Instituto Politécnico Nacional

Escuela Superior de Cómputo

Web Client and Backend Development Frameworks.

Sistema de Asignación de Casilleros

Desarrollo de Back-End, Documentación Swagger y Despliegue en Supabase

Profesor: M. en C. José Asunción Enríquez Zárate

Alumnos:

- *Cabrera García Daniel*
- *López Jiménez Angello Michael*
- *Martínez Pedraza Roberto Yahir*

mchllopezjimenez@gmail.com

7CM1

Junio 2025

Índice

1. Introducción

El presente documento describe la arquitectura y la implementación del **Sistema de Asignación de Casilleros** para la ESCOM. El objetivo es automatizar el proceso de registro de alumnos, validación de documentos, pagos y asignación de casilleros, garantizando trazabilidad, equidad y disminuyendo la carga administrativa.

La solución se compone de:

- Un back-end **Node.js + Express** con API REST.
- Base de datos PostgreSQL desplegada gratuitamente en **Supabase**.
- Documentación interactiva con **Swagger UI (OpenAPI 3.0)** en `/api-docs`.
- Front-end estático (HTML/CSS/JS) que consume las API vía `fetch`.

2. Modelo de Datos

Partiendo del análisis funcional se identificaron las entidades *Usuario*, *Solicitud*, *Documento*, *Pago*, *Casillero* y *AsignaciónCasillero*. El diagrama ER se describe a continuación:

```
[Diagrama conceptual]
Usuario (1) --- (N) Solicitud (1) --- (1) Documento
Usuario (1) --- (N) Pago
Casillero (1) --- (1) AsignaciónCasillero (N) --- (1) Usuario
```

Cada tabla se crea en Supabase con el siguiente *snippet* (*extracto*; ver script completo en el anexo):

Listing 1: Tabla `Usuario`

```sql
CREATE TABLE Usuario (
  id_usuario    SERIAL PRIMARY KEY,
  correo        VARCHAR(255) UNIQUE NOT NULL,
  contrasena    VARCHAR(255) NOT NULL,
  rol           VARCHAR(20)  NOT NULL CHECK (rol IN ('coordinador','alumno'))
);
```

Todas las claves foráneas se definen con `ON DELETE CASCADE` para mantener integridad referencial.

3. Back-End con Express

3.1. Estructura de carpetas

```
backend/
  docs/            # swagger.json
  routes/
     usuarios.js
     solicitudes.js
     documentos.js
     pagos.js
     casilleros.js
     asignaciones.js
  login.js         # endpoint de autenticación
  db.js            # pool de PostgreSQL (Supabase)
  index.js         # servidor principal
```

3.2. Conexión a Supabase (`db.js`)

```js
const { Pool } = require('pg');
require('dotenv').config();

module.exports = new Pool({
  connectionString: process.env.DATABASE_URL,
  ssl: { rejectUnauthorized: false }   // Requerido por Supabase
});
```

3.3. Servidor principal (index.js)

```
require('dotenv').config();
const express    = require('express');
const cors       = require('cors');
const swaggerUi  = require('swagger-ui-express');
const swaggerDoc = require('./docs/swagger.json');

const app = express();
app.use(cors());
app.use(express.json());

// Rutas automatizadas
app.use('/usuarios',     require('./routes/usuarios'));
app.use('/solicitudes',  require('./routes/solicitudes'));
app.use('/documentos',   require('./routes/documentos'));
app.use('/pagos',        require('./routes/pagos'));
app.use('/casilleros',   require('./routes/casilleros'));
app.use('/asignaciones', require('./routes/asignaciones'));
app.use('/login',        require('./routes/login'));

// Swagger
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDoc));

app.listen(3000, () => console.log('Servidor en http://localhost:3000'));
```

3.4. Ejemplo de ruta: creación de usuario

Listing 2: routes/usuarios.js (POST)

```
router.post('/', async (req, res) => {
  const { correo, contrasena, rol } = req.body;
  if (!correo || !contrasena) return res.status(400).json({ error:'Faltan campos' });

  // Se recomienda almacenar el hash (ejemplo con bcrypt):
  const hash = await bcrypt.hash(contrasena, 10);
  const query = 'INSERT INTO Usuario (correo, contrasena, rol) VALUES ($1,$2,$3) RETURNING *';
  const { rows } = await pool.query(query, [correo, hash, rol || 'alumno']);
  res.status(201).json(rows[0]);
});
```

4. Documentación Swagger (OpenAPI 3.0)

Se generó el fichero `docs/swagger.json`; un fragmento se muestra a continuación:

