BlockFile

Versión	Fecha	Descripción	Elaboradores
1.0	07/09/2025	Agregado de los capítulos 1 y 3.1, y avance de los requerimientos funcionales	Guetat Concha, Adriano Armando Incacutipa Choque, Juan Fernando Murillo Castillo, Alexander
1.1	15/09/2025	Agregado de los capítulos 2 y 3, Sección Administrativa y MER	Guetat Concha, Adriano Armando Incacutipa Choque, Juan Fernando Murillo Castillo, Alexander
1.2	21/09/2025	Agregado de MF-001 y correción de MER-001. Avances en desarrollo de la página web	Guetat Concha, Adriano Armando Incacutipa Choque, Juan Fernando Murillo Castillo, Alexander

Contents

Chapter 1

Introducción

Los videojuegos ofrecen entretenimiento a las personas, pero algunos poseen la capacidad de implementar sus propios sistemas para generar ingresos utilizando los videojuegos para obtener su público objetivo. En este caso, el proyecto **BlockFile** ofrece la oportunidad de tener un sistema propio web y movil para promocionar y vender mundos inmersivos para el videojuego Minecraft, ya sea para un creador de mundos inmersivos, conocido como *Builder*, o para un equipo de *Builders*.

1.1 Antecedentes del proyecto

Desde la publicación del videojuego Minecraft en el año 2009 hasta la fecha actual 2025, Minecraft ofrece a sus jugadores realizar creaciones estructurales con total libertad.

En los años de **2009** Minecraft se actualiza a la versión clásica donde se introduce el multijugador [1] y las creaciones de los primeros servidores como nerd.nu [2] donde la comunidad se une con fines de entretenimiento y se realizan las primeras construcciones con las limitadas variaciones de materiales que ofrecía la actualización.

En los años **2010** Minecraft entra a su fase beta [3]. El juego genera la necesidad de facilitar las creaciones, de esa forma nace la primera herramienta para la ayuda y facilidad de construir dentro del juego, llamado WorldEdit [4] y gracias a ello la cantidad de construcciones aumenta, generando el interés de más jugadores y creándose por primera vez la página web Planet Minecraft [5] donde la comunidad y los *Builders* pueden compartir y monetizar sus creaciones y ser publicidad y/o compradas por la comunidad de Minecraft.

En los años **2012** Minecraft se actualiza a la versión 1.2 que trae por primera vez variaciones de colores [6] y por lo que los *builders* logran creaciones más creativas, generando en los servidores la necesidad de una construcción estética y funcional donde los jugadores puedan aparecer al entrar y puedan conocer los servicios ofrecidos. Dichas construcciones serían conocidas como ”Spawn” y seguido de la modalidad de juego, como ejemplo ”Spawn Survival” [7]. Naciendo el servidor más conocido y popular en la actualidad, Hypixel [8].

En los años **2014** Minecraft sufre un estancamiento con sus actualizaciones, pero con una popularidad alta y potencial, Microsoft opta por comprar a la compañía de Minecraft [9] aumentando las actualizaciones por año. También nace la segunda página web para publicitar y vender construcciones llamada MC-Market actualmente conocida como BuitByBit [10]

aumentando significativamente la cantidad y necesidad de construcciones para los creadores de servidores en Minecraft.

En los años **2017** nace por primera vez la comunidad profesional de *Builders* llamada Builder Refuge [11] donde anualmente generan concursos del mejor *Builder* y usado también como publicación y venta de creaciones en su red social Discord, llamado Builder's Refuge. La gran cantidad de servidores e ideas para ser exclusivos generaron una gran cantidad de mini-juegos [12], [13] y con ello la necesidad de contratar *Builders* o buscar en las páginas *web* construcciones originales y/o propias para destacar. En la comunidad de modificadores del juego minecraft creando *mods* y con ello mapas de aventuras crecen, como ejemplo tenemos al creador de mapas de aventura llamado KillerCreeper55 con su juego de historia y aventura llamado Nostalgia [14] alcanzando un gran alcance dentro la comunidad de Minecraft y los creadores de contenido de la plataforma Youtube.

En los años **2022** nace por primera vez el estudio de creadores de eventos para Minecraft llamado Eufonia con su primer evento patrocinado por Twitch Rivals, y los creadores de contenido Auronplay, Rubius y Komanche [15]. Dicho estudio y estudios futuros necesitan de una gran variedad de profesionales, dentro de ellos están los *Builders*.

En la actualidad **2025** los *Builders* o equipos de *Builders* tienen como opción publicitar y/o monetizar sus creaciones en páginas *web* como Planet Minecraft [5] o BuiltByBit [16] siendo páginas *Marketplace*, ya sea por no contar con un sistema propio o para alcanzar mayor visibilidad de personas. Algunos creadores independientes como Horace Creations [17] o equipos grandes que cuentan con una comunidad alta de personas, como Varuna Builds [18] o Everbloom Games [19] optan por tener sus propios sistemas *web* para tener mayor personalización, automatizar pedidos y/o evitar las tarifas o suscripciones impuestas por las páginas *web Marketplace*.

1.2 Planteamiento del problema

El problema de las páginas *web Marketplace* es la alta subida de creaciones por distintos creadores, generando un problema en la visibilidad para los nuevos *Builders* que no cuentan con una forma de publicitar sus productos, o en caso contrario, si se cuenta con una gran publicidad o exposición, los *Maketplace* tienen tarifas por vender productos o la necesidad de pagar suscripciones para tener funciones adicionales y personalizar los productos. Siendo una necesidad la publicidad y customización del porfolio del *Builder* o equipo de *Builders* es que se opta por una página *Web* propia.

Algunas páginas web propias no cuentan con una exposición de sus productos en los buscadores de navegadores web, siendo una característica básica de *Web 3.0* para alcanzar mayor visibilidad y no cuentan con estadísticas como son los *rankings* que dan mayor visibilidad a los productos dentro de la página *web* e incentivan a las personas a comprar.

1.3 Justificación

Ante la problemática de las páginas *web Marketplace* actuales que presentan limitaciones que afectan directamente a los *Builders*, como la baja visibilidad por la saturación de productos, costos elevados por comisiones y suscripciones, escasas opciones de personalización del portafolio, carencia de una *Web* semántica y falta de estadísticas para mejorar las ventas. Dichas condiciones afectan a los nuevos creadores independientes y limitan sus oportunidades

de posicionamiento en un entorno cada vez más profesionalizado y demandante dentro de *Minecraft*.

BlockFile propone una plataforma propia que combine con las características básicas de las páginas *web Marketplace* (visualización, calificar, descargar y comentar productos), con un sistema para incentivar la compra y enriquecimiento de exposición de los productos en los navegadores *web* que es una característica de la *Web* 3.0, dicho sistema para un equipo de *Builders* o *Builders* independientes.

1.4 Objetivos

Como objetivo general, buscamos tener una página web propia y personalizada para un *Builder* o un equipo de *Builders*, la cual debe cumplir con los siguientes objetivos específicos:

1. Identificar los requerimientos funcionales y no funcionales del sistema, como la necesidad de visibilidad de productos y personalización de portafolios.
2. Definir la arquitectura del sistema web y móvil, y las herramientas de desarrollo, estructurando módulos como catálogo, autenticación, rankings, compras y comentarios, así como la integración de RDF para mejorar la visibilidad en buscadores, para poder desarrollar y cumplir con los requerimientos establecidos.
3. Desarrollar el proyecto con las metodologías y plan de trabajo para poder implementar las funcionalidades principales del sistema em el plazo fijado.
4. Validar la usabilidad, seguridad y rendimiento del sistema mediante pruebas y de aceptación de usuario.
5. Implementar el proyecto en un entorno productivo y el desarrollo del manual de usuario.

1.5 Alcance del proyecto

El proyecto **BlockFile** abarca todas las actividades y procesos necesarios para entregar el sistema *web* propio de compra pagada de creaciones hechas por *Builders*. Esto incluye:

- Análisis detallado de los requisitos del usuario y del sistema para comprender a fondo las necesidades y expectativas.
- Diseño integral del sistema, que comprende la arquitectura de software, las interfaces de usuario (tanto para clientes como para administradores), el modelo de la base de datos y los protocolos de seguridad.
- Análisis de los módulos para el desarrollo del sistema, incluyendo:
  - Módulos para la gestión de cuentas de usuarios clientes
  - Módulos para la gestión del catálogo de contenidos
  - Módulo de gestión de calificaciones y visualización de rankings.
  - Módulo de administración para la gestión de contenidos, categorías y usuarios.

- Implementación de rigurosas pruebas en todas las etapas del desarrollo para asegurar la calidad, funcionalidad, rendimiento, seguridad y usabilidad del sistema (pruebas funcionales, no funcionales).
- Preparación y ejecución del despliegue e implementación del sistema en el entorno de producción.
- Gestión integral del proyecto, que incluye la planificación de todas las fases, el seguimiento del progreso, el control de los cambios, la gestión de los riesgos, la gestión de las comunicaciones y la gestión de los recursos (humanos y físicos).
- Elaboración y mantenimiento de la documentación del proyecto, incluyendo el plan de trabajo, especificaciones de requisitos y diseño del sistema.
- Gestión formal del alcance del proyecto, mediante la implementación de un proceso estructurado para la recepción, evaluación, aprobación o rechazo, y documentación de cualquier cambio al trabajo necesario para entregar el sistema, asegurando que se mantenga dentro de los límites definidos.

Chapter 2

Análisis de Proyecto

En esta sección se detalla el análisis de **BlockFile**, abarcando tanto los requisitos funcionales como los no funcionales, el público objetivo y el modelo de diagrama de flujo de procesos, proporcionando una visión integral de la estructura y funcionamiento del sistema.

2.1 Requerimientos funcionales y no funcionales

Los requisitos funcionales del sistema empiezan en Tabla 2.1 y los requisitos no funcionales del sistema, en la Tabla 2.10.

Para la codificación de los requerimientos funcionales se usa la sintaxis inicial ”RF” con una enumeración secuencial iniciando con ”-001” y si hay requerimientos hijos se agrega la enumeración secuencial iniciando con ”-1”. Del mismo modo, para los requerimientos no funcionales, cambiando la sintaxis inicial siendo ”RNF”.

