

Documento de Entrega para Prueba Técnica

Candidato: Andrés Santiago Ulloa Martínez

a. Tiempo Planeado de Solución

El tiempo planeado total de la solución fue de **7.0 horas**, subdividida en las siguientes actividades:

Tarea	Tiempo Estimado (h)
I. Diseño de Arquitectura y Base de Datos: Diseño del esquema y definición de la arquitectura.	0.5
II. Backend PHP (API): Configuración de Docker, conexión a DB, desarrollo de validación, transacción, inserción y desarrollo de consultas y agrupación.	2.5
III. Frontend NextJs: Desarrollo del formulario de carga, lógica de POST/GET, redirección y maquetación de la página de resultados con componentes de tabla.	3.0
IV. Pruebas, Ajustes y Documentación: Pruebas funcionales y preparación de los entregables	1.0
Total Planeado	7.0 horas

b. Explicación de la solución

a. Arquitectura y tecnologías

La solución implementa una arquitectura Cliente-Servidor, cumpliendo con los requisitos no funcionales:

- *Backend:* PHP y MySQL, utilizando PDO para una conexión segura y transaccional.
- *Frontend:* NextJs con Material UI (MUI) para la interfaz.
- *Despliegue:* Se utiliza Docker para contener el entorno PHP y MySQL, facilitando instalación y portabilidad.

b. Flujo de Carga y Validación

- *Recepción y Validación de Formato:* El script **upload.php** recibe el archivo mediante POST y lo procesa línea por línea. Si una línea no contiene exactamente 4 valores separados por coma, si el email es inválido, o si el código no es 1, 2 o 3, el proceso se detiene.
- *Manejo de Errores:* Si se detectan errores de formato, el script devuelve un código HTTP 400 y el mensaje requerido: "El formato interno del archivo no es válido. Por favor, revisa la documentación". El frontend de NextJs lo captura y lo muestra al usuario.
- *Transacciones:* El script utiliza Transacciones PDO para asegurar que todo el lote de inserciones se complete con éxito. Si una inserción de base de datos falla, toda la operación es revertida.

- *Manejo de Duplicados*: La tabla users tiene una clave única compuesta ux_email_codigo (email + codigo). Para garantizar que los datos válidos se procesen sin detener la carga por duplicados ya existentes, se utiliza la sentencia INSERT IGNORE. Esto ignora silenciosamente las filas que violan la restricción de unicidad, asegurando la integridad sin frustrar la experiencia de usuario con un rechazo total.
- *Redirección*: Tras el éxito, el frontend redirecciona a la vista de resultados

c. *Flujo de Visualización*

- *Consulta y Agrupación*: La página /results realiza una llamada GET a **list.php**. El script ejecuta una consulta y agrupa los resultados en tres arreglos (activos, inactivos, espera) en el servidor, utilizando el campo código.
- *Renderizado en el Frontend*: NextJs recibe el JSON con los datos preagrupados y renderiza tres tablas separadas mediante una tabla modular para los estados 1, 2 y 3.

c. *Manual de Instalación*

Este manual asume el uso de **Docker Compose** para configurar el entorno completo de PHP y MySQL, simplificando la instalación.

Requisitos Previos

- **Docker y Docker Compose** (para el backend).
- **Node.js y npm** (o Yarn) (para el frontend NextJs).

Pasos Detallados de Ejecución

1. Clonar Repositorio

git clone <https://github.com/CaptainLevi20/gema-sas-app.git>

2. Configuración de Variables de Entorno:

Crea un archivo .env en la raíz del proyecto para definir las credenciales de la base de datos (utilizadas por config.php):

```
# .env
DB_HOST=db
DB_NAME=gema_sas
DB_USER=user_gema
DB_PASSWORD=secret_password
NEXT_PUBLIC_API_URL=http://localhost:8080
```

3. Configuración del Backend (PHP y MySQL):

Levanta los servicios de Docker:

```
docker-compose up -d
```

4. Ejecución del Frontend (NextJs):

Instala las dependencias y ejecuta el servidor de desarrollo:

```
cd frontend-nextjs
```

npm install
npm run dev

5. Acceso a la Aplicación

La aplicación estará disponible en: <http://localhost:3000>

El API de PHP estará disponible en: <http://localhost:8080>