Listing 3: Extracto de `swagger.json`

```
{
  "openapi":"3.0.0",
  "info":{"title":"API Casilleros ESCOM","version":"1.0.0"},
  "paths":{
    "/usuarios":{
      "post":{
        "summary":"Crear usuario",
        "requestBody":{
          "required":true,
          "content":{
            "application/json":{
              "schema":{
                "$ref":"#/components/schemas/UsuarioIn"
              }
            }
          }
        },
        "responses":{
          "201":{"description":"Creado"}
        }
      }
    }
  },
  "components":{
    "schemas":{
      "UsuarioIn":{
        "type":"object",
        "properties":{
          "correo":{"type":"string","format":"email"},
          "contrasena":{"type":"string","format":"password"},
          "rol":{"type":"string","enum":["alumno","coordinador"]}
        },
        "required":["correo","contrasena"]
      }
    }
  }
}
```

Al ejecutar el servidor, la UI queda disponible en `http://localhost:3000/api-docs`, mostrando una interfaz similar a:

```
[Interfaz Swagger UI]
+---------------------------+
| API Casilleros ESCOM v1.0 |
+---------------------------+
| POST /usuarios            |
| GET /usuarios/{id}        |
| POST /login               |
| ...                       |
+---------------------------+
```

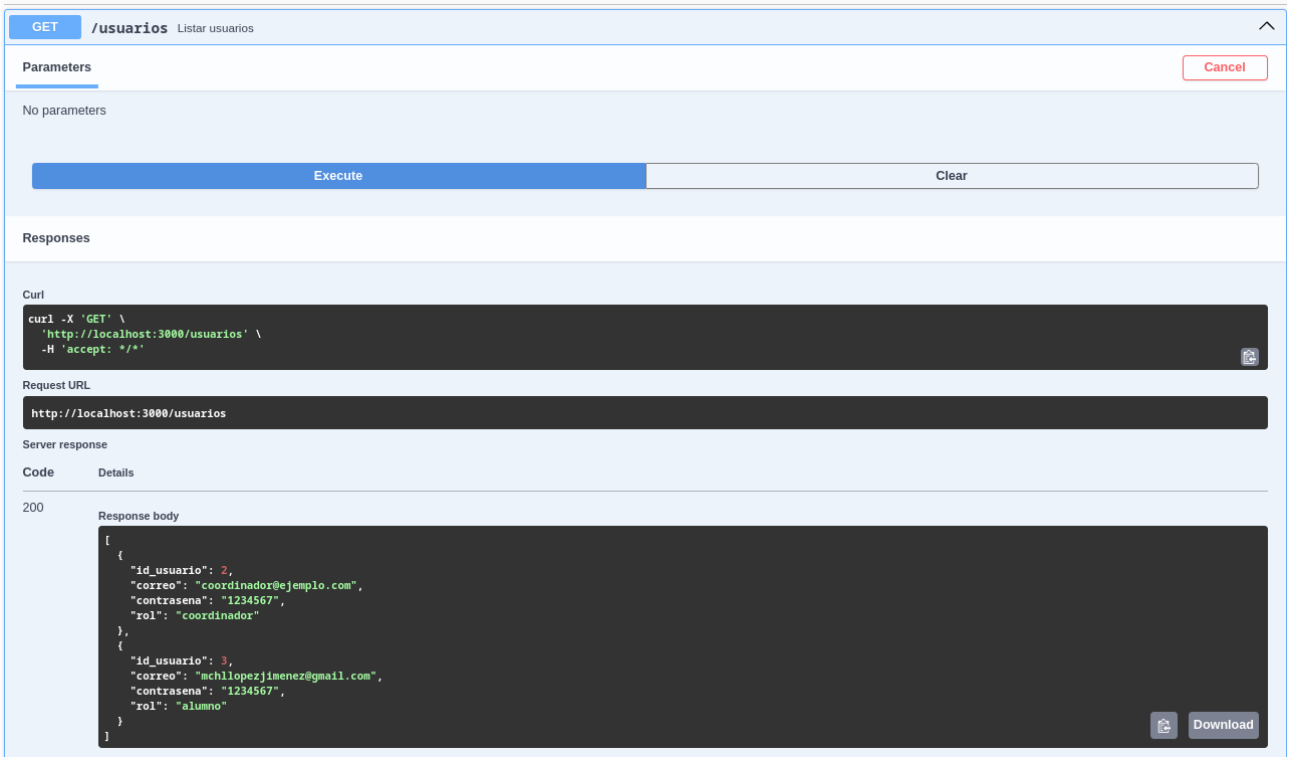

Figura 1: Documentado utilizando Swagger UI.

