Plan de pruebas de software

Sistema de gestión de gastos

Integrantes:

Ricardo Coronado Mera
Sebastián Lara Barría
Diego Sandoval Burgos

Fecha:

15 de Enero, 2021.

Objetivos

El presente documento describe el plan de pruebas de la aplicación web *Sistema de Gestión de Gastos para una Empresa Constructora*.

El plan de pruebas pretende diagnosticar problemas en la ejecución de la aplicación para solucionarlos oportunamente.

Arquitectura

La aplicación está diseñada para ejecutarse sobre el software de integración contínua Jenkins y consta de 3 partes: el frontend, el backend y la base de datos. Cada una de estas partes se ejecutará sobre su respectivo contenedor Docker. Estos contenedores se comunicarán entre sí usando la funcionalidad *network* de Docker..

1. El frontend es una aplicación desarrollada en el framework React.
2. El backend es una API REST desarrollada en Node.js con Express.js
3. La base de datos es una instancia de PostgreSQL.

El backend depende de la API y el frontend depende del backend, por lo tanto las pruebas se realizarán primero sobre la base de datos, luego sobre el backend y finalmente en el frontend.

Pruebas

Base de datos

1. Test de operación:
   Objetivo: verificar que la base de datos está operativa
   Procedimiento: Ejecutar el comando psql dentro del contenedor Docker y listar
   Automatización: Añadir una sección al pipeline de Jenkins que realice la tarea:

   ```
   $ docker exec sggastos_db pg_ctl status
   ```

   Resultado exitoso: El comando informa que la base de datos está operativa.

2. Test de conexión

   Objetivo: verificar que la base de datos acepta conexiones por los medios que el backend usará para comunicarse con ella.
   Procedimiento: Desde el contenedor del backend, conectarse a la base de datos
   Automatización: Añadir un paso al pipeline y conectarse usando psql
   Resultado exitoso: postgres nos da una shell para ejecutar sentencias SQL

3. Test modelo relacional

   Objetivo: verificar que la base de datos creó correctamente las tablas del modelo relacional
   Procedimiento: usar comandos postgres \d y SELECT
   Automatización: ejecutar los comandos en el contenedor docker de la base de datos.

```
$ docker exec sggastos_db echo "\c sgg" > test.sql
$ docker exec sggastos_db echo "\d obras" > test.sql
$ docker exec sggastos_db echo "\d gastos" > test.sql
$ docker exec sggastos_db echo "select (id, nombre, \
ubicacion) from obras;" > test.sql
$ docker exec sggastos_db echo "select (id, nombre, \
valor, proveedor) from gastos;" > test.sql
$ docker exec sggastos_db psql -U postgres -a -f
test.sql
```

   Resultado exitoso: Postgres no retorna errores.

Backend

1. Test operación servidor web

   Objetivo: verificar que el servidor web del backend de datos está operativo.
   Procedimiento: Enviar una solicitud GET
   Automatización: Usar una herramienta como Postman o alguna alternativa más liviana para enviar solicitud GET a la URL raiz de la API.
   Resultado exitoso: El servidor envía un mensaje HTTP 200 OK.

2. Test API

   Objetivo: verificar que la API está funcionando y retornando la informacion de la base de datos

Procedimiento: Enviar una solicitud GET a una ruta definida en la api, por ejemplo: `/api/obra/1/gasto`
Automatización: Usar una herramienta como Postman o alguna alternativa más liviana para enviar solicitud GET a una ruta definida de la API.
Resultado exitoso: El servidor envía una respuesta HTTP 200 OK y la información requerida en el entity body.

3. Test API (escritura)
   Objetivo: verificar que la API guarda la información cuando se hace una solicitud POST a una ruta definida.
   Procedimiento: Enviar una solicitud GET a una ruta definida en la api, por ejemplo: POST `/api/obra/1/gasto`, con un objeto apropiado en el entity-body de la solicitud.
   Automatización: Usar una herramienta como Postman o alguna alternativa más liviana para enviar solicitud GET a una ruta definida de la API.
   Resultado exitoso: El servidor envía un mensaje HTTP 200 OK y la información de escritura exitosa en el entity body.

Frontend

1. Despliegue de información
   Objetivo: Verificar que la aplicación frontend despliega apropiadamente la informacion obtenida de la API servidor web del backend de datos está operativo.
   Procedimiento: Visitar la aplicación en /Dashboard y /Obra/1
   Resultado exitoso: se despliegue la información en lugar de un error.
2. Agregar obra
   Objetivo: Verificar que la agregar una obra desde la aplicación, esta se guarda exitosamente.
   Procedimiento: Visitar la aplicación en /Dashboard, hacer clic en el botón de crear obra, ingresar los datos y confirmar el formulario.
   Resultado exitoso: se retorna a /Dashboard y la obra nueva aparece en la lista de obras.
3. Agregar gasto
   Objetivo: Verificar que al agregar un gasto desde la aplicación, este se guarda exitosamente
   Procedimiento: Visitar la aplicación en /Dashboard, hacer clic en una obra, rellenar el formulario de crear gasto y presionar el botón.
   Resultado exitoso: el gasto es agregado dinámicamente al sitio web.

4. Eliminar obra

Objetivo: Verificar que al eliminar una obra desde la aplicación, esta se elimina exitosamente.
Procedimiento: Visitar la aplicación en /Dashboard, hacer clic en una de las obras, hacer clic en el botón "Eliminar Obra".
Resultado exitoso: se retorna a /Dashboard y la obra nueva aparece en la lista de obras.

5. Eliminar gasto
   Objetivo: Verificar que al eliminar un gasto desde la aplicación, este se elimina exitosamente
   Procedimiento: Visitar la aplicación en /Dashboard, hacer clic en una obra, y hacer clic en el boton 'x' a la derecha de un gasto en la lista
   Resultado exitoso: el gasto es removido dinámicamente al sitio web.