IES LOS SAUCES
BENAVENTE (ZAMORA)

TÍTULO GRADO SUPERIOR DE

DESARROLLO DE APLICACIONES WEB

PIXEL

App de gestión de proyectos de desarrollo de videojuegos

DEPARTAMENTO:	INFORMÁTICA
TÍTULO PROYECTO:	PIXEL: App de gestión de proyectos de desarrollo de videojuegos
AUTOR:	Cristian Mateos Vega
CURSO:	2025/26
PERIODO:	JUNIO
TIPO DE PROYECTO:	
CÓDIGO DE PROYECTO:	
TUTORÍA COLECTIVA:	Hermelinda Ramos Osorio
TUTORÍA INDIVIDUAL:	Heraclio Borbujo Morán
TUTORÍA COLABORADOR:	

PIXEL: App de gestión de proyectos de desarrollo de videojuegos

Desarrollo de una plataforma de gestión de proyectos, tareas, equipos y archivos basada en Laravel 12, PHP 8.2, MySQL, Tailwind CSS 4 y Vite 7

Proyecto Final DAW2

Cristian Mateos Vega

El presente proyecto de investigación y desarrollo cuyo título es "PIXEL: App de gestión de proyectos de desarrollo de videojuegos" del módulo PROYECTO ha sido realizado por Cristian Mateos Vega, alumno del IES LOS SAUCES del Ciclo Formativo Grado Superior Desarrollo de aplicaciones web con el fin de la obtención del Título de Técnico Superior en Desarrollo de Aplicaciones Web.

La tutoría individual de dicho proyecto ha sido llevada a cabo por D./Dª. Heraclio Borbujo Morán profesor de segundo curso de CFGS Desarrollo de Aplicaciones web.

En Benavente, 5 de junio de 2026

Autor	**Vº Bº Tutoría colectiva**	**Vº Bº Tutoría individual**	**Vº Bº Colaboración**
Fdo. Cristian Mateos Vega	Fdo. Hermelinda Ramos Osorio	Fdo. Heraclio Borbujo Morán	Fdo.

Introducción

El presente proyecto consiste en el desarrollo de una aplicación web de gestión de proyectos colaborativa denominada PIXEL, construida con el framework PHP **Laravel** 12. La aplicación permite a equipos de desarrolladores de videojuegos organizar proyectos, crear y asignar tareas, controlar el acceso por áreas de trabajo, invitar colaboradores mediante enlaces únicos y gestionar archivos adjuntos de una forma más concreta y especializada que la que ofrecen aplicaciones de gestión más generales como Trello o Notion.

Desde el punto de vista técnico, la aplicación implementa el patrón **MVC** propio de **Laravel**, utilizando **Eloquent ORM** para la gestión de la base de datos **MySQL**, el motor de plantillas **Blade** para las vistas y **Tailwind CSS 4** compilado con **Vite 7** para el frontend responsivo.

La documentación abarca seis objetivos: el estudio comparativo de **Laravel** frente a otros frameworks, la explicación de los componentes de una aplicación **Laravel** y sus relaciones, la puesta en marcha del entorno de desarrollo desde cero, el desarrollo incremental de la aplicación con ejemplos de código reales, el uso de **GitHub** para documentar la evolución del proyecto y, finalmente, el despliegue en el entorno de explotación Ionos con Plesk.

Como funcionalidad destacada, la aplicación incluye un foro de discusión por proyecto con chat en tiempo real basado en **WebSockets** (**Laravel Reverb** en desarrollo, **Pusher** en producción) y **Livewire**, que permite a los miembros de un proyecto intercambiar mensajes y archivos de forma instantánea sin necesidad de recargar la página.

La elección de este proyecto responde a un interés personal por el desarrollo de videojuegos y a la dificultad de encontrar herramientas de gestión de proyectos que se puedan utilizar para desarrollar un videojuego de forma clara, ordenada y organizada, siento todas demasiado generales y hasta abrumadoras para proyecto con tanta carga de trabajo a organizar como un videojuego. Esto motivó la creación de PIXEL como solución orientada específicamente a equipos pequeños y medianos de desarrollo de videojuegos, donde cada miembro solo ve y gestiona lo que le corresponde según su área estando todo organizado de forma más clara, guiada y estructurada que en otras plataformas que ofrecen mucha más libertad a la hora de personalizar la organización de un proyecto, lo cual puede acabar siendo confuso.

Agradecimientos

A D. Heraclio Borbujo Morán, tutor del proyecto, por la orientación y el seguimiento durante todo el proceso de desarrollo. Al IES Los Sauces de Benavente por los medios y la formación recibida a lo largo del ciclo formativo.

Objetivos

El presente proyecto tiene como finalidad la aplicación práctica de los conocimientos adquiridos durante el Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web, concretamente en el ámbito del desarrollo web con el framework **Laravel**. Los objetivos específicos son:

1. Estudiar el framework **Laravel**, sus características principales y realizar una comparativa con otros frameworks de desarrollo web del mercado (Symfony, Django, Ruby on Rails, Spring Boot, Express).
2. Explicar los componentes de una aplicación basada en **Laravel** (modelos, vistas, controladores, rutas, migraciones, **middleware**) y la relación entre ellos siguiendo el patrón **MVC**.
3. Documentar la puesta en marcha de un entorno de desarrollo **Laravel** desde cero, partiendo de un equipo sin software instalado: PHP con **XAMPP**, **Composer**, Node.js, Git y **MySQL**.
4. Plantear y desarrollar una aplicación web funcional con **Laravel** orientada a la gestión de proyectos colaborativa para equipos de desarrollo de videojuegos (PIXEL), mostrando ejemplos reales de cada tipo de fichero y la secuencia de desarrollo seguida. El negocio de la aplicación es proporcionar a estudios independientes y equipos pequeños una plataforma centralizada donde organizar el trabajo por áreas técnicas, controlar el acceso de cada colaborador y exportar datos del juego en formatos compatibles con los principales motores.
5. Utilizar el repositorio en **GitHub** para guardar el trabajo realizado, documentando la evolución del proyecto mediante commits.
6. Desplegar la aplicación desarrollada en el entorno de explotación Ionos con Plesk, describiendo todo el proceso de implantación en producción.

Parte Teórica: Estudio del Framework

1 Presentación de Laravel

Laravel es un framework de código abierto para el desarrollo de aplicaciones web con PHP, creado por Taylor Otwell en 2011. Sigue el patrón arquitectónico **MVC** (Modelo-Vista-Controlador) y está diseñado para hacer el desarrollo web más rápido, elegante y seguro. En este proyecto se utiliza **Laravel** 12, la versión más reciente en el momento del desarrollo.

2 Características principales de Laravel

A continuación, se describen las características más relevantes del framework, todas ellas utilizadas en el desarrollo de PIXEL:

- **Eloquent ORM** — Mapeo objeto-relacional que permite interactuar con la base de datos usando clases PHP en lugar de SQL puro. En PIXEL se usa para gestionar todas las relaciones: proyectos, tareas, participaciones, invitaciones, archivos, etc.
- **Motor de plantillas Blade** — Directivas @if, @foreach, @extends, @yield para generar HTML dinámico de forma expresiva. Todas las vistas de PIXEL están implementadas con **Blade**.
- **Artisan CLI** — Herramienta de línea de comandos para generar código, ejecutar migraciones y gestionar la aplicación.
- **Migraciones** — Control de la base de datos mediante código PHP. Cualquier desarrollador obtiene la misma estructura con **php artisan** migrate.
- **Sistema de autenticación y autorización** — Guards, policies, **middleware** y gates. En PIXEL se usa para controlar el acceso a áreas de proyectos.
- **Sistema de rutas expresivo** — Define URLs con closures o controladores de forma clara y legible.
- **Cola de trabajos (Jobs & Queues)** — Ejecución asíncrona de tareas. En PIXEL se usa en desarrollo para procesar eventos de broadcasting (envío de mensajes del foro a través de **WebSockets**).

3 Licencia — Precios

Laravel es software libre distribuido bajo la licencia MIT, lo que permite su uso, modificación y distribución de forma gratuita, tanto en proyectos personales como comerciales. Los servicios opcionales del ecosistema (Forge, Vapor, Nova) son de pago, pero no son necesarios para el desarrollo básico.

4 Comparativa con otros frameworks — Grado de implantación

La siguiente tabla resume la comparativa entre **Laravel** y los principales frameworks web del mercado:

Característica	Laravel (PHP)	Symfony (PHP)	Django (Python)	Ruby on Rails	Spring Boot (Java)	Express (Node.js)
Curva de aprendizaje	Media-baja	Alta	Media	Media	Alta	Baja
Patrón arquitectónico	MVC	MVC	MVT	MVC	MVC	Sin patrón fijo
ORM incluido	Eloquent (activo)	Doctrine (data mapper)	Django ORM (activo)	ActiveRecord	Hibernate/JPA	No (usa librerías)
Sistema de plantillas	Blade	Twig	Jinja2	ERB	Thymeleaf	No (usa librerías)
CLI propio	Artisan	Console	manage.py	Rails CLI	Spring Initializr	No
Ecosistema	Muy amplio	Amplio	Amplio	Amplio	Muy amplio	Amplio (npm)
Rendimiento	Alto	Muy alto	Alto	Medio	Muy alto	Muy alto
Popularidad	Muy alta	Alta	Muy alta	Media	Muy alta	Muy alta
Licencia	MIT	MIT	BSD	MIT	Apache 2.0	MIT

Laravel es el framework PHP más descargado según Packagist, con más de 300 millones de instalaciones, y se sitúa entre los más valorados en las encuestas anuales de Stack Overflow. Destaca frente a Symfony por ser más accesible; frente a Django la diferencia principal es el lenguaje (PHP vs Python); frente a Spring Boot la curva de aprendizaje es significativamente menor.

5 Ejemplos de aplicaciones que utilizan Laravel

- **Laracasts** — Plataforma de formación en vídeo sobre desarrollo PHP y **Laravel**.
- **Invoice Ninja** — Aplicación de gestión de facturas de código abierto.
- **Statamic** — Sistema de gestión de contenidos (CMS) moderno.
- **Forge** y **Envoyer** — Herramientas del propio ecosistema **Laravel** para despliegue y orquestación de servidores.
- Multitud de startups, agencias de desarrollo web y sistemas SaaS en todo el mundo.

Componentes de una Aplicación Laravel

Una aplicación Laravel se organiza en componentes bien definidos que siguen el patrón MVC. A continuación, se describen estos componentes, con explicación de su función, sus relaciones y ejemplos tomados directamente de PIXEL y partiendo de una instalación limpia del entorno de desarrollo.

6 Frameworks y librerías usadas

6.1 PHP — Producción (composer.json)

- laravel/framework ^12.0 — El framework completo (routing, ORM, auth, etc.).
- laravel/tinker ^2.10 — REPL interactivo para ejecutar código **Laravel** desde terminal.
- livewire/livewire ^3.0 — Framework PHP reactivo para componentes de UI sin escribir JS. Usado en el chat del foro (ChatHilo.php).
- laravel/reverb — Servidor **WebSocket** oficial de **Laravel**. Usado en desarrollo para el chat en tiempo real.
- pusher/pusher-php-server — SDK PHP de **Pusher**. Usado en producción como alternativa a **Reverb** para el broadcasting.

6.2 PHP — Desarrollo

- fakerphp/faker — Generar datos falsos para pruebas.
- laravel/pail — Ver logs en tiempo real en terminal.
- laravel/pint — Formateador de código PHP.
- laravel/sail — Entorno Docker para desarrollo.
- phpunit/phpunit — Framework de tests.

6.3 JavaScript / CSS (package.json)

- tailwindcss ^4.0 — Framework CSS utility-first.
- @tailwindcss/vite ^4.0 — Plugin de **Tailwind** para **Vite**.
- vite ^7.0 — Bundler/compilador del frontend.
- laravel-vite-plugin ^2.0 — Integración entre **Laravel** y **Vite**.
- axios ^1.11 — Cliente HTTP para peticiones desde JavaScript.
- concurrently ^9.0 — Ejecutar múltiples procesos npm en paralelo.
- pusher-js — Cliente **WebSocket** de **Pusher** para el navegador. Usado internamente por **Laravel Echo**.
- laravel-echo — Librería JS que simplifica la escucha de canales **WebSocket**. Gestiona la conexión con **Reverb** o **Pusher** desde el navegador.

7 Preparación del entorno de desarrollo desde cero

El punto de partida es un equipo sin ningún software de desarrollo instalado. Los componentes deben instalarse en un orden concreto, ya que existen dependencias entre ellos.

Para el desarrollo de PIXEL se utilizó el siguiente entorno local:

- Sistema operativo: Windows con **XAMPP** instalado.
- Servidor web: Apache 2.4 incluido en **XAMPP**.
- PHP: versión 8.2 proporcionada por **XAMPP**.
- Base de datos local: **MySQL** 8.0 gestionada con **phpMyAdmin**.
- Frontend: Node.js LTS con npm para ejecutar **Vite** y compilar **Tailwind CSS**.
- Control de versiones: Git con publicación en **GitHub**.
- IDE: Visual Studio Code con extensiones PHP Intelephense, **Laravel Blade** Snippets, GitLens y **Tailwind CSS** IntelliSense.

7.1 Orden de instalación y dependencias

Orden	Componente	Versión	Función / Razón del orden
1º	PHP 8.2 (XAMPP)	8.2+	Base de todo el stack. XAMPP incluye también Apache y MySQL, por lo que con un solo instalador obtenemos tres componentes.
2º	Composer	Última	Gestor de dependencias PHP. Requiere PHP instalado previamente para funcionar.
3º	Node.js + npm	LTS 22.x	Entorno JavaScript necesario para compilar CSS (Tailwind) y JavaScript con Vite. Se instala después de PHP porque es independiente pero lo necesitamos para el frontend.
4º	Git	Cualquiera	Control de versiones. Necesario para crear el repositorio y para que Composer y npm descarguen paquetes de GitHub.
5º	Laravel (via Composer)	12.x	El propio framework se instala via Composer una vez que todo lo anterior está listo.

