

MEMORIA FINAL DEL PROYECTO

BURNING CREST



CICLO FORMATIVO DE GRADO SUPERIOR

DESARROLLO DE APLICACIONES WEB

AUTOR

DANIEL ABELLÁN ZÁRATE

TUTOR Y COORDINADOR

JESÚS VIVES CESPEDES

ÍNDICE

1.	INTRODUCCIÓN	4
2.	INTRODUCCIÓN EN INGLÉS	5
3.	OBJETIVOS	6
3.1.	Objetivos Fase Actual	6
3.2.	Objetivos Fases Futuras	6-7
4.	PLANIFICACIÓN	8
4.1.	Tabla de hitos	8
4.2.	Diagrama de Gantt	8
5.	ANÁLISIS	
5.1.	Estado del arte	9
5.2.	Funcionalidades	9-10
6.	DISEÑO	11
6.1.	Requisitos técnicos	11
6.2.	Arquitectura web	12-13
6.3.	Diseño back-end	13
6.3.1.	Modelo de datos	14-16
6.3.2.	Paquetes adicionales	16
6.4.	Diseño front-end	17
6.4.1.	Mock-ups	17-21
6.4.2.	Guía de estilos	21-22
6.4.3.	Capturas de la aplicación	22-24
7.	IMPLEMENTACIÓN	24
7.1.	Servidor	25-29
7.2.	Cliente	30-34
8.	DESPLIEGUE	35
8.1.	Modelo de despliegue utilizado	35
8.2.	Datos iniciales y configuración	35
8.3.	Pasos para el despliegue	35
8.4.	Proveedores y servicios utilizados	36

9. HERRAMIENTAS UTILIZADAS	37
10. MANUAL DE USUARIO	38-51
11. CRÉDITOS DE RECURSOS	52

1. INTRODUCCIÓN

En este documento se recoge toda la información relacionada con el módulo del Proyecto final del CFGS de DAW (Desarrollo de Aplicaciones Web). En dicho proyecto se han hecho uso de los conocimientos adquiridos durante el ciclo, como el análisis, planificación, diseño, organización, implementación y ejecución de un servicio web.

Burning Crest consiste en un proyecto sobre la realización de un videojuego RPG con una estética retro, en la que sus usuarios pueden participar en combates contra otros usuarios o contra la CPU, fortalecer sus personajes mediante mecánicas de subidas de nivel, obtención de habilidades que puedan o no impactar al rendimiento de dichos personajes en combate y cambios de “clases” u “oficios”, afectando a múltiples parámetros y permitiendo la obtención de habilidades adicionales, con el objetivo final de brindar a los usuarios un alto nivel de personalización de su batallón personal y el desarrollo de numerosas estrategias de combate distintas.

2. INTRODUCCIÓN EN INGLÉS

This document gathers all the info related with the Final Project module from the DAW (Desarrollo de Aplicaciones Web) CFGS. In said project, all of the knowledge acquired during the grade has been used, like analysis, planification, design, management, implementation and deployment of a web service.

Burning Crest's project consists in the making of a retro-style RPG videogame, where its users can participate in battles against each other or against the CPU, make their characters stronger via level up mechanics, obtention of skills which may or may not affect their performance during battles and "class" or "job" changes, which affect multiple parameters and allowing the acquisition of additional skills, with the final objective of bringing the users a high capability of customization for their personal army and the development of numerous unique combat strategies.

3. OBJETIVOS

El objetivo de este proyecto es el de crear un videojuego RPG, en el que los usuarios tendrán una serie de personajes que podrán desplegar en mapas, enfrentándose en combate a múltiples enemigos u otros usuarios, empleando múltiples estrategias con el movimiento, posicionamiento y habilidades de sus personajes para obtener la victoria. Gracias al sistema de clases y armas, se pretende aportar a los usuarios un gran abanico de posibilidades de personalización para sus personajes, modificando las probabilidades de subida de estadísticas durante las subidas de nivel, obteniendo sinergias entre habilidades y armas y moldeándolos para que cumplan roles específicos que se acomoden a sus gustos personales (un personaje que se encargue de recibir ataques sufriendo el menor daño posible, otro personaje que se dedique exclusivamente a hacer el mayor daño posible sin tener en cuenta su fragilidad, etc.).

3.1. Objetivos Fase Actual

Los objetivos de la fase actual del proyecto es establecer la base funcional en lo que respecta a todo el procesado relacionado con el sistema de combate, el crecimiento de los personajes mediante las subidas de nivel, el cambio de clases iniciales a clases avanzadas y la obtención de múltiples habilidades y el impacto de las mismas en los cálculos de combate. También se implementa el sistema de victorias/derrotas y la obtención de recompensas correspondientes.

3.2. Objetivos Fases Futuras

Para las fases futuras, la primera adición sería la de incorporar el sistema de armas y objetos, además de un inventario para los usuarios y uno particular para las unidades, permitiendo que lleven múltiples armas que puedan cambiar durante el desarrollo de los mapas, así como objetos pasivos o consumibles que les afecten de múltiples formas. Posteriormente, la incorporación de los mapas en los que se desplegarán las unidades y donde transcurrirán los enfrentamientos, tras lo que se centraría el proyecto en la

incorporación del sistema online para que los múltiples usuarios puedan interactuar y combatir entre sí.

4. PLANIFICACIÓN

4.1. Tabla de hitos

- 3 a 9 de Marzo -> Planificación e investigación sobre tecnologías y arquitectura a utilizar para el proyecto
- 10 de Marzo a 13 de Abril -> Diseño del modelo de datos y realización del CRUD
- 14 de Abril a 12 de Mayo -> Realización de la parte de Administración, utilizando los métodos de CRUD creados previamente para aportar funcionalidad a las diferentes acciones que podrán realizarse
- 12 a 31 de Mayo -> Realización de la parte funcional de la vista de Cliente, incluyendo la pantalla animada del combate, así como los sistemas de victorias/derrotas, promoción de clases y obtención de nuevas unidades
- 1 a 7 de Junio -> Aplicación de los diseños planeados para generar el aspecto de la vista del Cliente mediante CSS y JavaScript
- 7 a 9 de Junio -> Revisión del correcto funcionamiento del proyecto, así como búsqueda de posibles errores o detalles erróneos

4.2. Diagrama de Gantt



Imagen 1. Diagrama de Gantt del proyecto

5. ANÁLISIS

5.1. Estado del arte

Existen muchos otros juegos de ámbito online con mecánicas similares en cuanto a la obtención de nuevos personajes respecta, pero en general todo el sistema de funcionamiento de este proyecto se ha pensado desde un punto de vista propio sin fijarse en las tendencias actuales del mercado y la monetización de los mismos

5.2. Funcionalidades

- Sistema de inicio y cierre de sesión: Permite a los usuarios tanto registrarse en la web como iniciar sesión con sus datos
- Página inicial: Una vez iniciada sesión, los usuarios pueden acceder a distintos apartados, donde podrán realizar diversas acciones, siendo éstas “My Units”, “Matchmaking”, “Promotion” y “Shop”.
- Apartado “My Units”: Permite al usuario comprobar todas las unidades que tiene actualmente, pudiendo acceder a información sobre cada una si clica en ellas
- Información de unidad: Seleccionando una unidad en la sección previa, podrán verse con alto nivel de detalle, como su nivel, su clase, sus estadísticas y crecimientos actuales y las habilidades que tienen actualmente equipadas. También se podrá acceder al cambio de las posibles habilidades equipadas
- Cambio de habilidades: Habiendo accedido desde la sección de información de una unidad, se pueden cambiar las habilidades activas de la misma de entre todas las opciones disponibles
- Apartado “Matchmaking”: Los usuarios acceden a una pantalla de selección de sus unidades, donde elegirán la unidad con la que querrán realizar el combate. Tras ello, serán trasladados a una pantalla de selección de rival, donde saldrá una lista del resto de los usuarios con una de las unidades que tengan seleccionada, indicando la que han elegido para combatir con el usuario. Tras haber seleccionado el rival, se procederá a la pantalla de combate, donde, una vez realizados los cálculos, se mostrarán por pantalla las estadísticas relevantes

y las acciones de ambas unidades. Una vez terminado el combate, se pasará finalmente a la pantalla de resultados finales, con la información relevante en cuando a subidas de nivel o recompensas.

- Apartado “Promotion”: Entrando aquí, se mostrarán las unidades que pueden promover de clase en el momento, ninguna se muestra si no se da el caso o un aviso al usuario si carece del objeto necesarios para realizarlas. En el caso de poder y ver alguna unidad, al pulsar sobre ella se pasará a una pantalla en la que se podrá elegir entre todas las posibles promociones. Tras darle a una de las opciones, se pasará a una pantalla informativa sobre la promoción realizada.
- Apartado “Shop”: Una vez accedida a esta sección, podrá accederse a la opción de consumir un vale de lotería para poder obtener una unidad nueva. Si se pulsa dicha opción, se pasará a una pantalla informativa con la nueva unidad obtenida.

6. DISEÑO

6.1. Requisitos técnicos

En cuanto a lo que respecta a requisitos técnicos para el funcionamiento de la aplicación, para poder realizar el sistema de control de sesiones se hace uso de las Sesiones de PHP, mientras que la pantalla de cambio de habilidades hace uso de JavaScript para permitir actualizar los iconos y las descripciones conforme el usuario realiza el cambio en el selector. Por último, las animaciones de combate, así como los sonidos del mismo, requieren del uso de JavaScript y CSS.

