

Informe de Cambios y Funcionalidad

Melany Monroy Icaza

Introducción

En el levantamiento del proyecto se encontraron algunos errores que se han corregido satisfactoriamente para que el proyecto sea levantado y funcional. En el presente informe se recogen los errores específicos y las medidas que se tomaron para abordarlos. Se clasifican estos errores por ambientes: backend y frontend, y se adjunta evidencia del funcionamiento tras las correcciones del modo solicitado.

Backend - Spring Boot

Field *name* Missing

La prueba de funcionamiento comenzó levantando el backend. El primer error que se encontró al ejecutar el comando `./mvnw spring-boot:run` indicaba que faltaba un campo requerido en la *entity*, "name". Para solucionar este error se aumentó el campo "name" en el archivo que se halla en el directorio `/sevuelo-backend/src/main/java/ec/sevolutivo/sevuelos/sevuelos/domain/Request.java` como se ilustra a continuación.

```
@NotNull
@Size(max = 100)
@Column(name = "name", length = 100, nullable = false)
private String name;

...

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}
```

Database configuration

Una vez arreglado el error, se pudo notar que en el archivo "application.properties" no había contenido alguno y que no había una conexión a una base de datos regular (mySql, Oracle, Postgres). La elección de Postgres como base de datos requería incluir en el archivo "pom.xml" la dependencia necesaria, como figura a continuación.

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
```

Siguiendo esta elección, se eliminó la dependencia a "h2" que además generaba conflicto. A continuación, se detallaron las configuraciones adecuadas en application.properties, de la siguiente manera.

```
server.port=${port:8080}
spring.datasource.url=jdbc:postgresql://localhost:5432/db_prueba
spring.datasource.username=theda
spring.datasource.password=postgretheda
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.hibernate.ddl-auto=create
```

nótese que en la línea `spring.jpa.hibernate.ddl-auto=create` se especifica la opción `create` para que la base de datos se cree (o sobrescriba) una vez se corra el comando para instalar `./mvnw clean`

`install`. En caso se quiera únicamente actualizar una base de datos ya existente se debe especificar la opción `update`. Se ejecutó el comando:

```
./mvnw clean install -DskipTests
```

Para instalar la aplicación y crear la base de datos. Se usó la bandera `DskipTests` para saltar las pruebas unitarias y evitar así conflictos por falta de configuración en estas.

Una vez realizado todo esto el servidor en el puerto 8080 para el backend fue levantado exitosamente.

Frontend - ReactJS

Package version

Se hallaron dos errores en el archivo `package-lock.json`:

1. El primero se encontraba en la versión de `react` especificado en "`dependencies`". La línea original de código se mostraba `"react": " 18.2.0",`, lo que se cambió por `18.2.0`.
2. El segundo error era en la especificación de `babel-loader` y `webpack`. Estas especificaciones causaron conflicto porque al ejecutar `npm install` para instalar la aplicación levantada en react, el módulo `react-scripts` ya cargaba una versión actualizada del `webpack` y `babel-loader`.

Una vez corregidos ambos errores se ejecutó primero `npm install` seguido de `npm start`.

Http Request

La aplicación indicaba una pestaña titulada "New Request" y otra que se mostraba al ingreso (establecida «por defecto») titulada "Request". Estaba claro que el formulario en New Request tenía que ingresar datos en la tabla que se mostraría en Requests. Se realizó la prueba y el primer error que hubo es respecto al CORS. Para solucionar esto, en el backend, en el archivo `RequestResource.java` se añadió la URL que estaba siendo denegada en la anotación `@CrossOrigin`: `http://localhost:3000/requests`. Se procedió a realizar la prueba nuevamente, obteniendo un error con el status http 500, error interno del servidor. Esto indicaba que había un problema ya sea con el cuerpo de la petición http hacia el server, o con el modo en que el server procesaba tal petición. Para esto se imprimió el http request que se generaba al presionar "save" en "New Request" y se verificó que los datos ingresados en el formulario estaban siendo procesados sin problema desde el frontend. Para verificar el error específico en el server se hizo uso de Postman. Al hacer la petición http a través de Postman se pudo visualizar nuevamente el error de código http 500. Revisando el terminal se pudo observar lo siguiente

