éxito este proyecto, aún considero que llevo la placa L de “*Learning*” en mi carrera y currículum profesional como desarrollador de aplicaciones web.

9. Vías futuras

Para dar fin a esta memoria como ya habíamos comentado durante el punto de planificación y el de conclusiones finales, ha habido algunas funcionalidades de la aplicación web que me hubiera gustado agregar, pero por los motivos anteriormente expuestos (complejidad, fechas de entrega, realización de FCT’s, etc...) no ha sido posible.

A continuación de detallarán un listado de funciones generales que se plantean agregar a lo largo de los próximos meses:

1. **Subida a un alojamiento *hosting* remoto** con una correspondiente compra de dominio web preferiblemente y si está disponible “harvestmooncommunity.es”.
2. **Realizar e implementar un estudio SEO** con referencia al motor de búsquedas de Google para lograr un buen posicionamiento y aumentar el tráfico de navegantes.
3. **Añadir un chat privado entre usuarios** con el que podrán comunicarse como si se tratase de un modelo funcional de mensajería instantánea.
4. **Habilitar un sistema de comentarios de las entradas** del blog donde cada uno de los distintos usuarios pueda editar, borrar y crear sus propios comentarios interactuando así más con la comunidad.

5. **Reestructurar el diseño de la web** para dar cabida a anuncios monetizados externos que generen el tráfico de usuarios y visitantes de la web.
6. **Crear una sección para organizar eventos concretos y actividades temporales** dentro de la comunidad.
7. **Implementar la sección de Tienda** donde podremos comprar finalmente *merchandaising* no oficial de esta saga de videojuegos.

10. Glosario

- **Harvest Moon:** Del título original en japonés "Historia de granja". Saga de videojuegos japonesa a manos de Natsume como empresa localizadora en América, más tarde romperían el acuerdo y Marvelous Interactive (desarrolladora) del videojuego perdería los derechos de la marca registrada cedida a Natsume.
- **Story Of Seasons:** títulos consecutivos localizados y desarrollado por Marvelous Interactive después de romper acuerdos de licencia con Natsume.
- **Fandom:** término anglófono haciendo referencia a "Fan Kingdom", Reino fan en Español, esto se entiende como pasatiempo o hobby de todo aquello relacionado del campo del fandom que trate.
- **Fanarts:** subtermino de Fandom relacionado con el arte hecho y producido por fans de una comunidad.
- **Reviews:** término anglófono referido al análisis de alguna obra o producto.
- **Offtopic:** contribuciones generalmente más conocido en el mundo digital para definir aquellos temas de discusión no relacionados con el tema principal del que parte.
- **Kanji:** ideograma procedente de china y adaptado a la lengua japonesa.
- **Burnout:** síndrome del trabajador quemado, se da a la cronificación del estrés laboral, desembocando en agotamiento físico y mental. Altera el rendimiento, la autoestima y la calidad de vida del trabajador.
- **MVC:** Modelo Vista-Controlador, patrón de arquitectura de software en el que se hace una abstracción entre la parte de datos, la lógica de negocio y los módulos que se encargan de gestionar las demás comunicaciones.
- **Project manager:** Jefe de Proyecto en Español, encargado de mantener comunicaciones con el cliente además de organizar, asignar fechas y carga de trabajo a los diferentes equipos desarrolladores.
- **Merchandaising:** término anglosajón referido a aquellos productos de promoción comercial, generalmente asociado a una marca o empresa reconocida, para aumentar la visibilidad de esta y satisfacer a aquellos consumidores más acérrimos a la gama de productos.

11. Bibliografía/Webgrafía

- A. (2021a, enero 30). *Font Awesome ¿Qué es y cómo se usa?* Aquí hay dominios.
<https://www.aquihaydominios.com/blog/font-awesom-que-es-y-como-se-usa/>
- A. (2021b, mayo 8). *¿Qué es la metodología Kanban y cómo funciona?* • Asana.
<https://asana.com/es/resources/what-is-kanban>
- Atlassian. (s. f.). *¿Qué es un diagrama de Gantt?*
<https://www.atlassian.com/es/agile/project-management/gantt-chart>
- *Características de requisitos funcionales y requisitos no funcionales.* (s. f.).
myservername.com. <https://es.myservername.com/features-functional-requirements>
- colaboradores de Wikipedia. (2021, 30 agosto). *PhpMyAdmin*. Wikipedia, la enciclopedia libre. <https://es.wikipedia.org/wiki/PhpMyAdmin>
- colaboradores de Wikipedia. (2022a, enero 4). *CSS*. Wikipedia, la enciclopedia libre.
<https://es.wikipedia.org/wiki/CSS>
- colaboradores de Wikipedia. (2022b, abril 25). *PHP*. Wikipedia, la enciclopedia libre.
<https://es.wikipedia.org/wiki/PHP>
- colaboradores de Wikipedia. (2022c, abril 30). *MySQL*. Wikipedia, la enciclopedia libre.
<https://es.wikipedia.org/wiki/MySQL>
- colaboradores de Wikipedia. (2022d, mayo 4). *Mozilla Firefox*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Mozilla_Firefox
- colaboradores de Wikipedia. (2022e, mayo 5). *HTML*. Wikipedia, la enciclopedia libre.
<https://es.wikipedia.org/wiki/HTML>
- Mas, D. (2019, 5 junio). *Metodología Kanban: Pros y contras en la gestión de proyectos*. FHIOS Consultoría Estratégica. <https://www.fhios.es/metodologia-kanban-pros-y-contras/>