> Laravel no necesita Node.js para ejecutarse, pero sí para compilar el frontend. Tailwind CSS 4 y Vite 7 son herramientas del ecosistema Node.js. Sin ellas, las hojas de estilo no se procesan y el CSS no llega al navegador. Node.js actúa únicamente en tiempo de desarrollo y build; en producción solo sirven los archivos compilados.

7.2 Instalación de PHP con XAMPP

XAMPP es el método más sencillo para instalar PHP en Windows, ya que incluye en un solo paquete PHP, el servidor web Apache y **MySQL**. Tras instalarlo y arrancar Apache y **MySQL** desde el Panel de Control de **XAMPP**, disponemos inmediatamente de:

- PHP 8.2 disponible en línea de comandos.
- Servidor web Apache en http://localhost.
- **MySQL** 8.0 con interfaz visual **phpMyAdmin** en http://localhost/phpmyadmin.

Para una app **Laravel** en **XAMPP** normalmente hay que tocar dos cosas:

1. Apache — habilitar mod_rewrite en httpd.conf:

```
LoadModule rewrite_module modules/mod_rewrite.so  ← quitar el #
```

Sin esto las rutas de **Laravel** no funcionan (todo devolvería 404).

2. PHP — extensiones en php.ini:

```
extension=pdo_mysql      ← base de datos
extension=mbstring       ← strings multibyte (Laravel lo requiere)
extension=openssl        ← cifrado, tokens, Reverb
extension=fileinfo       ← subida de archivos
extension=zip            ← Composer lo necesita
```

7.3 Instalación de Composer

Composer es el gestor de dependencias de PHP, equivalente a npm para JavaScript. Se descarga desde getcomposer.org y durante la instalación se debe indicar la ruta al ejecutable PHP de **XAMPP**. Una vez instalado, el comando composer queda disponible globalmente.

7.4 Instalación de Node.js y npm

Node.js se descarga desde nodejs.org (versión LTS). Su instalación incluye automáticamente npm (Node Package Manager). Tras instalar Node.js disponemos de las herramientas necesarias para compilar **Tailwind CSS** y ejecutar **Vite**.

7.5 Instalación de Git

Git se descarga desde git-scm.com. Durante la instalación se recomienda seleccionar VS Code como editor por defecto y permitir el uso de Git desde el terminal de Windows. Tras instalar, se configura el usuario:

```
git config --global user.name "Nombre Apellido"
git config --global user.email "tu@email.com"
```

7.6 Creación de un proyecto Laravel desde cero

Con todo el entorno preparado, se crea el proyecto:

```
# Instalar el creador de proyectos Laravel globalmente
composer global require laravel/installer

# Crear el proyecto
laravel new CMVProyectoFinalLaravel

# Entrar en la carpeta y configurar el entorno
cd CMVProyectoFinalLaravel
# Editar .env con los datos de la base de datos
php artisan key:generate    # Genera la clave de cifrado
php artisan migrate         # Crea las tablas iniciales
npm install                 # Instala Tailwind, Vite y demás dependencias JS
npm run dev                 # Compila y vigila cambios en CSS/JS
php artisan serve           # Arranca el servidor de desarrollo
```

8 Estructura de directorios de un proyecto Laravel

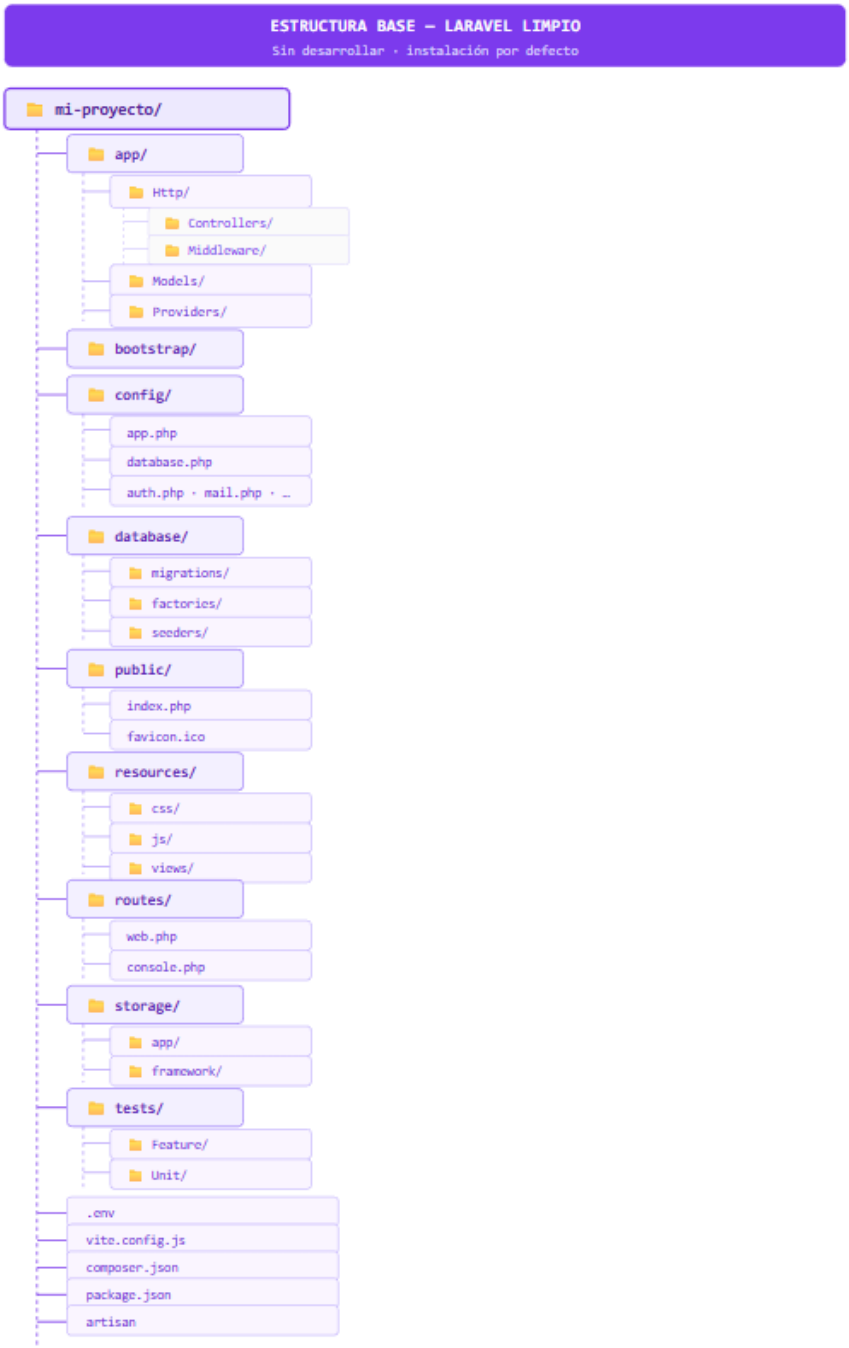
Cuando se crea un nuevo proyecto con el comando laravel new, **Laravel** genera automáticamente una estructura de carpetas que separa cada responsabilidad. La tabla siguiente muestra los directorios más importantes:

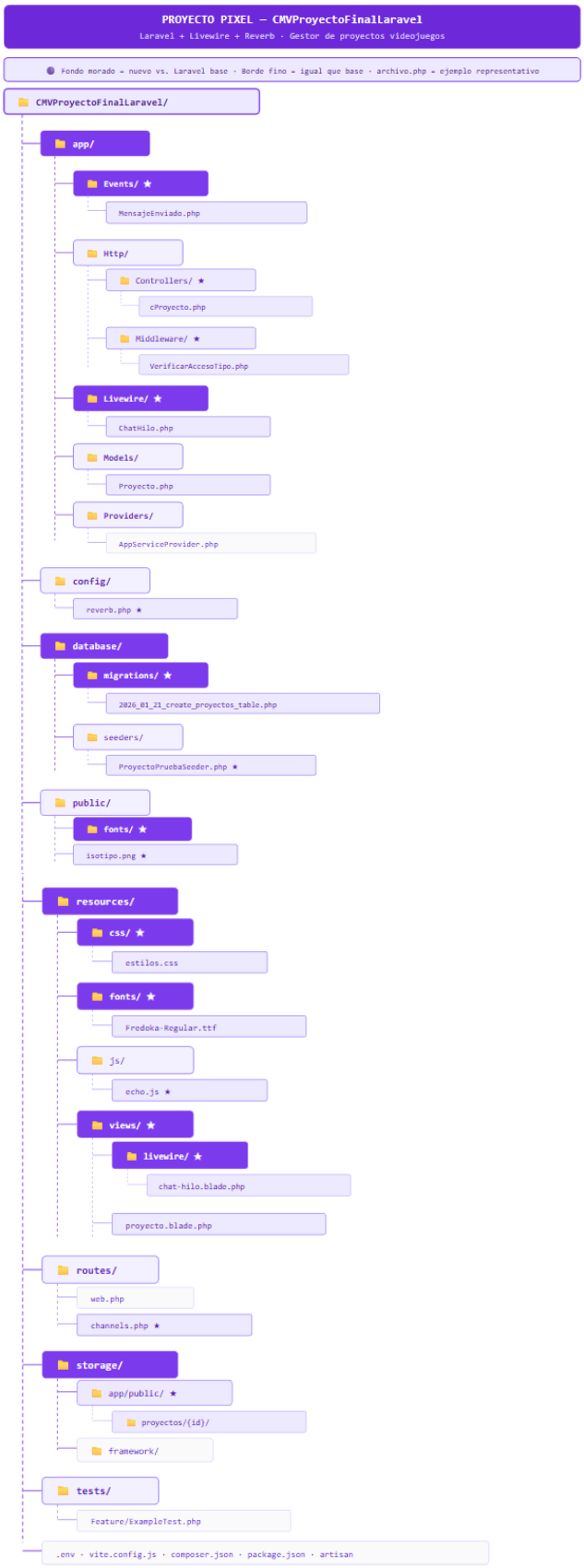
Directorio / Fichero	Componente	Función
app/Models/	Modelos (M)	Clases PHP que representan tablas de la base de datos y definen relaciones Eloquent.
app/Http/Controllers/	Controladores (C)	Reciben peticiones HTTP, ejecutan la lógica de negocio y devuelven respuestas.
resources/views/	Vistas Blade (V)	Plantillas HTML con lógica de presentación usando directivas Blade.
routes/web.php	Rutas	Mapa de URLs → controladores. Define qué URL ejecuta qué acción.
database/migrations/	Migraciones	Definen la estructura de la base de datos en código PHP versionado.
database/seeders/	Seeders	Insertan datos iniciales (tipos de área, estados de tarea, etc.).
app/Http/Middleware/	Middleware	Filtros que se ejecutan antes de llegar al controlador (autenticación, permisos).
.env	Configuración	Variables de entorno: credenciales de BD, clave de aplicación, modo debug.
composer.json	Dependencias PHP	Lista de paquetes PHP gestionados con Composer.
package.json	Dependencias JS/CSS	Lista de paquetes Node.js gestionados con npm (Tailwind, Vite...).
vite.config.js	Bundler frontend	Configuración de Vite para compilar CSS y JavaScript del frontend.

Los archivos subidos por los usuarios en PIXEL se almacenan en:

```
storage/app/public/proyectos/{proyectoId}/{nombre generado}
storage/app/public/foro/{hiloId}/{nombre_generado} ← archivos adjuntos del chat
```

El nombre en disco es generado automáticamente por **Laravel** para evitar colisiones; el nombre original se conserva en la columna original_name. El acceso desde el navegador se hace a través del enlace simbólico public/storage (**php artisan** storage:link). Los archivos se clasifican automáticamente por extensión: texto (txt, doc, docx, csv), pdf, audio (mp3, wav, ogg), imagen (jpg, png, gif, svg) y video (mp4, mov, webm).