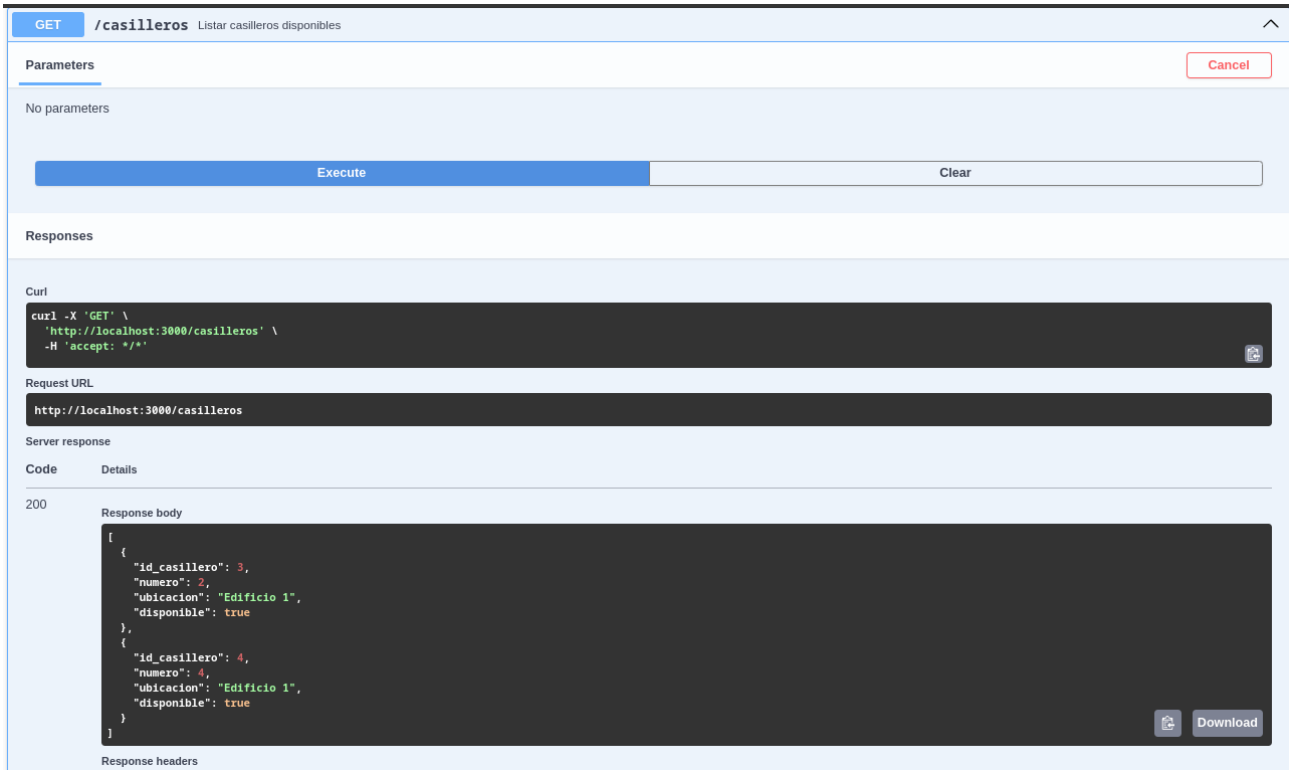

Figura 2: Listar Casilleros.

5. Front-End y Conexión con la API

5.1. Archivo conexion.js

Para evitar duplicación de código, se implementó un script unificado que maneja tanto el registro como el login:

Listing 4: Script unificado de conexion (conexion.js)

```
const API = 'http://localhost:3000';

/* ---------- Registro ---------- */
const registroForm = document.getElementById('register-form');
if (registroForm) {
  registroForm.addEventListener('submit', async (e) => {
    e.preventDefault();
    const datosRegistro = {
      correo: registroForm['register-correo'].value,
      contrasena: registroForm['register-contrasena'].value,
      rol: 'alumno' // Rol por defecto
    };

    try {
      const respuesta = await fetch(`${API}/usuarios`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(datosRegistro)
      });

      if (respuesta.ok) {
        localStorage.setItem('correo', datosRegistro.correo);
        window.location.href = 'dashboard.html';
      } else {
        alert('Error en el registro: ' + await respuesta.text());
      }
    } catch (error) {
      console.error('Error:', error);
      alert('Error de conexi n');
    }
  });
}

/* ---------- Login ---------- */
const loginForm = document.getElementById('login-form');
if (loginForm) {
  loginForm.addEventListener('submit', async (e) => {
    e.preventDefault();
    const credenciales = {
      correo: loginForm['login-correo'].value,
      contrasena: loginForm['login-contrasena'].value
    };

    try {
      const respuesta = await fetch(`${API}/login`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(credenciales)
      });

      if (respuesta.ok) {
        localStorage.setItem('correo', credenciales.correo);
        window.location.href = 'dashboard.html';
      } else {
        alert('Credenciales incorrectas');
      }
    } catch (error) {
      console.error('Error:', error);
```

```
59        alert('Error de conexi n');
60      }
61    });
62  }
```

