# DOCUMENTACIÓN TÉCNICA - WEBSHIELD

## 1. DATOS GENERALES DEL PROYECTO

### 1.1 Lenguajes de Programación

**Frontend:**

* **JavaScript (ES6+):** Lenguaje principal para la lógica del cliente
* **JSX:** Extensión de JavaScript para React components
* **HTML5:** Estructura y marcado semántico
* **CSS3:** Estilos y diseño responsivo
* **Tailwind CSS:** Framework de utilidades CSS

**Backend:**

* **JavaScript (Node.js):** Lenguaje del servidor y API REST
* **SQL:** Lenguaje de consulta para base de datos PostgreSQL

**Base de Datos:**

* **PostgreSQL:** Sistema de gestión de base de datos relacional

**DevOps y Configuración:**

* **Docker:** Containerización de aplicaciones
* **Shell Script:** Scripts de automatización y despliegue

### **1.2 Uso de APIs o Librerías**

**Backend (Node.js) - Dependencias Principales:**

| ****Librería**** | ****Versión**** | ****Propósito**** |
| --- | --- | --- |
| express | 4.21.2 | Framework web para Node.js, manejo de rutas HTTP |
| bcryptjs2.4.3 | 2.4.3 | Encriptación segura de contraseñas con hash |
| jsonwebtoken | 9.0.2 | Generación y validación de tokens JWT |
| cors | 2.8.55 | Manejo de Cross-Origin Resource Sharing |
| helmet | 7.2.0 | Middleware de seguridad HTTP headers |
| express-rate-limit | 6.11.2 | Limitación de velocidad de requests |
| validator | 13.15.15 | Validación de datos de entrada |
| joi | 17.13.3t | Esquemas de validación de objetos |
| pg (Pool) | 8.16.3 | Cliente PostgreSQL para Node.js |
| dotenv | 16.6.1 | Carga de variables de entorno |

**Frontend (React) - Dependencias Principales:**

| ****Librería**** | ****Versión**** | ****Propósito**** |
| --- | --- | --- |
| react | 19.1 | Biblioteca principal para UI components |
| react-dom | 19.1 | Rendering de React en el DOM |
| lucide-react | 0.5 | Iconografía moderna y consistente |
| tailwindcss | 4.1 | Framework CSS utility-first |

**API Propia:**

* **WebShield REST API:** API personalizada desarrollada en Express.js
* **Endpoints principales:**
  + /api/auth/\* - Autenticación y autorización
  + /api/projects/\* - CRUD de proyectos web
  + /api/elements/\* - Gestión de elementos de página
  + /api/payments/\* - Procesamiento de pagos
  + /api/stats/\* - Estadísticas y métricas

### **1.3 Modelo de Desarrollo**

**Metodología Ágil - Scrum Adaptado:**

* **Sprints:** Iteraciones de 2 semanas
* **Daily Standups:** Reuniones diarias de 15 minutos
* **Sprint Planning:** Planificación al inicio de cada sprint
* **Sprint Review:** Demostración de funcionalidades completadas
* **Retrospective:** Análisis de mejoras para siguiente sprint

**Flujo de Desarrollo:**

1. Feature Request → 2. Design → 3. Development → 4. Testing → 5. Code Review → 6. Deployment

**Control de Versiones:**

* **Git:** Sistema de control de versiones distribuido
* **GitHub:** Plataforma de alojamiento y colaboración
* **Branching Strategy:** Git Flow con ramas feature, develop y main

**Arquitectura de Software:**

* **Patrón MVC:** Model-View-Controller en backend
* **Component-Based:** Arquitectura de componentes en React
* **RESTful API:** Arquitectura REST para comunicación cliente-servidor
* **Microservicios Ligeros:** Separación de responsabilidades por módulos