ESTRUCTURA BASE — LARAVEL LIMPIO
Sin desarrollar · instalación por defecto
mi-proyecto/
app/
Http/
Controllers/
Middleware/
Models/
Providers/
bootstrap/
config/
app.php
database.php
auth.php · mail.php · …
database/
migrations/
factories/
seeders/
public/
index.php
favicon.ico
resources/
css/
js/
views/
routes/
web.php
console.php
storage/
app/
framework/
tests/
Feature/
Unit/
.env
vite.config.js
composer.json
package.json
artisan


PROYECTO PIXEL — CMVProyectoFinalLaravel
Laravel + Livewire + Reverb · Gestor de proyectos videojuegos
Fondo morado = nuevo vs. Laravel base · Borde fino = igual que base · archivo.php = ejemplo representativo
CMVProyectoFinalLaravel/
app/
Events/ ★
MensajeEnviado.php
Http/
Controllers/ ★
cProyecto.php
Middleware/ ★
VerificarAccesoTipo.php
Livewire/ ★
ChatHilo.php
Models/
Proyecto.php
Providers/
AppServiceProvider.php
config/
reverb.php ★
database/
migrations/ ★
2026_01_21_create_proyectos_table.php
seeders/
ProyectoPruebaSeeder.php ★
public/
fonts/ ★
isotipo.png ★
resources/
css/ ★
estilos.css
fonts/ ★
Fredoka-Regular.ttf
js/
echo.js ★
views/ ★
livewire/ ★
chat-hilo.blade.php
proyecto.blade.php
routes/
web.php
channels.php ★
storage/
app/public/ ★
proyectos/{id}/
framework/
tests/
Feature/ExampleTest.php
.env · vite.config.js · composer.json · package.json · artisan

9 Descripción de cada componente con ejemplos de PIXEL

9.1 Modelos — Eloquent ORM

Los modelos son clases PHP que representan una tabla de la base de datos. Con **Eloquent ORM** se definen las relaciones entre tablas sin escribir SQL. Ejemplo del modelo Tarea en PIXEL:

```
// app/Models/Tarea.php
class Tarea extends Model {
  protected $table = 'tareas';
  protected $fillable = ['title','description','project_id',
    'type_id','status_id','estimated_hours','start_date','end_date','is_milestone'];

  // Un proyecto tiene muchas tareas
  public function proyecto() { return $this->belongsTo(Proyecto::class, 'project_id'); }
  // Una tarea pertenece a un tipo (área) y a un estado
  public function tipo()    { return $this->belongsTo(Tipo::class, 'type_id'); }
  public function status()  { return $this->belongsTo(Estado::class, 'status_id'); }
  // Muchos usuarios pueden estar asignados a una tarea
  public function usuarios() { return $this->belongsToMany(Usuario::class,'participaciones'); }
  // Una tarea puede depender de otras tareas (relación autorreferencial)
  public function dependencias() {
    return $this->belongsToMany(Tarea::class,'tarea_dependencias','task_id','depends_on_id');
  }
  // Método de dominio: ¿está bloqueada por alguna dependencia sin terminar?
  public function isBlocked(): bool {
    if (!$this->relationLoaded('dependencias') || $this->dependencias->isEmpty()) {
      return false;
    }
    return $this->dependencias->contains(fn($d) => !$d->status || $d->status->name !==
'Terminada');
  }
}
```

9.2 Migraciones — Control de la BD

Las migraciones definen la estructura de la base de datos en código PHP. Cada cambio en el esquema se representa como un fichero de migración con fecha, lo que permite a cualquier desarrollador reproducir la misma estructura ejecutando **php artisan** migrate. Ejemplo de la migración de la tabla tareas:

```
// database/migrations/2026_01_21_000005_create_tareas_table.php
public function up(): void {
  Schema::create('tareas', function (Blueprint $table) {
    $table->id();
    $table->string('title');
    $table->text('description')->nullable();
    $table->foreignId('project_id')->constrained('proyectos')->onDelete('cascade');
    $table->foreignId('type_id')->constrained('tipos');
    $table->foreignId('status_id')->constrained('estados');
    $table->decimal('estimated_hours', 8, 2)->default(0);
    $table->date('start_date')->nullable();
    $table->date('end_date')->nullable();
    $table->boolean('is_milestone')->default(false);
```

```
        $table->timestamps();
    });
}
```

9.3 Controladores — Lógica de negocio

El controlador recibe la petición HTTP, valida los datos de entrada, interactúa con los modelos y devuelve una vista o redirección. Ejemplo del método store del controlador cTarea:

```
// app/Http/Controllers/cTarea.php
public function store(Request $request) {
  $request->validate([
    'title'          => 'required|max:255',
    'project_id'     => 'required|exists:proyectos,id',
    'type_id'        => 'required|exists:tipos,id',
    'status_id'      => 'required|exists:estados,id',
    'estimated_hours'=> 'nullable|numeric|min:0',
    'start_date'     => 'nullable|date',
    'end_date'       => 'nullable|date|after_or_equal:start_date',
    'depends_on'     => 'nullable|array',
    'depends_on.*'   => 'exists:tareas,id',
  ]);
  $tarea = Tarea::create($request->only([...campos...]));
  $tarea->dependencias()->sync($request->depends_on ?? []);
  return redirect()->back()->with('success', 'Tarea creada correctamente.');
}
```

9.4 Rutas — routes/web.php

El archivo web.php actúa como mapa del sitio, definiendo qué URL ejecuta qué controlador y qué **middleware** se aplica:

```
// routes/web.php
// Rutas públicas
Route::get('/', [PageController::class, 'index'])->name('home');
Route::get('/login', [cUsuario::class, 'showLogin'])->name('login');
Route::post('/login', [cUsuario::class, 'login'])->name('login.post');

// Rutas protegidas (requieren autenticación)
Route::middleware('auth')->group(function () {
  Route::get('/proyectos', [PageController::class, 'proyectos'])->name('proyectos');
  Route::post('/tareas', [cTarea::class, 'store'])->name('tareas.store');
  // Ruta con middleware adicional de control de acceso por área:
  Route::get('/proyecto/{id}/tipo/{tipoId}', [PageController::class, 'tareasPorTipo'])
    ->middleware('area.access')->name('proyecto.tipo');
});
```

9.5 Vistas Blade — recursos/views/

Las vistas **Blade** generan el HTML con directivas propias del framework. Permiten herencia de plantillas (@extends, @section), bucles (@foreach), condicionales (@if) e inclusión de componentes. Fragmento del formulario de creación de tareas:

```
{{-- resources/views/tareas.blade.php --}}
@if($errors->any())
  <div class="bg-red-100 text-red-700 p-3 rounded">
    @foreach($errors->all() as $error) <p>{{ $error }}</p> @endforeach
  </div>
@endif
<form method="POST" action="{{ route('tareas.store') }}">
  @csrf
  <input type="text" name="title" value="{{ old('title') }}" required>
  <select name="status_id">
    @foreach($estados as $estado)
      <option value="{{ $estado->id }}">{{ $estado->name }}</option>
    @endforeach
  </select>
  <button type="submit">Crear Tarea</button>
</form>
```

9.6 Middleware — Filtros de petición

El **middleware** intercepta la petición antes de que llegue al controlador. En PIXEL se usa el **middleware** VerificarAccesoTipo (registrado como area.access) para controlar el acceso granular a cada área del proyecto:

Paso	Condición	Resultado
1	¿Es el usuario el owner del proyecto?	SÍ → acceso total
2	¿Tiene filas en proyecto_accesos para este proyecto?	NO (cero filas) → acceso completo (modo legacy)
3	¿Existe fila con (proyecto_id, user_id, tipo_id)?	SÍ → acceso / NO → abort(403)

10 El patrón MVC y el flujo de una petición

Cuando el navegador realiza una petición HTTP (por ejemplo, GET /proyectos), **Laravel** sigue este flujo interno:

1. El **Middleware** verifica condiciones previas: ¿está el usuario autenticado? ¿tiene permisos de área?
2. El Router (routes/web.php) identifica qué controlador y método debe manejar la petición.
3. El Controlador recibe la petición, valida datos y consulta los Modelos.
4. Los Modelos interactúan con la Base de Datos a través de **Eloquent ORM**.
5. El Controlador pasa los datos obtenidos a la Vista **Blade** (En desarrollo se tienen que compilar los assets css/js pero en explotación se sirven de la carpeta public ya compilados previamente).
6. La Vista genera el HTML final que se envía al navegador.

Caso Práctico: Análisis y Diseño

11 Análisis y diseño

11.1 Catálogo de requisitos

El catálogo de requisitos recoge el compromiso funcional y técnico del desarrollador con el cliente o institución evaluadora. Todos los requisitos están numerados, son verificables y están alineados con los modelos de datos, casos de uso y arquitectura implementada. Las restricciones externas, normativa aplicable, plazos y coste estimado forman parte del compromiso documentado en esta sección.

Requisitos funcionales:

ID	Nombre	Descripción
RF-01	Registro de usuario	El sistema permite crear una cuenta nueva introduciendo username (único, obligatorio, máx. 50 car.), email (único, formato válido) y contraseña (mín. 4 car.). La contraseña se hashea concatenando username+contraseña, lo que impide ataques de diccionario con contraseñas compartidas. El usuario puede registrarse directamente o hacerlo a través de un enlace de invitación (?code=PROY-XXXXXXXX), en cuyo caso el código se preserva en sesión y se procesa automáticamente al completar el registro para unirle al proyecto.
RF-02	Autenticación y sesión	El sistema permite iniciar sesión con username y contraseña. El cierre de sesión (logout) destruye el token de sesión del servidor. Las rutas protegidas redirigen al login si el usuario no está autenticado. Los intentos fallidos no bloquean la cuenta en esta versión (sin límite de intentos).
RF-03	Gestión del perfil	El usuario autenticado puede editar desde /perfil: username (único, obligatorio), email (único), descripción personal (texto libre, opcional) y contraseña (requiere confirmación). Los campos email y username se validan contra duplicados en la base de datos antes de guardar.
RF-04	Gestión de proyectos	Cualquier usuario autenticado puede crear proyectos. El creador pasa a ser owner. El owner puede editar nombre (obligatorio, máx. 255 car.) y descripción del proyecto, y eliminarlo (se borran en cascada todas las tareas, archivos, hilos, miembros y datos predefinidos asociados). La pantalla /proyectos divide en "Mis proyectos" (owner) y "Proyectos en los que participo" (colaborador). El dashboard /proyecto/{id} muestra: info de progreso (% de tareas terminadas/total), número de miembros y accesos rápidos a las áreas habilitadas para el usuario.
RF-05	Gestión de tareas	El owner puede crear, editar y eliminar tareas en cualquier área. Cada tarea contiene: título (obligatorio, máx. 255 car.), descripción (texto libre, opcional), área/tipo (obligatorio; valores fijos del sistema: Desarrollo, Diseño, Audio, Narrativa, Marketing, Arte), estado (obligatorio; valores fijos: Pendiente, En Proceso, Terminada), fecha inicio (opcional), fecha fin (opcional, validada ≥ fecha inicio si ambas se informan), horas estimadas (decimal ≥ 0, por defecto 0) y flag de hito (booleano, por defecto false). Las tareas pueden declarar dependencias sobre otras tareas del mismo proyecto (relación autorreferencial muchos-a-muchos vía tabla tarea_dependencias). El sistema detecta automáticamente si una tarea está bloqueada (alguna de sus dependencias no está en estado Terminada) y lo refleja visualmente.
RF-06	Asignación de miembros a tareas	El owner puede asignar miembros a tareas de forma individual o masiva por área (función assignDepartamento). Restricciones: el owner no puede ser asignado a tareas. Solo puede asignarse a un miembro a una tarea si ese miembro tiene acceso al área (tipo) de dicha tarea según la tabla proyecto_accesos. Las asignaciones se almacenan en la tabla participaciones (task_id, user_id).

RF-07	Control de acceso por área	Cada miembro del proyecto solo puede acceder a las áreas para las que fue invitado, según las filas de proyecto_accesos. El middleware area.access aplica la siguiente lógica en orden: (1) si el usuario es owner → acceso total sin restricción; (2) si no existen filas en proyecto_accesos para ese usuario y proyecto → acceso completo (3) si existen filas → solo accede a las áreas registradas, devuelve HTTP 403 para el resto. Dentro de un área, el miembro no-owner solo ve las tareas en las que está asignado; las estadísticas (totales de tareas, progreso) se calculan siempre sobre el total del área.
RF-08	Sistema de invitaciones	El owner genera desde /proyecto/{id}/miembros un enlace de invitación con código único en formato PROY-XXXXXXXX (8 caracteres hexadecimales aleatorios). Al generar la invitación el owner selecciona las áreas que tendrá el invitado. La invitación caduca a los 7 días desde su creación (campo expires_at). Flujo de unión: (a) usuario autenticado accede a /join?code=PROY-XXXXXXXX → se valida el código y se crean las filas en proyecto_accesos; (b) usuario no autenticado → se guarda el código en sesión → se redirige a /registro?code=... → al registrarse se procesa el código. Si el usuario ya era miembro del proyecto, no se crea ningún registro duplicado y no se incrementa el contador uses_count. El campo areas de la tabla invitaciones almacena en JSON el array de tipo_id habilitados.
RF-09	Gestión de archivos del proyecto	Desde /proyecto/{id}/archivos los miembros con acceso al proyecto pueden subir archivos mediante interfaz drag-and-drop. Límite por archivo: 50 MB. El nombre en disco es generado por Laravel (UUID) para evitar colisiones; el nombre original se conserva en la columna original_name. Cada archivo puede tener nombre personalizable (máx. 255 car.) y descripción opcional. Los archivos se etiquetan por áreas a través de la tabla pivot archivo_tipo. La categoría se determina automáticamente por extensión: texto (txt, doc, docx, csv, rtf, odt → previsualización en modal, primeros 5.000 caracteres), imagen (jpg, jpeg, png, gif, webp, svg → miniatura inline), pdf (modal iframe), audio (mp3, wav, ogg, m4a, flac, aac → modal reproductor), video (mp4, mov, avi, webm, mkv → modal reproductor). El archivo físico se almacena en storage/app/public/proyectos/{proyectoId}/. El uploader y el owner pueden eliminar un archivo; la eliminación borra el fichero físico del storage y el registro de la base de datos.
RF-10	Vista Gantt	El diagrama de Gantt en /proyecto/{id}/gantt muestra todas las tareas del proyecto con fechas definidas. Cada tarea se representa como una barra horizontal proporcional a su duración (start_date → end_date). Las dependencias entre tareas se dibujan como líneas visuales que conectan el extremo derecho de la tarea predecesora con el extremo izquierdo de la dependiente. Los hitos (is_milestone = true) se representan con marcado visual diferenciado (rombo o icono especial). Las tareas bloqueadas (con dependencias no terminadas) se resaltan con color diferente. El Gantt es de solo lectura; no permite arrastrar barras. Solo se muestra en resoluciones de escritorio (oculto en móvil mediante clase CSS responsive). El owner ve todas las tareas; el miembro solo ve las de sus áreas.
RF-11	Vista Calendario	La vista mensual en /proyecto/{id}/calendario muestra cada tarea en los días comprendidos entre su fecha de inicio y fecha de fin. Los hitos tienen marcado visual especial. El usuario puede navegar entre meses con controles anterior/siguiente. Las tareas sin fechas definidas no aparecen en el calendario. El owner ve todas las tareas; el miembro solo ve las asignadas a sus áreas.
RF-12	Foro de discusión	Cada proyecto dispone de un foro en /proyecto/{id}/foro con hilos de conversación. Los hilos se ordenan por updated_at descendente. El listado muestra: título, autor, fecha del último mensaje y contador de respuestas. Cualquier miembro del proyecto puede crear hilos y publicar respuestas. Al crear un hilo se puede adjuntar uno o varios archivos (almacenados en el mensaje raíz interno del hilo). Solo el autor del hilo o el owner del proyecto pueden eliminar un hilo; la eliminación borra en cascada todos sus mensajes y archivos físicos. Solo el autor de un mensaje o el owner pueden eliminar ese mensaje y sus archivos asociados.

