Documentacion del sistema de gestion de talleres

Fecha de creación	@2 de julio de 2025 15:07
Etiquetas	

1. README.md - Documentación Principal

- **Tecnologías utilizadas**: Laravel, Livewire, Tailwind CSS, Alpine.js, SQLite
- **Arquitectura del proyecto**: Estructura de directorios y componentes
- **Modelos de datos**: User, Workshop, Task, WorkshopEnrollment
- **Funcionalidades por rol**: Admin y Student
- **Componentes de interfaz**: Componentes Livewire y vistas Blade
- **Sistema de estilos**: Tailwind personalizado con modo oscuro
- **Instalación y configuración**: Pasos detallados
- **Características UX/UI**: Responsivo, accesible, interactivo

2. ARCHITECTURE.md - Arquitectura Técnica

- **Patrones de diseño**: MVC, Repository, Middleware, Observer
- **Flujo de datos**: Diagramas de autenticación y gestión
- **Arquitectura de base de datos**: Relaciones e integridad
- **Componentes Livewire**: Ciclo de vida y comunicación
- **Seguridad**: Capas y flujos de autorización
- **Frontend**: Estructura de estilos y JavaScript
- **Rendimiento**: Optimizaciones y estrategias de caché
- **Escalabilidad**: Consideraciones horizontales y verticales

3. API_DOCUMENTATION.md - Documentación de API

- **Rutas actuales**: Web routes para admin y student

- **Componentes Livewire**: Endpoints AJAX y métodos
- **Modelos de datos**: Estructuras JSON completas
- **Autenticación**: Sistema actual y futuro con Sanctum
- **API REST futura**: Endpoints propuestos
- **Códigos de estado**: HTTP status codes
- **Configuración**: CORS y Rate Limiting

🛠️ Tecnologías Principales:

Backend:

- Laravel 11 (Framework PHP)
- Livewire 3 (Componentes reactivos)
- SQLite (Base de datos)
- Laravel Sanctum (Autenticación)

Frontend:

- Tailwind CSS (Estilos)
- Alpine.js (Interactividad)
- Blade Templates (Vistas)
- Vite (Build tool)

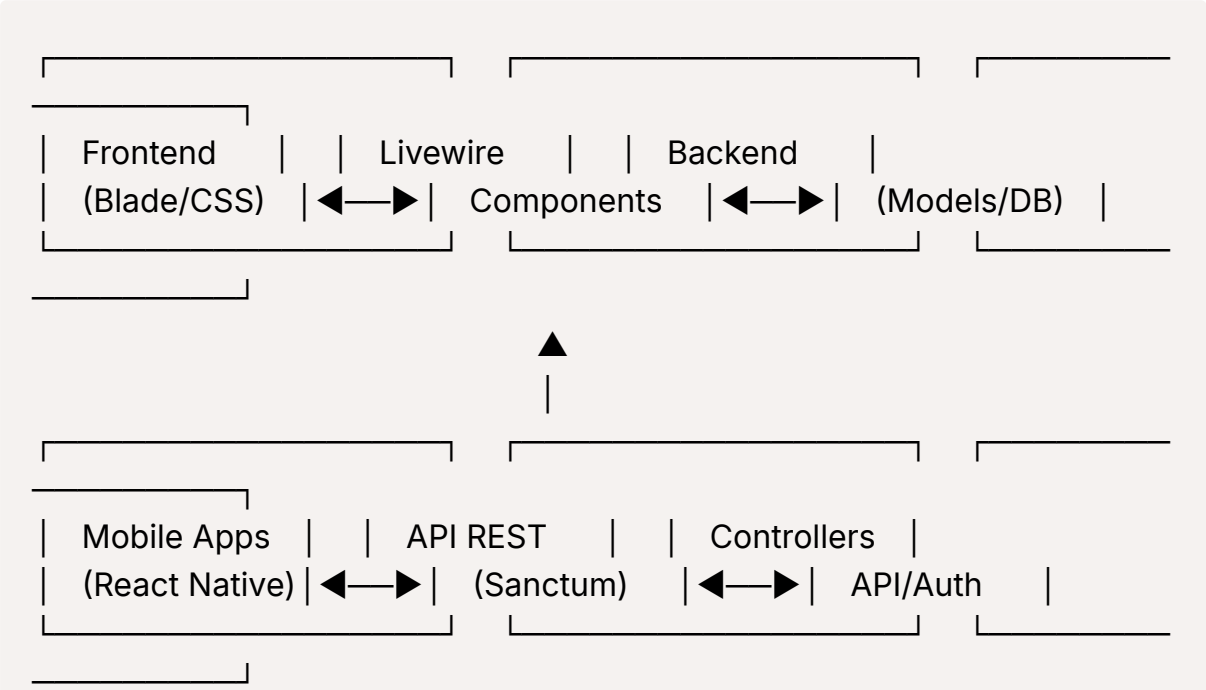
Funcionalidades:

- Sistema de roles (Admin/Student)
- Gestión de talleres y tareas
- Dashboard personalizado por rol
- Diseño responsivo
- Validación en tiempo real

📚 Documentación Detallada de Componentes

🏗️ Arquitectura General del Sistema

El sistema está construido con **Laravel + Livewire + APIs REST** siguiendo una arquitectura híbrida:



🎯 1. MODELOS DE DATOS (Models)

📋 User.php - Gestión de Usuarios

Función: Maneja usuarios (administradores y estudiantes) con soporte para APIs

```
// Ejemplo de uso en Livewire
$admin = User::create([
    'name' ⇒ 'Juan Pérez',
    'email' ⇒ 'admin@test.com',
    'role' ⇒ 'admin',
    'password' ⇒ bcrypt('password')
]);
```

```
// Ejemplo de uso en API
// POST /api/auth/register
{
   "name": "Juan Pérez",
   "email": "admin@test.com",
   "password": "password",
   "password_confirmation": "password",
   "role": "admin"
}

// Verificar rol (usado en ambos contextos)
if ($user->isAdmin()) {
   // Acceso a funciones de administrador
}

// Obtener talleres del estudiante
$workshops = $user->workshops; // Relación many-to-many

// Obtener tareas asignadas
$tasks = $user->tasks; // Relación one-to-many
```