2.1.1 Requerimientos Funcionales

Table 2.1 – Requisitos funcionales para Gestión del acceso al portal del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-001	Gestión del acceso al portal	El sistema debe permitir el inicio de sesión y registro para acceder al sistema	Alta
RF-001-1	Ingreso al Portal	El sistema debe permitir a los clientes y administradores llenar los campos de nombre de usuario y contraseña para poder ingresar al portal, solo si poseen una cuenta ya creada.	Alta

Table 2.1.a Requisitos funcionales para Gestión del acceso al portal del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-001-1-1	Formulario de acceso al portal	El formulario debe contar con los campos para ser llenados por el usuario: • Nombre de usuario. • Contraseña: Este campo debe mostrar ocultar la escritura con los carácteres "***". También debe contar con los siguientes botones: • "Iniciar Sesión" que permite ser presionado una vez completado los campos de usuario. • "Registrarse" que permite ser presionado para poder crear una cuenta a los clientes.	Alta
RF-001-1-2	Validación de credenciales en la base de datos	El sistema debe validar que el nombre de usuario y la contraseña existan en la base de datos.	Alta
RF-001-1-3	Redirección del acceso según Rol	El sistema debe dirigir a los usuarios ya sea administrador o cliente a su respectiva interfaz. Para los clientes se redirige a la interfaz de vista de productos; y para los administradores, al perfil de administrador	Alta
RF-001-2	Registro de Cliente	El sistema debe permitir a los usuarios clientes poder registrarse al sistema si no se registraron con anterioridad.	Alta
RF-001-2-1	Formulario de registro al portal	El formulario debe contar con los campos para ser llenados por el usuario: • Nombre de usuario. • Correo electrónico. • Contraseña. También debe contar con los siguientes botones: • "Registrarse" que permite ser presionado una vez completado los campos de usuario. • "Iniciar Sesión" que permite ser presionado para poder cambiar a la interfaz de iniciar sesión.	Alta
RF-001-2-2	Validación de credenciales de registro de usuario	El sistema debe validar que el nombre de usuario y el correo electrónico sean únicos y la contraseña sea mayor a 4 caracteres. El nombre de usuario no debe contener caracteres especiales. De ser validaron los datos de nombre de usuario, contraseña y correo electrónico deben ser registrados en la base de datos.	Alta
RF-001-2-3	Redirección de registro de clientes	El sistema debe dirigir a los clientes a la interfaz de vista de productos	Alta

Table 2.2 – Requisitos funcionales para Vista general para Usuarios del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-002	Vista del *Header* para Clientes	En todas las interfaces para clientes en el *Header* debe tener el nombre del proyecto y los siguientes botones: • "Inicio": Muestra la vista principal de los clientes, siendo la interfaz de muestra del catálogo. • "Ranking": Muestra las estadísticas de los productos y clientes • "Perfil": Muestra los atributos del cliente y su historial. • "Cerrar Sesión": Muestra la vista de "Inicio de Sesión" El sistema debe validar las acciones de los botones trayendo la información de la base de datos y cargando las respectivas interfaces al Usuario.	Alta
RF-003	Vista del *Header* para Administradores	En todas las interfaces para Administradores en el *Header* debe tener el nombre del proyecto y los siguientes botones: • "Perfil": Muestra los atributos del Administrador y cerrar sesión. • "Inventario": Muestra la gestión para los productos. • "Categorías": Muestra la gestión para las categorías • "Usuarios": Muestra la gestión para los usuarios" El sistema debe validar las acciones de los botones trayendo la información de la base de datos y cargando las respectivas interfaces al Usuario.	Alta
RF-004	Información mostrada y resumida para un producto	Los productos mostrados y agrupados deben mostrar la siguiente información: • Imagen principal del producto. • Precio del producto. • Nombre del producto. • Autor del producto. • calificación del producto. • Compras totales del producto.	Alta
RF-005	Paginador de información	Para una gran información mostrada en las interfaces el sistema debe permitir la paginación de la información con un paginador que contiene un máximo de 10 dígitos empezando desde el "1". Cada dígito debe ser seleccionable para el Usuario.	Alta

Table 2.3 – Requisitos funcionales para catalogo de productos del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-006	Cantidad Mostrada de productos en el Catálogo	Los productos con información resumida deben ser mostrados con un máximo de 12 productos, siendo una malla de 3 columnas por 4 filas. Para navegar por mas productos se debe usar el paginador de información.	Alta
RF-006-1	Filtro de búsqueda de Productos	Sección para realizar el filtro de búsqueda de los productos que contienen los siguientes campos de filtro: • ”Nombre”: Nombre del producto. • ”Autor”: Autor del productor. • ”Categoría”: Categoría del producto. Con un botón ”Filtrar” que al ser presionado aplica los filtros donde el sistema debe validar las distintas combinaciones de los campos llenados y omitiendo los campos vacíos trayendo la información de los productos de la base de datos y cargando en forma resumida en la interfaz del Usuario.	Alta

Table 2.4 – Requisitos funcionales para gestión de categorías del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-007	Filtrado de búsqueda para la gestión de Categorías	Mediante un botón ”Buscar Por” abre la Sección para realizar el filtro de búsqueda de las categorías que contienen los siguientes campos de filtro: • ”Id”: Id de la categoría. • ”Nombre”: Nombre de la categoría. • ”Descripción”: Descripción de la categoría. Con un botón ”Filtrar” que al ser presionado aplica los filtros donde el sistema debe validar las distintas combinaciones de los campos llenados y omitiendo los campos vacíos trayendo la información de las categorías de la base de datos y siendo cargados en la interfaz del Usuario.	Alta
RF-008	Tabla de información de las categorías	Las categorías deben ser mostradas en una tabla con los siguientes *headers*: 1. Id: Id de la categoría. 2. Nombre: Nombre de la categoría. 3. Descripción: Descripción de la categoría. 4. Editar: Botón para editar la información de la categoría. La cantidad de categorías por tabla deben ser máximo 20 filas y su navegación es con el paginador de información.	Alta

Table 2.4.a Requisitos funcionales para gestión de categorías del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-008-1	Botón para editar categoría	Al ser presionado muestra una sección para realizar la edición de la categoría que contienen los siguientes campos: • "Id": Id de la categoría (No editable). • "Fecha de creación": fecha de creación de la categoría (No editable). • "Nombre": Nombre de la categoría. • "Descripción": Descripción de la categoría. La sección contiene botones de "Aceptar" y "Borrar" para aceptar los cambios o para borrar la categoría de la base de datos.	Alta
RF-008-2	Botón para agregar categorías	Debe contener un botón "Agregar" que permite agregar categorias con la misma información de RF-008-1. El sistema debe guardar la información en la base de datos	Alta

Table 2.5 – Requisitos funcionales para gestión del perfil de usuario del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-009	Perfil de Administrador	Debe mostrar los atributos del administrador siendo los siguientes: • Nombre de Usuario. • Correo. • Contraseña. También debe mostrar los siguientes botones "Editar" y "Cerrar Sesión", donde el botón de "Editar" permite modificar los atributos des administrador. El botón "Cerrar Sesión" redirige al usuario a la interfaz de Inicio de Sesión.	Alta
RF-010	Perfil de Cliente	Debe mostrar los atributos y estadisticas del cliente siendo los siguientes: • Nombre de Usuario. • Correo. • Contraseña. • Saldo. • Número de compras: cantidad total de compras realizadas. También debe tener el botón "Editar" que permite modificar los atributos del cliente Nombre de usuario, correo y contraseña.	Alta

Table 2.5.a Requisitos funcionales para gestión del perfil de usuario del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-010-1	Historial de Cliente	En el historial de cliente debe mostrar la información resumida de los productos comprados; la presentación de los productos debe ser con un máximo de 12 productos distribuidos en una malla de 3 columnas y 4 filas; asi también su navegación debe ser con el paginador de información.	Alta

Table 2.6 – Requisitos funcionales para la gestión de productos del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-011	Filtrado de búsqueda para la gestión de Productos	Mediante un botón "Buscar Por" abre la Sección para realizar el filtro de búsqueda de los productos que contienen los siguientes campos de filtro: • "Id": Id del producto. • "Nombre": Nombre del producto. • "Autor": Autor del producto. • "Categoría": Categoría del producto. Con un botón "Filtrar" que al ser presionado aplica los filtros donde el sistema debe validar las distintas combinaciones de los campos llenados y omitiendo los campos vacíos trayendo la información de los productos de la base de datos y siendo cargados en la interfaz del Usuario.	Alta
RF-012	Tabla de información de los productos	Los productos para su gestión deben ser mostrados en una tabla con los siguientes *headers*: 1. "Id": Id del producto. 2. "Nombre": Nombre del producto. 3. "Autor": Autor del producto. 4. "Categoría": Categoría del producto. 5. Editar: Botón para editar la información del producto. La cantidad de productos por tabla deben ser máximo 20 filas y su navegación es con el paginador de información.	Alta

Table 2.6.a Requisitos funcionales para la gestión de productos del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-012-1	Botón para editar el producto	Al ser presionado muestra una sección para realizar la edición del producto que contienen los siguientes campos: • ”Id”: Id del producto (No editable). • ”Fecha de creación”: fecha de creación del producto (No editable). • ”Calificación”: calificación del producto (No editable). • ”Nombre”: Nombre de la categoría. • ”Autor”: Autor del producto. • ”Descripción”: Descripción del producto. • ”Versión”: Versión del producto. • ”Categoría”: Categoría del producto. • ”Precio”: Precio del producto. • Sección de gestión de imágenes para el producto. La sección contiene botones de ”Aceptar” y ”Borrar” para aceptar los cambios o para borrar la categoría de la base de datos.	Alta
RF-012-1-1	Sección de imágenes para el producto	Debe contener un botón ”Agregar Imágenes” que permite agregar imágenes del producto con un máximo de 10 imágenes. Dichas imágenes deben ser ordenadas y enumeradas por orden de subida donde la primera imagen es la imagen principal. Cada imagen debe tener un botón ”Borrar imagen” para eliminar la imagen de la lista; también debe contener el campo de texto para ingresar el número de otra imagen existente en la lista para cambiar el orden.	Alta
RF-012-2	Botón para agregar productos	Debe contener un botón ”Agregar” que permite agregar productos con la misma información de RF-012-1. El sistema debe guardar la información en la base de datos	Alta

Table 2.7 – Requisitos funcionales para gestión de usuarios del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-013	Filtrado de búsqueda para la gestión de Usuarios	Mediante un botón "Buscar Por" abre la Sección para realizar el filtro de búsqueda de los usuarios que contienen los siguientes campos de filtro: • "Id": Id del usuario. • "Nombre": Nombre del usuario. • "Saldo": Saldo del usuario. Con un botón "Filtrar" que al ser presionado aplica los filtros donde el sistema debe validar las distintas combinaciones de los campos llenados y omitiendo los campos vacíos trayendo la información de las usuarios de la base de datos y siendo cargados en la interfaz del Usuario.	Alta
RF-014	Tabla de información de los usuarios	Los usuarios para su gestión deben ser mostrados en una tabla con los siguientes *headers*: 1. "Id": Id del usuario. 2. "Nombre": Nombre del usuario. 3. "Saldo": Saldo del usuario que permita la búsqueda por un rango, por lo que necesita dos campos para ser llenado y determinar el intervalo de búsqueda. 4. Editar: Botón para editar la información del usuario. La cantidad de productos por tabla deben ser máximo 20 filas y su navegación es con el paginador de información.	Alta

Table 2.7.a Requisitos funcionales para gestión de usuarios del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-014-1	Botón para editar el usuario	Al ser presionado muestra una sección para realizar la edición del usuario que contienen los siguientes campos: • "Id": Id del usuario (No editable). • "Fecha de creación": fecha de creación del usuario (No editable). • "Nombre de Usuario": (No editable). • "correo": Correo del usuario (No editable). • "Saldo": Saldo del Usuario. La sección contiene botones de "Aceptar" y "Borrar" para aceptar los cambios o para marcar al usuario como "excliente" en la base de datos.	Alta

Table 2.8 – Requisitos funcionales para rankings del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-015	Sección para los rankings	En esta sección debe mostrar los siguientes botones para redirigir a las tablas de los ranking correspondientes: • ”Productos más comprados” • ”Mejores Compradores” • ”Productos Mejores Calificados”	Alta
RF-015-1	Ranking productos más comprados	Se debe mostrar la información de los productos mas comprados a lo largo del tiempo. Dicha información debe ser presentada en orden de mayor a menor en una tabla con los siguientes *headers*: • ”Top”: Muestra el orden del producto en el ranking. • ”Nombre”: Nombre del producto. • ”Autor”: Autor del producto. • ”Categoría”. Categoría del producto. • ”Precio”: Precio del producto. • ”Compras”: Número total de compras del producto. La cantidad de productos en la tabla debe ser un máximo de 10 productos y para la navegabilidad de la información debe ser hecha por un paginador de información.	Alta
RF-015-2	Ranking de mejores compradores	Se debe mostrar la información de los productos mejor calificados a lo largo del tiempo luego debe aplicarse una segunda ordenación con el número de calificaciones que recibió el producto. Dicha información debe ser presentada en orden de mayor a menor en una tabla con los siguientes *headers*: • ”Top”: Muestra el orden de los productos en el ranking. • ”Nombre”: Nombre del producto. • ”Autor”: Autor del producto. • ”Categoría”. Categoría del producto. • ”Precio”: Precio del producto. • ”Número de calificaciones”: Número total de calificaciones del producto. • ”Calificación”: calificación promedio del producto. La cantidad de productos en la tabla debe ser un máximo de 10 productos y para la navegabilidad de la información debe ser hecha por un paginador de información.	Alta