6.2. Arquitectura web

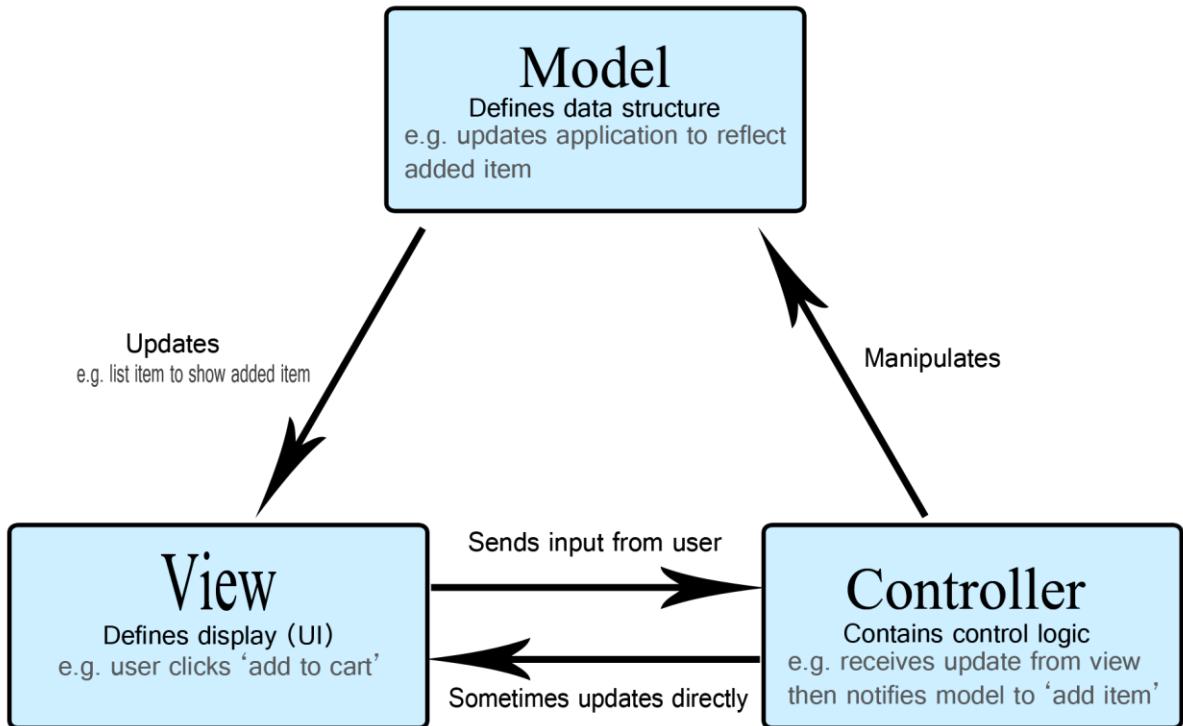


Imagen 2. Esquema de la arquitectura MVC (Modelo Vista Controlador)

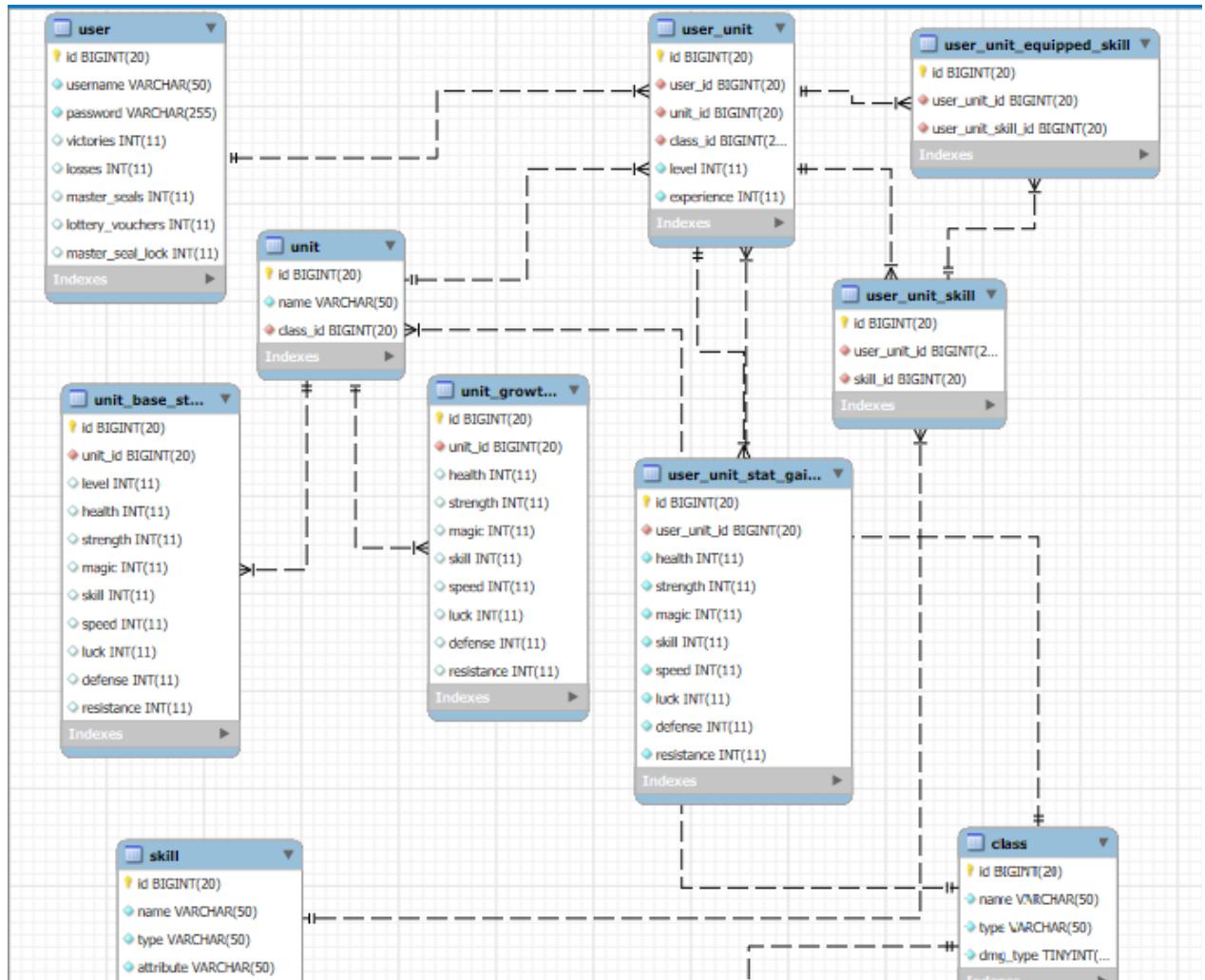
Para el proyecto, se ha hecho uso de la estructura Modelo Vista Controlador (MVC). Ésta consiste en la separación en capas de las distintas partes de la aplicación para aumentar la segmentación de las distintas partes de la aplicación en la máxima medida de lo posible, aumentando además la seguridad. La capa del Modelo es en la que se definen las operaciones sobre la base de datos, es decir, el CRUD, y la parte más sensible a nivel de seguridad de la aplicación. La parte de la Vista es la que define todas las páginas HTML o PHP relacionadas con lo que verá el usuario durante su uso de la aplicación, y que será totalmente desconocedora de los métodos y funcionalidades internas del resto de la aplicación. Por último, el Controlador es la parte que actúa como intermediaria entre el Modelo y la Vista, permitiendo a la Vista recibir datos de la base de datos, pero sin saber