RF-13	Chat en tiempo real	Dentro de cada hilo, los mensajes se envían y reciben en tiempo real mediante WebSockets. En desarrollo: servidor Reverb (ws://localhost:8080). En producción: Pusher (wss://ws-eu.pusher.com:443). El componente Livewire ChatHilo re-renderiza la lista de mensajes al recibir el evento MensajeEnviado, sin recargar la página. Los canales son privados: el servidor verifica que el usuario sea miembro del proyecto antes de autorizar la suscripción (routes/channels.php). La cola de trabajos (QUEUE_CONNECTION=database en desarrollo, sync en producción) despacha el evento al servidor WebSocket de forma asíncrona.
RF-14	Archivos en el chat	En cada respuesta del foro se pueden adjuntar múltiples archivos (límite 50 MB por archivo). Los archivos se almacenan en storage/app/public/foro/{hiloId}/{mensajeId}/. La categoría y la previsualización siguen las mismas reglas que los archivos del proyecto (RF-09): imagen inline en la burbuja, PDF en modal iframe, audio/vídeo en modal reproductor, texto en modal con primeros 5.000 caracteres. Al eliminar un mensaje sus archivos físicos se borran del storage. Cualquier miembro puede descargar archivos del foro.
RF-15	Datos predefinidos del juego	Cada proyecto dispone de un catálogo de entidades predefinidas del videojuego accesible desde /proyecto/{id}/predefinicion. El catálogo es independiente por proyecto: las entidades de un proyecto no son visibles ni modificables desde otro. El CRUD completo (crear, editar, eliminar) está disponible tanto para el owner como para cualquier miembro con acceso al proyecto. Se gestionan tres tipos de entidad: (1) Personaje — identificador único en el juego (game_id, texto, único por proyecto, obligatorio), nombre (obligatorio, máx. 150 car.), vida (entero ≥ 0), ataque (entero ≥ 0), defensa (entero ≥ 0), velocidad (decimal(8,2) ≥ 0). La unicidad de game_id se garantiza a nivel de base de datos con índice unique(game_id, proyecto_id). (2) Ítem — game_id (único por proyecto), nombre (obligatorio), descripción (opcional), precio (decimal(10,2) ≥ 0, por defecto 0), tipo (enumerado obligatorio: Arma, Consumible o Misión). (3) Diálogo — pertenece a un Personaje (personaje_id FK obligatorio); contiene id_conversacion (texto, agrupa líneas de la misma conversación), orden (entero ≥ 1, posición dentro de la conversación), texto del diálogo (obligatorio). Los diálogos de una conversación se recuperan ordenados por el campo orden.
RF-16	Exportación de datos a motores de juego	Los datos predefinidos (RF-15) pueden exportarse como archivo JSON desde la misma pantalla /proyecto/{id}/predefinicion. La exportación se genera íntegramente en el navegador (JavaScript client-side) sin ninguna petición adicional al servidor. Se soportan tres formatos: (a) Unity — campos con prefijo m_ (m_ID, m_Nombre, m_Health, m_Attack, m_Defense, m_Speed para personajes; m_ID, m_Nombre, m_Description, m_Price, m_Type para ítems; m_ConversationID, m_Order, m_CharacterID, m_Text para diálogos), agrupados en PersonajeList.Items, ItemList.Items y DialogoList.Items. Compatible con ScriptableObjects y JsonUtility de Unity. (b) Unreal Engine — array de objetos con clave "Name" igual al game_id, compatible con la importación de DataTables en el editor de Unreal. (c) Godot — diccionario anidado por game_id, cargable directamente con JSON.parse() en GDScript. El archivo descargado recibe el nombre gamedata_{motor}_{proyecto_id}.json. No se requiere ninguna librería PHP de exportación (ni maatwebsite/excel ni DomPDF).
RF-17	Notas de seguimiento de tarea	Cualquier miembro con acceso a una tarea puede añadir notas de seguimiento desde la vista de detalle /tareas/{id}. Campos: contenido (texto libre, obligatorio). La nota queda asociada a la tarea y al usuario que la crea (timestamps automáticos). El autor de la nota o el owner del proyecto pueden eliminarla; la eliminación solo borra el registro de la tabla notas_tarea, sin efecto sobre la tarea. Las notas se muestran ordenadas por fecha de creación ascendente.
RF-18	Panel de miembros	El owner accede a /proyecto/{id}/miembros para consultar el listado completo de colaboradores del proyecto, con su username y las áreas de acceso configuradas. Solo el owner puede acceder a esta vista. Desde ella el owner también puede gestionar invitaciones.

Requisitos no funcionales:

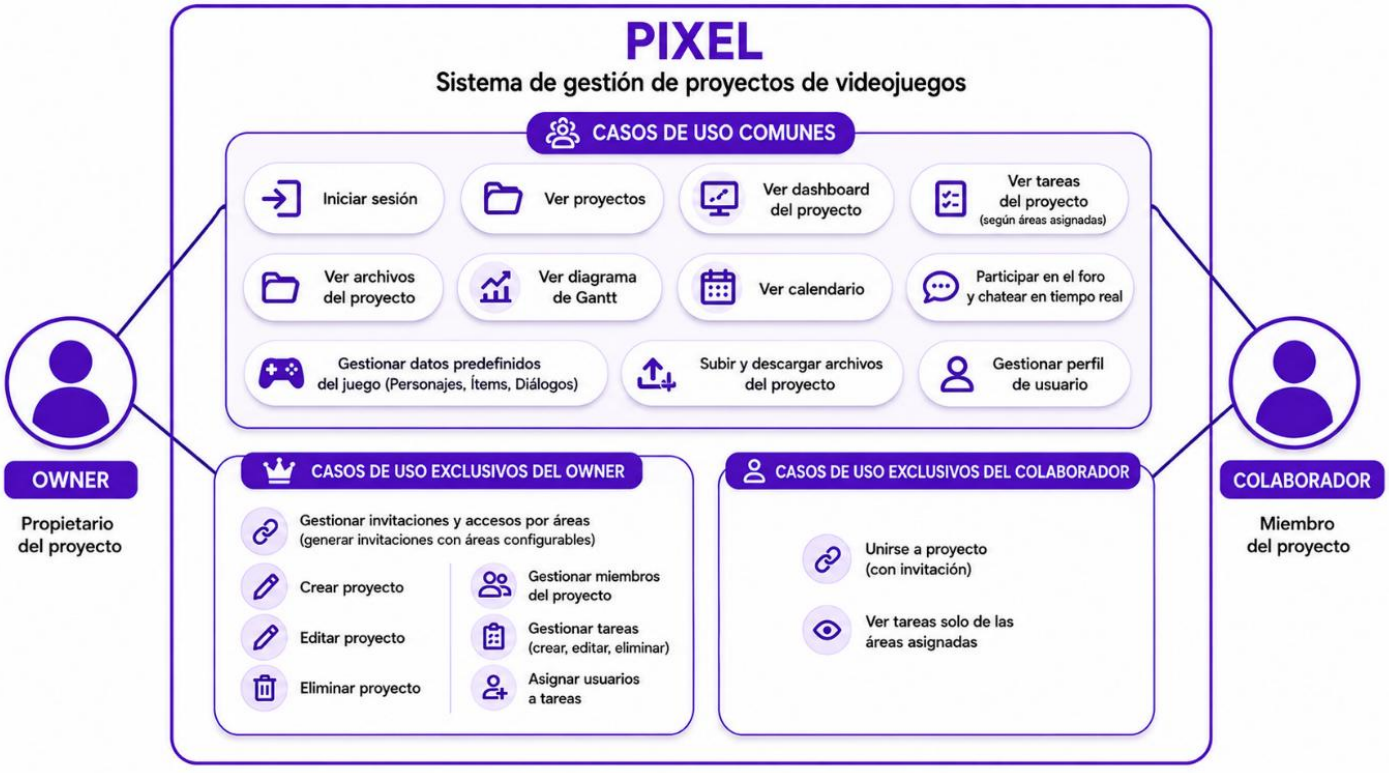
ID	Categoría	Descripción
RNF-01	Protección CSRF	Todos los formularios que modifiquen estado (POST, PUT, DELETE) incluyen el token CSRF de Laravel. Las peticiones sin token válido son rechazadas con HTTP 419. No se aplica a la ruta GET de descarga de archivos ni a las rutas de previsualización de solo lectura.
RNF-02	Control de acceso por roles	El sistema reconoce dos roles: owner y miembro. El owner tiene acceso irrestricto a todas las operaciones de su proyecto. El miembro solo puede operar sobre las áreas para las que fue invitado. Cualquier intento de acceso a un recurso no autorizado devuelve HTTP 403. Las rutas que requieren ser owner validan auth()->id() === $proyecto->created_by antes de ejecutar la operación.
RNF-03	Canales WebSocket privados	Los canales de chat (chat.{hiloId}) son privados. El servidor verifica la membresía del usuario antes de autorizar la suscripción mediante el endpoint POST /broadcasting/auth. Un usuario no autenticado o que no pertenezca al proyecto no puede suscribirse al canal ni recibir mensajes en tiempo real.
RNF-04	Seguridad de contraseñas	Las contraseñas se almacenan con Bcrypt (función Hash::make de Laravel). El mecanismo de hash concatena username+contraseña antes de hashear, añadiendo una capa extra de singularidad frente a contraseñas compartidas entre usuarios. No se almacena ninguna contraseña en texto plano ni en logs.
RNF-05	Usabilidad y diseño responsivo	La interfaz es funcional en dispositivos móviles (≥ 320 px) y escritorio. Se implementa con Tailwind CSS 4 (utility-first, sin framework JS adicional). El diagrama de Gantt se oculta en resoluciones móviles (hidden md:block o equivalente) por limitaciones de espacio. El tiempo medio de aprendizaje de un usuario nuevo no debe superar 10 minutos para las operaciones básicas (crear proyecto, crear tarea, invitar miembro).
RNF-06	Rendimiento	El tiempo de carga de cualquier página bajo condiciones normales (servidor IONOS, conexión de 10 Mbps) no debe superar 2 segundos. La exportación JSON de datos predefinidos se ejecuta en el cliente sin round-trip al servidor, por lo que es instantánea independientemente del tamaño del catálogo. Las consultas Eloquent de listas largas (tareas, archivos, mensajes) usan lazy loading paginado o carga eager para evitar el problema N+1.
RNF-07	Mantenibilidad	El código sigue el patrón MVC de Laravel con convenciones estándar (controladores en app/Http/Controllers/, modelos en app/Models/, vistas en resources/views/). La categorización de archivos (extensiones → categoría) se centraliza en el Trait CategorizaArchivos (app/Traits/CategorizaArchivos.php), compartido entre los controladores cArchivo y cForo mediante use CategorizaArchivos. Añadir un nuevo tipo de extensión solo requiere modificar el método resolverCategoria() en ese único Trait.. Las migraciones están versionadas en orden cronológico y permiten reproducir el esquema completo con php artisan migrate.
RNF-08	Compatibilidad de navegadores	La aplicación es compatible con las versiones actuales de Chrome, Firefox, Edge y Safari (desktop). Las funcionalidades de WebSocket requieren soporte de la API WebSocket en el navegador (disponible en todos los navegadores modernos desde 2012). No se garantiza compatibilidad con Internet Explorer.
RNF-09	Normativa de protección de datos (RGPD)	La aplicación almacena datos personales de usuarios (username, email, descripción) en un servidor propio (IONOS, UE). No se comparten datos con terceros salvo el servicio Pusher (proveedor WebSocket externo, con servidores en la UE según configuración). No se implementan cookies de seguimiento ni publicidad. El usuario puede eliminar su cuenta y todos sus datos asociados. El proyecto se enmarca en un entorno académico (IES Los Sauces, Benavente); en un entorno de producción real sería obligatorio redactar una política de privacidad conforme al Reglamento (UE) 2016/679 (RGPD) y la LOPDGDD (Ley Orgánica 3/2018).

RNF-10	Restricción de entorno de producción	El hosting de producción (IONOS, plan compartido con Plesk) no permite mantener procesos persistentes propios (no se puede levantar Reverb como servicio). Por ello, en producción el broadcasting WebSocket se delega a Pusher (servicio externo gestionado). La cola de trabajos usa QUEUE_CONNECTION=sync en producción (sin worker propio), lo que significa que los eventos se envían de forma síncrona al final de cada petición HTTP. Esta decisión es una restricción arquitectónica del entorno de hosting elegido.
RNF-11	Plazos de desarrollo	El proyecto se desarrolló entre enero y mayo de 2026 (~240 horas totales distribuidas en 13 fases): fase 1-3 (enero-febrero): estructura base, migraciones, autenticación. Fase 4-7 (marzo-abril): CRUD proyectos/tareas, control de acceso, invitaciones. Fase 8-10 (abril): Gantt, calendario, archivos, datos predefinidos. Fase 11-13 (mayo): foro con chat WebSocket, notas de tarea, ajustes de frontend. Entrega: junio 2026.
RNF-12	Coste del proyecto	Herramientas de desarrollo: Laravel (MIT), PHP (MIT), MySQL (GPL/comercial), Tailwind CSS (MIT), Vite (MIT), Git (LGPL), VS Code (MIT) — coste 0 €. Infraestructura: hosting IONOS ~4-6 €/mes; dominio ~10-15 €/año; Pusher (plan gratuito, hasta 200.000 mensajes/día, 100 conexiones simultáneas). Asistencia IA: Claude Code Pro ~20 $/mes × 5 meses activos ~ 95 €. Coste total de infraestructura primer año: ~200 €. Mano de obra equivalente (desarrollador junior en España, ~12 €/h × 240 h): ~2.880 €. Coste total estimado del proyecto: ~3.080 €.

