HW1: Mid-term assignment report

Joao Santos[110555], v09

1 Introduction

1.1 Overview of the work

This report presents the midterm individual project required for TQS, covering both the software product features and the adopted quality assurance strategy.

The main purpose of BusConnection Application is to provide users with the ability to Query bus connections between specific origin and/or destination locations, along with filtering options such as departure date and currency. They can also book reservations choosing what seat they want and check them with the token generated.

1.2 Current limitations

About limitations, I tried to implement a persistence database, but I couldn´t, I´ve tried with SQL, with PostgreSQL, with hibernate but my application never connected to the database

server, so I gave up and use spring hibernate non persistent database. So, whenever I shutdown the Spring Application I lose all progress on the database.

Also, I'm using spring cache to store the API information, so the cache is emptied every time Spring Application is shut downed.

2 Product specification

2.1 Functional scope and supported interactions

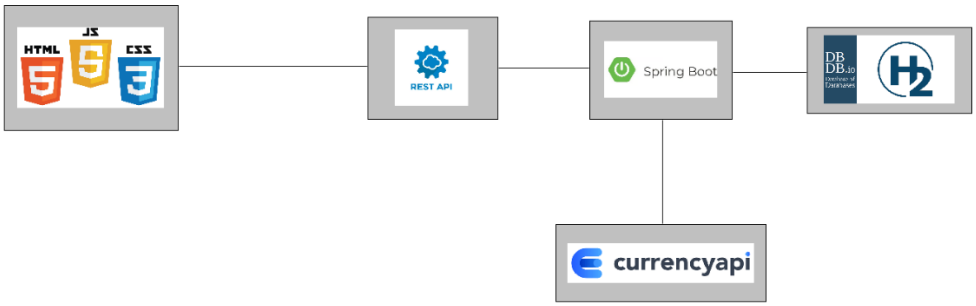
The main usage scenario is a regular person that has interest in travelling from Lisbon to Porto by bus and wants to book a ticker online. For that the person will fill the from with the origin and destination and see the available trips. Then the person will book the reservation and reserve a seat. A token will be generated, and the person must save it. Then if the person wants to check their reservation, he must use the token previously generated.

2.2 System architecture

The application can be divided into 2 big parts, front-end and back-end.

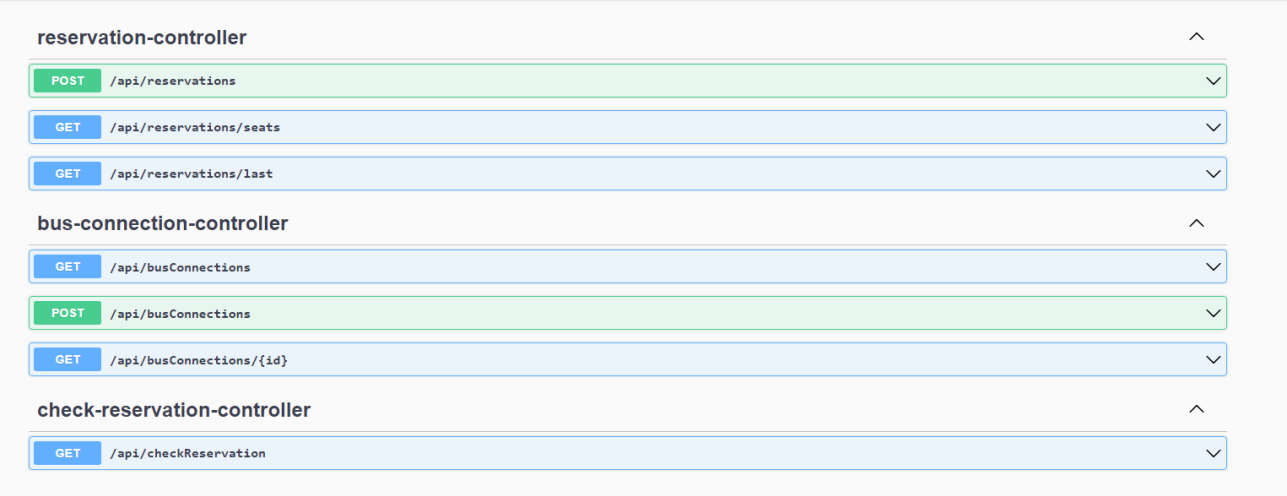
The front-end was made via HTML, CSS and JS.