Relaciones Clave:

```
// Un usuario puede estar en múltiples talleres
public function workshops()
{
   return $this->belongsToMany(Workshop::class, 'workshop_enrollments');
}

// Un usuario puede tener múltiples tareas
public function tasks()
{
   return $this->hasMany(Task::class, 'assigned_to');
}

// Métodos para autenticación API
public function createToken($name)
```

```
{
    return $this->tokens()->create([
        'name' => $name,
        'token' => hash('sha256', $plainTextToken = Str::random(40)),
    ]);
}
```

🏢 Workshop.php - Gestión de Talleres

Función: Representa los talleres extracurriculares con soporte web y API

```
// Crear un taller (Livewire)
$workshop = Workshop::create([
    'name' => 'Taller de Programación Web',
    'description' => 'Aprende desarrollo web moderno',
    'instructor' => 'Prof. Ana García',
    'capacity' => 25,
    'start_date' => '2024-02-01',
    'end_date' => '2024-04-01',
    'location' => 'Lab A',
    'status' => 'active'
]);

// Crear un taller (API REST)
// POST /api/workshops
{
    "name": "Taller de Programación Web",
    "description": "Aprende desarrollo web moderno",
    "instructor": "Prof. Ana García",
    "capacity": 25,
    "start_date": "2024-02-01",
    "end_date": "2024-04-01",
    "location": "Lab A",
    "status": "active"
}

// Inscribir estudiante (Livewire)
$workshop->students()->attach($student->id);
```

```
// Inscribir estudiante (API)
// POST /api/workshops/{id}/enroll
```

Atributos Calculados:

```
// Estudiantes inscritos
public function getEnrolledCountAttribute()
{
    return $this->students()->count();
}

// Cupos disponibles
public function getAvailableSpotsAttribute()
{
    return $this->capacity - $this->enrolled_count;
}

// Verificar estado activo
public function isActive()
{
    return $this->status === 'active';
}
```

✅ Task.php - Gestión de Tareas

Función: Representa las tareas asignadas a estudiantes con soporte completo API

```
// Crear una tarea (Livewire)
$task = Task::create([
    'title' => 'Crear página web personal',
    'description' => 'Desarrolla una página con HTML y CSS',
    'workshop_id' => $workshop->id,
    'assigned_to' => $student->id,
    'assigned_by' => $admin->id,
    'due_date' => now()->addDays(7),
    'priority' => 'high',
```

```
    'status' ⇒ 'pending'
]);

// Crear una tarea (API REST)
// POST /api/tasks
{
    "title": "Crear página web personal",
    "description": "Desarrolla una página con HTML y CSS",
    "workshop_id": 1,
    "assigned_to": 2,
    "due_date": "2024-02-15T23:59:59",
    "priority": "high",
    "status": "pending"
}

// Completar tarea (Livewire)
$task→update([
    'status' ⇒ 'completed',
    'completed_at' ⇒ now()
]);

// Completar tarea (API)
// POST /api/tasks/{id}/complete
{
    "completion_notes": "Tarea completada exitosamente"
}
```

Métodos Útiles:

```
// Verificar vencimiento
public function isOverdue()
{
    return $this→due_date && $this→due_date→isPast() && $this→status !=
= 'completed';
}

// Colores dinámicos para UI
public function getPriorityColorAttribute()
```

```
{
    return match($this->priority) {
        'high' => 'text-red-600 bg-red-100',
        'medium' => 'text-yellow-600 bg-yellow-100',
        'low' => 'text-green-600 bg-green-100',
    };
}

// Colores para estado
public function getStatusColorAttribute()
{
    return match($this->status) {
        'completed' => 'text-green-600 bg-green-100',
        'in_progress' => 'text-blue-600 bg-blue-100',
        'pending' => 'text-gray-600 bg-gray-100',
    };
}
```

⚡ 2. COMPONENTES LIVEWIRE (Interfaz Web)

🎛️ Admin/Dashboard.php - Panel Administrativo

Función: Dashboard principal para administradores con estadísticas

```
class Dashboard extends Component
{
    // Propiedades reactivas
    public $totalStudents;
    public $totalWorkshops;
    public $totalTasks;
    public $completedTasks;

    public function mount()
    {
        $this->loadStats(); // Cargar datos al inicializar
    }

    public function loadStats()
```

```
    {
        // Consultas optimizadas (mismas que usa la API)
        $this->totalStudents = User::where('role', 'student')->count();
        $this->totalWorkshops = Workshop::count();
        $this->totalTasks = Task::count();
        $this->completedTasks = Task::where('status', 'completed')->count();
    }
}
```

Equivalente en API:

```
GET /api/dashboard/admin
Authorization: Bearer {token}

Response:
{
    "success": true,
    "data": {
        "stats": {
            "total_students": 25,
            "total_workshops": 4,
            "total_tasks": 15,
            "completed_tasks": 8
        }
    }
}
```

🏢 Admin/WorkshopManager.php - CRUD de Talleres

Función: Gestión completa de talleres (crear, editar, eliminar)

```
class WorkshopManager extends Component
{
    use WithPagination;

    // Propiedades del formulario
    public $showModal = false;
    public $editMode = false;
```

```
    public $name = '';
    public $description = '';
    // ... más propiedades

    public function save()
    {
        $this->validate();

        $data = [
            'name' => $this->name,
            'description' => $this->description,
            // ... más campos
        ];

        if ($this->editMode) {
            Workshop::find($this->workshopId)->update($data);
        } else {
            Workshop::create($data);
        }

        $this->closeModal();
        session()->flash('message', 'Taller guardado exitosamente.');
    }
}
```

Equivalente en API:

```
# Crear taller
POST /api/workshops
Authorization: Bearer {token}
Content-Type: application/json

{
    "name": "Nuevo Taller",
    "description": "Descripción del taller",
    "instructor": "Prof. García",
    "capacity": 20,
    "start_date": "2024-03-01",
```

```
    "end_date": "2024-05-01",
    "location": "Lab A"
}

# Actualizar taller
PUT /api/workshops/1
Authorization: Bearer {token}
Content-Type: application/json

{
    "name": "Taller Actualizado",
    "capacity": 25
}
```