Table 2.8.a Requisitos funcionales para rankings del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-015-3	Ranking de productos mejores calificados	Se debe mostrar la información de los mejores clientes compradores de productos a lo largo del tiempo. Dicha información debe ser presentada en orden de mayor a menor en una tabla con los siguientes *headers*: • "Top": Muestra el orden de los clientes en el ranking. • "Nombre": Nombre del cliente. • "Número de compras": Número total de compras que realizo el cliente. La cantidad de clientes en la tabla debe ser un máximo de 10 clientes y para la navegabilidad de la información debe ser hecha por un paginador de información.	Alta

Table 2.9 – Requisitos funcionales para la vista del producto del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-016	Vista general del producto	Se debe mostrar la información principal del producto, la opción de compra, las características del producto, los comentarios del producto y una sección de opciones una vez comprado el producto.	Alta
RF-016-1	Información principal del producto para su vista general	Debe mostrar la siguiente información; • Nombre del producto. • Imágenes del producto, con la imagen principal resaltada a la vista. • Descripción del producto.	Alta
RF-016-2	Comentarios del producto en su vista general	Debe mostrar los comentarios realizados al producto una vez adquiridos por el cliente. La información de los comentarios que debe aparecer individualmente es: • Nombre del cliente. • Calificación del producto dada por el cliente. • fecha de envío del comentario. • Descripción del comentario. Los comentarios deben ser colocados en orden de llegada y se debe mostrar un máximo de 10 comentarios y para su navegación debe usarse un paginador de información.	Alta

Table 2.9.a Requisitos funcionales para la vista del producto del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RF-016-3	Botón de compra del producto en su vista general	Debe aparecer la siguiente información en la vista general del producto: • Precio del producto. • Saldo del cliente. • Botón de comprar el producto. Si el cliente no posee saldo para comprar el producto el administrador debe poder aumentar el saldo del cliente en el administrador de usuarios.	Alta
RF-016-4	Características del producto en su vista general	Debe aparecer la siguiente información del producto: • Calificación promedio del producto. • Cantidad de compras del producto. • Autor del producto. • Versión del producto. • Categoría del producto. • Fecha de publicación del producto.	Alta
RF-016-5	Opciones del producto comprado en la vista general del producto	Esta sección debe aparecer al cliente solo cuando adquirió el producto donde aparecen los siguientes botones: • ”Descargar”: Descargar el archivo del producto .schem. • ”Calificar”: Para calificar el producto en un rango de 1-5 estrellas siendo enteros. • ”Comentar”: Para comentar el producto donde debe aparecer una sección para colocar el comentario con un máximo de 100 palabras.	Alta

2.1.2 Requerimientos No Funcionales

Table 2.10 – Requisitos no funcionales del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RNF-001	Seguridad	El sistema debe implementar medidas de seguridad que garanticen la confidencialidad e integridad de la información en procesos como registro, inicio de sesión, gestión de contraseñas, manejo de saldo y administración de usuarios. En particular, debe cumplir con lo siguiente: **- Control de acceso a través de roles de usuario:** El sistema debe emplear un esquema de roles (administrador/cliente) que limite el acceso a funcionalidades o información según el tipo de usuario.	Alta

Table 2.10.a Requisitos no funcionales del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RNF-002	Usabilidad	El sistema debe proporcionar una experiencia de uso intuitiva, coherente y agradable para todos los usuarios (clientes y administradores). Para ello, debe cumplir con los siguientes criterios: **- Diseño consistente:** El diseño debe ser homogéneo en todas las pantallas (botones, menús, tipografía, colores). **- Organización clara:** Los contenidos deben presentarse de forma estructurada (3 contenidos por columna con un máximo de 10 filas). **- Facilidad de aprendizaje:** El sistema debe ser sencillo de entender y debe permitir a un usuario nuevo realizar con facilidad las acciones básicas en menos de 2 minutos (registro, inicio de sesión, búsqueda, compra, descarga, calificación y comentario de contenidos). **- Manejo de errores:** Cuando se produzcan errores o excepciones, el sistema debe proporcionar mensajes que expliquen la causa de manera clara.	Alta
RNF-003	Portabilidad	El sistema debe funcionar correctamente en diversos entornos para garantizar que los usuarios puedan utilizar la aplicación web independientemente del navegador o sistema operativo que usen. El aplicativo *movil* debe ser portable para la versión 16 de Android. **- Soporte mínimo:** El sitio web debe ser funcional en versiones actuales de navegadores modernos (Google Chrome, Mozilla Firefox, Microsoft Edge, Opera y Safari). **- Sistemas operativos compatibles:** El sitio web debe ser accesible desde Windows, MacOS y Linux a través de navegadores compatibles.	Alta
RNF-004	Escalabilidad	El sistema debe ser capaz de adaptarse al crecimiento en cuanto a la cantidad de usuarios concurrentes y el volumen de contenidos, manteniendo tiempos de respuesta eficientes y estables. **- Rango estimado:** El sistema debe ser capaz de manejar un incremento de 5 a 20 usuarios simultáneos y de 10 a 50 contenidos sin disminuir su rendimiento de manera significativa. **- Velocidad de consultas:** Las operaciones como búsqueda, listado de contenidos y acceso a contenidos deben ejecutarse en un tiempo medio entre 10 ms y 1 s.	Alta

Table 2.10.b Requisitos no funcionales del sistema BlockFile

Código	Nombre	Descripción	Prioridad
RNF-005	Mantenibilidad	El sistema debe estar diseñado de forma que facilite su mantenimiento, permitiendo que las actualizaciones o correcciones se completen en un máximo de 3 días. Para ello debe cumplir con lo siguiente: **- Comentarios:** Mantener comentarios en el código que describan la funcionalidad de cada sección. **- Control de versiones:** Usar un sistema de control de versiones (git) que permita registrar los cambios y revertir modificaciones en caso de errores.	Alta

2.2 Público objetivo

El público objetivo principal lo conforman los *Builders* independientes y equipos de *Builders* dedicados a la creación de mundos inmersivos en el videojuego Minecraft. Requiriendo un sistema *web* propio y personalizado que les permita mostrar sus creaciones, gestionar pedidos, obtener mayor visibilidad en buscadores mediante la *Web 3.0* e incrementar sus oportunidades de ingresos sin depender de los costos y limitaciones de los *Marketplace* tradicionales.

Así también, el público objetivo del que adquiera el sistema son los siguientes:

- **Creadores de servidores públicos o privados**, que buscan construcciones estéticas y funcionales (como *spawns* o mapas temáticos) para diferenciar su propuesta y atraer jugadores.
- **Creadores de contenido en redes sociales** (YouTube, Twitch, etc.), que necesitan mundos originales para eventos, historias y experiencias inmersivas para incrementar su interacción y alcance con la audiencia.
- **Visibilidad para estudios y comunidades** como Eufonia o Builder's Refuge, que organizan concursos, eventos patrocinados, requieren confiabilidad en los *Builders* postulantes, una *Web* propia que exponga sus creaciones y proyectos incrementa en el profesionalismo del *Builder*.
- **Compradores de la comunidad Minecraft**, quienes están interesados en adquirir productos para personalizar su experiencia dentro del juego o con sus amigos.

2.3 Diagrama de flujo de procesos

Para el flujo de procesos se modela con los diagramas de secuencia fueron elaborados utilizando la herramienta Enterprise Arquitect [20]. La imagen 2.1 muestra el diagrama de secuencia DS-007 PerfilCliente.

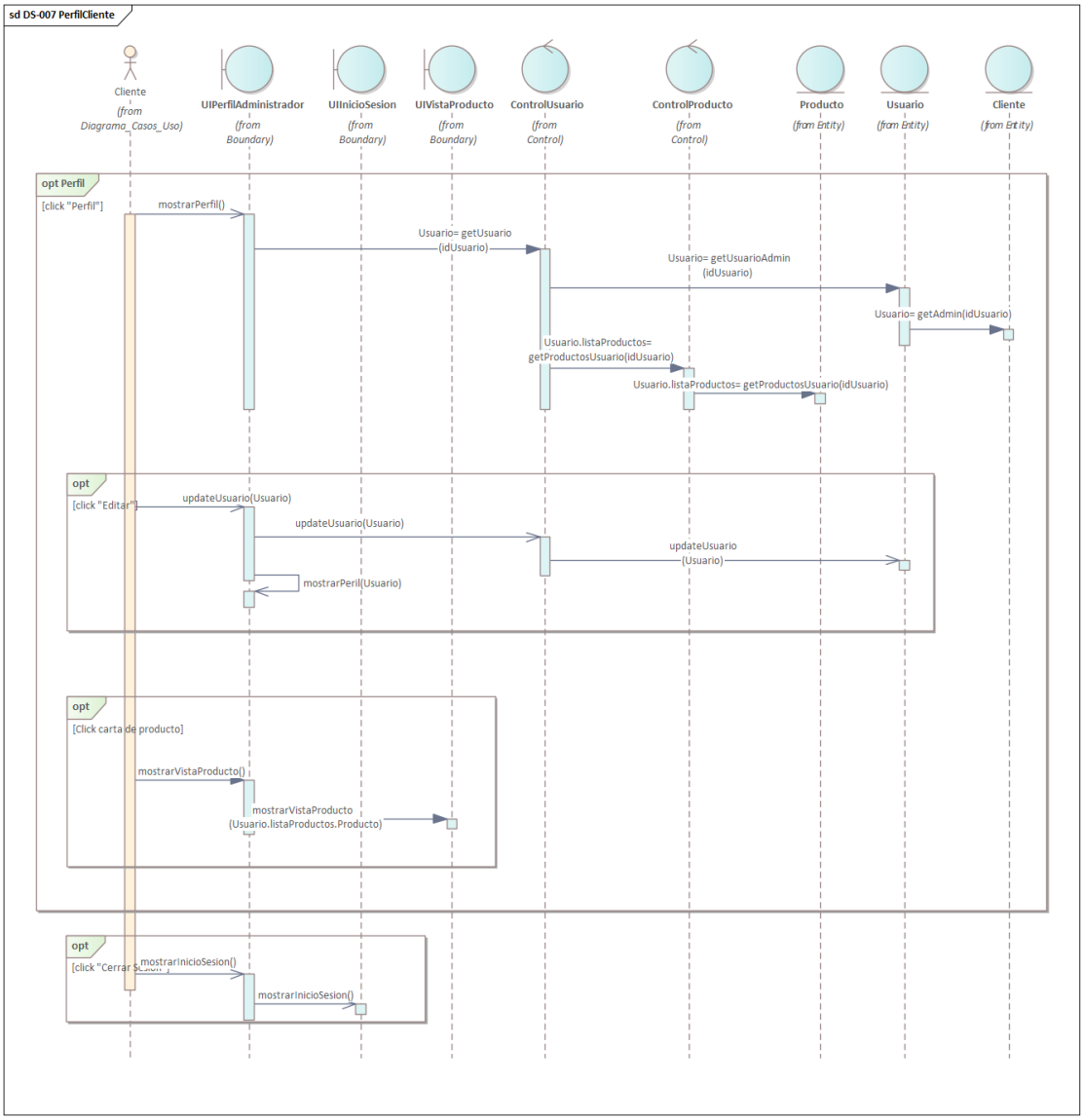

Figure 2.1 – Diagrama de secuencia DS-007

Link de acceso: **Acceso al archivo de los Diagramas de Secuencia**

Pasos de ejecución:

1. Ingresar al repositorio en GitHub usando el link proporcionado y descargar la carpeta ”Navegabilidad”.
2. Entrar en la carpeta descargada y abrir el archivo ”index.htm”.
3. En el apartado izquierdo en la raíz ”Proyect” seleccionar ”Diagramas_Secuencia” y en los directorios (”Acceso”, ”Administrador” o ”Cliente”) entrar al directorio con codificación deseada y abrir el archivo con el mismo nombre del directorio padre.