The backend was developed with Spring boot, and it has a controller layer to communicate with the front end. The service layer to handle the service logic and functions. The repository layer to have functions to interact with the database. The entity layer to create the entities and provide a definition for the database table. The component layer that has some data initialization and some functions to help the internal system.

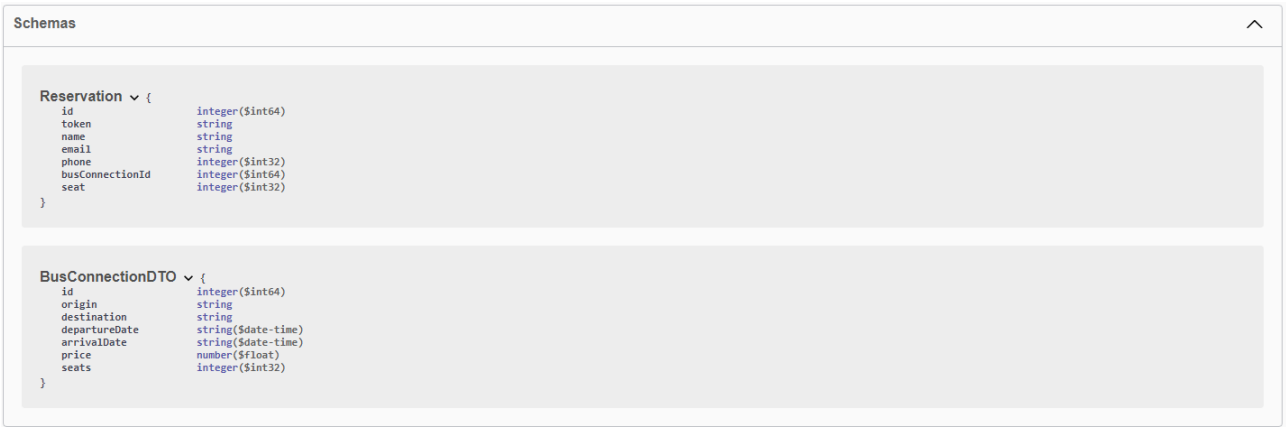


To launch the application run mvn spring-boot:run and access the port http://localhost:8080.

2.3 API for developers



This API documentation can be accessed through http://localhost:8080/swagger-ui/index.html.´



The schemas are also present.

3 Quality assurance

3.1 Overall strategy for testing

Since the application was organized in layers, I used integration tests to test a single layer component to then start another one. For that I used mocks and simulate the behavior of the layers that were not implemented yet. I also used cucumber

3.2 Unit and integration testing

As unit tests I tested some validate functions when initializing the entities such as valid phone number or email address.

```java
// Testing the id
@Test
public void testId() {
    busConnection.setId(id:10L);
    assertThat(busConnection.getId()).isEqualTo(expected:10L);
}

// Testing the invalid price
@Test
public void testInvalidPrice() {
    assertThrows(expectedType:IllegalArgumentException.class, () -> busConnection.setPrice(-10));
}

// Testing the invalid seats
// if the number of seats is greater than 50, throws an exception
public void testInvalidSeats() {
    List <Integer> seats = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        seats.add(i);
    }
    assertThrows(expectedType:IllegalArgumentException.class, () -> new BusConnection().setSeats(seats));
}

// Testing the invalid seats
// If the number of seats is less than 0, the number of seats is set to 0
@Test
public void testInvalidSeats2() {
    List <Integer> seats = new ArrayList<>();
    for (int i = 0; i < -10; i++) {
        seats.add(i);
    }
    assertThrows(expectedType:IllegalArgumentException.class, () -> new BusConnection().setSeats(seats));
```

```java
@Test
public void testInvalidSeat() {
    assertThrows(expectedType:IllegalArgumentException.class, () -> reservation.setSeat(seat:0));
    assertThrows(expectedType:IllegalArgumentException.class, () -> reservation.setSeat(-1));
}

@Test
public void testInvalidPhone() {
    assertThrows(expectedType:IllegalArgumentException.class, () -> reservation.setPhone(phone:123));
    assertThrows(expectedType:IllegalArgumentException.class, () -> reservation.setPhone(phone:1234567890));
}

@Test
public void testInvalidEmail() {
    assertThrows(expectedType:IllegalArgumentException.class, () -> reservation.setEmail(email:"email"));
}
```