✅ Admin/TaskManager.php - CRUD de Tareas

Función: Asignación y gestión de tareas

```
class TaskManager extends Component
{
    public function save()
    {
        $this->validate();

        $data = [
            'title' => $this->title,
            'description' => $this->description,
            'workshop_id' => $this->workshop_id,
            'assigned_to' => $this->assigned_to,
            'assigned_by' => auth()->id(),
            'due_date' => $this->due_date,
            'priority' => $this->priority,
            'status' => $this->status,
        ];

        Task::create($data);
        $this->closeModal();
```

```
    }
}
```

Equivalente en API:

```
POST /api/tasks
Authorization: Bearer {token}
Content-Type: application/json

{
    "title": "Nueva Tarea",
    "description": "Descripción de la tarea",
    "workshop_id": 1,
    "assigned_to": 2,
    "due_date": "2024-02-15T23:59:59",
    "priority": "medium"
}
```

👨‍🎓 Student/Dashboard.php - Panel del Estudiante

Función: Dashboard personalizado para estudiantes

```
class Dashboard extends Component
{
    public function markTaskCompleted($taskId)
    {
        $task = Task::find($taskId);

        if ($task && $task->assigned_to === auth()->id()) {
            $task->update([
                'status' => 'completed',
                'completed_at' => now(),
            ]);

            $this->loadStats();
            session()->flash('message', 'Tarea completada.');
        }
```

```
    }
}
```

Equivalente en API:

```
POST /api/tasks/1/complete
Authorization: Bearer {token}
Content-Type: application/json

{
    "completion_notes": "Tarea completada exitosamente"
}
```

🚀 3. CONTROLADORES API REST

🔐 Api/AuthController.php - Autenticación

Función: Maneja autenticación con tokens Sanctum

```
class AuthController extends Controller
{
    public function login(Request $request)
    {
        $request->validate([
            'email' => 'required|email',
            'password' => 'required',
        ]);

        $user = User::where('email', $request->email)->first();

        if (!$user || !Hash::check($request->password, $user->password)) {
            throw ValidationException::withMessages([
                'email' => ['Credenciales incorrectas.'],
            ]);
        }

        $token = $user->createToken('api-token')->plainTextToken;
```

```
        return response()->json([
            'success' => true,
            'data' => [
                'user' => $user,
                'token' => $token,
                'token_type' => 'Bearer',
            ]
        ]);
    }
}
```

Flujo de Autenticación:

```
1. Cliente envía credenciales
   ↓
2. Servidor valida credenciales
   ↓
3. Si son válidas, genera token Sanctum
   ↓
4. Retorna token al cliente
   ↓
5. Cliente usa token en siguientes peticiones
```

🏢 Api/WorkshopController.php - Gestión de Talleres

Función: CRUD completo de talleres con autorización

```
class WorkshopController extends Controller
{
    public function store(Request $request)
    {
        $this->authorize('create', Workshop::class); // Policy check

        $request->validate([
            'name' => 'required|string|max:255',
            'description' => 'required|string',
            // ... más validaciones
        ]);
```

```
        $workshop = Workshop::create($request->all());

        return response()->json([
            'success' => true,
            'message' => 'Taller creado exitosamente',
            'data' => ['workshop' => $workshop]
        ], 201);
    }

    public function enroll(Request $request, $id)
    {
        $workshop = Workshop::findOrFail($id);
        $user = $request->user();

        // Validaciones de negocio
        if (!$user->isStudent()) {
            return response()->json([
                'success' => false,
                'message' => 'Solo estudiantes pueden inscribirse'
            ], 403);
        }

        // Crear inscripción
        WorkshopEnrollment::create([
            'user_id' => $user->id,
            'workshop_id' => $workshop->id,
            'enrollment_date' => now(),
        ]);

        return response()->json([
            'success' => true,
            'message' => 'Inscripción exitosa'
        ]);
    }
}
```

✅ Api/TaskController.php - Gestión de Tareas

Función: CRUD de tareas con filtros por rol

```
class TaskController extends Controller
{
    public function index(Request $request)
    {
        $user = $request->user();
        $query = Task::with(['assignedTo', 'workshop']);

        // Filtrar por rol
        if ($user->isStudent()) {
            $query->where('assigned_to', $user->id);
        }

        // Filtros opcionales
        if ($request->has('status')) {
            $query->where('status', $request->status);
        }

        $tasks = $query->paginate($request->get('per_page', 15));

        return response()->json([
            'success' => true,
            'data' => [
                'tasks' => $tasks->items(),
                'pagination' => [
                    'current_page' => $tasks->currentPage(),
                    'total' => $tasks->total(),
                ]
            ]
        ]);
    }

    public function complete(Request $request, $id)
    {
        $task = Task::findOrFail($id);
        $user = $request->user();
```

```
        // Verificar permisos
        if ($task->assigned_to !== $user->id) {
            return response()->json([
                'success' => false,
                'message' => 'Sin permisos'
            ], 403);
        }

        $task->update([
            'status' => 'completed',
            'completed_at' => now(),
            'completion_notes' => $request->completion_notes,
        ]);

        return response()->json([
            'success' => true,
            'message' => 'Tarea completada'
        ]);
    }
}
```

📊 Api/DashboardController.php - Dashboards

Función: Proporciona datos de dashboard según el rol

```
class DashboardController extends Controller
{
    public function dashboard(Request $request)
    {
        $user = $request->user();

        if ($user->isAdmin()) {
            return $this->adminDashboard($request);
        } else {
            return $this->studentDashboard($request);
        }
    }
```

```
    public function adminDashboard(Request $request)
    {
        $stats = [
            'total_students' => User::where('role', 'student')->count(),
            'total_workshops' => Workshop::count(),
            'total_tasks' => Task::count(),
            'completed_tasks' => Task::where('status', 'completed')->count(),
        ];

        $recentTasks = Task::with(['assignedTo', 'workshop'])
                    ->latest()
                    ->take(10)
                    ->get();

        return response()->json([
            'success' => true,
            'data' => [
                'stats' => $stats,
                'recent_tasks' => $recentTasks,
            ]
        ]);
    }
}
```