2.4 Modelo Entidad-Relación

El modelo entidad-relación describe la estructura lógica del sistema **BlockFile**, identificando las principales entidades, sus atributos y las relaciones existentes entre ellas.

2.4.1 Diagrama del modelo entidad-relación

La figura 2.2 presenta el diagrama del modelo entidad-relación. Este modelo sirve como base para el posterior diseño de la base de datos relacional. Este modelo fue elaborado utilizando la herramienta Enterprise Arquitect [20].

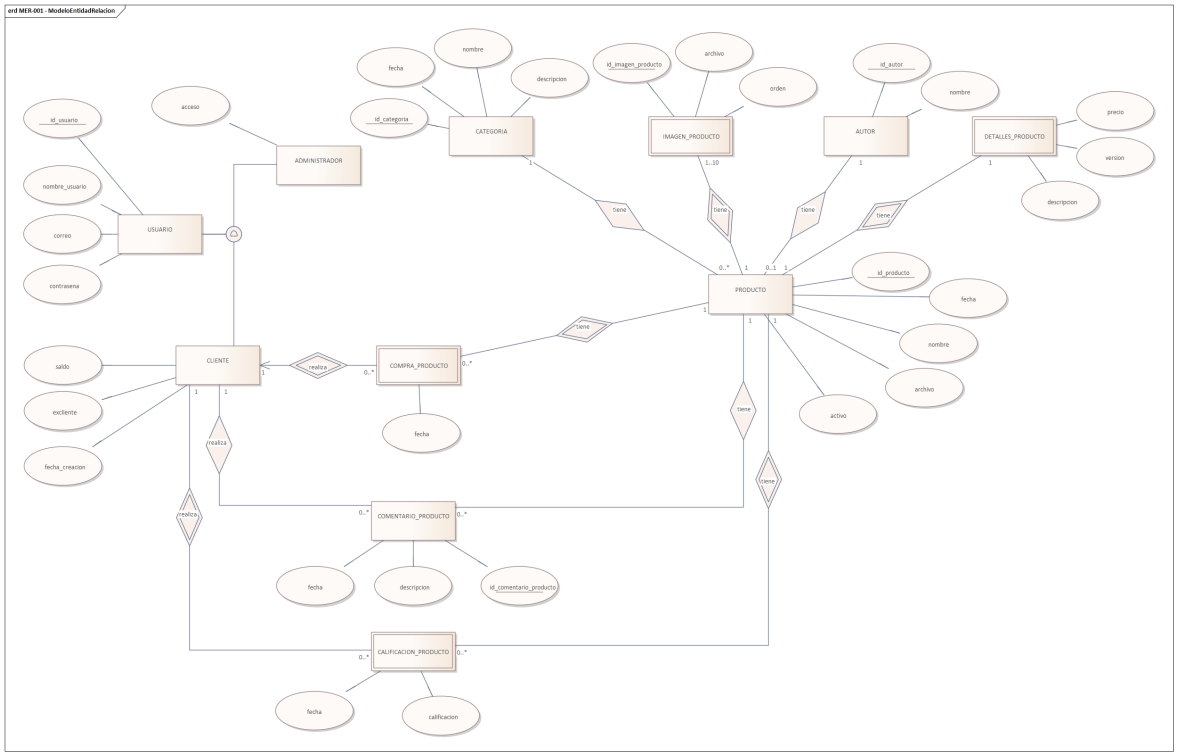

Figure 2.2 – Modelo entidad-relación (MER-001) del sistema BlockFile

Link de acceso: **Acceso al archivo del Moderlo entidad-relación**

Pasos de ejecución:

1. Ingresar al repositorio en GitHub usando el link proporcionado y descargar la carpeta ”Navegabilidad”.
2. Entrar en la carpeta descargada y abrir el archivo ”index.htm”.
3. En el apartado izquierdo en la raíz ”Proyect” seleccionar Modelo_de_Datos, seleccionar el directorio ”ModeloEntidadRelacion” y seleccionar el archivo llamado ”MER-001 - ModeloEntidadRelacion”.

Chapter 3

Diseño del Sistema

En esta sección se detalla el diseño de **BlockFile**, abarcando tanto la arquitectura del sistema como las herramientas y tecnologías empleadas para el desarrollo correcto y finalización del proyecto. También se detalla el diseño final de la página *web* y *movil* cumpliendo los requerimientos del sistema y usuario.

3.1 Arquitectura del sistema

Las herramientas usadas en la realización y finalización del proyecto en las áreas de frontend, backend y base de datos son las siguientes:

3.1.1 Frontend

A continuación mencionaremos las herramientas para el desarrollo del *Frontend*:

- HTML, CSS y JS.
- Editor de codigo Visual Studio Code versión 1.103.

3.1.2 Backend

A continuación mencionaremos las herramientas para el desarrollo del *Backend*:

- Django versión 5.2.6.
- IDE (Entorno de desarrollo integrado) PyCharm versión 2024.3.5 (Professional Edition).
- Python v3.13.7.

3.1.3 Base de Datos

A continuación mencionaremos las herramientas para el desarrollo para la *Base de Datos*:

- PostgreSQL versión 17.6.

3.2 Tecnologías y herramientas

La elección de cada herramienta se debe a los aspectos de estabilidad, escalabilidad, soporte por la comunidad y de acuerdo a los requerimientos del proyecto.

3.2.1 Frontend

- **HTML, CSS y JS:** Tecnologías base para poder desarrollar el *frontend* dentro del proyecto mediante Django.
- **Visual Studio Code v1.103:** Editor flexible que gracias a su comunidad brinda extensiones que facilita el desarrollo del *Frontend*.

3.2.2 Backend

- **Django v5.2.6:** *Framework* en Python para el desarrollo en *backend* ofreciendo seguridad y escalabilidad, con un sistema de capas MTV (Model–Template–View), brindando una estructura ordenada que simplifica la gestión de datos y acelera la implementación de funcionalidades.
- **PyCharm 2024.3.5 (Professional Edition):** IDE especializado en Python para aumentar la productividad del desarrollo, con herramientas como integración con la base de datos, integración con GitHub, extensiones especializadas, depuración y facilidad de implementar entornos virtuales.

3.2.3 Base de Datos

- **PostgreSQL v17.6:** Elegida por ser una base de datos relacional escalable y por su facilidad para implementar consultas complejas. También, gracias a su *Multiversion Concurrency Control* (MVCC) para la lectura y escrituras de los productos que es donde hay mas interacción por los usuarios.

3.3 Prototipos UI/UX

Los prototipos de presentación es un diagrama de mapeo, donde se visualizan las interfaces y el mapeo de su navegabilidad del proyecto **BlockFile**.

3.3.1 Wireframes

En los *Wireframes* mostramos el esqueleto de las interfaces del proyecto como se muestra en la figura 3.1 y fueron diseñados con la herramienta Enterprise Arquitect [20] para la navegabilidad interactiva y el diagrama de mapeo de los *Wireframes* se realizó con la herramienta Figma [21].

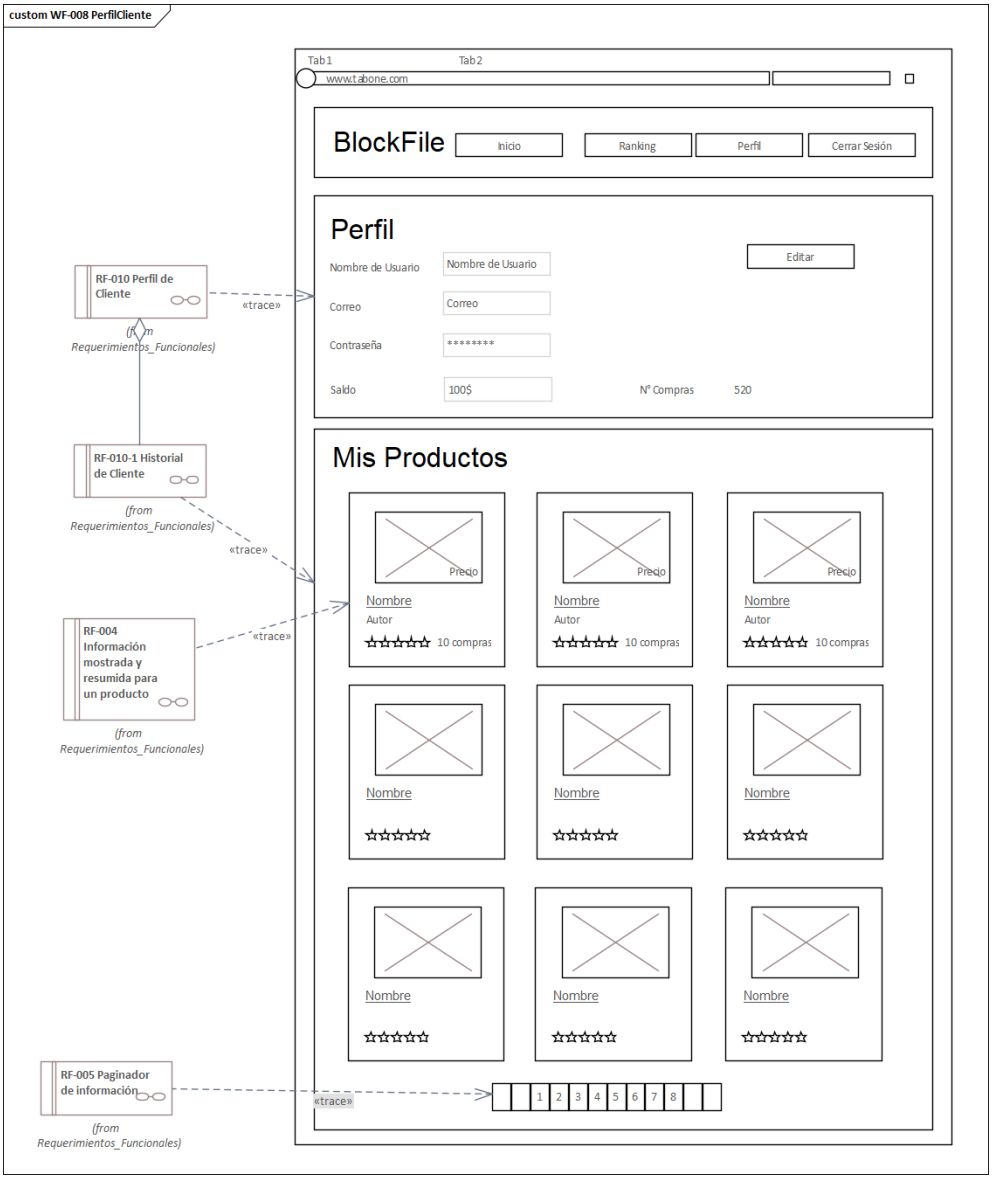

Figure 3.1 – *Wireframe* del Perfil de cliente

Link de acceso: **Acceso al archivo de los Diagramas de Secuencia**

Pasos de ejecución:

1. Ingresar al repositorio en GitHub usando el link proporcionado y descargar la carpeta ”Navegabilidad”.
2. Entrar en la carpeta descargada y abrir el archivo ”index.htm”.
3. En el apartado izquierdo en la raíz ”Proyect” seleccionar ”Wireframes”.
4. Puede seleccionar el archivo ”GeneralWireframe” donde tendra una vista general de todos los wireframes, o puede ir en los directorios (”Acceso”, ”Administrador” o ”Cliente”) entrar al directorio con codificación deseada y abrir el archivo con el mismo nombre del

directorio padre.