After that I went on a multi-layer application test were I mock the behavior of some layers and test internal functions of a single layer. When I' am assured that the layer is tested I start developing the next layer.

```java
@Mock(lenient = true)
private BusConnectionRepo busConnectionsRepo;

@InjectMocks
private BusConnectionServiceIMPL busConnectionsService;

@BeforeEach
void setUp() {
    List<Integer> seats = Arrays.asList(...a:1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20);

    BusConnection busConnection1 = new BusConnection(origin:"Porto", destination:"Lisboa", new Date(year:2021, month:10, date:10, hrs:10, min:0), new Date( 2021, 10, 1
    BusConnection busConnection2 = new BusConnection(origin:"Porto", destination:"Lisboa", new Date(year:2021, month:10, date:10, hrs:8, min:0), new Date( 2021, 10, 10
    BusConnection busConnection3 = new BusConnection(origin:"Porto", destination:"Lisboa", new Date(year:2021, month:10, date:10, hrs:12, min:0), new Date( 2021, 10, 1
    BusConnection busConnection4 = new BusConnection(origin:"Lisboa", destination:"Porto", new Date(year:2021, month:10, date:10, hrs:14, min:0), new Date( 2021, 10, 1


    List<BusConnection> busConnections1 = Arrays.asList(busConnection1);
    List<BusConnection> busConnections2 = Arrays.asList(busConnection1, busConnection2, busConnection3);

    Mockito.when(busConnectionsRepo.findByOriginAndDestinationAndDepartureDate(busConnection1.getOrigin(), busConnection1.getDestination(), busConnection1.getDepartureD
    Mockito.when(busConnectionsRepo.findByOriginAndDestination(busConnection1.getOrigin(), busConnection1.getDestination())).thenReturn(busConnections2);
    Mockito.when(busConnectionsRepo.findByOrigin(busConnection1.getOrigin())).thenReturn(busConnections2);
    Mockito.when(busConnectionsRepo.findByDestination(busConnection1.getDestination())).thenReturn(busConnections2);
    Mockito.when(busConnectionsRepo.findAll()).thenReturn(Arrays.asList(busConnection1, busConnection2, busConnection3, busConnection4));
    Mockito.when(busConnectionsRepo.findById(busConnection1.getId())).thenReturn(java.util.Optional.of(busConnection1));
    when(busConnectionsRepo.save(any(type:BusConnection.class))).thenReturn(busConnection1);
    Mockito.when(busConnectionsRepo.findByOriginAndDestinationAndDepartureDate(origin:"Joao", destination:"Santos", new Date(year:2021, month:10, date:10, hrs:10, min:0
    Mockito.when(busConnectionsRepo.findByOriginAndDestination(origin:"Joao", destination:"Santos")).thenReturn(Arrays.asList());
    Mockito.when(busConnectionsRepo.findByOrigin(origin:"Joao")).thenReturn(Arrays.asList());
    Mockito.when(busConnectionsRepo.findByDestination(destination:"Santos")).thenReturn(Arrays.asList());

}
```

```java
@Test
void getBusConnections_WhenOriginDestinationAndDepartureDate_thenReturnBusConnections() {
    List<Integer> seats = Arrays.asList(...a:1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20);
    BusConnection busConnection1 = new BusConnection(origin:"Porto", destination:"Lisboa", new Date(year:2021, month:10, date:10, hrs:10, min:0), new Date( 2021, 10, 10, 15,
    List<BusConnection> found = busConnectionsService.getBusConnections(origin:"Porto", destination:"Lisboa", new Date(year:2021, month:10, date:10, hrs:10, min:0));


    assertThat(found).hasSize(expected:1);
    assertThat(found.get(index:0).getOrigin()).isEqualTo(busConnection1.getOrigin());
    assertThat(found.get(index:0).getDestination()).isEqualTo(busConnection1.getDestination());
    assertThat(found.get(index:0).getDepartureDate()).isEqualTo(busConnection1.getDepartureDate());
    assertThat(found.get(index:0).getArrivalDate()).isEqualTo(busConnection1.getArrivalDate());

}

void whenSearchInvalidOriginDestinationAndDepartureDate_thenReturnEmptyList() {
    List<BusConnection> found = busConnectionsService.getBusConnections(origin:"Porto", destination:"Lisboa", new Date(year:2021, month:10, date:10, hrs:10, min:0));

    assertThat(found).isEmpty();
```

```java
@Test
void getBusConnections_WhenOrigin_thenReturnBusConnections() {
    List<Integer> seats = Arrays.asList(...a:1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20);
    BusConnection busConnection1 = new BusConnection(origin:"Porto", destination:"Lisboa", new Date(year:2021, month:10, date:10, hrs:10, min:0), new Date( 2021, 10, 10, 15,
    BusConnection busConnection2 = new BusConnection(origin:"Porto", destination:"Lisboa", new Date(year:2021, month:10, date:10, hrs:8, min:0), new Date( 2021, 10, 10, 13,
    BusConnection busConnection3 = new BusConnection(origin:"Porto", destination:"Lisboa", new Date(year:2021, month:10, date:10, hrs:12, min:0), new Date( 2021, 10, 10, 17,
    List<BusConnection> found = busConnectionsService.getBusConnections(origin:"Porto", destination:"", departureDate:null);


    assertThat(found).hasSize(expected:3);
    assertThat(found.get(index:0).getOrigin()).isEqualTo(busConnection1.getOrigin());
    assertThat(found.get(index:1).getOrigin()).isEqualTo(busConnection2.getOrigin());
    assertThat(found.get(index:2).getOrigin()).isEqualTo(busConnection3.getOrigin());
}
```