5.2. Pruebas End-to-End

El flujo completo de pruebas incluye:

Paso	Acción y Validación
1	Iniciar back-end: `node backend/index.js`
	Verificar: Mensaje "Servidor en http://localhost:3000.[en] consola
2	Abrir `index.html` en navegador usando Live Server
	Verificar: Formularios de registro/login se muestran correctamente
3	Registrar nuevo usuario (correo válido + contraseña)
	Verificar: 1) Redirección a dashboard.html 2) Nuevo registro en tabla Usuario de Supabase
4	Cerrar sesión (eliminar `localStorage`) y realizar login
	Verificar: 1) Acceso con credenciales 2) Bloqueo con credenciales incorrectas
5	En `dashboard.html`: Completar formulario de solicitud
	Verificar: 1) Nueva entrada en tabla Solicitud 2) Actualización en tiempo real

Cuadro 1: Protocolo de pruebas integrales

Nota técnica: Para pruebas automatizadas, se recomienda:

- Usar `jest` para tests unitarios del backend
- Configurar `Postman` para validar los endpoints API

6. Conclusiones

Se logró implementar un back-end robusto en Express, documentado con OpenAPI y persistiendo en PostgreSQL sobre Supabase. Swagger facilitó las pruebas y la verificación de requisitos. El front-end, aunque sencillo, demuestra el flujo completo desde el registro hasta el envío de la solicitud, sentando las bases para futuras ampliaciones (JWT, roles avanzados, panel de coordinador, etc.).

A. Script completo de creación de tablas (SQL)

Listing 5: Script SQL completo

```sql
CREATE TABLE Usuario (
    id_usuario SERIAL PRIMARY KEY,
    correo VARCHAR(255) UNIQUE NOT NULL,
    contrasena VARCHAR(255) NOT NULL,
    rol VARCHAR(20) CHECK (rol IN ('coordinador', 'alumno')) NOT NULL
);

CREATE TABLE Solicitud (
    id_solicitud SERIAL PRIMARY KEY,
    id_usuario INT NOT NULL,
    numero_boleta VARCHAR(20) UNIQUE NOT NULL,
    nombre_completo VARCHAR(255) NOT NULL,
    semestre_actual INT NOT NULL,
    correo_personal VARCHAR(255) NOT NULL,
    numero_celular VARCHAR(20) NOT NULL,
    estado VARCHAR(20) CHECK (estado IN ('pendiente', 'aprobada', 'rechazada')) DEFAULT 'pendien
    motivo_rechazo TEXT,
    FOREIGN KEY (id_usuario) REFERENCES Usuario(id_usuario) ON DELETE CASCADE
);

CREATE TABLE Documento (
    id_documento SERIAL PRIMARY KEY,
    id_solicitud INT NOT NULL,
    tipo VARCHAR(30) CHECK (tipo IN ('comprobante horario', 'credencial vigente')) NOT NULL,
    ruta_archivo TEXT NOT NULL,
    FOREIGN KEY (id_solicitud) REFERENCES Solicitud(id_solicitud) ON DELETE CASCADE
);

CREATE TABLE Pago (
    id_pago SERIAL PRIMARY KEY,
    id_solicitud INT NOT NULL,
    validado_por_coordinador BOOLEAN DEFAULT FALSE,
    estado_pago VARCHAR(20) CHECK (estado_pago IN ('no pagado', 'pagado')) DEFAULT 'no pagado',
    FOREIGN KEY (id_solicitud) REFERENCES Solicitud(id_solicitud) ON DELETE CASCADE
);

CREATE TABLE Casillero (
    id_casillero SERIAL PRIMARY KEY,
    numero INT UNIQUE NOT NULL,
    ubicacion VARCHAR(255) NOT NULL,
    disponible BOOLEAN DEFAULT TRUE
);

CREATE TABLE AsignacionCasillero (
    id_asignacion SERIAL PRIMARY KEY,
    id_pago INT NOT NULL,
    id_casillero INT NOT NULL,
    fecha_asignacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_pago) REFERENCES Pago(id_pago) ON DELETE CASCADE,
    FOREIGN KEY (id_casillero) REFERENCES Casillero(id_casillero) ON DELETE CASCADE
);
```