### **1.4 Herramientas de Desarrollo**

**Entorno de Desarrollo Integrado (IDE):**

* **Visual Studio Code:** Editor principal con extensiones para React y Node.js
* **Extensiones:**
  + ES7+ React/Redux/React-Native snippets
  + Prettier - Code formatter
  + ESLint
  + Thunder Client (API testing)

**Gestión de Dependencias:**

* **npm:** Package manager para Node.js
* **package.json:** Configuración de dependencias y scripts

**Containerización y Despliegue:**

* **Docker:** Containers para desarrollo y producción
* **docker-compose.yml:** Orquestación multi-container
* **Render.com:** Plataforma de despliegue cloud

**Testing y Calidad:**

* **Jest:** Framework de testing para JavaScript
* **React Testing Library:** Testing de componentes React
* **Postman:** Testing de API endpoints

**Scripts de Automatización:**

* run-webshield.sh - Script principal de ejecución
* test-backend.sh - Script de testing del backend

## **2. ESTRUCTURA DEL SISTEMA**

### **2.1 Mock-ups y Diseño de Interfaz**

**Pantallas Principales del Sistema:**

### **2.2 Definición de la Base de Datos**

**Esquema de Base de Datos PostgreSQL:**

**-- Tabla de usuarios**

**CREATE TABLE IF NOT EXISTS usuarios (**

**id SERIAL PRIMARY KEY,**

**email VARCHAR(255) UNIQUE NOT NULL,**

**password VARCHAR(255) NOT NULL,**

**plan VARCHAR(20) DEFAULT 'free' CHECK (plan IN ('free', 'pro', 'premium')),**

**created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP**

**);**

**-- Tabla de proyectos**

**CREATE TABLE IF NOT EXISTS proyectos(**

**id SERIAL PRIMARY KEY,**

**name VARCHAR(255) NOT NULL,**

**user\_id INTEGER NOT NULL,**

**created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**FOREIGN KEY (user\_id) REFERENCES usuarios(id) ON DELETE CASCADE**

**);**

**-- Tabla de elementos**

**CREATE TABLE IF NOT EXISTS elementos (**

**id SERIAL PRIMARY KEY,**

**project\_id INTEGER NOT NULL,**

**type VARCHAR(50) NOT NULL CHECK (type IN ('text', 'image', 'button', 'form', 'gallery', 'menu')),**

**settings JSONB DEFAULT '{}',**

**order\_position INTEGER NOT NULL DEFAULT 0,**

**created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**FOREIGN KEY (project\_id) REFERENCES proyectos(id) ON DELETE CASCADE**

**);**

**-- Tabla de pagos**

**CREATE TABLE IF NOT EXISTS pagos (**

**id SERIAL PRIMARY KEY,**

**user\_id INTEGER NOT NULL,**

**plan VARCHAR(20) NOT NULL CHECK (plan IN ('free', 'pro', 'premium')),**

**amount DECIMAL(10,2) NOT NULL DEFAULT 0.00,**

**status VARCHAR(20) DEFAULT 'pending' CHECK (status IN ('pending', 'completed', 'failed', 'cancelled')),**

**stripe\_payment\_id VARCHAR(255),**

**created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**FOREIGN KEY (user\_id) REFERENCES usuarios(id) ON DELETE CASCADE**