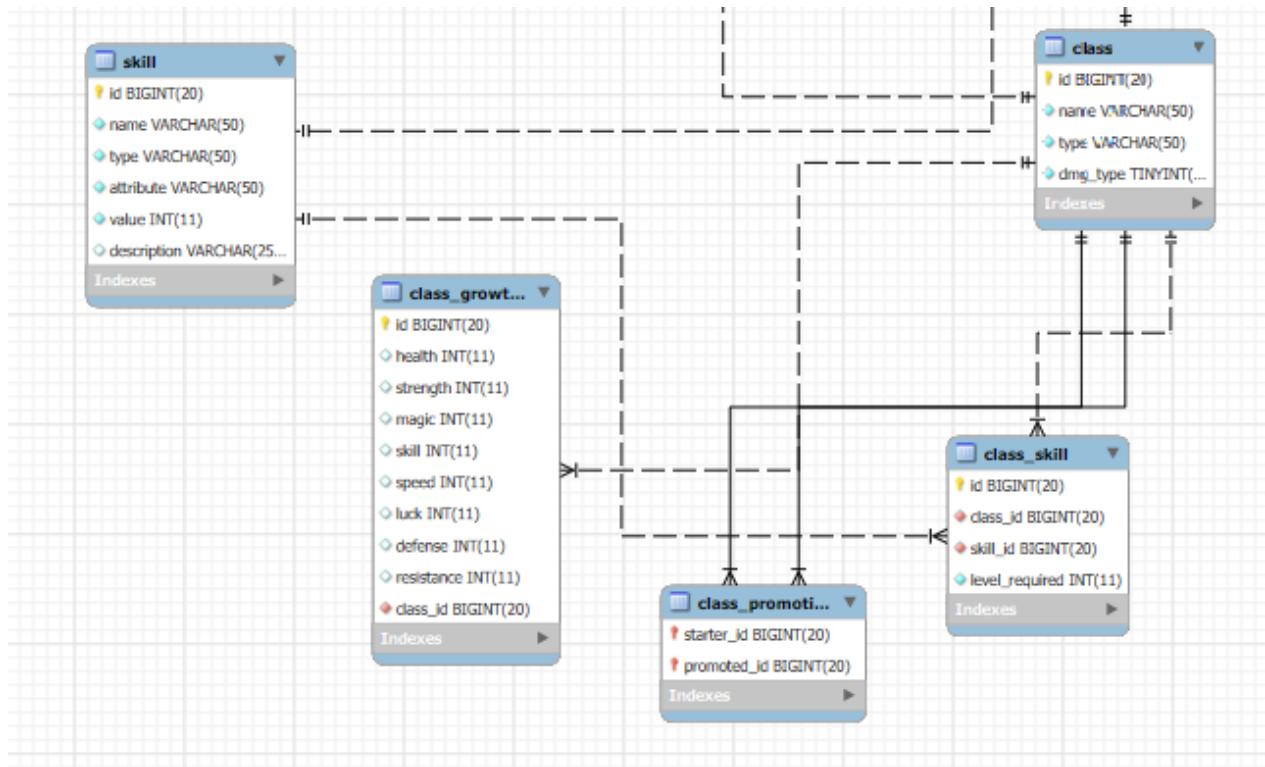
exactamente cómo se están obteniendo o modificando dichos datos. Ésta parte es la que asegura la mayor seguridad posible entre los datos sensibles del servidor y las acciones del usuario.

6.3. Diseño back-end

En el caso de este proyecto, por ciertos motivos de restricciones de tiempo, he utilizado PHP básico para realizar el funcionamiento del back-end, creando un pseudo-enrutador que funciona mediante la toma de ciertos parámetros (siempre estarán presentes, y en caso de la falta de alguno de ellos, se redirigirá a una página por defecto) en la URL de la web, haciendo saber a la aplicación en qué página se encuentra en el momento actual. También se ha creado un fichero especializado para mantener el control de la sesión iniciada por parte del usuario, así como un fichero para realizar la conexión con la base de datos siempre que sea necesario

6.3.1. Modelo de datos





Imágenes 3 y 4. Modelo de datos del proyecto

Proporcionando una sencilla explicación del modelo de datos mostrado en la imagen, empezando por la tabla más esencial y básica: la de usuarios “*user*”. En ella se guardan tanto los parámetros de nombre de usuario y contraseña como los de control de victorias, derrotas y objetos de promoción de clase y de lotería de unidades.

Después está la tabla “*class*”, donde se especifican los detalles básicos de las clases que podrán tener las distintas unidades, destacando un valor para definir si es de tipo inicial o avanzada y otro para indicar si el daño que realice la unidad será físico o mágico. Después se encuentra la tabla “*class_growths*”, conteniendo los crecimientos específicos de cada una de las clases. Los crecimientos son porcentajes que indicarán la probabilidad de subir estadísticas en un punto a la hora de subir de nivel. Por último, la tabla “*class_promotion*” mantiene relaciones entre dos clases, donde se indica las clases avanzadas a las que podrá evolucionar una clase inicial.

Continuando, la tabla “*skill*” se utiliza para almacenar las distintas habilidades que podrán obtenerse. La variable de tipo se utiliza para establecer cuándo se encontrará activa la habilidad (cuando inicia el combate o no, si es un simple bono de estadísticas,

etc.), mientras que la de atributo a qué estadística afectará (acierto, fuerza, defensa, etc.). La clase “*class_skill*” se utiliza para atribuir habilidades a las clases, estableciendo a partir de qué nivel podrán obtenerse subiendo de nivel en dicha clase.

Siguiendo, la tabla “*unit*” se utiliza para almacenar las distintas unidades que habrán disponibles en el juego. La siguiente, “*unit_base_stats*” sirve para establecer las estadísticas iniciales que tendrá cada una de las unidades. Después está “*unit_growths*”, que al igual que con las clases, establecen los crecimientos de las unidades. Al subir de nivel, la probabilidad de que la unidad suba el valor de una estadística será el de sus crecimientos sumados a los de la clase en los que se encuentre actualmente.

Terminando con la última serie de tablas, “*user_unit*” se utiliza para establecer qué unidades tendrá cada uno de los usuarios. Esta tabla es clave en el funcionamiento de la aplicación, pues debido a la naturaleza de cómo funcionan las subidas de niveles, dos usuarios pueden fácilmente tener la misma unidad con estadísticas diferentes, ya que pueden haber tenido aumentos de estadísticas distintos en la misma cantidad de niveles cada uno. Además, almacena el nivel y puntos de experiencia actuales de la misma. La siguiente tabla, “*user_unit_stat_gains*”, se utiliza precisamente para almacenar esos aumentos de estadísticas específicos de cada unidad obtenidos con cada nivel ganado. La tabla “*user_unit_skill*” almacena todas las habilidades que cada una de las unidades de un usuario haya ido aprendiendo conforme ha subido de nivel estando en clases específicas. Por último, “*user_unit_equipped_skill*” sirve para indicar, entre esas habilidades que ha aprendido, cuáles están equipadas actualmente por acción directa del usuario.

6.3.2. Paquetes adicionales

En este proyecto únicamente se han utilizado dos paquetes adicionales: Bootstrap para aportar un aspecto básico en el lado de la administración y JQuery para el lado del cliente, permitiendo realizar un par de funcionales adicionales de forma más sencilla que con JavaScript básico.

6.4. Diseño front-end

En la parte de front-end, se han utilizado templates básicas de HTML para mostrar el contenido, así como JavaScript para añadir algunas funcionalidades adicionales, como la actualización de datos instantánea a la hora de cambiar las habilidades de una unidad o permitir la animación de los combates. También se ha realizado una cierta comunicación entre back y front mediante el envío de datos recogidos de la base de datos desde PHP a JavaScript, quien los ha recogido y utilizado, siendo el caso las descripciones e iconos de las habilidades para permitir su cambio instantáneo como ha sido mencionado previamente. Se ha utilizado SASS para establecer el aspecto de CSS de forma más sencilla y organizada.

6.4.1. Mockups



Imagen 5. Mockup de la pantalla Home o página inicial



Imagen 6. Mockup de la página "Unit Info" o información de una unidad del usuario



Imagen 7. Mockup de la página de "Change Skills" o cambio de habilidades de una unidad

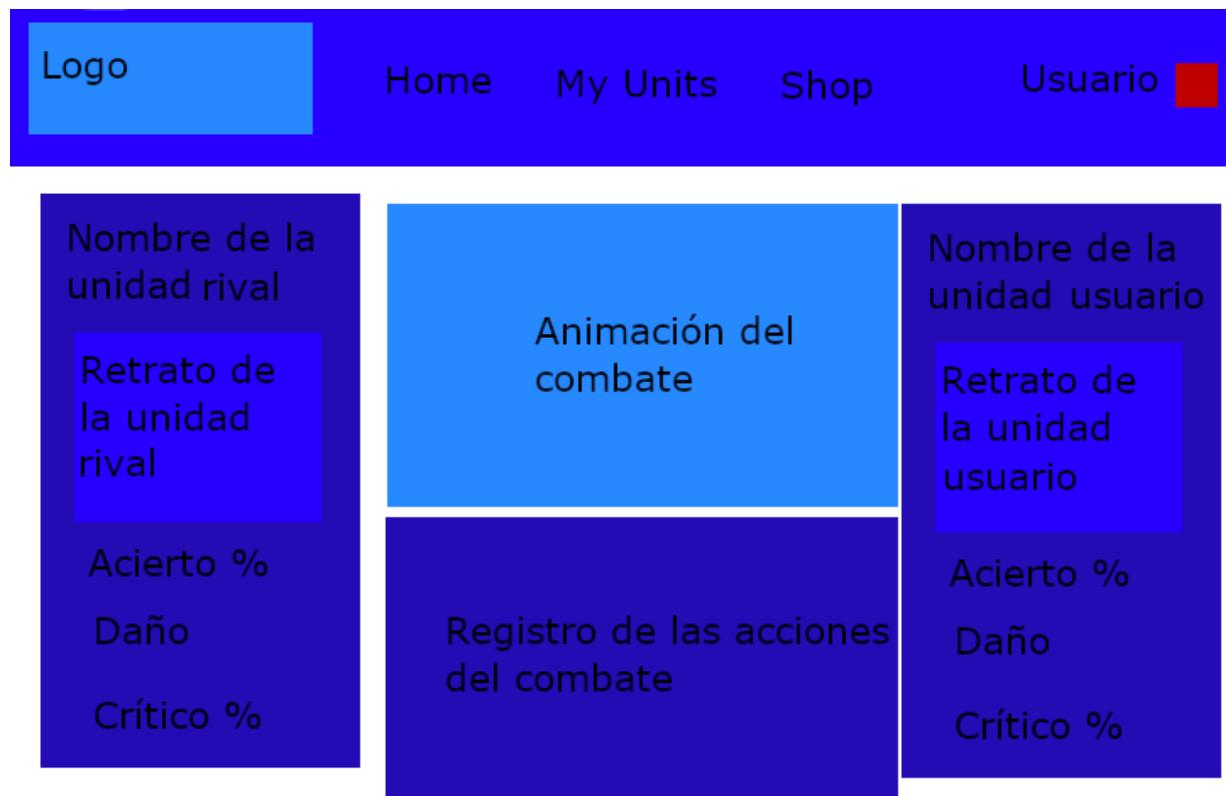


Imagen 8. Mockup de la pantalla del combate entre dos unidades



Imagen 9. Mockup de la pantalla "Results" o resultado final tras un combate

6.4.2. Guía de estilos

Al querer obtenerse una estética retro de videojuego, se han utilizado colores sencillos que recuerdan a la época de las consolas de finales de los 80 y principios de los 90 (como la SNES), por lo que se han seleccionado gamas de grises y azules para los distintos elementos de la página.