```
jakarta.validation.ConstraintViolationException: Validation failed for
classes [ec.sevolutivo.sevuelos.sevuelos.domain.Request] during persist
time for groups [jakarta.validation.groups.Default, ]
List of constraint violations:[
    ConstraintViolationImpl{interpolatedMessage='must not be null',
    propertyPath=name, rootBeanClass=class
    ec.sevolutivo.sevuelos.sevuelos.domain.Request,
    messageTemplate='{jakarta.validation.constraints.NotNull.message}'}
]
```

Esto indicaba que, por supuesto, al incluir el campo `name` tal como se indica en la sección *Backend* el campo no podría estar nulo. En este punto, se podría haber escogido dos caminos: modificar la entity para que el campo `name` acepte valores nulos, o mandar en el http request el nombre además del pasajero y el destino. Por simpleza se escogió modificar el http request, con la siguiente modificación:

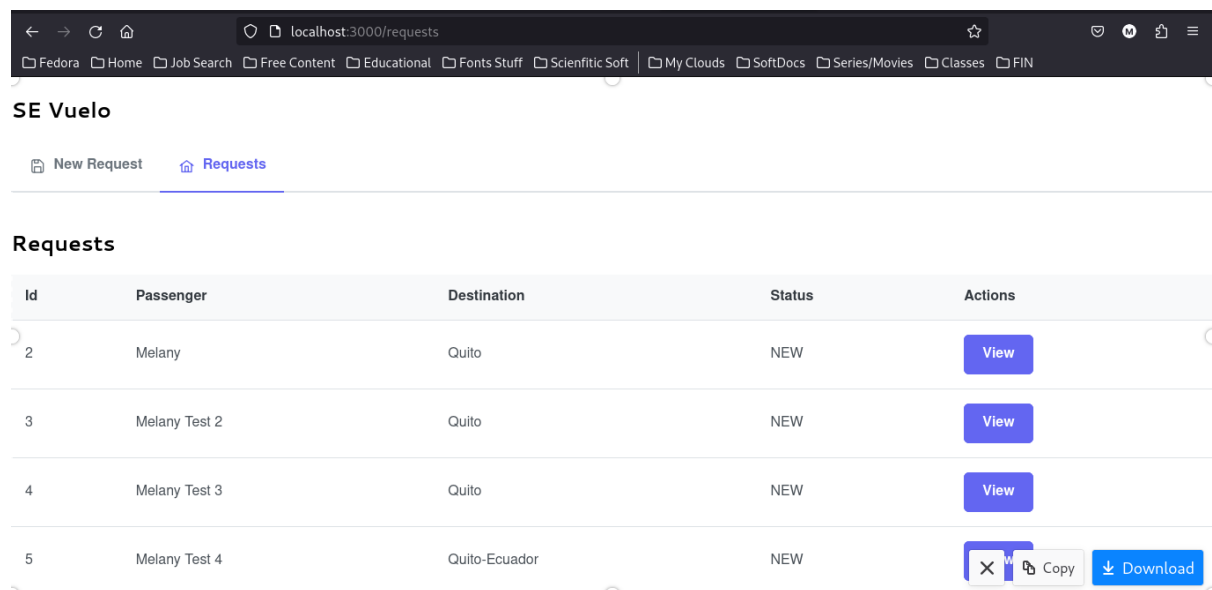
```
const onSubmit = (data: any) => {
  const newRequest = defaultValue;
  newRequest.name = data.passenger;
  newRequest.passenger = data.passenger;
  newRequest.destination = data.destination;
  createRequest(newRequest);
  reset();
};
```

Nótese la línea ingresada `newRequest.name = data.passenger`; donde se especifica que el nombre va a ser el mismo que el ingresado en el campo `passenger`, y la línea borrada `window.location.href =`

"/requests". Habiendo hecho las dos primeras pruebas se procedió a verificar si se puede obtener la respuesta de la petición http con estado 200 para que la página "refresque" una vez que el payload ha sido exitosamente procesado y la base de datos cargada. Al ser evidente en la prueba 3 que esto no sería posible en el archivo *new-request.tsx* se procedió a modificar la función importada desde *request.service.ts* *createRequest* de la siguiente manera.

```
export const createRequest = (request: IRequest) => {
  const requestUrl = `${apiUrl}/requests`;
  return axios.post(requestUrl, request).then((response: any) => {
    if (response.status === 200) {
      window.location.href = "/requests"
    }
  });
};
```

Con esto se realizó la prueba número 4 y se pudo verificar que el aplicativo funcione acorde a lo esperado. En la Figura 1 se puede verificar el resultado.



Id	Passenger	Destination	Status	Actions
2	Melany	Quito	NEW	<button>View</button>
3	Melany Test 2	Quito	NEW	<button>View</button>
4	Melany Test 3	Quito	NEW	<button>View</button>
5	Melany Test 4	Quito-Ecuador	NEW	<button>X</button> <button>Copy</button> <button>Download</button>

Figure 1: Captura de pantalla con los datos siendo ingresados adecuadamente.

En la Figura 2 se puede apreciar que las entradas efectivamente provienen y han sido guardadas exitosamente en la base de datos.

```
db_prueba=# SELECT * FROM request;
 id | destination | name       | passenger | status
-----+-----+-----+-----+-----
  2 | Quito      | Melany     | Melany    | NEW
  3 | Quito      | Melany Test 2 | Melany Test 2 | NEW
  4 | Quito      | Melany Test 3 | Melany Test 3 | NEW
  5 | Quito-Ecuador | Melany Test 4 | Melany Test 4 | NEW
(4 rows)

db_prueba=#
```

Figure 2: Captura de pantalla con los datos siendo ingresados adecuadamente.