11.2 Modelo de casos de uso

Los actores del sistema son el Owner (propietario del proyecto) y el Miembro (colaborador invitado).

Caso de uso	Owner (propietario)	Miembro (colaborador)
Proyectos	Crear, editar y eliminar proyectos propios	Ver proyectos en los que participa
Miembros	Gestionar miembros y permisos de área	—
Invitaciones	Generar enlaces con áreas configurables	Unirse al proyecto con el enlace o código
Tareas	Crear, editar y eliminar tareas en cualquier área	Ver tareas de sus áreas; agregar actualizaciones
Asignaciones	Asignar miembros a tareas específicas	Ver sus propias tareas asignadas
Archivos	Subir y eliminar archivos del proyecto	Subir archivos; descargar cualquier archivo
Visualizaciones	Ver Gantt y Calendario de todo el proyecto	Ver sus tareas asignadas en Gantt y Calendario
Datos predefinidos	Crear, editar y eliminar Personajes, Ítems y Diálogos del proyecto	Crear, editar y eliminar Personajes, Ítems y Diálogos del proyecto
Foro	Crear hilos y participar en ellos	Crear hilos y participar en ellos



11.3 Diagrama de clases — Modelos Eloquent

A continuación, se detallan las clases del sistema (modelos **Eloquent**) y sus relaciones principales:

Clase (Modelo)	Relaciones	Atributos principales
Usuario	hasMany Proyecto (creados), acceso via ProyectoAcceso, hasMany Participacion	id, username, description, email, password
Proyecto	belongsTo Usuario (creador) hasMany Tarea belongsToMany Usuario via proyecto_accesos hasMany Archivo	id, name, description, created_by
Tarea	belongsTo Proyecto, Tipo y Estado belongsToMany Usuario via participaciones belongsToMany Tarea via tarea_dependencias (autorreferencial)	id, title, description, project_id, type_id, status_id, estimated_hours, start_date, end_date, is_milestone
Invitacion	belongsTo Proyecto y Usuario (creador)	id, proyecto_id, codigo, created_by, expires_at, uses_count, areas (JSON)
Archivo	belongsTo Proyecto y Usuario (subidor) belongsToMany Tipo via archivo_tipo	id, proyecto_id, uploaded_by, name, original_name, path, mime_type, size, categoria
Participacion	belongsTo Tarea y Usuario	task_id, user_id
ForoHilo	belongsTo Proyecto y Usuario	id, proyecto_id, user_id, titulo, contenido
ForoMensaje	belongsTo ForoHilo y Usuario	id, hilo_id, user_id, contenido
ForoArchivo	belongsTo ForoMensaje y Usuario	id, mensaje_id, user_id, original_name, path, mime_type, size, categoria
Personaje	belongsTo Proyecto	game_id, nombre, vida, ataque, defensa, velocidad
Item	belongsTo Proyecto	game_id, nombre, descripcion, precio, tipo
Dialogo	belongsTo Proyecto y Personaje	id_conversacion, orden, personaje_id, texto
NotaTarea	belongsTo Tarea belongsTo Usuario (autor)	id, task_id, user_id, contenido, created_at

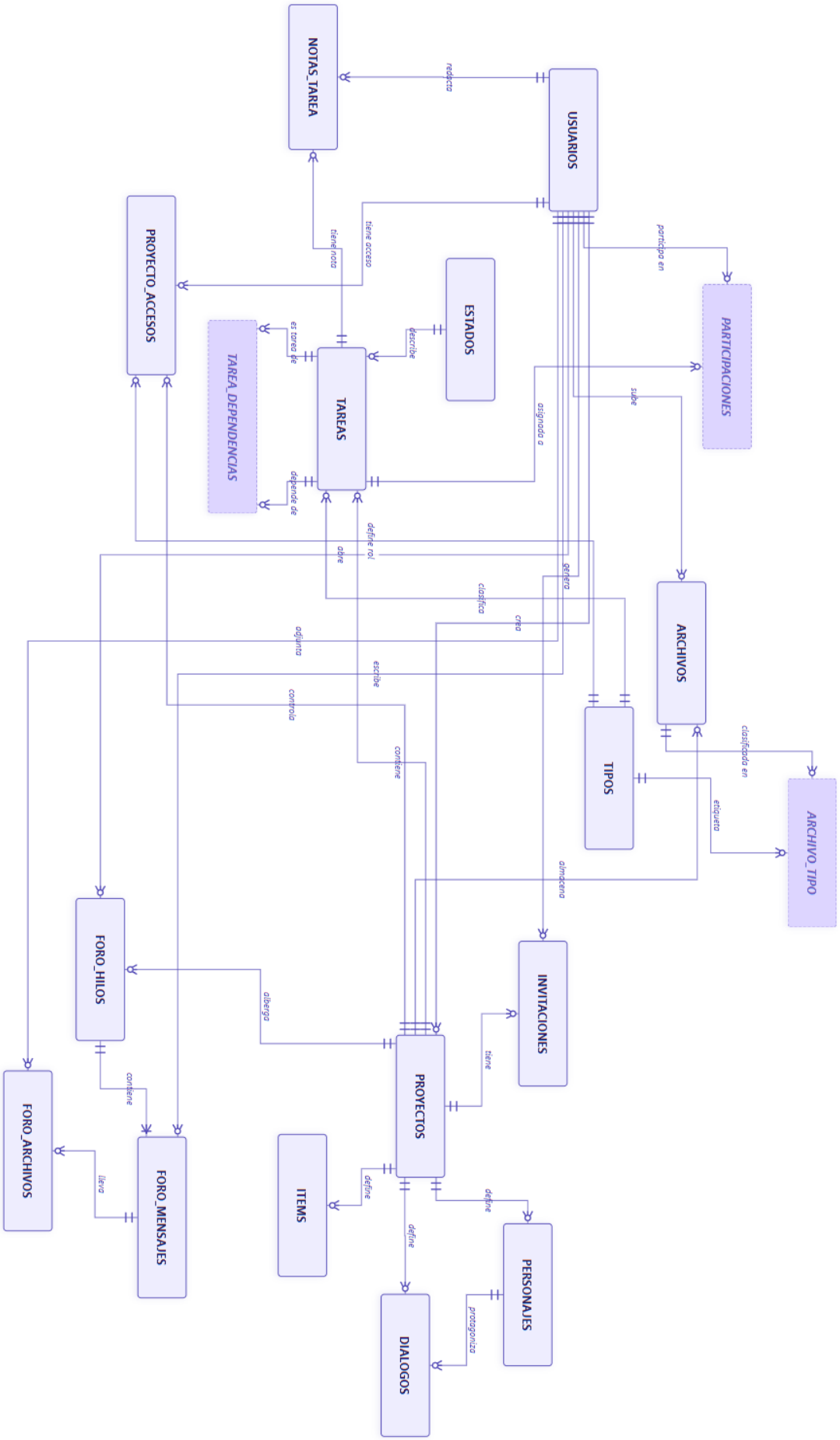
11.4 Modelo físico de datos

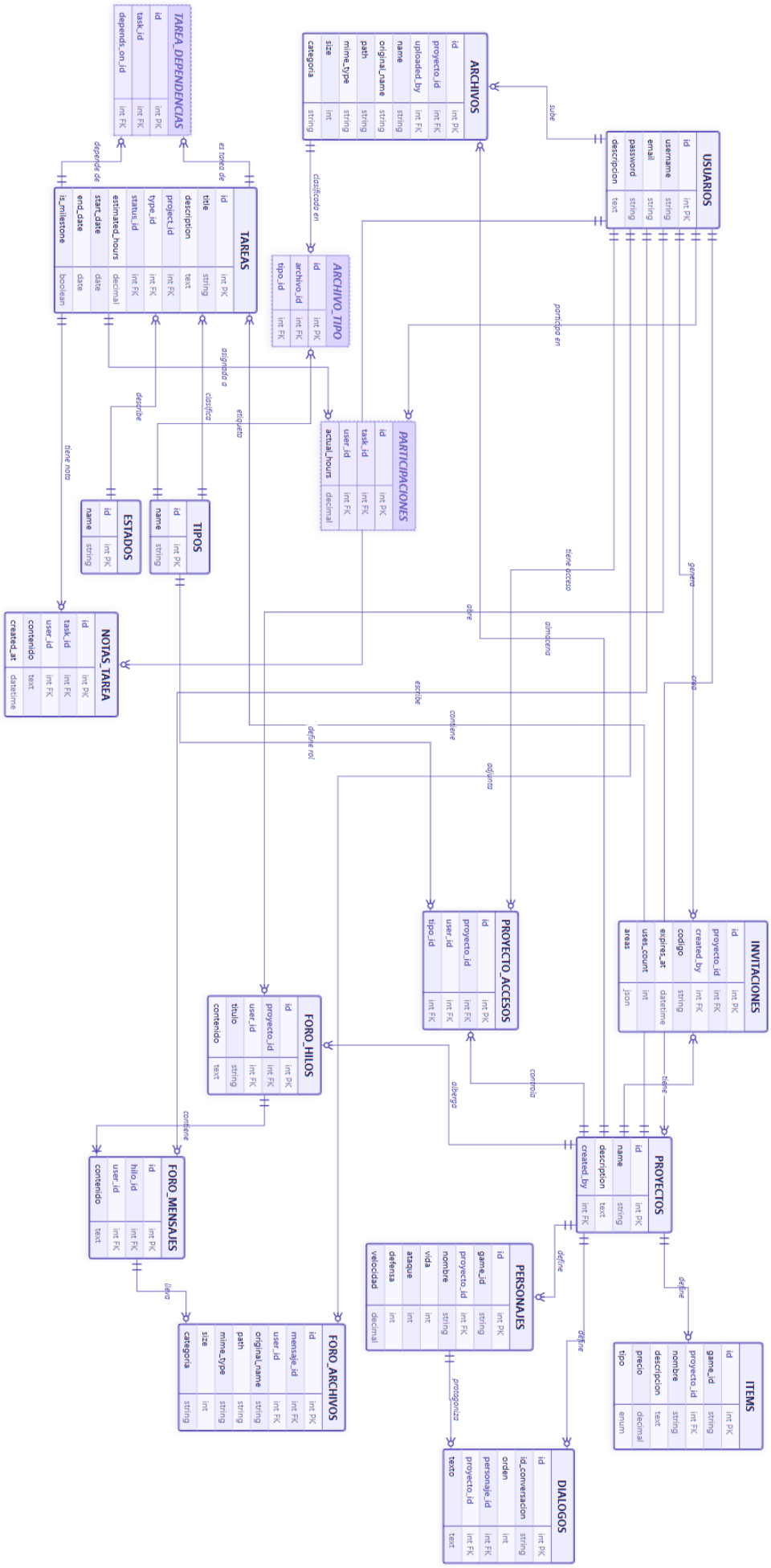
Tablas de la base de datos **MySQL**:

Tabla	Migración (fecha)	Descripción y campos clave
tipos	2026_01_20_000001	Áreas: id, name (Desarrollo, Diseño, Audio, Narrativa, Marketing, Arte).
estados	2026_01_20_000002	Ciclo de vida: id, name (Pendiente, En Proceso, Terminada).
usuarios	2026_01_21_000001	id, username, description, email, password, remember_token.
proyectos	2026_01_21_000002	id, name, description, created_by (FK usuarios).
tareas	2026_01_21_000005	id, title, description, project_id, type_id, status_id, estimated_hours, start_date, end_date, is_milestone.
participaciones	2026_01_21_000006	task_id, user_id.
invitaciones	2026_04_25_000001	id, proyecto_id, codigo, created_by, expires_at, uses_count, areas (JSON).
proyecto_accesos	2026_04_25_000003	Control por área: id, proyecto_id, user_id, tipo_id.
tarea_dependencias	2026_04_25_000005	Autorreferencial: task_id, depends_on_id.
archivos	2026_04_28_000001	id, proyecto_id, uploaded_by, name, description, original_name, path, mime_type, size, categoria.
archivo_tipo	2026_04_28_000001	Pivot archivos-áreas: archivo_id, tipo_id.
foro_hilos	2026_05_14_000001	id, proyecto_id, user_id, titulo, contenido.
foro_mensajes	2026_05_14_000002	id, hilo_id, user_id, contenido.
foro_archivos	2026_05_14_000003	id, mensaje_id, user_id, original_name, path, mime_type, size, categoria.
personajes	2026_05_04_000001	Datos predefinidos de personaje: id, game_id (unique), proyecto_id, nombre, vida, ataque, defensa, velocidad (decimal).
items	2026_05_04_000002	Datos predefinidos de item: id, game_id (unique), proyecto_id, nombre, descripcion, precio, tipo (enum: Arma/Consumible/Mision).
dialogos	2026_05_04_000003	Datos predefinidos de dialogo: id, id_conversacion, orden, personaje_id, proyecto_id, texto.
notas_tarea	2026_05_11_000002	Notas de seguimiento de tarea: id, task_id (FK tareas), user_id (FK usuarios), contenido (text), timestamps.

jobs / job_batches / failed_jobs	0001_01_01_000002	Cola de trabajos para broadcasting WebSocket en tiempo real. La tabla jobs almacena los eventos MensajeEnviado pendientes de enviar a Reverb/Pusher de forma asíncrona (id, queue, payload, attempts, reserved_at, available_at, created_at). job_batches gestiona grupos de jobs. failed_jobs registra los trabajos que fallaron con su traza de error (uuid, connection, queue, payload, exception, failed_at). El worker queue:work consume esta cola y despacha los mensajes al canal WebSocket privado del hilo.