#2700ff

#240cb5

#07031c

#e9e9e9

#dfdfdf

En cuanto al tamaño de la letra, se ha utilizado un tamaño estándar de 16px, aumentándolo para secciones de título u opciones de menú.

Por último, como iconos se han utilizado recursos de juegos ya preexistentes para dar un aspecto cercano al que se obtendría al final, donde los iconos y resto de recursos sería realizado por un equipo de diseño.

6.4.3. Capturas de la aplicación

Las capturas correspondientes a los mockups presentados en la sección anterior son los siguientes:



Imagen 10. Sección "Home" o página principal

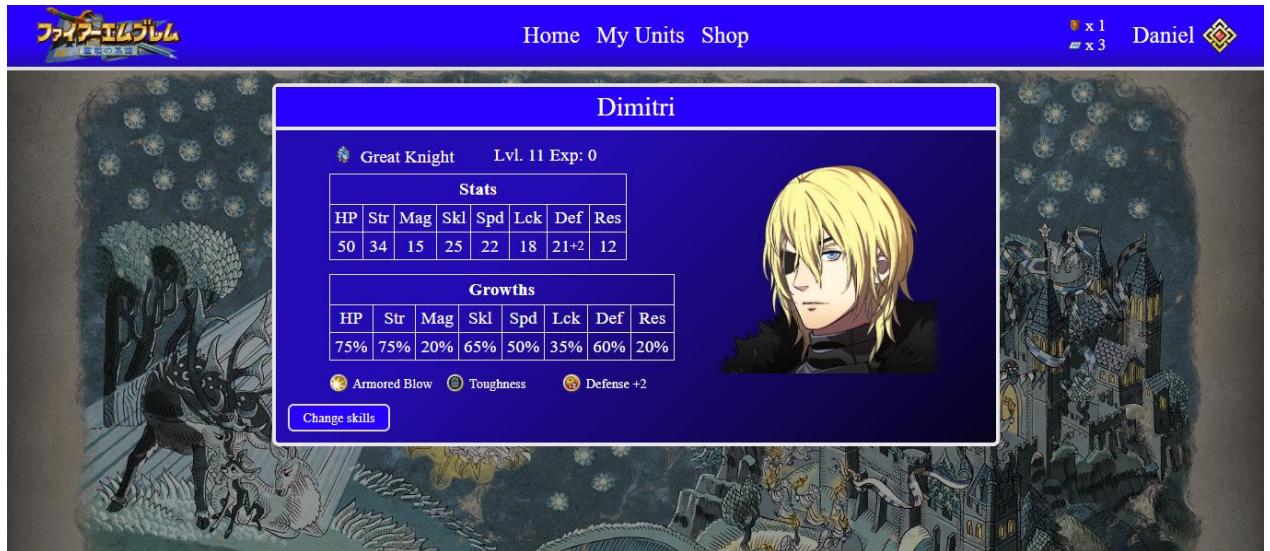


Imagen 11. Sección "Unit Info" o información de una unidad específica



Imagen 12. Sección "Change Skills" o cambio de habilidades de una unidad



Imagen 13. Sección "Battle" o vista del combate entre dos unidades



Imagen 14. Sección "Results Screen" o pantalla de resultados del combate

7. IMPLEMENTACIÓN

7.1. Servidor

Para comenzar, se ha establecido un fichero para mantener la conexión con la base de datos, realizado de la siguiente forma:

```
<?php
class db{
    const HOST = "localhost";
    const DBNAME = "tfg";
    const USER = "root";
    const PASSWORD = "";

    static public function connection(){
        $connection = null;

        try {
            $options = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
            $connection = new PDO('mysql:host=localhost; dbname=' . self::DBNAME, self::USER, self::PASSWORD, $options)
        } catch (Exception $e) {
            echo "There's been a problem with the database: " . $e->getMessage();
        }

        return $connection;
    }
}
```

Imagen 15. Muestra del fichero de configuración de la conexión con la BD

Tras ello, para permitir el funcionamiento de la web, se ha creado un fichero de enrutado que funciona mediante los parámetros obtenidos en la URL de la web. Si no son correctos, se redirigirá a una pantalla por defecto:

```
<?php
function router (){
    $url = $_SERVER["REQUEST_URI"];

    //If nothing is added in the URL, the user is redirected to the home page
    if (substr($url,-1)!=""){
        return "views/user/home.php";
    }

    //If the route format is wrong, the user is redirected to the home page
    if (strpos($url,"index.php")){
        return "views/user/home.php";
    }

    //If no table has been defined, the user is redirected to the home page
    if (!isset($_REQUEST["table"])){
        return "views/user/home.php";
    }

    $table = $_REQUEST["table"];

    $tables = [
        "user" => [
            "login" => "login.php",
            "authentication" => "authentication.php",
            "register" => "register.php",
            "registration" => "registration.php",
            "home" => "home.php",
            "logout" => "logout.php"
        ],
        "units" => [
            "showAll" => "showAll.php",
            "unitDetail" => "unitDetail.php"
        ],
        "shop" => [
            "main" => "main.php",
            "obtainUnit" => "obtainUnit.php"
        ]
    ];

    //If the table introduced is not registered, the user is redirected to the home page
    if (!isset($tables[$table])){
        return "views/user/home.php";
    }

    //If no action is defined, the user is redirected to the home page
    if (!isset($_REQUEST["action"])){
        return "views/user/home.php";
    }

    $action = $_REQUEST["action"];

    //If the action introduced is not registered, the user is redirected to the home page
    if (!isset($tables[$table][$action])){
        return "views/user/home.php";
    }

    return "views/{$table}/{$tables[$table][$action]}";
}
```

Imágenes 16 y 17. Muestra del enrutador, mostrando las acciones permitidas y la forma de operar que tendrá en caso de que los parámetros introducidos no sean válidos

Una vez establecidos los ficheros básicos de la aplicación, se han realizado los ficheros de los Modelos conectados a la base de datos, conteniendo todas las operaciones CRUD necesarias para el funcionamiento de la aplicación:

```
<?php
require_once('config/db.php');
require_once('baseModel.php');

class ClassModel implements BaseModel{
    private $connection;
    public function __construct(){
        $this->connection = db::connection();
    }

    /**
     * Inserts a new entry in the database's "class" table
     *
     * @param array $class Contains the values for each field in the table
     */
    public function insert(array $class): int | null {
        try {
            $sqlQuery = "INSERT INTO `class`(`name`, `type`) VALUES (:name, :type);";
            $preparedQuery = $this->connection->prepare($sqlQuery);
            $dataArray = [
                ":name" => $class["name"],
                ":type" => $class["type"]
            ];
            $result = $preparedQuery->execute($dataArray);

            return ($result == true) ? $this->connection->lastInsertId() : null;
        } catch (PDOException $e) {
            return null;
        }
    }
}
```

Imagen 18. Ejemplo del Modelo de la entidad "Class", mostrando en este caso el método de inserción de una nueva fila en la tabla de la base de datos

Tras realizar los ficheros para el Modelo, se han realizado los ficheros para manejar los Controladores, que permitirán la conexión segura entre la Vista y el Modelo:

```

<?php
require_once "models/classModel.php";
require_once "assets/php/functions.php";

class ClassController{
    private $model;

    public function __construct(){...}

    /**
     * Finds an entry in the database's "class" table and returns its info
     *
     * @param int $id ID of the row whose information will be retrieved
     *
     * @return stdClass|null Returns an "stdClass" type if a matching row was found, null if it wasn't
     */
    public function read(int $id): stdClass | null {
        return $this->model->findById($id);
    }

    /**
     * Finds an entry in the database's "class_growths" table and returns its info
     *
     * @param int $id ID of the row whose information will be retrieved
     *
     * @return stdClass|null Returns an "stdClass" type if a matching row was found, null if it wasn't
     */
    public function readGrowths(int $id): stdClass | null {
        return $this->model->findbyIdGrowths($id);
    }
}

if($dataIsValid) {
    //Format checks
    $dataFormatisValid = true;
    $growthCheckList = [
        "health_growth",
        "strength_growth",
        "magic_growth",
        "skill_growth",
        "speed_growth",
        "luck_growth",
        "defense_growth",
        "resistance_growth"
    ];

    $dataFormatisValid = isValidFormDataFormat("growth", $growthChecklist, $classdataArray);
    if($dataFormatisValid) {
        $newClassId = $this->model->insert($classdataArray);

        if($newClassId != null){
            $newClassGrowthsId = $this->model->addGrowths($newClassId, $classdataArray);

            if($newClassGrowthsId != null){
                header("location:index.php?table=class&action=show&id=" . $newClassId);
                exit();
            }else{
                header("location:index.php?table=class&action=create&error=true");
                exit();
            }
        }else{
            header("location:index.php?table=class&action=create&error=true");
            exit();
        }
    }
}