🛡️ 4. SISTEMA DE SEGURIDAD

🔐 Middleware de Autenticación

AdminMiddleware.php (Para rutas web):

```
class AdminMiddleware
{
    public function handle(Request $request, Closure $next): Response
    {
        if (!auth()->check() || !auth()->user()->isAdmin()) {
            abort(403, 'Acceso denegado');
        }
        return $next($request);
```

```
    }
}
```

Laravel Sanctum (Para APIs):

```
// En routes/api.php
Route::middleware('auth:sanctum')->group(function () {
    Route::get('dashboard', [DashboardController::class, 'dashboard']);
    Route::apiResource('workshops', WorkshopController::class);
});
```

🔒 Policies para Autorización

WorkshopPolicy.php:

```
class WorkshopPolicy
{
    public function create(User $user): bool
    {
        return $user->isAdmin();
    }

    public function update(User $user, Workshop $workshop): bool
    {
        return $user->isAdmin();
    }

    public function delete(User $user, Workshop $workshop): bool
    {
        return $user->isAdmin();
    }
}
```

TaskPolicy.php:

```
class TaskPolicy
{
    public function view(User $user, Task $task): bool
```

```
    {
        return $user->isAdmin() || $task->assigned_to === $user->id;
    }

    public function update(User $user, Task $task): bool
    {
        return $user->isAdmin() ||
               ($user->isStudent() && $task->assigned_to === $user->id);
    }
}
```

🔄 5. FLUJOS DE DATOS COMPLETOS

📊 Flujo: Crear Nueva Tarea

Livewire (Web):

```
1. Admin va a /admin/tasks
   ↓
2. TaskManager::render() carga datos
   ↓
3. Admin hace clic "Nueva Tarea"
   ↓
4. openModal() → $showModal = true
   ↓
5. Admin llena formulario y envía
   ↓
6. save() → validate() → Task::create()
   ↓
7. Livewire re-renderiza automáticamente
```

API REST (Móvil/Integración):

```
1. Cliente hace POST /api/tasks
   ↓
2. Middleware auth:sanctum verifica token
   ↓
3. TaskController::store() ejecuta
```

↓
4. Policy verifica permisos (admin)
↓
5. Valida datos de entrada
↓
6. Task::create() guarda en BD
↓
7. Retorna JSON con tarea creada

📝 Flujo: Estudiante Completa Tarea

Livewire (Web):

1. Estudiante va a /student/tasks
↓
2. Vista carga: auth()→user()→tasks()
↓
3. Estudiante hace clic "Completar"
↓
4. POST a route('student.task.complete')
↓
5. Controlador actualiza BD
↓
6. Redirect con mensaje de éxito

API REST (Móvil):

1. App móvil hace POST /api/tasks/1/complete
↓
2. Sanctum verifica token
↓
3. TaskController::complete() ejecuta
↓
4. Verifica que tarea pertenece al usuario
↓
5. Actualiza status y completed_at
↓
6. Retorna JSON con tarea actualizada

🎨 6. SISTEMA DE RESPUESTAS API

📋 Estructura Estándar de Respuestas

Respuesta Exitosa:

```
{
   "success": true,
   "message": "Operación exitosa",
   "data": {
      "user": {...},
      "token": "1|abc123..."
   }
}
```

Respuesta de Error:

```
{
   "success": false,
   "message": "Error de validación",
   "errors": {
      "email": ["El campo email es obligatorio"],
      "password": ["Mínimo 6 caracteres"]
   }
}
```

🔧 Middleware de Respuestas API

ApiResponseMiddleware.php:

```
class ApiResponseMiddleware
{
   public function handle(Request $request, Closure $next): Response
   {
      $response = $next($request);

      // Solo para rutas API
      if (!$request->is('api/*')) {
```

```
            return $response;
        }

        // Agregar headers CORS
        $response->headers->set('Access-Control-Allow-Origin', '*');
        $response->headers->set('Access-Control-Allow-Methods', 'GET, PO
ST, PUT, DELETE');

        return $response;
    }
}
```

🔗 7. CONEXIONES ENTRE SISTEMAS

🌐 Arquitectura Híbrida

```
┌──────────────────────────────────────────────────────────────────────
──────────┐
│                 LARAVEL BACKEND                    │
├──────────────────────────────────────────────────────────────────────
──────────┤
│  Models (User, Workshop, Task, WorkshopEnrollment)      │
│  ├─ Eloquent Relationships                          │
│  ├─ Accessors & Mutators                            │
│  └─ Business Logic                                  │
├──────────────────────────────────────────────────────────────────────
──────────┤
│                 DUAL INTERFACE                     │
├──────────────────────────┬──────────────────────────────────────────
──────────┤
│   LIVEWIRE      │          API REST                 │
│   (Web App)     │        (Mobile/Integration)          │
├──────────────────────────┼──────────────────────────────────────────
──────────┤
│ • Components     │ • Controllers                     │
│ • Blade Views    │ • JSON Responses                     │
│ • Session Auth   │ • Token Auth (Sanctum)               │
```

```
│ • CSRF          │ • CORS Headers                    │
│ • Flash Messages│ • Structured Errors               │
└─────────────────┴─────────────────────────────────────┘
```

🔄 Compartición de Lógica

Modelos Compartidos:

```
// Mismo modelo usado por Livewire y API
class Task extends Model
{
    // Lógica de negocio compartida
    public function isOverdue()
    {
        return $this->due_date && $this->due_date->isPast()
            && $this->status !== 'completed';
    }

    // Usado en Livewire para UI
    public function getPriorityColorAttribute()
    {
        return match($this->priority) {
            'high' => 'text-red-600 bg-red-100',
            'medium' => 'text-yellow-600 bg-yellow-100',
            'low' => 'text-green-600 bg-green-100',
        };
    }
}
```