**);**

**-- Índices para optimización**

**CREATE INDEX IF NOT EXISTS idx\_usuarios\_email ON usuarios(email);**

**CREATE INDEX IF NOT EXISTS idx\_proyectos\_user\_id ON proyectos(user\_id);**

**CREATE INDEX IF NOT EXISTS idx\_elementos\_project\_id ON elementos(project\_id);**

**CREATE INDEX IF NOT EXISTS idx\_elementos\_order ON elementos(project\_id, order\_position);**

**CREATE INDEX IF NOT EXISTS idx\_pagos\_user\_id ON pagos(user\_id);**

**CREATE INDEX IF NOT EXISTS idx\_pagos\_status ON pagos(status);**

**-- Función para actualización automática de timestamps**

**CREATE OR REPLACE FUNCTION update\_updated\_at\_column()**

**RETURNS TRIGGER AS $**

**BEGIN**

**NEW.updated\_at = CURRENT\_TIMESTAMP;**

**RETURN NEW;**

**END;**

**$ language 'plpgsql';**

**-- Triggers para updated\_at automático**

**CREATE TRIGGER IF NOT EXISTS update\_usuarios\_updated\_at**

**BEFORE UPDATE ON usuarios FOR EACH ROW**

**EXECUTE FUNCTION update\_updated\_at\_column();**

**CREATE TRIGGER IF NOT EXISTS update\_proyectos\_updated\_at**

**BEFORE UPDATE ON proyectos FOR EACH ROW**

**EXECUTE FUNCTION update\_updated\_at\_column();**

**CREATE TRIGGER IF NOT EXISTS update\_elementos\_updated\_at**

**BEFORE UPDATE ON elementos FOR EACH ROW**

**EXECUTE FUNCTION update\_updated\_at\_column();**

**CREATE TRIGGER IF NOT EXISTS update\_pagos\_updated\_at**

**BEFORE UPDATE ON pagos FOR EACH ROW**

**EXECUTE FUNCTION update\_updated\_at\_column();**

**TIPO DE RELACIONES DE BASE DE DATOS:**

**Modelo Relacional Normalizado - Tercera Forma Normal (3NF)**

**Relaciones Entre Tablas:**

1. **usuarios → proyectos (Relación 1:N)**
   * **Tipo:** One-to-Many (Uno a Muchos)
   * **Cardinalidad:** 1 usuario puede tener N proyectos
   * **Integridad:** ON DELETE CASCADE (eliminar usuario elimina sus proyectos)
   * **Clave Foránea:** proyectos.user\_id → usuarios.id
2. **proyectos → elementos (Relación 1:N)**
   * **Tipo:** One-to-Many (Uno a Muchos)
   * **Cardinalidad:** 1 proyecto puede tener N elementos
   * **Integridad:** ON DELETE CASCADE (eliminar proyecto elimina sus elementos)
   * **Clave Foránea:** elementos.project\_id → proyectos.id
   * **Ordenamiento:** Incluye order\_position para secuencia de elementos
3. **usuarios → pagos (Relación 1:N)**
   * **Tipo:** One-to-Many (Uno a Muchos)
   * **Cardinalidad:** 1 usuario puede tener N pagos (historial de transacciones)
   * **Integridad:** ON DELETE CASCADE (eliminar usuario elimina su historial de pagos)
   * **Clave Foránea:** pagos.user\_id → usuarios.id

### **2.3 Flujo de Funcionamiento del Sistema**

**Estructura de Directorios:**

webshield-fullstack/

├── backend/

│ ├── src/

│ │ ├── controllers/ -- Lógica de negocio

│ │ ├── middleware/ --comunicacion

│ │ ├── routes/ -- Definición de rutas API

│ │ └── utils/ --Utilidades y helpers

│ ├── server.js --Punto de entrada del servidor

│ └── package.json --Dependencias del backend

├── frontend/

│ ├── src/

│ │ ├── api/ --Configuración de API (config.js)

│ │ ├── App.js --Componente principal React

│ │ └── index.js --Punto de entrada React

│ ├── public/ --Archivos estáticos

│ └── package.json --Dependencias del frontend

├── database/

│ └── schema.sql --Esquema de base de datos

├── scripts/

│ ├── run-webshield.sh --Script de ejecución

│ └── test-backend.sh -- Script de testing

├── docker-compose.yml --Orquestación containers

├── Dockerfile.backend --Container del backend

├── Dockerfile.frontend --Container del frontend

└── README.md --Documentación del proyecto

**Flujo de Datos del Sistema:**

1. Usuario accede → Frontend (React)

2. Frontend → API Request → Backend (Express.js)

3. Backend → Validación → Middleware de seguridad

4. Backend → Base de datos → PostgreSQL

5. PostgreSQL → Respuesta → Backend

6. Backend → JSON Response → Frontend

7. Frontend → Actualización UI → Usuario

**Ciclo de Vida de una Request:**

sequenceDiagram

participant U as Usuario

participant F as Frontend

participant B as Backend

participant DB as PostgreSQL

U->>F: Acción (crear proyecto)