```

Imágenes 19 y 20. En la primera se muestra cómo el Controlador hace llamadas simples al Modelo. En la segunda, se muestra el método de creación de una nueva entrada en la BD,

viendo cómo se realizan comprobaciones y validaciones, tras lo que se hacen llamadas (o no) al modelo dependiendo de los resultados obtenidos en ellas

Por último, los ficheros HTML/PHP dedicados a la muestra del contenido a los usuarios se han realizado de una forma similar a la siguiente:

```
<?php
require_once "config/config.php";
require_once "assets/php/functions.php";
require_once "controllers/UserUnitController.php";

$userId = $_SESSION["user"]["id"];
$userUnitController = new UserUnitController();
$unitList = $userUnitController->listByUserId($userId);
?>

<div class="my-units">
    <?php
    foreach($unitList as $index => $unitData){
        $portraitRoute = "assets/img/unit/" . $unitData->unit_id . "/" . strtolower($unitData->class_type) . "/default.png";
    ?>
        <div class="my-units_unit">
            <div class="unit_link">
                <a href="index.php?table=units&action=info&id=<?= $unitData->id ?>"></a>
            </div>
            <div class="unit_name">
                <span><?= $unitData->name ?></span>
            </div>
            <div class="unit_image-wrapper">
                name ?>">
            </div>
        </div>
    <?php } ?>
</div>
```

Imagen 21. Vista de una de las páginas más sencillas a nivel de estructura de la web: la que muestra todas las unidades de un usuario. Puede comprobarse cómo se añaden pequeños trozos de código PHP para formar la estructura final HTML que será vista por el usuario, y cómo los datos específicos se han obtenidos gracias a la comunicación con el Controlador desde la Vista

Por último, se ha creado un fichero de “functions.php” que contiene varias funciones misceláneas, como las comprobaciones de formatos, la comprobación de si el usuario tiene o no alguna unidad o el procesado de todos los cálculos requeridos en el combate, aunque no se aportarán capturas en este preciso documento debido a la longitud del código de dichos cálculos es bastante extenso

7.2. Cliente

Para el lado del cliente, los ficheros de JavaScript necesarios para las funcionalidades adicionales se han colocado en los mismos directorios que contienen aquellas vistas a las que van a afectar:

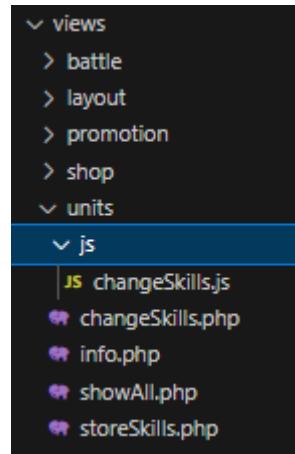


Imagen 22. Localización de ejemplo de dónde se ubican en el proyecto aquellos ficheros JavaScript que afectarán a una vista concreta. En este caso, el fichero “changeSkills.js” afectará a la vista “changeSkills.php”, que es el fichero que contiene la estructura HTML

En todos sus usos se hace una espera a que el contenido de la página se haya cargado, tras lo que se pondrán en marcha. En el caso del selector de las habilidades, se recogen los datos y se colocan EventListeners:

```
window.onload = () => {
    let referencesDiv = document.getElementById("skill-references");
    let referencesDataJSON = referencesDiv.innerHTML;
    referencesDiv.remove();

    let descriptionsDiv = document.getElementById("skill-descriptions");
    let descriptionsDataJSON = descriptionsDiv.innerHTML;
    descriptionsDiv.remove();

    let references = JSON.parse(referencesDataJSON);
    let descriptions = JSON.parse(descriptionsDataJSON);

    let skillSelectors = document.querySelectorAll("select");
    let skillDescriptions = document.getElementsByClassName('skill-selector__description');

    for(let index in skillSelectors){
        let selectorElement = skillSelectors[index];
        let iconElement = selectorElement.previousElementSibling;
        let descriptionElement = skillDescriptions[index];

        selectorElement.onchange = () => {
            let userUnitSkillId = selectorElement.value;
            let newIconRoute = "";
            let newDescription = "";
            if(userUnitSkillId != 0){
                newIconRoute = "assets/img/skill/" + references[userUnitSkillId] + "/icon.png";
                newDescription = descriptions[references[userUnitSkillId]];
            }else{
                newIconRoute = "assets/img/icon/unknownSkill.png";
                newDescription = "No skill selected";
            }
            iconElement.setAttribute("src", newIconRoute);
            descriptionElement.innerHTML = newDescription;
        }
    }
}
```

Imagen 23. Parte del código del fichero "changeSkills.js", donde se establece a cada selector un comportamiento para actualizar su ícono y descripción correspondientes conforme el usuario va cambiando la opción seleccionada

Por otro lado, el funcionamiento del combate funciona principalmente con captura de información y la realización de distintas acciones, marcadas por esperas de tiempo para permitir la sincronización de todas las acciones de forma secuencial:

```
window.onload = () => {
    let divWithData = document.getElementById("battle-data");
    let battleDataJSON = divWithData.innerHTML;
    divWithData.remove();

    let battleLogArray = JSON.parse(battleDataJSON);
    let battleLogDiv = document.getElementById("battle-log");

    > const classNamesArray = [...];
    >

    let userUnit = document.getElementById("user-unit");
    let userIdleImg = userUnit.getAttribute("src");
    let userAttackImg = userIdleImg.replace("idle.png", "attack.gif");
    let userCriticalImg = userIdleImg.replace("idle.png", "critical.gif");

    let rivalUnit = document.getElementById("rival-unit");
    let rivalIdleImg = rivalUnit.getAttribute("src");
    let rivalAttackImg = rivalIdleImg.replace("idle.png", "attack.gif");
    let rivalCriticalImg = rivalIdleImg.replace("idle.png", "critical.gif");

    let hitSfx = new Audio("assets/sound/hit.wav");
    hitSfx.loop = false;
    let criticalSfx = new Audio("assets/sound/critical.wav");
    criticalSfx.loop = false;
    let missSfx = new Audio("assets/sound/miss.wav");
    missSfx.loop = false;
    let unitDefeatedSfx = new Audio("assets/sound/unitDefeated.wav");
    unitDefeatedSfx.loop = false;

    let animationDelay = 500;
    let accumulatedAnimationDelay = 1000;

    for(let action in battleLogArray){
        let isUserAction = action.indexOf("user");
        let isRivalAction = action.indexOf("rival");
        let isAttack = action.indexOf("Attack");
        let isCritical = action.indexOf("Critical")
        let isMiss = action.indexOf("Miss");
        let isDefeat = action.indexOf("Defeat");
        let actionClass = "";

        for(let index in classNamesArray){ ... };
        >

        let animationLength = 0;
        let textDelay = 0;
        let textCritDelay = 0;
        switch(actionClass){
            case "Mercenary":
                if(isAttack != -1 || isMiss != -1){
                    animationLength = 2500;
                }else if(isCritical != -1){
                    animationLength = 4500;
                }
                textDelay = 0;
                textCritDelay = 1250;
                break;
            case "Myrmidon": ...
            case "Archer": ...
        }
    }
}
```

```

let animationTotalTime = animationLength + animationDelay;

//Setting up the animations in the correct order
setTimeout(() => {
    if(isUserAction != -1){
        if(isAttack != -1) {
            userUnit.setAttribute("src", userAttackImg);
        }else if (isCritical != -1){
            userUnit.setAttribute("src", userCriticalImg);
        }else if (isMiss != -1){
            userUnit.setAttribute("src", userAttackImg);
        }else if(isDefeat != -1){
            userUnit.classList.add("battle_unit--defeated");
        }
        userUnit.classList.add("battle_unit--active");
    } else if(isRivalAction != -1){
        if(isAttack != -1){
            rivalUnit.setAttribute("src", rivalAttackImg);
        }else if(isCritical != -1){
            rivalUnit.setAttribute("src", rivalCriticalImg);
        }else if(isMiss != -1){
            rivalUnit.setAttribute("src", rivalAttackImg);
        }else if(isDefeat != -1){
            rivalUnit.classList.add("battle_unit--defeated");
        }
        rivalUnit.classList.add("battle_unit--active");
    }
}, accumulatedAnimationDelay);

let extraTextDelay = isCritical != -1 ? textCritDelay : textDelay;
//Text Shown
setTimeout(() => {
    battleLogDiv.innerHTML += battleLogArray[action] + "<br>";
    if(isAttack != -1){
        hitSfx.play();
    }else if(isCritical != -1){
        criticalSfx.play();
    }else if(isMiss != -1){
        missSfx.play();
    }else if(isDefeat != -1){
        unitDefeatedSfx.play();
    }
}, (accumulatedAnimationDelay) + ((animationLength/2) + extraTextDelay));

//Idle Stance Return
setTimeout(() => {
    if(isUserAction != -1){
        userUnit.setAttribute("src", userIdleImg);
        userUnit.classList.remove("battle_unit--active");
    } else if(isRivalAction != -1){
        rivalUnit.setAttribute("src", rivalIdleImg);
        rivalUnit.classList.remove("battle_unit--active");
    }
}, (accumulatedAnimationDelay) + animationLength);

accumulatedAnimationDelay += animationTotalTime;
}

```

Imágenes 24, 25, 26 y 27. En la primera se puede observar la recuperación de los datos sobre las acciones realizadas en el combate enviada desde el back-end y la carga de los recursos necesarios, tanto imágenes como ficheros de sonido. En la segunda se puede observar cómo se establecen los tiempos requeridos para espaciar y permitir la sincronización de todas las acciones una por una. En la tercera puede verse cómo se establece qué animación se va a realizar dependiendo de los parámetros obtenidos en la acción que se está recorriendo actualmente. En la última puede verse cómo se establecen

tanto la aparición del texto de combate así como cuándo han de ejecutarse los efectos de sonido.