Validaciones Compartidas:

```
// Trait para validaciones comunes
trait TaskValidation
{
    protected function getTaskRules()
    {
        return [
```

```
            'title' => 'required|string|max:255',
            'description' => 'required|string',
            'workshop_id' => 'required|exists:workshops,id',
            'assigned_to' => 'required|exists:users,id',
            'due_date' => 'required|date|after:now',
            'priority' => 'required|in:low,medium,high',
        ];
    }
}

// Usado en Livewire
class TaskManager extends Component
{
    use TaskValidation;

    protected $rules = self::getTaskRules();
}

// Usado en API
class TaskController extends Controller
{
    use TaskValidation;

    public function store(Request $request)
    {
        $request->validate($this->getTaskRules());
    }
}
```

📱 8. EJEMPLOS DE INTEGRACIÓN

🔧 JavaScript/React Frontend

```
// Configuración del cliente API
class ApiClient {
    constructor() {
        this.baseURL = '<http://localhost:8000/api>';
```

```
    this.token = localStorage.getItem('auth_token');
  }

  async request(endpoint, options = {}) {
    const url = `${this.baseURL}${endpoint}`;
    const config = {
      headers: {
        'Content-Type': 'application/json',
        ...(this.token && { 'Authorization': `Bearer ${this.token}` }),
        ...options.headers,
      },
      ...options,
    };

    const response = await fetch(url, config);
    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.message || 'Error en la petición');
    }

    return data;
  }

  // Métodos específicos
  async login(email, password) {
    const response = await this.request('/auth/login', {
      method: 'POST',
      body: JSON.stringify({ email, password }),
    });

    this.token = response.data.token;
    localStorage.setItem('auth_token', this.token);
    return response.data.user;
  }

  async getTasks(filters = {}) {
    const params = new URLSearchParams(filters);
```

```
        return await this.request(`/tasks?${params}`);
    }

    async completeTask(taskId, notes) {
        return await this.request(`/tasks/${taskId}/complete`, {
            method: 'POST',
            body: JSON.stringify({ completion_notes: notes }),
        });
    }
}
```

📱 React Native App

```
// services/TalleresAPI.js
import AsyncStorage from '@react-native-async-storage/async-storage';

class TalleresAPI {
    constructor() {
        this.baseURL = '<http://localhost:8000/api>';
    }

    async getToken() {
        return await AsyncStorage.getItem('auth_token');
    }

    async request(endpoint, options = {}) {
        const token = await this.getToken();
        const url = `${this.baseURL}${endpoint}`;

        const config = {
            headers: {
                'Content-Type': 'application/json',
                ...(token && { 'Authorization': `Bearer ${token}` }),
                ...options.headers,
            },
            ...options,
        };
```

```
        const response = await fetch(url, config);
        const data = await response.json();

        if (!response.ok) {
            throw new Error(data.message);
        }

        return data;
    }

    async login(email, password) {
        const response = await this.request('/auth/login', {
            method: 'POST',
            body: JSON.stringify({ email, password }),
        });

        await AsyncStorage.setItem('auth_token', response.data.token);
        await AsyncStorage.setItem('user', JSON.stringify(response.data.use
r));

        return response.data.user;
    }

    async getMyTasks() {
        return await this.request('/tasks');
    }

    async getDashboard() {
        return await this.request('/dashboard');
    }
}

export default new TalleresAPI();
```

🖥️ Componente React

```
// components/TaskList.jsx
import React, { useState, useEffect } from 'react';
```

```jsx
import TalleresAPI from '../services/TalleresAPI';

function TaskList() {
  const [tasks, setTasks] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    loadTasks();
  }, []);

  const loadTasks = async () => {
    try {
      const response = await TalleresAPI.getMyTasks();
      setTasks(response.data.tasks);
    } catch (error) {
      console.error('Error loading tasks:', error);
    } finally {
      setLoading(false);
    }
  };

  const completeTask = async (taskId) => {
    try {
      await TalleresAPI.completeTask(taskId, 'Completada desde la app');
      loadTasks(); // Recargar lista
    } catch (error) {
      console.error('Error completing task:', error);
    }
  };

  if (loading) return <div>Cargando tareas...</div>;

  return (
    <div className="task-list">
      <h2>Mis Tareas</h2>
      {tasks.map(task => (
        <div key={task.id} className="task-item">
          <h3>{task.title}</h3>
```

```
                <p>{task.description}</p>
                <span className={`priority-${task.priority}`}>
                  {task.priority}
                </span>
                {task.status !== 'completed' && (
                  <button onClick={() => completeTask(task.id)}>
                    Completar
                  </button>
                )}
              </div>
            ))}
        </div>
    );
}

export default TaskList;
```

🧪 9. TESTING

🔬 Tests de API

AuthTest.php:

```
class AuthTest extends TestCase
{
    use RefreshDatabase;

    public function test_user_can_login()
    {
        $user = User::factory()->create([
            'email' => 'test@example.com',
            'password' => bcrypt('password123'),
        ]);

        $response = $this->postJson('/api/auth/login', [
            'email' => 'test@example.com',
            'password' => 'password123',
        ]);
```

```
        $response->assertStatus(200)
                ->assertJsonStructure([
                    'success',
                    'data' => [
                        'user',
                        'token',
                        'token_type'
                    ]
                ]);
    }
}
```

WorkshopTest.php:

```
class WorkshopTest extends TestCase
{
    public function test_admin_can_create_workshop()
    {
        $admin = User::factory()->create(['role' => 'admin']);
        $token = $admin->createToken('test')->plainTextToken;

        $response = $this->withHeaders([
            'Authorization' => 'Bearer ' . $token,
        ])->postJson('/api/workshops', [
            'name' => 'Test Workshop',
            'description' => 'Test Description',
            'instructor' => 'Test Instructor',
            'capacity' => 20,
            'start_date' => '2024-03-01',
            'end_date' => '2024-05-01',
            'location' => 'Test Location'
        ]);

        $response->assertStatus(201);
        $this->assertDatabaseHas('workshops', [
            'name' => 'Test Workshop'
        ]);
```

```
    }
}
```

🚀 10. VENTAJAS DE LA ARQUITECTURA HÍBRIDA

✅ Beneficios del Sistema Dual

Para Desarrollo Web (Livewire):