5. En ambos casos puede dar selección a los botones del wireframe par su navegabildiad.
6. Adicionalmente, pude dar selección a los requerimientos para visualizar el diagrama del requerimiento con sus respectivas descripciones.

Para acceder al mapa general de los *Wireframes* se debe acceder al siguiente link Acceso al **mapa general de Wireframes** , la página muestra una estructura general con todos *Wireframes* con las direcciones de los botones, como ejemplo muestra la sección de inicio de sesión al portal en la figura 3.2.

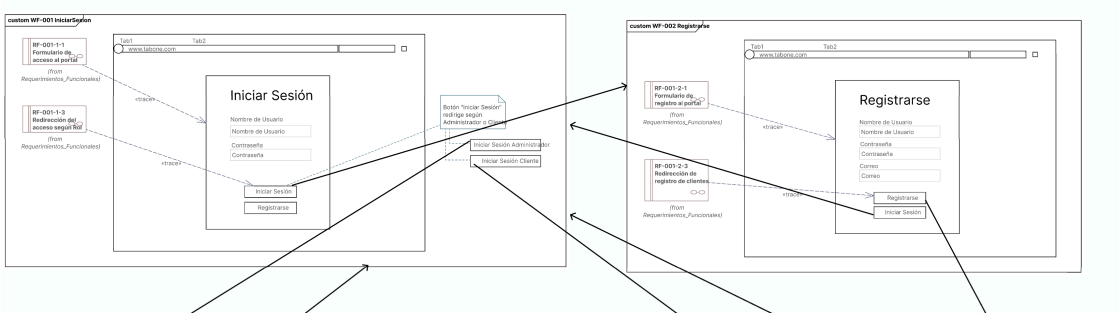

Figure 3.2 – *Wireframe* de inicio de sesión de usuarios

3.3.2 Mockups

En los *Mockups* mostramos el diseño final de las interfaces del proyecto como se muestra en la figura 3.3 y fueron diseñados con la herramienta Figma [21].

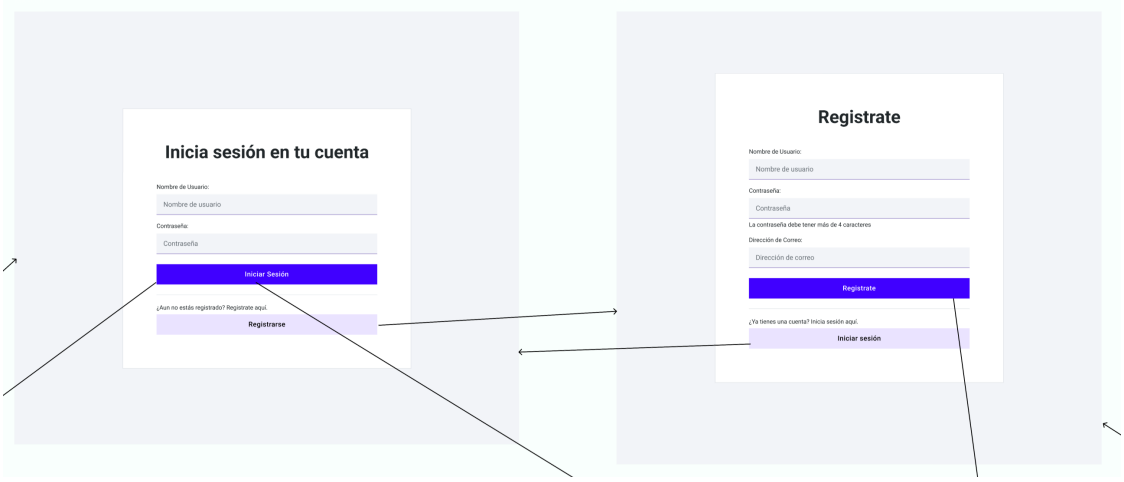

Figure 3.3 – *Mockup* del Perfil de cliente

Para acceder al mapa general de los *mockups* se debe acceder al siguiente link **Acceso al mapa general de *mockups*** , la página muestra una estructura general con todos *mockups* con las direcciones de los botones.

Chapter 4

Desarrollo e Implementación

En esta sección se detalla el desarrollo de **BlockFile**, abarcando las arquitecturas de desarrollo en el *backend*, así como en el *frontend* a lo largo del tiempo para garantizar el seguimiento del desarrollo correcto y la finalización del proyecto. También se detalla el Modelo Físico de la base de datos.

4.1 Metodología de desarrollo

El desarrollo se realizó con la metodología ágil **Scrum**, adaptado a los requerimientos del proyecto para obtener el mapa de las historias de usuario. Permitiendo repartir las tareas con seguimiento del proceso, como ejemplo esta la figura 4.1.

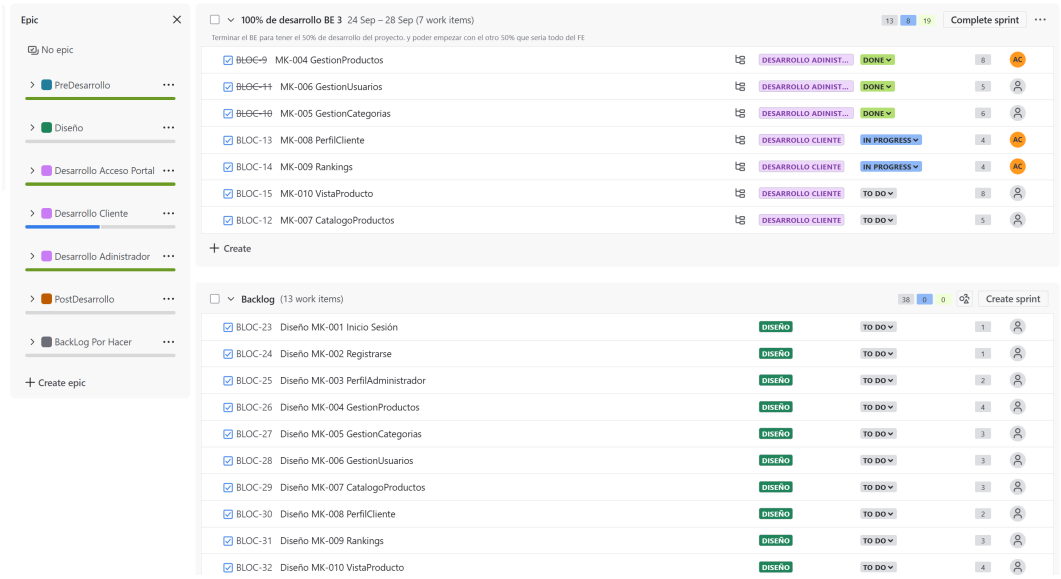

Figure 4.1 – Jira - Backlog del proyecto **BlockFile**

Roles y responsabilidades

El equipo estuvo compuesto por los siguientes roles:

- **Product Owner:** Define y prioriza las historias de usuario en el *Product Backlog*, asegurando el cumplimiento de los requisitos funcionales y no funcionales.
- **Scrum Master:** Facilita y gestiona impedimentos, asegurando el cumplimiento de la metodología ágil.
- **Equipo de Desarrollo:** Implementa las funcionalidades del sistema.

Metodología de Reuniones

- **Sprint Planning:** Definición de los objetivos e historias de usuario de cada *sprint* semanal, con sus tareas en Jira.
- **Sprint Review:** Reunión para la demostración de los resultados obtenidos para dar retroalimentación.
- **Sprint Retrospective:** Reunión de 3 días a la semana con duración de 15 minutos, mediante Discord para comunicar avances, problemas y siguientes aportes.

Cada sprint tuvo una duración de una semana, y se usó un tablero de Jira para el seguimiento de las tareas bajo los estados: *To Do*, *In Progress*, *Code Review* y *Done*.

4.2 Plan de trabajo y cronograma

Las actividades se definen en la siguiente tabla, esta contiene el código, la relación con la actividad anterior, fechas, recursos y costo de cada actividad.

Table 4.1 – Cronograma de actividades para el desarrollo del sistema BlockFile

Código	Relación	Actividades	Inicio	Final	Recursos	Costos
C-001	-	Funcionalidades, organización y diseño de mockups.	07/09/2025	17/09/2025	RH: 3 personas RF: 3 computadoras, Overleaf, Figma	S/.2,361.60
C-002	FS	plan de trabajo	17/09/2025	24/09/2025	RH: 3 personas RF: 3 computadoras, overleaf	S/.2,361.60
C-003	FS	Plan de Ejecución y Modelo de requisitos	24/09/2025	07/10/2025	RH: 5 personas RF: 5 computadoras, StarUML	S/.4,428.00

Continúa en la siguiente página

Table 4.1 – continuación desde la página anterior

Código	Relación	Actividades	Inicio	Final	Recursos	Costos
C-004	FS	Diagramas de Secuencia y de Modelo de datos	06/10/2025	16/10/2025	RH: 3 personas RF: 3 computadoras	S/.3,247.20
C-005	FS	Desarrollo Front-end, interfaz de usuario y administrador.	05/10/2025	10/10/2025	RH: 3 personas RF: 3 computadoras, Pycharm, Github	S/.5,904.00
C-006	FS	Desarrollo Back-end, lógica del servidor.	05/10/2025	10/10/2025	RH: 3 personas RF: 3 computadoras, Pycharm, Github	S/.6,494.40
C-009	FS	Documentación, despliegue y monitoreo del proyecto.	09/10/2025	20/10/2025	RH: 3 personas RF: 3 computadoras, Overleaf, Railway	S/.4,428.00

4.3 Desarrollo del backend y API

En esta sección se muestra el desarrollo del backend y las API's utilizadas. Para el desarrollo del *backend* usamos el modelo de datos físicos que describe la implementación concreta de la base de datos del sistema **BlockFile** en el sistema gestor de bases de datos elegido (PostgreSQL).

4.3.1 Diagrama del Modelo Físico

La Figura 4.2 presenta el diagrama del modelo físico **(MF-001)** de la base de datos de **BlockFile**, incluyendo los tipos de datos definidos para su implementación en PostgreSQL.