VER DIAGRAMA


NOTAS_TAREA
USUARIOS
redacta
PROYECTO_ACCESOS
tiene nota
tiene acceso
participa en
ESTADOS
PARTICIPACIONES
TAREA_DEPENDENCIAS
es tarea de
describe
TAREAS
sube
asignada a
depende de
define rol
abre
genera
clasifica
crea
ARCHIVOS
adjunta
escribe
controla
clasificada en
TIPOS
contiene
etiqueta
ARCHIVO_TIPO
almacena
FORO_HILOS
INVITACIONES
alberga
tiene
FORO_ARCHIVOS
contiene
PROYECTOS
ITEMS
define
define
define
PERSONAJES
lleva
FORO_MENSAJES
protagoniza
DIALOGOS


TAREA_DEPENDENCIAS
id int PK
task_id int FK
depends_on_id int FK
ARCHIVOS
id int PK
proyecto_id int FK
uploaded_by int FK
name string
original_name string
path string
mime_type string
size int
categoria string
USUARIOS
id int PK
username string
email string
password string
descripcion text
TAREAS
id int PK
title string
description text
project_id int FK
type_id int FK
status_id int FK
estimated_hours decimal
start_date date
end_date date
is_milestone boolean
ARCHIVO_TIPO
id int PK
archivo_id int FK
tipo_id int FK
PARTICIPACIONES
id int PK
task_id int FK
user_id int FK
actual_hours decimal
ESTADOS
id int PK
name string
TIPOS
id int PK
name string
NOTAS_TAREA
id int PK
task_id int FK
user_id int FK
contenido text
created_at datetime
PROYECTO_ACCESOS
id int PK
proyecto_id int FK
user_id int FK
tipo_id int FK
INVITACIONES
id int PK
proyecto_id int FK
created_by int FK
codigo string
expires_at datetime
uses_count int
areas json
FORO_HILOS
id int PK
proyecto_id int FK
user_id int FK
titulo string
contenido text
PROYECTOS
id int PK
name string
description text
created_by int FK
FORO_MENSAJES
id int PK
hilo_id int FK
user_id int FK
contenido text
PERSONAJES
id int PK
game_id string
proyecto_id int FK
nombre string
vida int
ataque int
defensa int
velocidad int
ITEMS
id int PK
game_id string
proyecto_id int FK
nombre string
descripcion text
precio decimal
tipo enum
FORO_ARCHIVOS
id int PK
mensaje_id int FK
user_id int FK
original_name string
path string
mime_type string
size int
categoria string
DIALOGOS
id int PK
id_conversacion string
orden int
personaje_id int FK
proyecto_id int FK
texto text
depende de
es tarea de
clasificado en
sube
pertenece en
asignado a
clasifica
etiqueta
describe
tiene nota
define
tiene acceso
genera
crea
almacena
escribe
contiene
define rol
configura
contiene
alberga
contiene
tiene
define
define
define
define
pertenece
protagoniza

11.5 Diseño de interfaz

Todas las vistas están implementadas con **Tailwind CSS 4** en diseño responsivo:

- index.blade.php — Página pública de inicio con acceso a login y registro.
- login.blade.php / registro.blade.php — Autenticación. El registro admite ?code= para unirse por invitación.
- proyectos.blade.php — Listado dividido en Mis proyectos y Proyectos en los que participo.
- proyecto.blade.php — Dashboard con barra de progreso, áreas accesibles y accesos rápidos.
- tareas.blade.php — Lista de tareas por área con estadísticas, dependencias y estado.
- tarea.blade.php — Vista de detalle individual de una tarea: descripción, asignados, dependencias, horas y notas de seguimiento.
- gantt.blade.php — Diagrama de barras horizontal con líneas de dependencia entre tareas.
- calendario.blade.php — Vista mensual de tareas con marcado especial para hitos.
- miembros.blade.php — Gestión de permisos por área (solo owner).
- archivos.blade.php — Drag-and-drop, filtros por categoría y previsualización de texto.
- hilo.blade.php — Chat en tiempo real de un hilo de foro con subida de archivos y previsualización.
- foro.blade.php — Listado de hilos del proyecto con opción de crear nuevo hilo.
- perfil.blade.php — Edición de username, email, descripción y contraseña.
- predefinicion.blade.php — Gestión CRUD de Personajes, Items y Diálogos del proyecto con panel de exportación JSON por motor de juego (Unity, Unreal Engine, Godot).

11.6 Árbol de navegación

```
/ ——— Inicio público ——▶ /login o /registro
/proyectos ——— Panel de proyectos (requiere autenticación)
/proyecto/{id}/predefinicion ——— Gestión y exportación de datos predefinidos del proyecto
/proyecto/{id} ——— Dashboard del proyecto
/proyecto/{id}/tipo/{tipoId} ——— Tareas por área (middleware area.access)
/tareas/{id} ——— Vista detalle de tarea individual (GET, cTarea@show)
/proyecto/{id}/gantt ——— Diagrama de Gantt
/proyecto/{id}/calendario ——— Vista Calendario
/proyecto/{id}/miembros ——— Gestión de miembros (solo owner)
/proyecto/{id}/archivos ——— Gestión de archivos
/proyecto/{id}/foro ——— Foro y chat en tiempo real del proyecto
/proyecto/{id}/foro/{hiloId} ——— Chat de un hilo concreto
/perfil ——— Edición del perfil de usuario
/join?code=XXXX ——— Enlace de invitación (público)
```

11.7 Web Services — API

La aplicación no expone una API REST pública. Las rutas que no devuelven HTML completo son: GET /archivos/{id}/preview (devuelve JSON con los primeros 5.000 caracteres de un archivo de texto para el modal de previsualización) y POST /broadcasting/auth (utilizada internamente por **Laravel Echo** para autorizar la suscripción a canales **WebSocket** privados del foro). Todas las demás rutas devuelven vistas HTML renderizadas por **Blade** o redirecciones HTTP estándar.

11.8 Presupuesto y financiación

- **Laravel** (MIT), PHP, **MySQL**, **Tailwind CSS**, **Vite**, Git, VS Code → coste 0 €.
- Hosting Ionos (plan compartido con PHP 8.2 y **MySQL**) → aprox. 4–6 €/mes.
- **Claude Code** (suscripción Pro, ~19 €/mes × 5 meses de desarrollo activo) → ~95 €.
- Dominio → aprox. 10–15 €/año.

Coste total de infraestructura: ~200 € (primer año).

Horas de desarrollo y coste estimado:

- Análisis y diseño (requisitos, diagramas, modelo de datos): ~20 h
- Backend (migraciones, modelos, controladores, autenticación, broadcasting): ~110 h
- Frontend (vistas Blade, Tailwind CSS, componentes Livewire, Gantt, Calendario): ~60 h
- Pruebas manuales y despliegue en Ionos/Plesk: ~25 h
- Documentación técnica: ~25 h

Total estimado: ~240 horas (enero–mayo 2026). Tomando como referencia una tarifa de desarrollador junior en España (~12 €/h), el coste de mano de obra equivalente sería de aproximadamente **2880 €**. Sumado al coste de infraestructura, el coste total estimado del proyecto asciende a unos **3080 €,** lo que lo sitúa como un proyecto viable para una startup o equipo independiente de desarrollo de videojuegos.

Precio de venta — como producto (modelo SaaS):

Comercializado como plataforma SaaS de suscripción mensual orientada a equipos de desarrollo de videojuegos independientes:

Plan	Límites	Precio
Gratis	1 proyecto, 5 miembros	0 €/mes
Indie	5 proyectos, 10 miembros	~15 €/mes por equipo
Studio	Proyectos y miembros ilimitados + soporte	~35 €/mes por equipo

Con una base de 100 equipos en el plan Indie, el ingreso mensual recurrente (MRR) sería de ~1.500 €, cubriendo los costes de infraestructura desde el primer mes.

Caso Práctico: Implementación

12 Sistemas internos de la aplicación

12.1 Sistema de autenticación

El sistema de autenticación de PIXEL usa una variación del hash estándar: concatena el username con la contraseña antes de hashear, añadiendo una capa extra de seguridad frente a contraseñas compartidas.

```
// Al registrar:
$passwordConcatenada = $request->username . $request->password;
$usuario->password = Hash::make($passwordConcatenada);

// Al verificar login:
$passwordConcatenada = $request->username . $request->password;
Hash::check($passwordConcatenada, $user->password); // true/false

// Ejemplo: si username es 'admin' y password es '1234',
// lo que se hashea es 'admin1234'.
```

12.2 Middleware y control de acceso

El **middleware** VerificarAccesoTipo (registrado como area.access en bootstrap/app.php) controla el acceso granular a las áreas de cada proyecto. Su lógica de tres pasos queda recogida en la tabla de la sección de componentes (9.6).

12.3 Sistema de invitaciones

El owner genera un enlace con código único en formato PROY-XXXXXXXX. Cualquier persona que acceda puede unirse al proyecto con los permisos de área configurados:

```
OWNER genera invitación → POST /invitaciones/generar/{proyecto}
  Código: PROY-A3F7B2C1 | Expira: +7 días | Areas: [1, 3, 5]

GET /join?code=PROY-A3F7B2C1
  ¿Válida? SÍ → ¿Autenticado?
    NO → guarda código en sesión → /registro?code=...
    SÍ → unirAlProyecto()
         → ProyectoAcceso::firstOrCreate() por cada área (acceso)
         → ProyectoAcceso::firstOrCreate() por cada área
```

12.4 Sistema de archivos

Categoría	Extensiones	Función
texto	txt, doc, docx, csv, rtf, odt	Previsualización de contenido en modal (primeros 5.000 caracteres)
pdf	pdf	Indicador visual PDF
audio	mp3, wav, ogg, m4a, flac, aac	Indicador visual Audio
imagen	jpg, jpeg, png, gif, webp, svg	Preview de imagen en miniatura
video	mp4, mov, avi, webm, mkv	Indicador visual Video

12.5 Sistema de foro y chat en tiempo real

PIXEL incorpora un foro de discusión por proyectos con chat en tiempo real basado en **WebSockets**. Cada proyecto tiene su propio foro con hilos de conversación; dentro de cada hilo, los mensajes aparecen en pantalla al instante para todos los miembros conectados, sin necesidad de recargar la página.

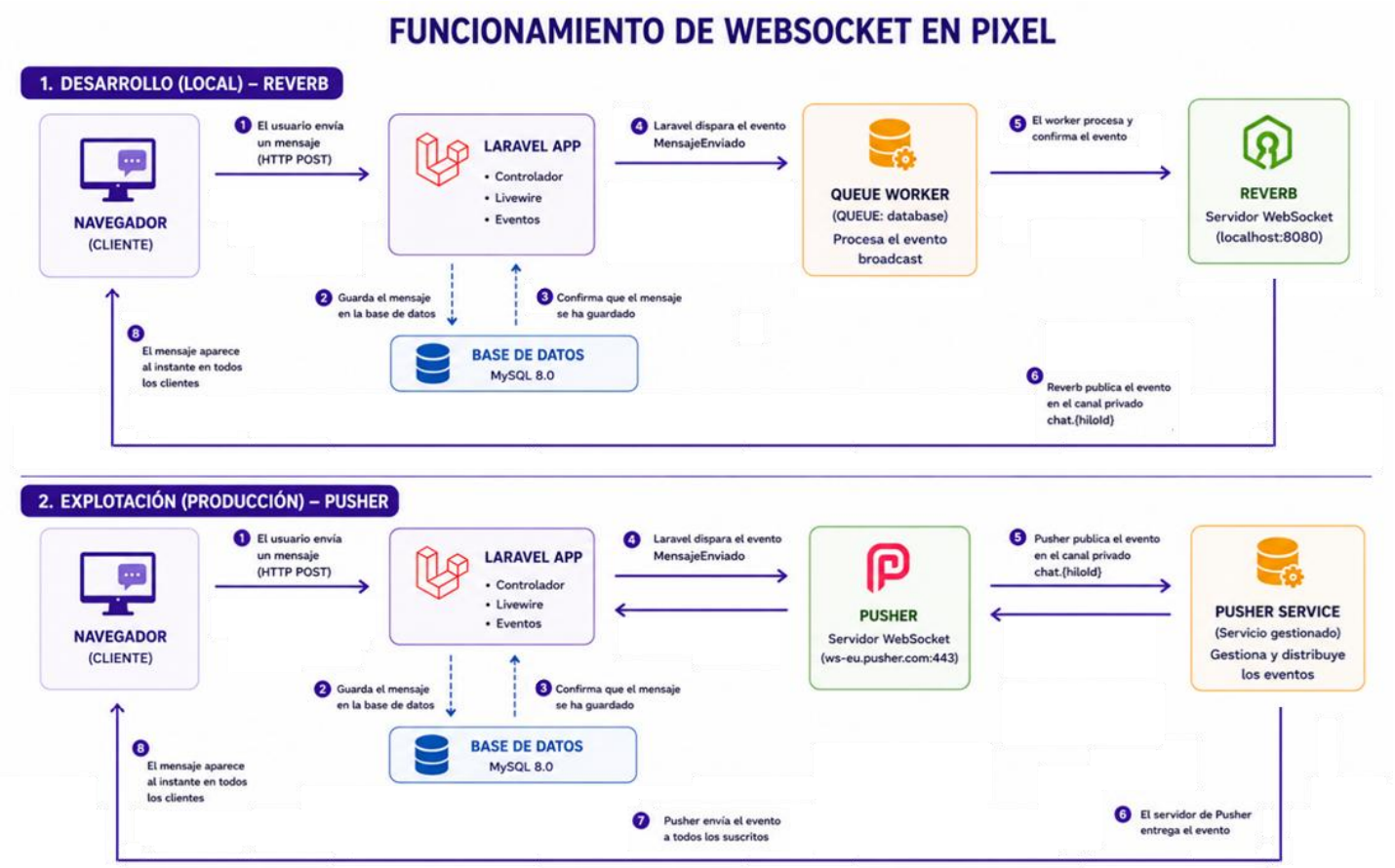
¿Qué es un WebSocket?