- ✅ Desarrollo rápido con componentes reactivos
- ✅ Sin necesidad de API para funcionalidad básica
- ✅ SEO amigable con server-side rendering
- ✅ Autenticación tradicional con sesiones

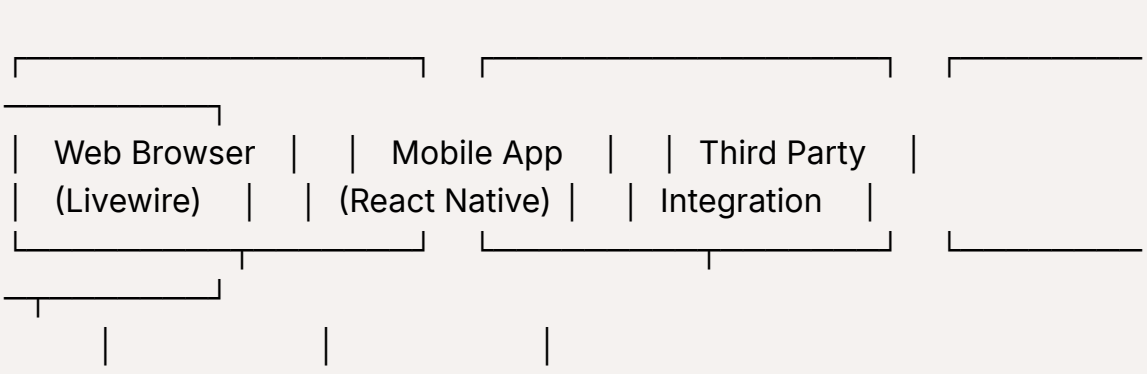
Para Aplicaciones Móviles (API REST):

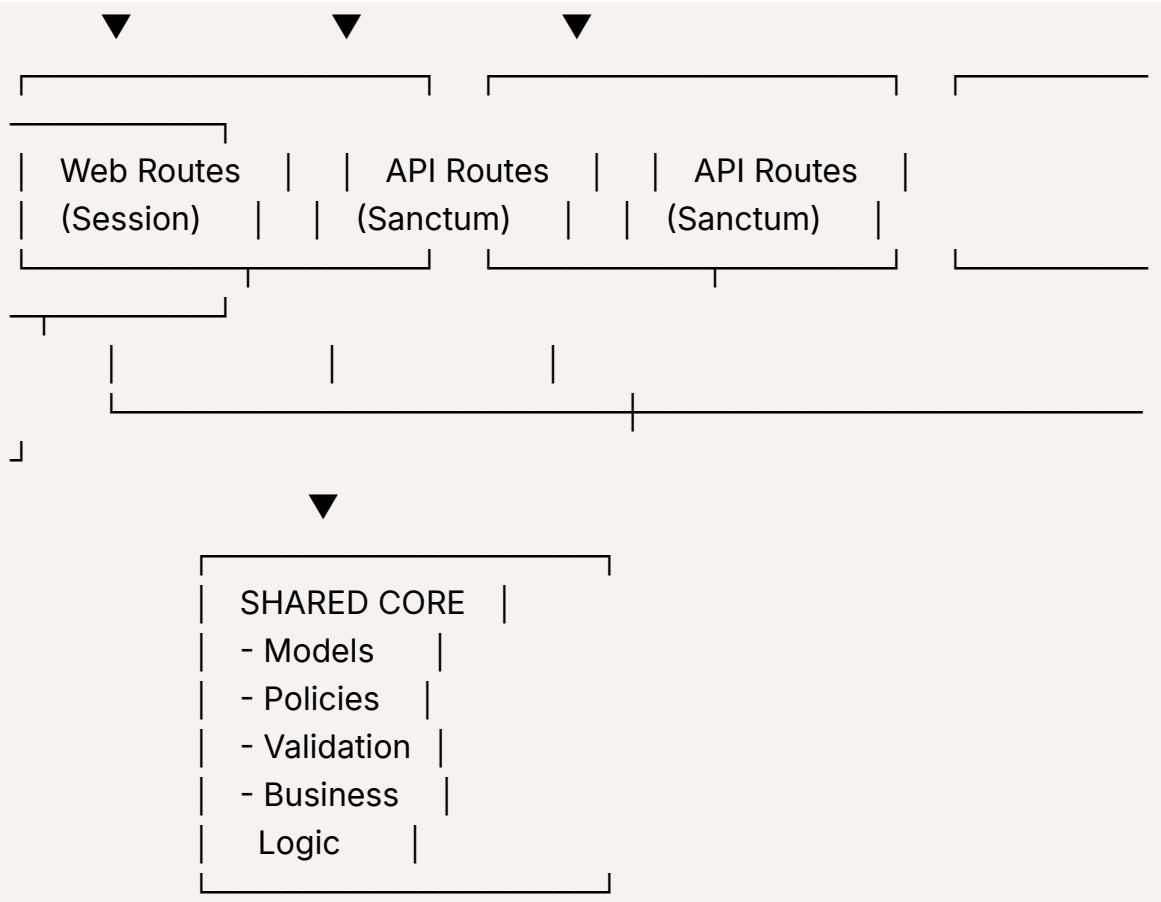
- ✅ Acceso completo desde cualquier plataforma
- ✅ Autenticación con tokens seguros
- ✅ Respuestas JSON estructuradas
- ✅ Escalabilidad para múltiples clientes

Compartido:

- ✅ Misma lógica de negocio en modelos
- ✅ Validaciones consistentes
- ✅ Base de datos unificada
- ✅ Políticas de autorización compartidas

🔄 Flujo de Datos Unificado





📈 11. ESCALABILIDAD Y MANTENIMIENTO

🔧 Optimizaciones Implementadas

Cache Compartido:

```
// Usado tanto en Livewire como API
public function getActiveWorkshops()
{
    return Cache::remember('active_workshops', 3600, function () {
        return Workshop::where('status', 'active')
                    ->with('students')
                    ->get();
    });
}
```

Eager Loading:

```
// Optimización para ambos sistemas
$tasks = Task::with(['assignedTo', 'workshop', 'assignedBy'])
        ->where('status', 'pending')
        ->get();
```