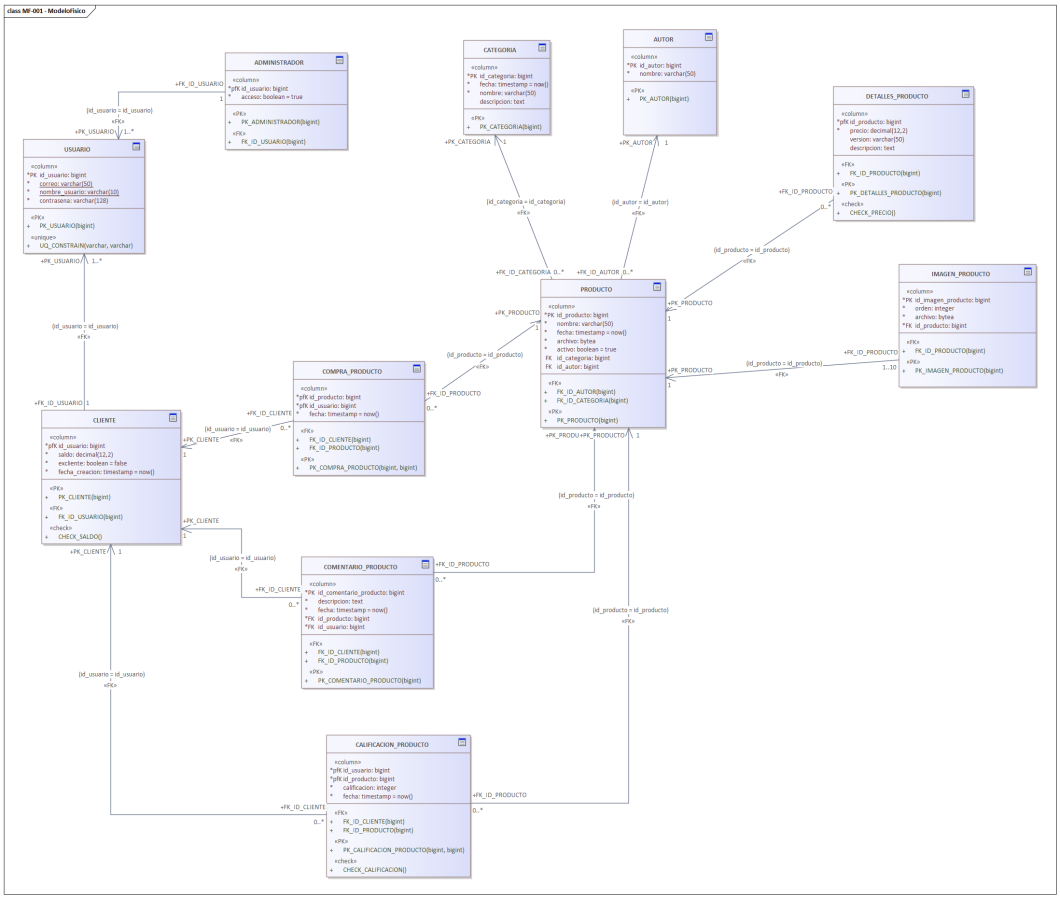

Figure 4.2 – Diagrama del modelo físico (MF-001) del sistema BlockFile

Link de acceso: **Acceso al archivo del Moderlo Físico**

Pasos de ejecución:

1. Ingresar al repositorio en GitHub usando el link proporcionado y descargar la carpeta ”Navegabilidad”.
2. Entrar en la carpeta descargada y abrir el archivo ”index.htm”.
3. En el apartado izquierdo en la raíz ”Proyect” seleccionar Modelo_de_Datos, seleccionar el directorio ”ModeloFisico” y seleccionar el archivo llamado ”MF-001 - ModeloFisico”.

4.3.2 Arquitectura

El backend fue desarrollado utilizando **Django** en **Python**, bajo el patrón **MTV (Model–Template–View)** de Django, pero extendido a una arquitectura multicapa organizada en:

- **Models:** Definición de las entidades del sistema conforme al modelo físico (MF-001) y los códigos BD-001 a BD-011 de la sección administrativa.
- **Repository:** Encargado del acceso y manipulación directa de datos, implementando consultas CRUD mediante SQL.
- **Services:** Lógica de negocio que coordina las operaciones entre repositorios y vistas.

- **Serializers:** Validación y transformación de datos de entrada/salida entre la API y el frontend.
- **Views (API):** Encargado de mandar la información solicitada por el *Frotnend* mediante *Django REST Framework*.

4.3.3 Implementación de aplicaciones Django

- **BlockFileBE (Configuración):** contiene las configuraciones del proyecto y las rutas principales de las aplicaciones
- **Utilidades del proyecto (Core):** Contiene los *models* del proyecto, variables y algoritmos que pueden ser utilizados en mas de una aplicación
- **Manejo de usuarios (users):** Permite el acceso al sistema, ya sea como cliente o administrador, permite el registro de clientes y muestra la información del administrador con capacidad de edición de sus datos.
- **Gestor de las categorías (gestorCategorias):** Permite la gestión de las categorías (crear, editar, eliminar y borrar) mostrando la información en tablas.
- **Gestor de Productos (gestorProductos):** Permite la gestión de los productos (crear, editar, eliminar y borrar) mostrando la información en tablas.
- **Gestor de usuarios (gestorUsuarios):** Permite la gestión de los usuarios permitiendo la edición de su saldo y barrado del usuario, mostrando la información en tablas.
- **Perfil del Cliente (PerfilCliente):** Permite la edición de la información del cliente como la vista de sus productos comprados.
- **Catálogo de productos (CatalogoProductos):** Contiene la vista principal de los productos en cartas de presentación, filtrador y paginador.
- **Tablas de estadísticas (Rankings):** Permite la visualización de los productos mas comprados, productos mejores calificados y mejores compradores.
- **vista principal del producto (vistaProducto):** Permite ver los comentarios y la información detallada de un producto, con la capacidad de comprarlo para permite la opción de descargar, calificar o comentar el producto.

4.3.4 APIs por aplicación Django

A continuación se describen las APIs expuestas por cada aplicación del proyecto, indicando método HTTP, ruta y comportamiento.

4.3.4.1 Proyecto `BlockFileBE` (enrutamiento raíz)

- Incluye las rutas de: users, gestorProductos, gestorCategorias, gestorUsuarios, catalogoProductos, vistaProducto, Rankings, PerfilCliente.

4.3.4.2 Aplicación users

- **[VISTA] GET/POST /iniciar/** → `iniciar_sesion_view`. Muestra formulario de Iniciar Sesión y, en POST, valida credenciales (serializer y service) y abre sesión (guarda id, nombre y correo en request.session); redirige según rol de usuario.

- **[VISTA] GET/POST /registrarse/** → `registrarse_view`. Muestra registro y, en POST, valida unicidad de nombre y correo, y crea al usuario (serializer y service); inicia sesión al crear.
- **[VISTA] GET/POST /perfil-admin/** → `perfil_admin_view`. Muestra y edita datos del administrador; en POST valida campos y modifica los cambios (serializer y service).
- **[VISTA] GET /logout/** → `logout_view`. Limpia la sesión y redirige a Iniciar Sesión.

4.3.4.3 Aplicación `gestorCategorias`

- **[VISTA] GET /gestor-categorias/gestion/** → `gestion_categorias_view`. Muestra la UI de categorías (tabla + filtros).
- **[API] GET /gestor-categorias/gestion/api/listar/** → `api_listar_categorias`. Lista de categorías con filtros (id, nombre, descripcion) y paginación (máx. 10 filas por página).
- **[API] GET /gestor-categorias/gestion/api/detalle/{id}/** → `api_detalle_categoria`. Devuelve detalle de una categoría para edición.
- **[API] POST /gestor-categorias/gestion/api/guardar/** → `api_guardar_categoria`. Crea o actualiza la categoría.
- **[API] POST /gestor-categorias/gestion/api/eliminar/{id}/** → `api_eliminar_categoria`. Elimina categoría.

4.3.4.4 Aplicación `gestorProductos`

- **[VISTA] GET /gestor-productos/gestion/** → `gestion_productos_view`. Muestra la UI de productos (tabla + filtros).
- **[API] GET /gestor-productos/gestion/api/listar/** → `api_listar_productos`. Lista productos con filtros (`id`, `nombre`, `autor`, `categoria`) y paginación (10 por página).
- **[API] GET /gestor-productos/gestion/api/detalle/{id}/** → `api_detalle_producto`. Devuelve detalle completo para edición.
- **[API] POST /gestor-productos/gestion/api/guardar/** → `api_guardar_producto`. Crea o actualiza el producto (nombre, autor, versión, categoría, precio, descripción, etc.).
- **[API] POST /gestor-productos/gestion/api/subir_imagen/{id}/** → `api_subir_imagen`. Sube una imagen del producto, detecta MIME, asigna orden secuencial y almacena el binario.
- **[API] POST /gestor-productos/gestion/api/borrar_imagen/{id}/** → `api_borrar_imagen`. Elimina una imagen del producto.
- **[API] POST /gestor-productos/gestion/api/reordenar_imagen/{id}/** → `api_reordenar_imagen`. Cambia el orden de una imagen existente.
- **[API] POST /gestor-productos/gestion/api/eliminar/{id}/** → `api_eliminar_producto`. Borra el producto marcando como inactivo.

- **[API] GET /gestor-productos/gestion/api/imagen/{id}/** → `api_imagen_producto`. Devuelve el binario de una imagen para mostrarla.
- **[API] POST /gestor-productos/gestion/api/subir_archivo/{id}/** → `api_subir_archivo_producto`. Sube el archivo descargable del producto, guarda contenido binario y nombre.
- **[API] GET/POST /gestor-productos/gestion/api/descargar_archivo/{producto_id}/** → `descargar_producto_view`. Retorna la descarga del archivo asociado, con nombre y `Content-Type`.

4.3.4.5 Aplicación `gestorUsuarios`

- **[VISTA] GET /gestor-usuarios/gestion/** → `gestion_usuarios_view`. Muestra la UI de usuarios (tabla + filtros).
- **[API] GET /gestor-usuarios/gestion/api/listar/** → `api_listar_usuarios`. Lista usuarios con filtros (`id`, `nombre` y `saldo`); paginación (10 por página).
- **[API] GET /gestor-usuarios/gestion/api/detalle/{id}/** → `api_detalle_usuario`. Devuelve datos de un usuario para modificarlos.
- **[API] POST /gestor-usuarios/gestion/api/guardar/** → `api_guardar_usuario`. Actualiza saldo del usuario.
- **[API] POST /gestor-usuarios/gestion/api/eliminar/{id}/** → `api_eliminar_usuario`. Marca como excliente al usuario.

4.3.4.6 Aplicación `catalogoProductos`

- **[VISTA] GET /catalogo-productos/catalogo/** → `catalogo_productos_view`. Muestra el catálogo para los clientes con filtros por `nombre`, `autor` y `categoria`.
- **[API] GET /catalogo-productos/catalogo/api/listar/** → `api_listar_productos`. Devuelve JSON del catálogo paginado (10 por página) con: id, nombre, autor, precio, id de imagen principal, calificación promedio y total de compras.
- **[API] GET /catalogo-productos/catalogo/api/imagen/{id}/** → `api_imagen_producto`. Retorna el binario de la imagen para ser mostrada.

4.3.4.7 Aplicación `vistaProducto`

- **[VISTA] GET/POST /vista-producto/producto/{producto_id}/** → `detalle_producto_view`. Muestra la vista detallada (nombre, autor, categoría, fecha, descripción, imágenes, calificación promedio, compras, etc.). Controla si el cliente ya compró para habilitar las opciones: *descargar*, *calificar*, *comentar*. En POST gestiona acciones de la vista (por ejemplo, comprar y registro en historial.
- **[API] GET /vista-producto/producto/api/imagen/{id}/** → `api_imagen_producto`. Devuelve binario de imagen del producto para mostrarlo.
- **[API] GET /vista-producto/producto/api/{producto_id}/descargar/** → `descargar_producto_view`. Entrega el archivo descargable del producto si el cliente lo compró.

- **[API] POST /vista-producto/producto/api/{producto_id}/calificar/** → `calificar_producto_view`. Registra calificación entera en rango [1-5].
- **[API] POST /vista-producto/producto/api/{producto_id}/comentar/** → `comentar_producto_view`. Registra el comentario ingresado por el cliente.

4.3.4.8 Aplicación `Rankings`

- **[VISTA] GET /rank/** → `rankings_view`. Muestra las tres tablas paginadas:
  1. **Productos más comprados**: Lista ordenada por productos con compras totales.
  2. **Mejores compradores**: clientes ordenados por compras realizadas.
  3. **Productos mejor calificados**: Lista ordenada por promedio de calificación y por número de calificaciones.