HTTP clásico es unidireccional: el navegador pregunta, el servidor responde y la conexión se cierra. Un **WebSocket** abre una conexión permanente bidireccional: servidor y cliente pueden enviarse mensajes en cualquier momento sin que nadie tenga que "preguntar" primero. Esto es lo que hace que un mensaje nuevo aparezca en pantalla de todos los participantes en el momento exacto en que se envía.

Piezas que intervienen

- **Laravel Reverb** — servidor **WebSocket** propio (usado en desarrollo, escucha en ws://localhost:8080).
- **Pusher** — servicio **WebSocket** externo gestionado por terceros (usado en producción).
- **Laravel Echo** — librería JS del navegador que gestiona la conexión al servidor **WebSocket**.
- **Livewire** — framework PHP reactivo que re-renderiza el chat cuando llega un evento.
- **Broadcasting + Queue** — sistema de eventos de **Laravel** que hace de puente entre PHP y el servidor **WebSocket**.

Flujo completo al enviar un mensaje:



Código clave — app/Events/MensajeEnviado.php

El evento implementa ShouldBroadcast. Al dispararse, **Laravel** lo envía al servidor **WebSocket** a través de un canal privado exclusivo de cada hilo.

```
class MensajeEnviado implements ShouldBroadcast
{
    public function broadcastOn(): array
    {
        return [new PrivateChannel("chat." . $this->hiloId)];
    }
    // Nombre corto: el listener de Echo lo escucha como '.MensajeEnviado' (con punto)
    public function broadcastAs(): string
    {
        return 'MensajeEnviado';
    }
}
```

Código clave — routes/channels.php

Antes de que un navegador escuche un canal privado, el servidor verifica que el usuario sea miembro del proyecto al que pertenece el hilo.

```
Broadcast::channel("chat.{hiloId}", function ($user, $hiloId) {
    $hilo = ForoHilo::find($hiloId);
    return $hilo && (
        $proyecto->created_by === $user->id ||
        $proyecto->miembros()->where("user_id", $user->id)->exists()
    );
});
```

Código clave — app/Livewire/ChatHilo.php

El componente **Livewire** escucha el canal **WebSocket**. Cuando llega un evento MensajeEnviado, **Livewire** re-renderiza automáticamente la lista de mensajes desde la base de datos.

```
#[On("echo-private:chat.{hiloId},.MensajeEnviado")]
public function refrescarMensajes(): void
{
    // Livewire recarga $mensajes desde BD y re-renderiza
}
```

Código clave — resources/js/echo.js

El navegador se conecta al servidor **WebSocket** usando **Laravel Echo**. El archivo soporta dos providers: **Reverb** (desarrollo) y **Pusher** (producción), seleccionando el correcto según el objeto window.__BROADCAST_CONFIG__ inyectado por el servidor en cada carga de la vista. Esto evita hardcodear el broadcaster en el JS compilado y permite cambiar de entorno sin recompilar los assets.

```
const cfg      = window.__BROADCAST_CONFIG__ ?? {};
const provider = cfg.provider ?? import.meta.env.VITE_CHANNELS_PROVIDER ?? 'reverb';

if (provider === 'pusher') {
    window.Echo = new Echo({
        broadcaster:  'pusher',
        key:          cfg.key     ?? import.meta.env.VITE_PUSHER_APP_KEY,
        cluster:      cfg.cluster ?? import.meta.env.VITE_PUSHER_APP_CLUSTER,
        forceTLS:     true,
        authEndpoint: (cfg.appBase ?? '') + '/broadcasting/auth',
    });
} else {
    window.Echo = new Echo({
        broadcaster:       'reverb',
        key:               cfg.key    ?? import.meta.env.VITE_REVERB_APP_KEY,
        wsHost:            cfg.host   ?? import.meta.env.VITE_REVERB_HOST,
        wsPort:            cfg.port   ?? import.meta.env.VITE_REVERB_PORT ?? 8080,
        wssPort:           cfg.port   ?? import.meta.env.VITE_REVERB_PORT ?? 8080,
        forceTLS:             (cfg.scheme ?? import.meta.env.VITE_REVERB_SCHEME ?? 'http') ===
'https',
        enabledTransports: ['ws', 'wss'],
        authEndpoint:      (cfg.appBase ?? '') + '/broadcasting/auth',
    });
}
```

Código clave — resources/views/hilo.blade.php

La vista del hilo inyecta en el HTML la configuración del provider activo en ese entorno. El **Blade** PHP lee el .env del servidor en tiempo de ejecución: si el broadcaster activo es pusher, inyecta clave y cluster de **Pusher**; si es reverb, inyecta host, puerto y esquema de **Reverb**.

```
@if(config('broadcasting.default') === 'pusher')
<script>
  window.__BROADCAST_CONFIG__ = {
    provider: 'pusher',
    key:      '{{ config("broadcasting.connections.pusher.key") }}',
    cluster:  '{{ config("broadcasting.connections.pusher.options.cluster") }}',
    appBase:  '{{ rtrim(parse_url(config("app.url"), PHP_URL_PATH) ?? "", "/") }}',
  };
</script>
@else
<script>
  window.__BROADCAST_CONFIG__ = {
    provider: 'reverb',
    key:      '{{ config("broadcasting.connections.reverb.key") }}',
    host:     '{{ config("broadcasting.connections.reverb.options.host") }}',
    port:      {{ config("broadcasting.connections.reverb.options.port", 443) }},
    scheme:   '{{ config("broadcasting.connections.reverb.options.scheme", "https") }}',
    appBase:  '{{ rtrim(parse_url(config("app.url"), PHP_URL_PATH) ?? "", "/") }}',
  };
</script>
@endif
```

Entorno de desarrollo — procesos necesarios

En desarrollo se necesitan cuatro procesos corriendo en paralelo:

```
# Terminal 1 — Servidor web Laravel
php artisan serve

# Terminal 2 — Servidor WebSocket Reverb
php artisan reverb:start

# Terminal 3 — Worker de cola (procesa los eventos)
php artisan queue:work

# Terminal 4 — Compilador frontend
npm run dev
```

Desarrollo vs. Producción

En producción (IONOS + Plesk) no se puede levantar un proceso **Reverb** persistente. Se usa **Pusher**, un servicio externo gratuito que actúa igual que **Reverb**.

Aspecto	Desarrollo	Producción
Servidor WebSocket	Reverb (localhost:8080)	Pusher (ws-eu.pusher.com:443)
Cola de trabajos	QUEUE_CONNECTION=database	QUEUE_CONNECTION=sync (sin worker)
Compilación JS	npm run dev (Vite en vivo)	Assets compilados en public/build/
Protocolo	ws:// (sin TLS)	wss:// (con TLS)

Estructura de la base de datos del foro

```
foro_hilos
  id, proyecto_id, user_id, titulo, contenido
  └── foro_mensajes
        id, hilo_id, user_id, contenido
        └── foro_archivos
              id, mensaje_id, user_id, original_name,
              path, mime_type, size, categoría
```

Categorías de archivo y previsualización

Categoría	Extensiones	Vista previa / comportamiento
imagen	jpg, png, gif, webp, svg...	Se muestra inline en la burbuja del mensaje
pdf	pdf	Modal con iframe de previsualización
audio	mp3, wav, ogg, m4a...	Modal con reproductor de audio
video	mp4, mov, avi, webm...	Modal con reproductor de vídeo
texto	txt, doc, docx, csv...	Modal con primeros 5.000 caracteres

Permisos del foro

Acción	Quién puede	Restricción
Ver el foro	Cualquier miembro del proyecto	—
Crear hilo	Cualquier miembro	—
Eliminar hilo	Autor del hilo o owner del proyecto	Los demás no pueden
Enviar mensaje	Cualquier miembro	—
Eliminar mensaje	Autor del mensaje o owner del proyecto	Los demás no pueden
Descargar archivo	Cualquier miembro	—

12.6 Módulo de datos predefinidos y exportación a motores de juego

PIXEL incluye un módulo de datos predefinidos que permite a los equipos definir entidades del videojuego directamente en la plataforma y exportarlas en un formato JSON compatible con los principales motores: Unity, Unreal Engine y Godot.

El módulo cubre tres tipos de entidad:

- **Personajes** — atributos estadísticos: game_id (identificador único en el juego), nombre, vida, ataque, defensa, velocidad.
- **Ítems** — objetos del inventario: game_id, nombre, descripción, precio, tipo (Arma, Consumible o Misión).
- **Diálogos** — líneas de conversación: id_conversacion, orden, personaje asociado y texto.

Cada proyecto gestiona sus propios datos de forma independiente. El CRUD completo (crear, editar, eliminar) está disponible para el owner y los miembros con acceso al proyecto desde la ruta /proyecto/{id}/predefinicion.

Formato de exportación por motor (generado en el navegador, sin petición al servidor):

- **Unity** — nomenclatura C# con prefijo m_ (m_Id, m_Nombre, m_Vida…). Los ítems se agrupan en una clave ItemList. Compatible con ScriptableObjects y JSON serializer de Unity.

```
{
  "PersonajeList": {
    "Items": [
      {
        "m_ID": "hero_01",
        "m_Nombre": "Kira",
        "m_Health": 100,
        "m_Attack": 45,
        "m_Defense": 30,
        "m_Speed": 1.8
      }
    ]
  },
  "ItemList": {
    "Items": [
      {
        "m_ID": "sword_02",
        "m_Nombre": "Espada Larga",
        "m_Description": "Arma de ataque base",
        "m_Price": 150,
        "m_Type": "Arma"
      }
    ]
  },
  "DialogoList": {
    "Items": [
      {
        "m_ConversationID": "conv_intro",
        "m_Order": 1,
        "m_CharacterID": "hero_01",
        "m_Text": "¿Quién eres tú?"
      }
    ]
  }
}
```

- Unreal Engine — array de objetos con clave "Name" para identificar el activo, compatible con la importación de DataTables a través del editor de UE.

```
{
  "Personajes": [
    {
      "Name": "hero_01",
      "Nombre": "Kira",
      "Vida": 100,
      "Ataque": 45,
      "Defensa": 30,
      "Velocidad": 1.8
    }
  ],
  "Items": [
    {
      "Name": "sword_02",
      "Nombre": "Espada Larga",
      "Descripcion": "Arma de ataque base",
      "Precio": 150,
      "Tipo": "Arma"
    }
  ],
  "Dialogos": [
    {
      "Name": "conv_intro_001",
      "ID_Conversacion": "conv_intro",
      "Orden": 1,
      "Personaje_ID": "hero_01",
      "Texto": "¿Quién eres tú?"
    }
  ]
}
```

- **Godot** — diccionario anidado donde la clave raíz es el game_id del personaje/ítem, listo para cargarse con JSON.parse() en GDScript.

```
{
  "personajes": {
    "hero_01": {
      "nombre": "Kira",
      "vida": 100,
      "ataque": 45,
      "defensa": 30,
      "velocidad": 1.8
    }
  },
  "items": {
    "sword_02": {
      "nombre": "Espada Larga",
      "descripcion": "Arma de ataque base",
      "precio": 150,
      "tipo": "Arma"
    }
  },
  "dialogos": {
    "conv_intro": {
      "1": {
        "personaje_id": "hero_01",
        "texto": "¿Quién eres tú?"
      }
    }
  }
}
```

El archivo descargado recibe el nombre gamedata_{motor}_{proyecto_id}.json. La exportación es inmediata y no requiere ninguna librería PHP adicional (maatwebsite/excel, DomPDF, etc.).

13 Secuencia de desarrollo

El desarrollo se realizó de forma incremental en 13 fases, con commits frecuentes en **GitHub** para documentar la evolución:

Fase	Nombre	Archivos clave creados	Descripción
1	Estructura base	composer.json, .env, package.json	laravel new y configuración inicial del .env y base de datos MySQL.
2	Migraciones y modelos	database/migrations/*.php, app/Models/*.php	Todas las migraciones en orden cronológico y modelos Eloquent con sus relaciones.
3	Autenticación	cUsuario.php, login.blade.php, registro.blade.php	Registro y login con hash personalizado (username+password), sesiones y CSRF.
4	CRUD Proyectos	cProyecto.php, proyectos.blade.php	Crear, editar, eliminar y listar proyectos con verificación de propiedad.
5	CRUD Tareas	cTarea.php, tareas.blade.php	Validación de fechas (end_date ≥ start_date), dependencias y horas estimadas.
6	Control de acceso	VerificarAccesoTipo.php, ProyectoAcceso.php	Middleware area.access y tabla proyecto_accesos para acceso granular por área.
7	Invitaciones	cInvitacion.php, Invitacion.php	Código único PROY-XXXXXXXX, expiración en 7 días y áreas configurables.
8	Gantt y Calendario	gantt.blade.php, calendario.blade.php	Diagrama de barras horizontal con líneas de dependencia y vista mensual.
9	Sistema de archivos	cArchivo.php, archivos.blade.php	Upload drag-and-drop, categorización automática y previsualización de texto.
10	Datos predefinidos y exportación	app/Models/Personaje.php, Item.php, Dialogo.php, app/Http/Controllers/cPredefinicion.php, predefinicion.blade.php	CRUD de Personajes, Items y Diálogos por proyecto. Exportación client-side a JSON en formato Unity, Unreal Engine y Godot.
11	Foro y Chat en tiempo real	app/Events/MensajeEnviado.php, app/Livewire/ChatHilo.php, foro.blade.php, hilo.blade.php, routes/channels.php	Chat por hilos con WebSockets (Reverb en dev, Pusher en prod), subida de archivos en mensajes y previsualización por categoría.
12	Notas de tarea	app/Models/NotaTarea.php, app/Http/Controllers/cNotaTarea.php, tarea.blade.php	Módulo de notas de seguimiento por tarea. Vista de detalle individual (tarea.blade.php). Simplificación de la tabla participaciones (migración 2026_05_11_000001).
13	Frontend Tailwind	resources/css/, vite.config.js	Refinamiento del diseño responsivo con Tailwind CSS 4 y Vite 7.