Paginación Consistente:

```
// Livewire
use WithPagination;
$tasks = Task::paginate(10);

// API
$tasks = Task::paginate($request->get('per_page', 15));
```

🚀 Futuras Mejoras

Notificaciones en Tiempo Real:

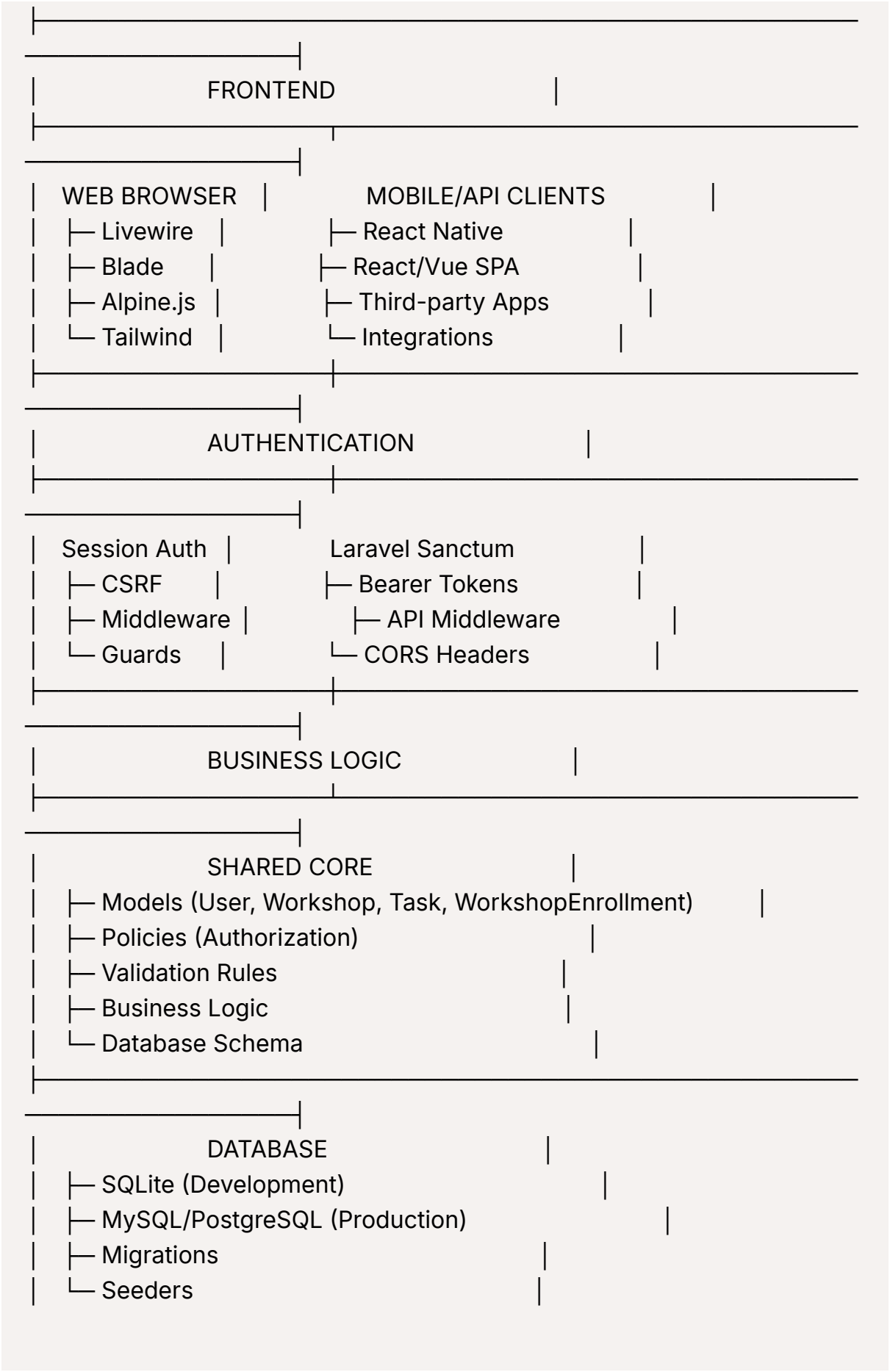
```
// Broadcasting para ambos sistemas
broadcast(new TaskAssigned($task))->toOthers();

// Livewire escucha eventos
protected $listeners = ['taskAssigned' => 'refreshTasks'];

// API puede usar WebSockets o Server-Sent Events
```

Queue Jobs:

```
// Procesamiento asíncrono compartido
dispatch(new SendTaskReminderJob($task));
dispatch(new GenerateReportJob($user));
```

🎯 Resumen de la Arquitectura Completa

```
┌─────────────────────────────────────────────────────────────────────────┐
│                        SISTEMA HÍBRIDO                                  │
```


FRONTEND
WEB BROWSER
Livewire
Blade
Alpine.js
Tailwind
MOBILE/API CLIENTS
React Native
React/Vue SPA
Third-party Apps
Integrations
AUTHENTICATION
Session Auth
CSRF
Middleware
Guards
Laravel Sanctum
Bearer Tokens
API Middleware
CORS Headers
BUSINESS LOGIC
SHARED CORE
Models (User, Workshop, Task, WorkshopEnrollment)
Policies (Authorization)
Validation Rules
Business Logic
Database Schema
DATABASE
SQLite (Development)
MySQL/PostgreSQL (Production)
Migrations
Seeders

```
└──────────────────────────────────────────────────────────────────────────────
──────────────────────────┘
```

Esta arquitectura híbrida proporciona **flexibilidad máxima** permitiendo:

1. **Desarrollo web rápido** con Livewire
2. **APIs REST completas** para móviles e integraciones
3. **Código compartido** para lógica de negocio
4. **Escalabilidad** para futuras necesidades
5. **Mantenimiento simplificado** con un solo backend

🏗️ ARQUITECTURA GENERAL

El sistema tiene **DOS INTERFACES** que comparten el mismo backend:

- **Web App** (Livewire) - Para usar en navegador
- **API REST** - Para apps móviles o integraciones

📊 MODELOS (Base de Datos)