4.3.4.9 Aplicación `PerfilCliente`

- **[VISTA] GET/POST /perfil-cliente/** → `perfil_cliente_view`. Muestra el perfil (nombre, correo, contraseña, saldo, número de compras) y el historial de compras. En POST permite actualizar nombre, correo y contraseña.

4.4 Desarrollo del frontend

El *Frontend* de **BlockFile** se implementa con **HTML, CSS y JavaScript** usando plantillas de Django, siguiendo la codificación de interfaces definida (MK-001 a MK-010) en la sección administrativa y consumiendo las APIs del backend mediante **fetch()**. El *Frontend* tiene la siguiente arquitectura:

- **Plantillas**: `templates/` con subcarpetas por uso: `base/`, `users/`, `products/`, `categories/`, `usuarios/`.
- **Estilos**: `static/css/` con un sistema modular: `variables.css`, `base.css`, `layout.css`, `components.css`, `utilities.css`.
- **JS**: comportamiento del *frontend* en las propias plantillas.

4.4.1 Plantillas y navegación

Se usa la herencia de plantillas para encabezados de navegación y marcos comunes:

- **Header Cliente**: `templates/base/HeaderCliente.html`. Contiene navegación para *Inicio de cliente* (catálogo), *Ranking*, *Perfil* y *Cerrar sesión*.
- **Header Administrador**: `templates/base/HeaderAdministrador.html`. Contiene navegación para *Perfil*, *Inventario* (gestión de productos), *Categorías* (gestión de categorías) y *Usuarios* (gestión de usuarios).

Cada página HTML extiende uno de los *headers* y define el bloque `{ % block content % }`.

Las interfaces son:

- **MK-001** (*Iniciar Sesión*): `users/IniciarSesion.html`.
- **MK-002** (*Registrarse*): `users/Registrarse.html`

- **MK-003** (*Perfil Administrador*): `users/PerfilAdministrador.html`
- **MK-004** (*Gestión de Productos*): `products/GestionProductos.html`
- **MK-005** (*Gestión de Categorías*): `categories/GestionCategorias.html`
- **MK-006** (*Gestión de Usuarios*): `usuarios/GestorUsuarios.html`
- **MK-007** (*Catálogo*): `users/Catalogo.html`
- **MK-008** (*Perfil Cliente*): `users/PerfilCliente.html`
- **MK-009** (*Rankings*): `products/Rankings.html`
- **MK-010** (*Vista de Producto*): `products/VistaProducto.html`

4.4.2 JavaScript por vistas

El proyecto centra el comportamiento en scripts embebidos al final de cada plantilla. El patrón es:

1. Captura de elementos con utilidades (`const $ = id => document.getElementById(id)`).
2. Construcción de *query strings* dinámicos desde formularios con `FormData` y `URLSearchParams`.
3. Llamadas `fetch()` (GET/POST), parseo `.json()`, y render de tabla o tarjetas.
4. Paginación con botones *Prev/Siguiente* y páginas numeradas.

Catálogo (MK-007, `users/Catalogo.html`).

- **Filtros**: nombre, autor y categoría (`#formBuscar`). Se construye la query eliminando campos vacíos antes de invocar `./api/listar/`.
- **Render**: se dibujan tarjetas de producto (máx. 12 por página, cuadrícula 3x4), incluyendo imagen principal (URL de imagen construida con { % url 'catalogoProductos:api_imagen' id % }), nombre, autor, precio y calificación, según los datos devueltos por la API.
- **Paginación**: componente `#pager` con navegación por páginas y límite de dígitos definido por los requisitos.
- **Detalle**: la ruta a detalle se forma sobre la plantilla de URL vistaProducto:detalle_producto.

Vista de Producto (MK-010, `products/VistaProducto.html`).

- **Imagenes producto**: imagen principal (`#mainImg`) y miniaturas organizadas en un carrusel de tipo `.carousel--thumbs` con pista `.carousel__track`. Al hacer *click* en una miniatura, actualiza `src` de la principal.
- **Compra y acciones post-compra**: muestra precio y saldo; si el cliente posee el producto, habilita *Descargar*, *Calificar* y *Comentar*.
- **Modales accesibles**: dos modales (`modalCalificar` y `modalComentar`) con apertura/cierre por teclas (`Escape`), cierre al *click* fuera del contenido.

- **Comentarios paginados**: hasta 10 por página, ordenados por llegada. Carga asíncrona sobre endpoint de comentarios y actualización del listado sin recargar.

Rankings (MK-009, `products/Rankings.html`).

- **Pestañas/tablas**: "Productos más comprados", "Productos mejor calificados" y "Mejores compradores".
- **Carga asíncrona**: cada pestaña consulta su endpoint y renderiza listas de máximo 10 filas con su paginación.

Gestión de Categorías (MK-005, `categories/GestionCategorias.html`).

- **Búsqueda**: filtros por id, nombre y descripción; invoca `./api/listar/` y llena la tabla administrativa.
- **CRUD**: Edición con el modal (`#modal`) y POST a `./api/guardar/`; borrado con `./api/eliminar/{id}/`.

Gestión de Productos (MK-004, `products/GestionProductos.html`).

- **Búsqueda**: filtros por id, nombre, autor y categoría.
- **Edición**: modal con campos (id, fecha, calificación no editable, nombre, autor, versión, categoría, precio, descripción).
- **Imágenes**: subida, borrado y reordenamiento por índice; vista previa y actualización del orden.
- **Archivo descargable**: endpoints para subir y descargar el archivo del producto.

Gestión de Usuarios (MK-006, `usuarios/GestorUsuarios.html`).

- **Búsqueda**: por id, nombre y saldo; tabla paginada y acciones por fila.
- **Edición**: actualización de saldo y estado ("excliente").

Perfil Cliente (MK-008, `users/PerfilCliente.html`).

- **Modo lectura/edición**: los campos *nombre*, *correo* y *contraseña* alternan entre *readOnly* y modo editable con botones `Editar`/`Cancelar`; se conservan valores originales por si se cancela.
- **Historial paginado**: visualización de compras con paginación.

4.4.3 Arquitectura de estilos (CSS)

El sistema de estilos es modular:

- `variables.css`: tipografías (`--font-sans`), escala de tamaños, espacios, paleta de colores, radios, sombras.
- `base.css`: Los estilos para etiquetas base (html, body, img, button, etc.) sin necesidad de parametros de *class*, *id*.
- `layout.css`: Los estilos para diseños estructurados, como `header` y paginadores.

- `components.css`: Los estilos para los componentes de cada página reutilizables, como tarjetas (`.card`), botones (`.btn`, variantes `--ghost`/`--accent`), formularios (`.form-row`), tablas y `.modal`.
- `utilities.css`: Los estilos individuales, como margin (`.mt-2`, `.mb-4`, partes para `.flex`, `.between`, `.wrap`, `.items-end`, y otros.

4.5 Integración mínima entre frontend y backend

Se describe el mecanismo de comunicación entre ambas capas del sistema **BlockFile**. El frontend, desarrollado con **HTML, CSS y JavaScript**, consume los servicios del backend implementado en **Django**, intercambiando datos mediante el protocolo HTTP con el formato **JSON**.

4.5.1 Comunicación y flujo de datos

El proceso general de comunicación sigue los siguientes pasos:

1. El usuario interactúa con la interfaz (por ejemplo, presiona un botón o envía un formulario).
2. El script JavaScript captura el evento y ejecuta una petición asíncrona `fetch()` hacia un endpoint REST del backend.
3. El backend recibe la solicitud, la procesa a través de sus capas (*view, serializer, service, repository, model*) y accede a la base de datos **PostgreSQL**.
4. El backend genera una respuesta en formato **JSON** o, en algunos casos, un archivo binario (por ejemplo, imágenes o descargas de productos).
5. El frontend interpreta la respuesta y actualiza el contenido de la página.

4.5.2 Protocolos y formato de intercambio

La comunicación se realiza mediante peticiones HTTP de tipo `GET` y `POST`:

- `GET`: Recupera información (por ejemplo, listado de productos, rankings o comentarios).
- `POST`: Envía datos o ejecuta operaciones (registro, compras, comentarios, CRUD de gestores).

Los datos se envian en formato **JSON**.

Ejemplo de flujo completo

Un ejemplo es la acción de comprar un producto:

1. El cliente hace clic en “Comprar”.
2. El script JS ejecuta `fetch('/vista-producto/producto/api/{id}/comprar', {method:'POST'})`.
3. El backend valida saldo y registro de compra; si es correcto, devuelve `{"ok":true,"nuevo_saldo":23.5}`.
4. El frontend actualiza la vista, desactiva el botón “Comprar” y habilita las opciones “Descargar”, “Calificar” y “Comentar”.

Chapter 5

Presentación y Conclusiones

5.1 Demostración del prototipo

En esta sección se muestra las vistas de los prototipos del proyecto **BlockFile**

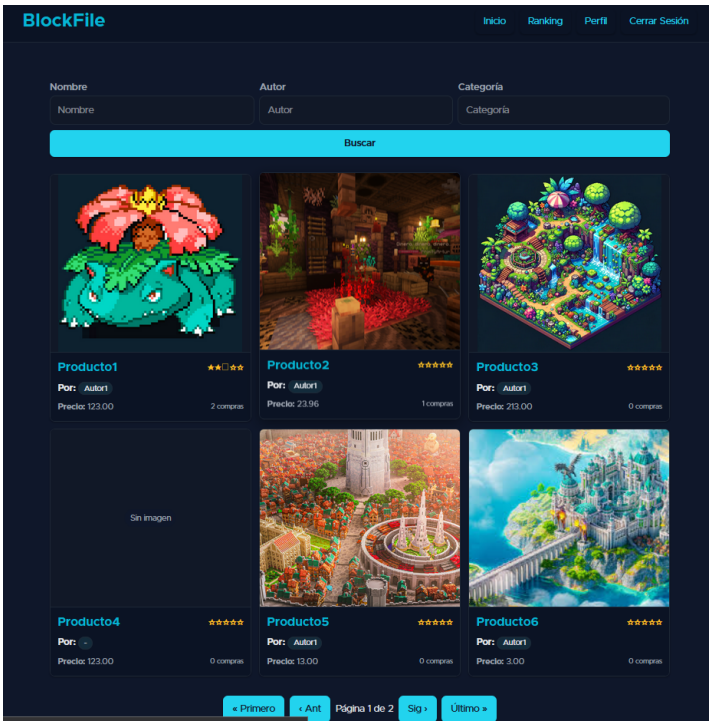

Figure 5.1 – Prototipo del catalogo de productos vista oscura desde del cliente

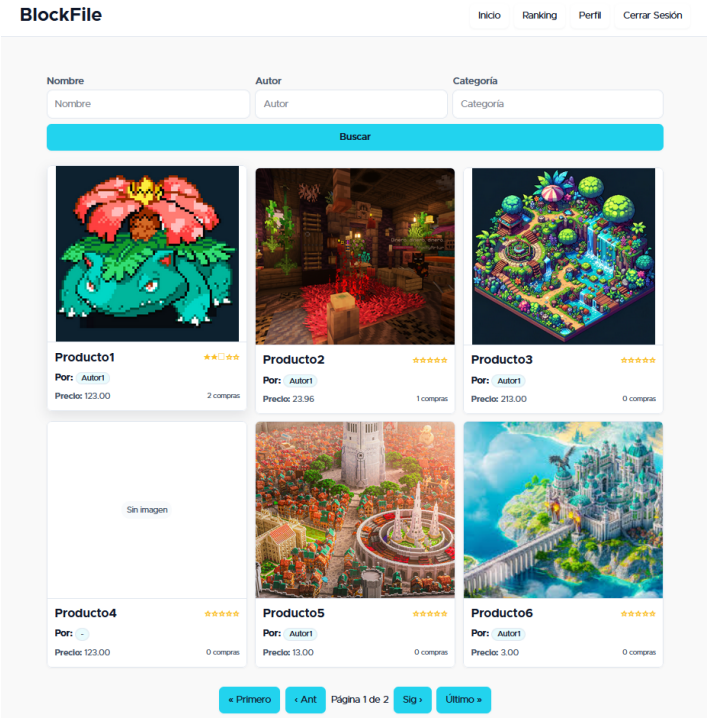

Figure 5.2 – Prototipo del catalogo de productos vista clara desde del cliente

5.2 Resultados y aprendizajes

Se obtuvo un desarrollo completo del backend y frontend usando django aprendiendo de estructuras y arquitecturas como el MVT y el trabajo por capas.