8. DESPLIEGUE

8.1. Modelo de despliegue utilizado

En el caso de este proyecto, se ha realizado una instalación directa en el mismo servidor, utilizando XAMPP para poder ejecutar Apache y desplegar la web.

8.2. Datos iniciales y configuración

Se han insertado unos datos iniciales en la base de datos respecto a todas las tablas: una serie de usuarios, incluyendo un administrador, así como una serie de clases, habilidades y unidades, todas con sus valores de crecimientos, estadísticas base, enlaces de promoción entre clases, etc.. También se han asignado ciertas unidades por defecto a algunos de los usuarios, permitiendo la funcionalidad directa de la aplicación sin tener que realizar modificaciones en la base de datos. Todo esto se encuentra en el mismo fichero de creación de las tablas en la base de datos incluido en los ficheros del proyecto.

8.3. Pasos para el despliegue

Para poder realizar el despliegue, se ha instalado XAMPP en la máquina del servidor, quien ya viene con defecto con PHP y Apache, facilitando el trabajo a la hora de reducir las tecnologías que han de instalarse para el funcionamiento del despliegue mismo. En el caso actual se han asignado permisos completos al usuario del tipo 777 sobre la carpeta del mismo XAMPP, aunque en un caso real habrían de establecerse permisos más restrictivos para evitar severos agujeros de seguridad. Una vez todo esto está realizado, se han colocado los ficheros del proyecto dentro un directorio que esté a su vez situado dentro de la carpeta “htdocs” del fichero raíz de XAMPP. Cuando se hayan colocado, bastará con arrancar los servicios de Apache y MySQL, instalar el fichero de

la base de datos con la herramienta SQL que guste (o con el mismo PHPMyAdmin que viene con XAMPP) y acceder a localhost para visualizar el servicio web.

8.4. Proveedores y servicios utilizados

Se ha utilizado un servicio del tipo EC2, aprovechando el uso de la creación de instancias y la aplicación de una IP elástica, para el despliegue del proyecto, mediante el uso de Amazon Web Service (AWS) con una licencia educativa para estudiantes.

9. HERRAMIENTAS UTILIZADAS

- PHP -> Lenguaje utilizado para el funcionamiento de la parte del back-end e inyección de código en las plantillas HTML
- JavaScript -> Lenguaje utilizado para añadir ciertas funcionalidades a la aplicación desde el lado del front-end y, especialmente, para la animación del combate
- CSS y SASS -> CSS es el lenguaje utilizado para aportar el aspecto a la aplicación. Para generar el fichero CSS maestro se ha utilizado SASS, que ha permitido una mejor organización y separación de los elementos que forman la estructura de las distintas vistas
- XAMPP -> Utilizado para montar la aplicación en un servidor simulado, facilita la implementación de PHP y Apache en el proyecto, así como un gestor de BD predeterminado con PHPMyAdmin
- Visual Studio Code -> Entorno de desarrollo utilizado para la creación del proyecto
- HeidiSQL -> Gestor de base de datos utilizado para realizar operaciones en base de datos de forma manual durante las fases iniciales del desarrollo del proyecto, así como durante la fase de testeo
- GIMP 2 -> Editor gráfico utilizado para obtener los recursos deseados de las listas de imágenes e iconos obtenidas en los recursos, así como la generación de los mockups de la parte del front-end
- Git y GitHub -> Utilizado como sistema de control de versiones, mientras que GitHub ha sido la página seleccionada para establecer los repositorios remotos
- Canva.com -> Realización del diagrama de Gantt
- MySQL Workbench -> Realización del esquema del Modelo de datos

10. MANUAL DE USUARIO

Nada más entrar al sitio web, puede observarse una página de inicio de sesión. En ella se pueden introducir los datos de usuario necesarios, tras lo que se pulsa el botón para comprobar que sean correctos. Otra opción es pulsar el link de la parte inferior de la ventana, lo que llevará a la página de creación de un nuevo usuario

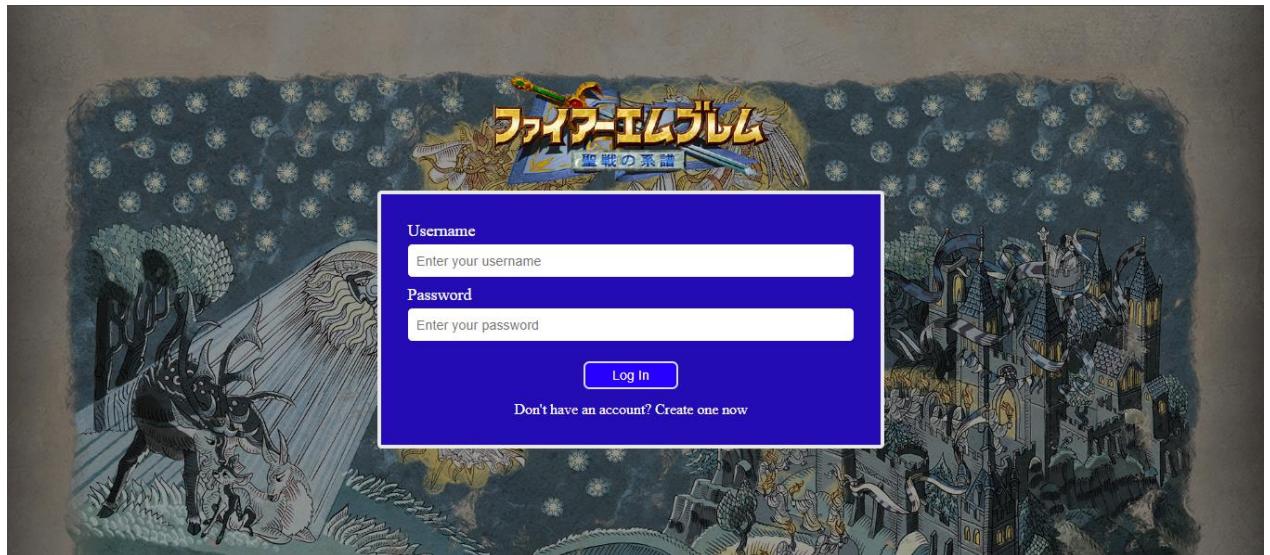


Imagen 28. Vista de la página de inicio de sesión

Si se pulsa la creación de un nuevo usuario, se mostrará un formulario. En él, el usuario introducirá el nombre de usuario que quiera tener (siempre que no se encuentre registrado de forma previa, caso en el que surgirá un aviso por pantalla). Tras introducir una contraseña y validarla repitiéndola en el segundo campo, puede pulsarse el botón de Crear una nueva cuenta. Si se cambia de opinión, se puede pulsar el botón para volver a la página de inicio de sesión



Imagen 29. Vista de la página de registro

Una vez se haya iniciado sesión, se verá la pantalla principal de la aplicación. Desde aquí, puede verse en la parte superior derecha los recursos que tiene el usuario (el ícono superior indica el número de promociones que puede realizar actualmente, mientras que el inferior indica el número de nuevas unidades que puede obtener en la tienda), así como su nombre de usuario y el ícono de cierre de sesión justo a su derecha. Las cuatro opciones principales se muestran en la parte central de la pantalla



Imagen 30. Vista de la página principal

Habiendo pulsado la opción “My Units” del menú principal, se pasará a ver todas las unidades que tiene el usuario. Inicialmente los usuarios no tienen ninguna unidad hasta que no inician sesión por primera vez, tras lo que la aplicación le asignará aleatoriamente 3 unidades entre todas las disponibles. Clicando en cualquier de ellas llevará al usuario a ver un cuadro de información específico sobre cada una de ellas. Nótese también como en la barra de navegación ha aparecido un menú de accesos directos: éste siempre será visible en cualquier página que no sea la principal



Imagen 31. Vista de la página ”My Units”

En esta página puede verse información concreta sobre la unidad seleccionada: su clase, nivel y puntos de experiencia actuales, las estadísticas que posee y sus crecimientos, que indican la probabilidad de incrementar cada una de las estadísticas cuando la unidad suba de nivel. El botón “Change skills” permite cambiar las habilidades activas de la unidad, siempre que la unidad las haya aprendido con anterioridad