👤 User.php

`// QUÉ HACE: Maneja usuarios (admins y estudiantes)$user = User::create(['name' => 'Juan', 'role' => 'student']);$user->isAdmin(); // true/false$user->workshops; // talleres del usuario$user->tasks; // tareas asignadas`

🏢 Workshop.php

`// QUÉ HACE: Representa talleres extracurriculares$workshop = Workshop::create(['name' => 'Programación Web', 'capacity' => 25]);$workshop->students; // estudiantes inscritos$workshop->available_spots; // cupos libres$workshop->isActive(); // si está activo`

✅ Task.php

`// QUÉ HACE: Tareas asignadas a estudiantes$task = Task::create(['title' => 'Crear página web', 'assigned_to' => $student->id]);$task->isOverdue(); // si está vencida$task->priority_color; // color para UI (rojo, amarillo, verde)`

📝 WorkshopEnrollment.php

`// QUÉ HACE: Conecta estudiantes con talleresWorkshopEnrollment::create(['user_id' => 1, 'workshop_id' => 2]);// Tabla intermedia para relación many-to-many`

⚡ COMPONENTES LIVEWIRE (Interfaz Web)

🎛️ Admin/Dashboard.php

// QUÉ HACE: Panel principal del administradorpublic function loadStats() { $this->totalStudents = User::where('role', 'student')->count(); $this->totalTasks = Task::count(); // Muestra estadísticas en tiempo real}

🏢 Admin/WorkshopManager.php

// QUÉ HACE: CRUD completo de tallerespublic function save() { Workshop::create(['name' => $this->name, 'instructor' => $this->instructor]); // Crear/editar talleres con modal}

✅ Admin/TaskManager.php

// QUÉ HACE: Asignar tareas a estudiantespublic function save() { Task::create(['title' => $this->title, 'assigned_to' => $this->assigned_to, 'due_date' => $this->due_date]); // Crear tareas y asignarlas}

👨‍🎓 Student/Dashboard.php

// QUÉ HACE: Panel personal del estudiantepublic function markTaskCompleted($taskId) { $task = Task::find($taskId); $task->update(['status' => 'completed']); // Completar tareas propias}

🚀 CONTROLADORES API REST

🔐 AuthController.php

// QUÉ HACE: Autenticación con tokens para mótilesPOST /api/auth/login → Genera tokenPOST /api/auth/register → Crea usuarioGET /api/auth/me → Info del usuarioPOST /api/auth/logout → Elimina token

🏢 WorkshopController.php

// QUÉ HACE: CRUD de talleres vía APIGET /api/workshops → Lista talleresPOST /api/workshops → Crea taller (solo admin)POST /api/workshops/1/enroll → InscribirseDELETE /api/workshops/1/enroll → Desinscribirse

✅ TaskController.php

// QUÉ HACE: Gestión de tareas vía APIGET /api/tasks → Lista tareas (filtradas por rol)POST /api/tasks → Crea tarea (solo admin)POST /api/tasks/1/complete → Completar tareaPATCH /api/tasks/1/status → Cambiar estado

📊 DashboardController.php

// QUÉ HACE: Datos de dashboard según rolGET /api/dashboard → Redirige según rolGET /api/dashboard/admin → Stats para adminGET /api/dashboard/student → Stats para estudiante

---

🛡️ SISTEMA DE SEGURIDAD

🔒 Middleware

// AdminMiddleware: Solo admins pueden pasar// StudentMiddleware: Solo estudiantes pueden pasar// ApiResponseMiddleware: Agrega headers CORS

🛡️ Policies

// WorkshopPolicy: ¿Puede crear/editar talleres?// TaskPolicy: ¿Puede ver/editar esta tarea?

🔑 Autenticación Dual

// WEB: Sesiones tradicionales// API: Tokens Sanctum (Bearer Token)

🔄 FLUJOS TÍPICOS

📝 Admin Crea Tarea

1. Admin va a /admin/tasks (Livewire)2. Hace clic "Nueva Tarea"3. Llena formulario4. TaskManager::save() ejecuta5. Task::create() guarda en BD6. Livewire actualiza la tabla automáticamente

📱 App Móvil Crea Tarea

1. App hace POST /api/tasks2. Sanctum verifica token3. TaskController::store() ejecuta4. Policy verifica permisos5. Task::create() guarda en BD6. Retorna JSON con tarea creada

✅ Estudiante Completa Tarea

WEB: Clic botón → route('student.task.complete') → actualiza BDAPI: POST /api/tasks/1/complete → verifica permisos → actualiza BD

🎨 INTERFAZ DE USUARIO

🖥️ Layout Principal (app.blade.php)

// QUÉ HACE:- Sidebar con navegación según rol- Modo oscuro con Alpine.js- Mensajes flash automáticos- Información del usuario logueado

🎨 Estilos (app.css)

/* QUÉ HACE: */- Animaciones suaves (fadeIn, slideIn)- Scrollbar personalizada- Estados hover y focus- Badges de colores para prioridades/estados- Modo oscuro completo

⚡ JavaScript (app.js)

// QUÉ HACE:- Alpine.js para interactividad- Persistencia de modo oscuro- Auto-ocultar mensajes flash- Estados de carga para Livewire

---

📱 EJEMPLOS DE USO

🌐 Frontend React

// Conecta con las APIsconst api = new ApiClient('http://localhost:8000/api');await api.login('student@test.com', 'password');const tasks = await api.getTasks();await api.completeTask(taskId, 'Completada!');

📱 App React Native

```
// Mismas APIs, diferente interfazimport TalleresAPI from './services/TalleresAPI';const user = await TalleresAPI.login(email, password);const dashboard = await TalleresAPI.getDashboard();
```

🔧 CONFIGURACIÓN

📦 Dependencias Principales

```
{  "laravel/framework": "^10.10",    // Backend  "livewire/livewire": "^3.0",      // Componentes web "laravel/sanctum": "^3.2"         // Autenticación API}
```

🗄️ Base de Datos

```
users (id, name, email, role, student_id)workshops (id, name, instructor, capacity, dates)tasks (id, title, workshop_id, assigned_to, due_date, status)workshop_enrollments (user_id, workshop_id)
```