F->>F: Validación local

F->>B: POST /api/projects

B->>B: Autenticación JWT

B->>B: Validación con Joi

B->>DB: INSERT INTO projects

DB->>B: Confirmation

B->>F: Response JSON

F->>F: Actualizar estado

F->>U: UI actualizada

### **2.4 Estructura de Seguridad**

**Autenticación y Autorización:**

**1. JWT (JSON Web Tokens):**

// Generación de token en backend

const token = jwt.sign(

{ userId: user.id, email: user.email, plan: user.plan },

process.env.JWT\_SECRET,

{ expiresIn: '24h' }

);

// Middleware de verificación

const verifyToken = (req, res, next) => {

const token = req.headers['authorization']?.split(' ')[1];

if (!token) return res.status(401).json({ error: 'Access denied' });

try {

const verified = jwt.verify(token, process.env.JWT\_SECRET);

req.user = verified;

next();

} catch (error) {

res.status(400).json({ error: 'Invalid token' });

}

};

**2. Encriptación de Contraseñas:**

// Hash de contraseña con bcrypt

const saltRounds = 12;

const hashedPassword = await bcrypt.hash(password, saltRounds);

// Verificación de contraseña

const isValidPassword = await bcrypt.compare(password, user.password\_hash);

**3. Middleware de Seguridad:**

// Helmet para headers de seguridad HTTP

app.use(helmet({

contentSecurityPolicy: false,

crossOriginEmbedderPolicy: false

}));

// Rate limiting

const limiter = rateLimit({

windowMs: 15 \* 60 \* 1000, // 15 minutos

max: 100 // límite de 100 requests por IP

});

app.use('/api/', limiter);

// CORS configurado

app.use(cors({

origin: process.env.FRONTEND\_URL,

credentials: true

}));

**4. Validación de Datos:**

// Esquemas de validación con Joi

const userSchema = Joi.object({

email: Joi.string().email().required(),

password: Joi.string().min(8).required(),

full\_name: Joi.string().min(2).max(100).required()

});

// Validación en endpoints

const { error, value } = userSchema.validate(req.body);

if (error) {

return res.status(400).json({ error: error.details[0].message });

}

**5. Seguridad de Base de Datos:**

// Conexión segura PostgreSQL

const pool = new Pool({

connectionString: process.env.DATABASE\_URL,

ssl: process.env.NODE\_ENV === 'production' ? { rejectUnauthorized: false } : false

});

// Consultas preparadas para prevenir SQL injection

const result = await pool.query(

'SELECT \* FROM users WHERE email = $1',

[email]

);

**6. Variables de Entorno (.env):**

JWT\_SECRET= no mostable.

DATABASE\_URL= no mostable.

FRONTEND\_URL=[https://webshield100.onrender.com](https://webshield100.onrender.com/)

BACKEND\_URL=https://webshield100-backend.onrender.com

NODE\_ENV=production

**Medidas de Seguridad Implementadas:**

* Autenticación JWT con expiración
* Contraseñas hasheadas con bcrypt (salt rounds: 12)
* Rate limiting para prevenir ataques de fuerza bruta
* Headers de seguridad con Helmet
* CORS configurado correctamente
* Validación de datos con Joi y validator
* Consultas SQL preparadas anti-injection
* Conexión SSL a base de datos en producción
* Variables de entorno para secretos
* Middleware de autorización por rutas
* Sanitización de inputs del usuario