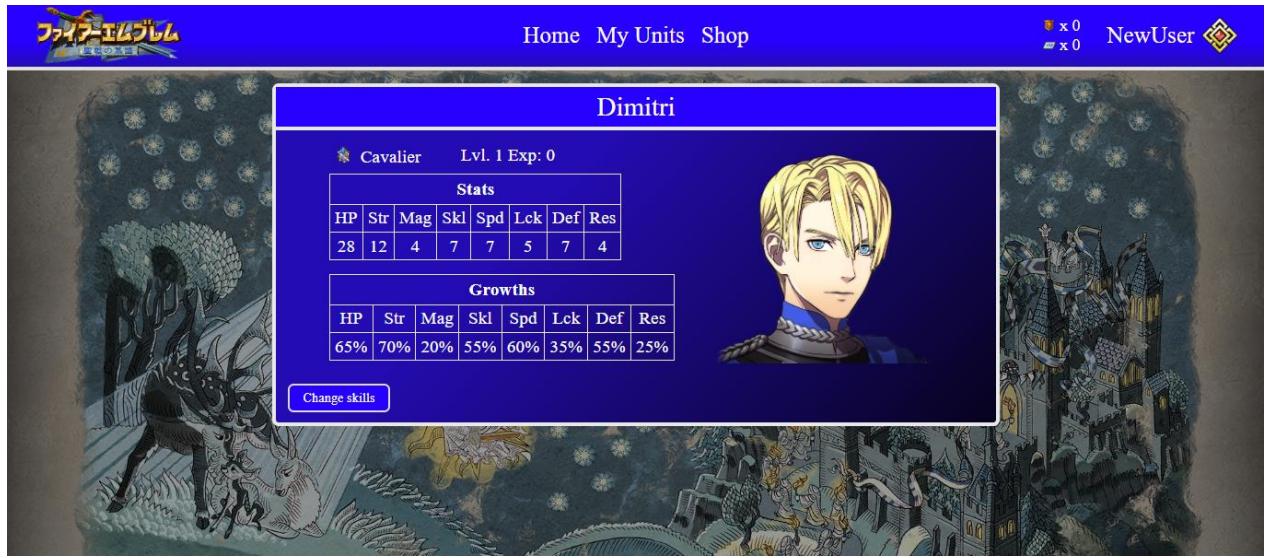


Imagen 32. Vista de la información detallada de una unidad

En esta pantalla el usuario podrá seleccionar las habilidades que la unidad utilizará durante la fase de juego. Nótese que aunque las habilidades puedan ser seleccionadas más de una vez, únicamente se harán efectivas una vez. Cuando se haya terminado de elegir, se puede pulsar el botón para volver a la página previa realizando los cambios

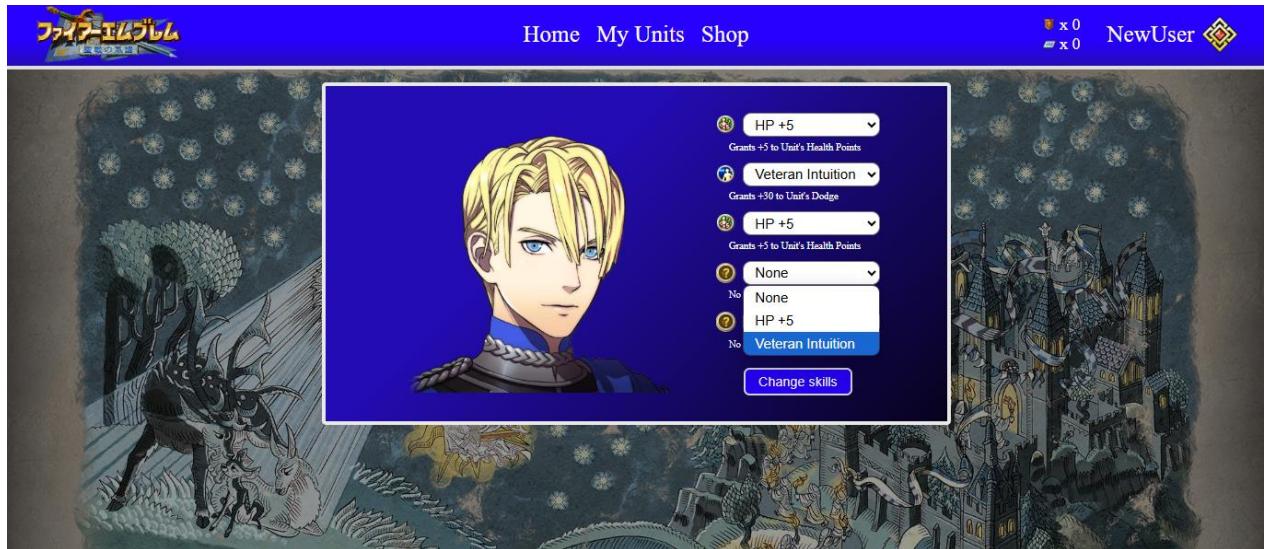


Imagen 33. Vista de la pantalla de cambio de las habilidades equipadas



Imagen 34. Vista de la actualización de la página de información al cambiar las habilidades

Volviendo al menú principal, si se pulsa la opción “Matchmaking”, comenzará la preparación de una batalla. La lista que se mostrará contendrá todas aquellas unidades del usuario, mostrando la información de sus estadísticas efectivas y los iconos de aquellas habilidades que tenga actualmente equipadas. Para seleccionar aquella con la que se quiera realizar el combate, bastará con clicar sobre ella

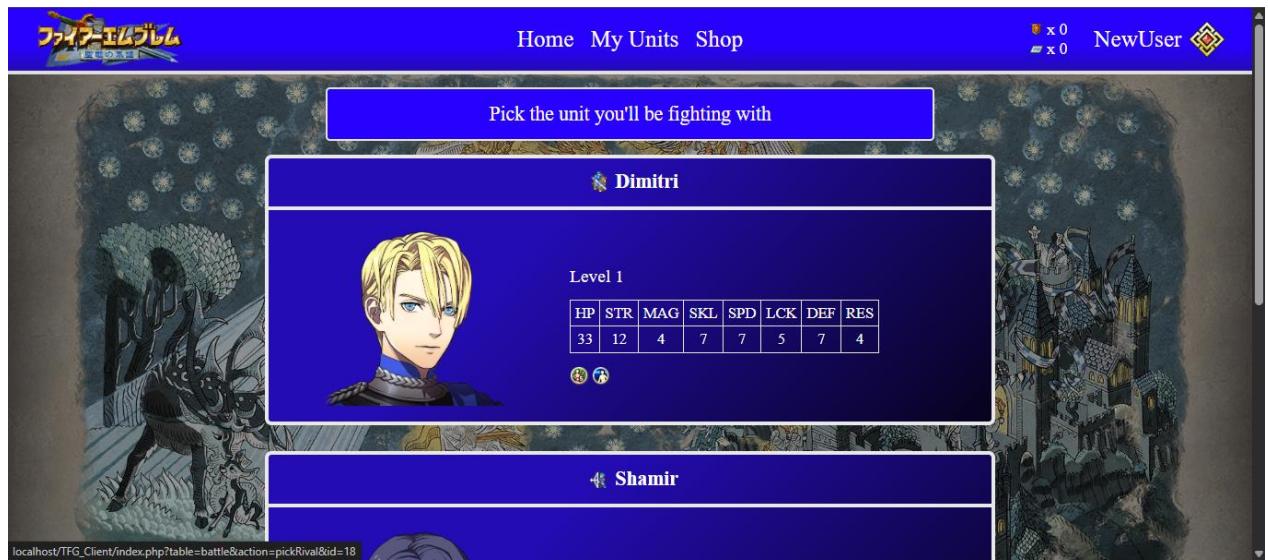


Imagen 35. Vista de la selección de una unidad para combatir

A continuación se mostrará una lista muy similar, excepto que en este caso se mostrará una unidad distinta por cada usuario que haya registrado en la aplicación. Al igual que en la página previa, se ha de clicar sobre aquel usuario contra el que se quiera combatir



Imagen 36. Vista de la lista de rivales disponibles para combatir

Una vez seleccionado, se observará la pantalla de combate, donde se podrá observar la información relevante de la unidad rival (lado izquierdo) y la de la unidad del usuario (lado derecho), ubicándose en el centro la animación del combate entre ambas y un registro de las acciones que van sucediendo durante el mismo. Una vez finalizado el combate, ya sea por derrota de uno de los lados o por límite de rondas, el usuario será trasladado tras unos segundos a la pantalla de resultados



Imagen 37. Vista de la interfaz del combate entre ambas unidades

En la pantalla de resultados, podrán observarse los incrementos de puntos de experiencia, nivel y de estadísticas de la unidad (estos dos últimos siempre que la unidad haya subido de nivel), así como la ganancia de habilidades, objetos de promoción de clase o vales de la tienda para poder obtener nuevas unidades. Una vez finalizado, el usuario puede pulsar el botón Home de la barra de navegación para volver al menú principal



Imagen 38. Vista de la pantalla de resultados

Si desde el menú principal se selecciona la opción de “Unit Promotion”, el usuario se verá trasladado a la pantalla de promociones. Si no tuviese objetos de promoción o unidades que pudiesen promover, se comunicaría mediante un mensaje en la pantalla. En caso de que ambas condiciones se cumplan, se listarán aquellas unidades capaces de promover actualmente