14 Repositorio de software

El código fuente completo, incluyendo todos los commits del proceso de desarrollo, está disponible en:

https://github.com/CrisMatVeg/CMVProyectoFinalLaravel

El historial de commits refleja la evolución completa del proyecto. El archivo .gitignore excluye automáticamente los ficheros sensibles o regenerables: .env, /vendor/, /node_modules/, /public/build/ y /storage/app/public/.

15 Aplicación desarrollada

La aplicación PIXEL está desplegada y accesible en producción en:

https://cristianmatveg.ieslossauces.es

Funcionalidades disponibles en la versión presentada:

- Autenticación completa: registro, login y edición de perfil.
- Creación y edición de proyectos con descripción y índices de progreso.
- Gestión de tareas con dependencias, hitos, fechas y horas estimadas.
- Asignación de usuarios a tareas.
- Control de acceso a proyectos por áreas de trabajo.
- Sistema de invitaciones con código único y expiración configurable.
- Subida y descarga de archivos adjuntos con previsualización de texto.
- Diagrama de Gantt interactivo con líneas de dependencia.
- Vista de Calendario mensual con marcado de hitos.
- Foro de discusión por proyecto organizado en hilos de conversación.
- Chat en tiempo real con **WebSockets**: los mensajes aparecen al instante para todos los miembros.
- Subida de archivos en el chat con previsualización integrada (imágenes inline, modal para PDF, audio y vídeo).
- Gestión de datos predefinidos del juego: Personajes, Ítems y Diálogos reutilizables por proyecto.
- Exportación de datos a Unity, Unreal Engine y Godot en formato JSON adaptado a cada motor, generada en el navegador sin intervención del servidor.

Proceso de Implantación — Ionos con Plesk

Una vez finalizado el desarrollo, la aplicación se despliega en el entorno de explotación: un servidor de Ionos gestionado con el panel de control Plesk.

La aplicación se despliega en producción en un servidor de Ionos gestionado con Plesk:

- **Proveedor: Ionos (1&1)** — hosting compartido.
- Panel de control: Plesk Obsidian.
- Servidor web: Apache 2.4 con PHP 8.2 en modo PHP-FPM.
- Base de datos: **MySQL** 8.0.
- Acceso: SSH habilitado + SFTP.
- Document Root: apunta a la carpeta public/ del proyecto **Laravel**.

16 Pasos del despliegue

7. Crear subdominio en Plesk (opcional; en este caso se usa directamente el dominio proporcionado por el IES).
8. Crear la base de datos **MySQL** desde Plesk → Bases de datos → Añadir base de datos.
9. Ir al apartado de **Laravel** (**Laravel** Toolkit) y darle a Instalar la aplicación.
10. Elegir el dominio y seleccionar la opción Instalar desde repositorio remoto, introduciendo la URL del repositorio de **GitHub**.
11. Una vez instalada, editar el fichero .env desde el panel de **Laravel** Toolkit (botón Editar) con los datos de la base de datos de producción.
12. Ejecutar las migraciones: **php artisan** migrate --force
13. Ir a la pestaña Despliegue y pulsar Desplegar, sin modificar las opciones marcadas por defecto.
14. Si es necesario, limpiar caché desde la pestaña **Artisan**: **php artisan** cache:clear o config:clear.

Conclusiones y Trabajo Futuro

El desarrollo de este proyecto ha permitido afianzar los conocimientos del ciclo formativo en un entorno real y funcional. Se ha demostrado que **Laravel** es un framework maduro y productivo que reduce significativamente el tiempo de desarrollo gracias a sus abstracciones (**Eloquent ORM**, **Blade**, **Artisan**), a la vez que mantiene el código organizado y mantenible siguiendo el patrón **MVC**.

La integración de herramientas modernas como **Tailwind CSS 4** y **Vite 7** ha permitido construir una interfaz responsiva y estéticamente cuidada sin necesidad de frameworks JavaScript complejos. El uso de Git y **GitHub** desde el inicio ha resultado fundamental para organizar el trabajo en fases y documentar la evolución del desarrollo.

El despliegue en Ionos con Plesk ha supuesto un aprendizaje valioso sobre los entornos de producción reales: configuración del servidor web, gestión de permisos, variables de entorno de producción y optimización de la aplicación para su puesta en marcha.

La implementación del foro con chat en tiempo real ha requerido integrar tecnologías adicionales (**WebSockets** con **Reverb**/**Pusher**, **Laravel Echo**, **Livewire** y el sistema de broadcasting de **Laravel**), ampliando significativamente las capacidades interactivas de la aplicación más allá de la gestión de proyectos estática.

El módulo de datos predefinidos aporta un valor diferencial claro respecto a otras plataformas de gestión de proyectos: la posibilidad de definir entidades del juego (Personajes, Ítems y Diálogos) directamente en la plataforma y exportarlas en formato JSON adaptado a Unity, Unreal Engine o Godot, eliminando la necesidad de herramientas externas para este paso del proceso de producción.

Comparativa con Trello y Notion

Trello y Notion son las referencias más habituales en gestión de proyectos para equipos creativos, por lo que resulta relevante situar PIXEL frente a ellas. Trello ofrece un sistema de tableros Kanban sencillo y visual, pero carece de control de acceso por roles dentro de un mismo proyecto, no distingue entre tipos de colaboradores y no dispone de ninguna integración con el flujo de desarrollo de videojuegos. Notion es más flexible: permite crear bases de datos, wikis y documentación dentro del mismo espacio, pero esa misma generalidad la convierte en una herramienta que requiere configuración manual para cada caso de uso. Ninguna de las dos permite restringir lo que ve cada miembro según su disciplina técnica, exportar datos de juego en formatos nativos para motores, visualizar dependencias entre tareas en un diagrama de Gantt interactivo o comunicarse en tiempo real mediante WebSockets dentro de la propia plataforma. PIXEL no pretende competir en amplitud de funcionalidades con estas herramientas de propósito general, sino ofrecer un flujo de trabajo específico y completo para estudios de videojuegos independientes, así como una plataforma original, única y novedosa, donde la especialización por áreas y la integración con motores de juego aportan un valor que Trello, Notion o cualquier otra herramienta de organización no cubre de forma nativa.

17 Trabajo futuro

- API REST completa con autenticación por tokens mediante **Laravel** Sanctum, para dar soporte a clientes móviles.
- Gráficas de rendimiento del equipo (horas trabajadas vs. estimadas por área).
- Internacionalización de la aplicación con el sistema de traducciones de **Laravel**.
- Tests automatizados con PHPUnit y Pest para garantizar la calidad del código en cada nueva versión.

18 Uso de IA (Claude Code) en el proyecto

Se construyeron manualmente los cimientos del proyecto (login/logout, navegación entre páginas y controles de acceso a estas, CRUD de proyectos, estructura de BD y configuraciones) en enero y febrero, obteniendo una versión básica de la app que permitía acceder y crear proyectos, así como eliminarlos y modificarlos. A partir de abril, la colaboración con IA enfocada al uso profesional y laboral aceleró el desarrollo x10, permitiendo implementar features complejas como WebSockets en tiempo real, sistema de permisos granular y responsive design completo en semanas en lugar de meses.

Durante el desarrollo de este proyecto se ha utilizado Claude Code como asistente de IA integrado directamente en VSCode. A diferencia de un chatbot genérico, Claude Code tiene acceso a los archivos del proyecto, puede ejecutar comandos en la terminal, leer y editar código, realizar commits y conectarse a servicios externos como GitHub. La versión utilizada es Claude Sonnet 4.6, configurada con nivel de esfuerzo high (alta calidad de razonamiento).

18.1 ¿Qué es Claude Code?

Herramienta de IA de Anthropic para desarrollo de software. Disponible como:

- Extensión de VSCode ← **la usada en este proyecto**
- CLI (terminal)
- App de escritorio (Mac/Windows)
- Web: claude.ai/code

18.2 Planes y precios (a mayo 2026)

Plan	Precio	Claude Code
Free	Gratis	No incluido
Pro (el utilizado en este proyecto)	~$20/mes	Incluido, con límite de uso mensual
Max (5x)	~$100/mes	Incluido, 5× más uso que Pro
Max (20x)	~$200/mes	Incluido, 20× más uso que Pro
Teams	~$30/usuario/mes	Incluido para todo el equipo
Enterprise	Precio personalizado	Incluido, sin límites

Los precios son orientativos. Verificar siempre en **claude.ai/pricing** porque cambian.

18.3 Cómo funciona el consumo

Claude Code **no cobra por mensaje** sino por **tokens** (fragmentos de texto):

- Cada vez que Claude lee un archivo > consume tokens de entrada
- Cada respuesta generada > consume tokens de salida
- Los planes Pro/Max incluyen una cuota mensual de tokens
- Si se supera la cuota > Claude Code se pausa hasta el siguiente ciclo

18.4 Archivos de configuración

- **CLAUDE.md** - Archivo en la raíz del proyecto que actúa como manual de instrucciones leído por Claude al inicio de cada sesión. El usuario define aquí el estilo de comunicación (español, directo, nivel principiante…), el formato de commits (convencional en español, siempre con confirmación previa), el flujo de merge (developerCMV a master con --no-ff, luego vuelta a developerCMV con fast-forward), la prohibición de abrir el navegador desde comandos y la política de archivos temporales (eliminación inmediata tras su uso).
- **.claude/settings.json** - Permisos a nivel de proyecto. Se permiten sin confirmación: php, composer, npm, npx, node y comandos git habituales (status, diff, log, add, commit, etc.), así como operaciones de sistema de archivos (ls, mkdir, rm, cp, mv). Están bloqueados los comandos de apertura de navegador (start http*, xdg-open, open http*). Incluye dos hooks (comandos que tras un determinado evento se ejecutan solos): Stop (detecta cambios sin commitear al terminar la respuesta y avisa automáticamente) y PreToolUse/Bash (bloquea cualquier intento de abrir el navegador antes de ejecutar un comando).

18.5 Memoria persistente entre sesiones

Claude dispone de un sistema de memoria que persiste entre conversaciones. Las memorias activas en este proyecto son:

- **feedback_start_servers.md:** Al inicio de cada sesión, Claude arranca automáticamente en background el servidor Reverb (WebSocket para chat en tiempo real) y el queue worker (colas de trabajo), sin que el usuario tenga que pedirlo.
- **feedback_merge_flow.md:** Flujo completo obligatorio tras cada merge a master: merge --no-ff desde developerCMV, merge fast-forward de vuelta, y push a ambas ramas.

18.6 Modo de trabajo y flujo de sesión

El flujo habitual de una sesión es: Claude Code se activa al abrir VSCode, arranca Reverb y queue:work en background, se describe la tarea en lenguaje natural, Claude analiza el código afectado, propone y ejecuta los cambios si tras supervisarlos son confirmados, el hook Stop detecta cambios sin commitear y avisa; se confirma y Claude genera el mensaje de commit en español. Si va a master, se ejecuta el flujo completo de merge y push en ambas ramas.

Para tareas complejas se activa el Plan Mode: Claude lee el código relevante, diseña un plan, lo escribe en un archivo y espera aprobación o correcciones del usuario antes de ejecutar cambios reales.

18.7 Restricciones establecidas

- Sin inglés: toda comunicación en español, incluidas las explicaciones técnicas.
- Sin navegador: Claude no puede abrir URLs (bloqueado por hook y lista de denegación).
- Sin commits automáticos: siempre se requiere confirmación explícita del usuario antes de commitear.
- Sin archivos basura: los scripts temporales se eliminan inmediatamente tras usarse.
- Sin features extra: Claude implementa exactamente lo pedido, sin añadir funcionalidad no solicitada.

18.8 Cómo se construyó esta configuración

La configuración no se estableció de golpe sino que fue evolucionando a lo largo de varias sesiones: primero se creó CLAUDE.md con las reglas base, después se configuraron los permisos en settings.json para evitar confirmaciones repetitivas, los hooks se añadieron para automatizar el aviso de commits y bloquear el navegador y la memoria de Reverb/queue se guardó cuando se indicó que se requería arranque automático. Cada corrección realizada durante una sesión se convierte en una memoria o regla para no repetir el error.

Webgrafías y Referencias

19 Documentación oficial

Laravel 12 — Documentación oficial: https://laravel.com/docs/12.x
Eloquent ORM: https://laravel.com/docs/12.x/eloquent
Tailwind CSS 4: https://tailwindcss.com/docs
Vite: https://vite.dev/guide/
Composer: https://getcomposer.org/doc/
PHP 8.2 — Manual oficial: https://www.php.net/manual/es/
MySQL 8.0: https://dev.mysql.com/doc/refman/8.0/en/
Claude Code: https://code.claude.com/docs/es/overview

20 Herramientas y entornos

XAMPP: https://www.apachefriends.org
Git: https://git-scm.com/doc
GitHub — Guías y documentación: https://docs.github.com
Plesk — Documentación de administración: https://docs.plesk.com
Pusher — Sitio oficial: https://pusher.com/

21 Referencias adicionales

Stack Overflow Developer Survey 2024 — Frameworks más usados: https://survey.stackoverflow.co/2024/
Packagist — Estadísticas de descargas de Laravel: https://packagist.org/packages/laravel/framework/stats