Se aprendio a realizar un documento de ejecución base para futuros proyectos mejorarlo.

En el frontend se uso los templates y static de django aprendiendo de html, css y java script.

Se aprendió de diseñar y desarrollar un modelo de datos y plasmarlo en postgresql usando consultas CRUD.

Se obtuvo el despliegue completo del proyecto en Railway aprendiendo como desplegar un proyecto en django desde cero.

5.3 Posibles mejoras futuras

Se pude migrar el frontend a un framework one page como es Angular permitendo trabajar con una arquitectura definida.

Se puede mejorar la estética css de los templates usando animaciones y un diseño a la carga de pantalla.

Se puede implementar microservicios como autenticación con correos de google y pasarelas de pago envez de que el administrador le tenga que agregar manualmente.

Chapter 6

Anexos

6.1 Código fuente y documentación básica

El código fuente es publicado en el control de versiones GitHub, donde se desarrolla el proyecto **BlockFile**, dichos accesos son los siguientes:

- **Acceso al Backend y Frontend**: Acceso al Backend
- **Acceso a la documentación**:

6.2 Capturas de pantalla del prototipo

Chapter 7

Sección Administrativa

Esta sección aborda los aspectos administrativos que permiten una gestión integral y ordenada de todos los elementos de **BlockFile**.

7.1 Codificación

A continuación se especifican las directrices para la asignación de nombres y códigos a los distintos componentes del sistema de **BlockFile**.

7.1.1 Requisitos

Para la codificación de los requerimientos funcionales se usa la sintaxis inicial ”RF” con una enumeración secuencial iniciando con ”-001” y si hay requerimientos hijos se agrega la enumeración secuencial iniciando con ”-1”. Del mismo modo, para los requerimientos no funcionales, cambiando la sintaxis inicial siendo ”RNF”.

Table 7.1 – Codificación de Requisitos

Código	**Nombre**
RF-001	Gestión del acceso al portal
RF-001-1	Ingreso al Portal
RF-001-1-1	Formulario de acceso al portal
RF-001-1-2	Validación de credenciales en la base de datos
RF-001-1-3	Redirección del acceso según Rol
RF-001-2	Registro de Cliente
RF-001-2-1	Formulario de registro al portal
RF-001-2-2	Validación de credenciales de registro de usuario
RF-001-2-3	Redirección de registro de clientes
RF-002	Vista del *Header* para Clientes
RF-003	Vista del *Header* para Administradores
RF-004	Información mostrada y resumida para un producto

Table 7.1.a Codificación de Requisitos

Código	**Nombre**
RF-005	Paginador de información
RF-006	Filtro de búsqueda de Productos
RF-006-1	Cantidad Mostrada de productos en el Catálogo
RF-007	Filtrado de búsqueda para la gestión de Categorías
RF-008	Tabla de información de las categorías
RF-008-1	Botón para editar categoría
RF-008-2	Botón para agregar categorías
RF-009	Perfil de Administrador
RF-010	Perfil de Cliente
RF-010-1	Historial de Cliente
RF-011	Filtrado de búsqueda para la gestión de Productos
RF-012	Tabla de información de los productos
RF-012-1	Botón para editar el producto
RF-012-1-1	Sección de imágenes para el producto
RF-012-2	Botón para agregar productos
RF-013	Filtrado de búsqueda para la gestión de Usuarios
RF-014	Tabla de información de los usuarios
RF-014-1	Botón para editar el usuario
RF-015	Sección para los rankings
RF-015-1	Ranking productos más comprados
RF-015-2	Ranking de mejores compradores
RF-015-3	Ranking de productos mejores calificados
RF-016	Vista general del producto
RF-016-1	Información principal del producto para su vista general
RF-016-2	Comentarios del producto en su vista general
RF-016-3	Botón de compra del producto en su vista general
RF-016-4	Características del producto en su vista general
RF-016-5	Opciones del producto comprado en la vista general del producto
RNF-001	Seguridad
RNF-002	Usabilidad
RNF-003	Portabilidad
RNF-004	Escalabilidad
RNF-005	Mantenibilidad

7.1.2 Interfaces

Las interfaces son codificadas con el nombre MK y se les asigna un número único y secuencial. En el caso de componentes específicos de los mockups, como las barras laterales, se utilizará el prefijo "MK", seguido de un guión y una secuencia de letras descriptivas (por ejemplo, BNA para la barra navegación del administración), y a continuación, otro guión y un número secuencial.

Table 7.2 – Codificación de Interfaces

Código	Nombre
MK-001	IniciarSesion
MK-002	Registrarse
MK-003	PerfilAdministrador
MK-004	GestionProductos
MK-005	GestionCategorias
MK-006	GestionUsuarios
MK-007	CatalogoProductos
MK-008	PerfilCliente
MK-009	Rankings
MK-010	VistaProducto
MK-BNA-1	BarraNavegacionAdmin
MK-BNC-1	BarraNavegacionCliente

7.1.3 Entidades

Las Entidades en las base de datos son codificadas con el nombre BD y se les asigna un número único y secuencial.

Table 7.3 – Codificación de Entidades

Código	Nombre
BD-001	USUARIO
BD-002	ADMINISTRADOR
BD-003	CLIENTE
BD-004	PRODUCTO
BD-005	COMPRA_PRODUCTO
BD-006	COMENTARIO_PRODUCTO
BD-007	CALIFICACION_PRODUCTO
BD-008	CATEGORIA

Table 7.3.a Codificación de Entidades

Código	Nombre
BD-009	IMAGEN_PRODUCTO
BD-010	AUTOR
BD-011	DETALLES_PRODUCTO

7.1.4 Diagramas de Secuencia

Los diagramas de secuencia son codificadas con el nombre DS y se les asigna un número único y secuencial.

Table 7.4 – Codificación de Diagramas de Secuencia

Código	Nombre
DS-001	AccesoPortal
DS-002	PerfilAdministrador
DS-003	GestionProductos
DS-004	GestionCategorias
DS-005	GestionUsuarios
DS-006	CatalogoProductos
DS-007	PerfilCliente
DS-008	Rankings
DS-009	VistaProducto

7.1.5 Modelo de datos

El modelo entidad-relación se codifica con las letras MER y un número único secuencial. El modelo relacional se codifica con las letras MR y un número único secuencial. Finalmente, el modelo físico se codifica con las letras MF y un número secuencial.

Table 7.5 – Codificación de Interfaces

Código	Nombre
MER-001	Modelo entidad-relación
MR-001	Modelo relacional
MF-001	Modelo físico

References

[1] Minecraft Wiki. "Java edition classic – features." Accedido el 6 de septiembre de 2025. [Online]. Available: https://minecraft.fandom.com/wiki/Java_Edition_Classic#Features.

[2] nerd.nu Wiki. "Main page – nerd.nu wiki." Accedido el 6 de septiembre de 2025. [Online]. Available: https://wiki.nerd.nu/wiki/Main_Page.

[3] Minecraft Wiki. "Java edition version history." Accedido el 6 de septiembre de 2025. [Online]. Available: https://minecraft.fandom.com/wiki/Java_Edition_version_history.

[4] M. Miller. "Introducing worldedit 7." Accedido el 6 de septiembre de 2025. [Online]. Available: https://madelinemiller.dev/blog/introducing-worldedit-7.

[5] Planet Minecraft. "Planet minecraft – community for creative gamers." Accedido el 6 de septiembre de 2025. [Online]. Available: https://www.planetminecraft.com.

[6] Minecraft Wiki. "Java edition 1.2.4." Accedido el 6 de septiembre de 2025. [Online]. Available: https://minecraft.fandom.com/wiki/Java_Edition_1.2.4.

[7] Minecraft Wiki. "Spawn." Accedido el 6 de septiembre de 2025. [Online]. Available: https://minecraft.fandom.com/wiki/Spawn.

[8] Hypixel Forums. "Hypixel history #4: The complete timeline of hypixel (2012–2024) – no longer updated." Accedido el 6 de septiembre de 2025. [Online]. Available: https://hypixel.net/threads/hypixel-history-4-the-complete-timeline-of-hypixel-2012-2024-no-longer-updated.2563451.

[9] BBC News Mundo. "Microsoft compra mojang, creadora de minecraft." Accedido el 6 de septiembre de 2025. [Online]. Available: https://www.bbc.com/mundo/ultimas_noticias/2014/09/140915_ultnot_microsoft_compra_mojang_minecraft_egn.

[10] BuiltByBit. "Who we are." Accedido el 6 de septiembre de 2025. [Online]. Available: https://builtbybit.com/wiki/who-we-are.

[11] Builder's Refuge. "Builder's refuge – creative minecraft community." Accedido el 6 de septiembre de 2025. [Online]. Available: https://buildersrefuge.com.

[12] Minecraft Wiki. "Mini games." Accedido el 6 de septiembre de 2025. [Online]. Available: https://minecraft.fandom.com/wiki/Mini_games.

[13] Planet Minecraft. "Minecraft projects – minigames." Accedido el 6 de septiembre de 2025. [Online]. Available: https://www.planetminecraft.com/projects/tag/minigames.

[14] YouTube Wiki. "Killercreeper55 – minecraft: Sus mapas." Accedido el 6 de septiembre de 2025. [Online]. Available: https://youtube.fandom.com/es/wiki/KillerCreeper55_-_Minecraft#Sus_Mapas.

[15] Eufonia Studio. "About – eufonia studio." Accedido el 6 de septiembre de 2025. [Online]. Available: https://eufonia.studio/about.

[16] BuiltByBit. "Minecraft builds – resources marketplace." Accedido el 6 de septiembre de 2025. [Online]. Available: https://builtbybit.com/resources/categories/minecraft-builds.3.

[17] H. Altman. "Horace altman – portfolio and creative works." Accedido el 6 de septiembre de 2025. [Online]. Available: https://horacealtman.com.

[18] V. Builds. "Varuna – professional minecraft builds." Accedido el 6 de septiembre de 2025. [Online]. Available: https://varunabuilds.com.

[19] Everbloom Games. "Minecraft maps – everbloom games." Accedido el 6 de septiembre de 2025. [Online]. Available: https://everbloomgames.com/minecraft-maps.

[20] Enterprise Arquitect. "Sparx systems - enterprise arquitect." Accedido el 10 de agosto de 2025. [Online]. Available: https://sparxsystems.com/products/ea/17.1.

[21] Figma Inc. "Figma – collaborative interface design tool." Accedido el 13 de agosto de 2025. [Online]. Available: https://www.figma.com.

[22] "OMG Unified Modeling Language (OMG UML), Version 2.5.1," Object Management Group, Tech. Rep., 2017, Disponible en línea. [Online]. Available: https://www.omg.org/spec/UML/.