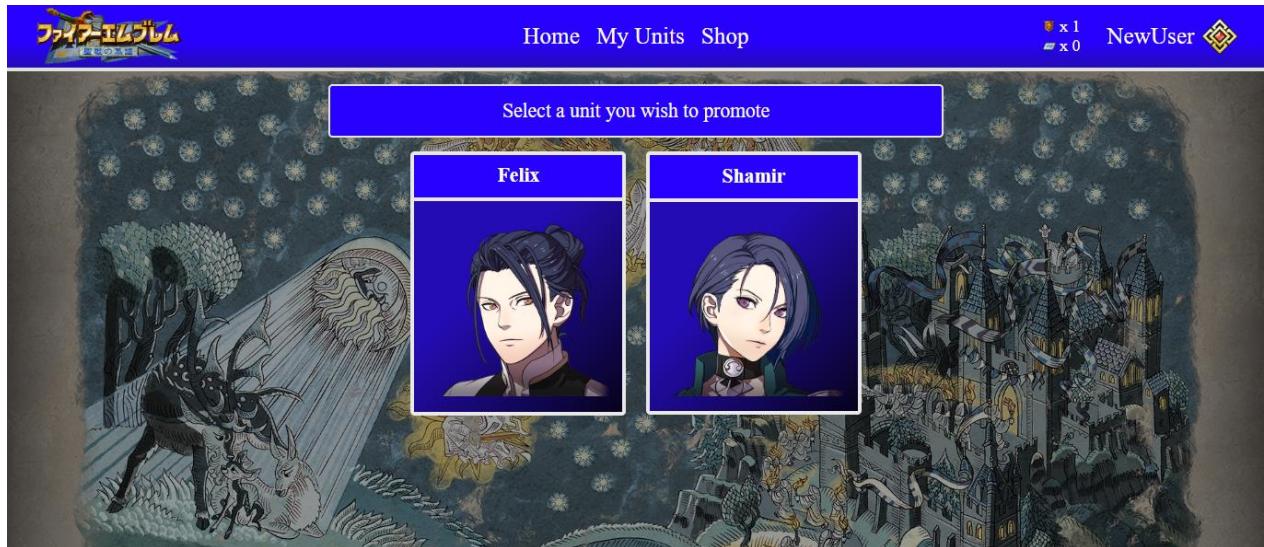


Imagen 39. Vista de las unidades disponibles para promover

Una vez seleccionada la unidad, aparecerán aquellas clases a las que puede promover desde su clase actual. Para elegir una de ellas, se clica sobre la que se deseé



Imagen 40. Vista de las clases disponibles para promover de la unidad seleccionada

Una vez seleccionada, se mostrará un mensaje informativo sobre cambio de clase y el consecuente reinicio de nivel (vuelta al nivel 1, pero sin eliminar las ganancias de estadísticas ni de habilidades) al usuario, tras lo que podrá volver al menú principal pulsando el botón Home de la barra de navegación

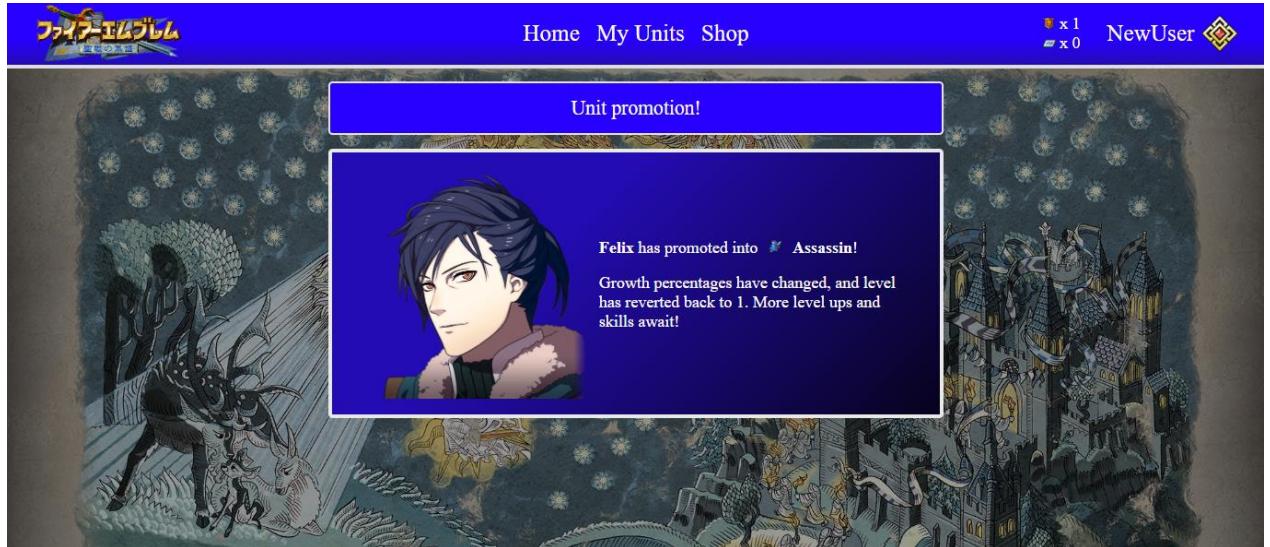


Imagen 41. Información de la promoción realizada

Como última opción del menú, el usuario puede pulsar la opción “Shop” para acceder a la tienda



Imagen 42. Opciones disponibles en la tienda accedida por el apartado "Shop"

Una vez dentro, puede seleccionar la opción “Unit Lottery” para obtener aleatoriamente una nueva unidad, siempre que tenga un vale. En caso de que no lo tenga, al intentar entrar en la opción se le avisará mediante un mensaje por pantalla. En caso de que sí lo tenga, se le mostrará la unidad que ha obtenido, junto a información básica sobre la misma, como su clase y estadísticas base

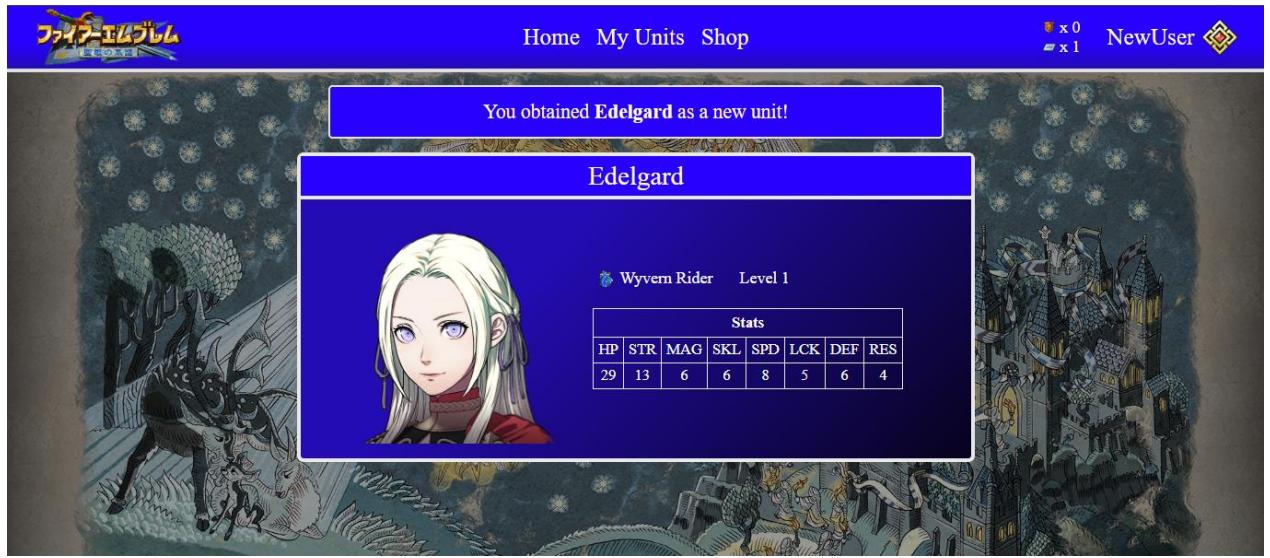


Imagen 43. Unidad obtenida tras pulsar la opción de ”Unit Lottery”

Por último, si el usuario pulsa el botón rojo situado a la derecha de su nombre de usuario, cerrará la sesión y será devuelto a la página de inicio de sesión

11. CRÉDITOS DE RECURSOS

- Nintendo e Intelligent Systems, propietarios de la saga Fire Emblem, de donde proceden todos los recursos externos utilizados en el proyecto (logo, retratos de personajes, sprites, imagen de fondo, datos de estadísticas y crecimientos, idea en sí del juego, etc.)
- Rips de los sprites, fondos de combate y diapositivas de historia de Fire Emblem: Binding Blade, Fire Emblem: Blazing Sword y Fire Emblem: The Sacred Stones por los usuarios Lexou, Drakewing e Indogutsu Tenbuki
- Rip de los retratos de personajes y elementos de interfaz de Fire Emblem: Three Houses por los usuarios Mikom, warisheck, bisaflor y Dakress
- Rip de las imágenes de los murales de Fire Emblem: Three Houses por el usuario Dakress de The Spriters Resource
- Iconos de las imágenes de los iconos de habilidades de Fire Emblem Awakening y Fire Emblem Fates por los usuarios Ploaj y trainboy2019 de The Spriters Resource
- Rip de los ficheros de sonido de Fire Emblem GBA por el usuario SuperFlomm de “The Sound Resource”