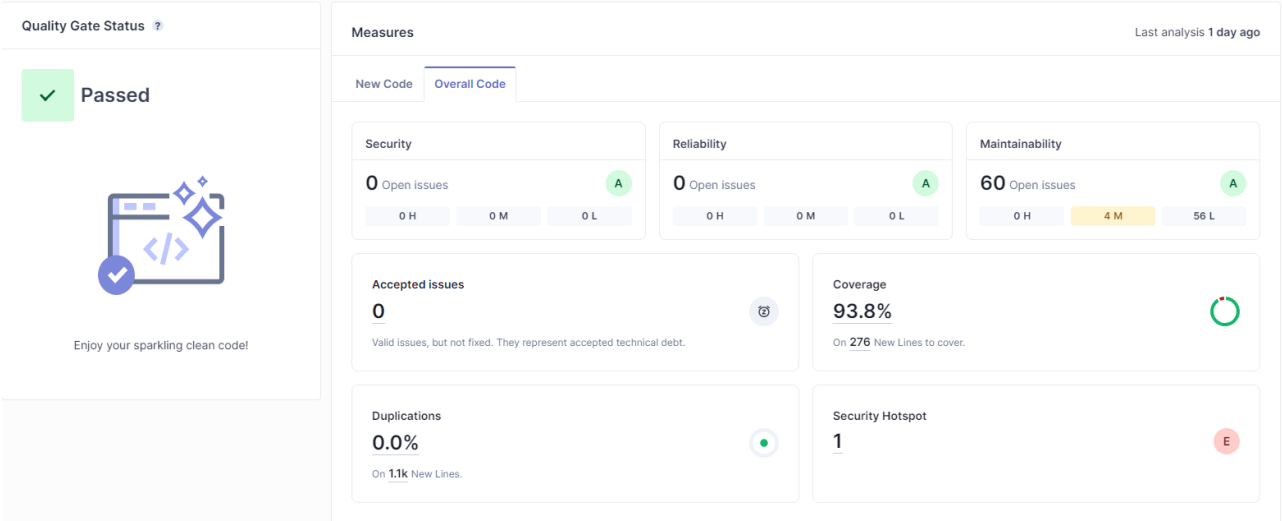
```java
void testUpdateCurrencyData() {
    // Simulando a resposta da API
    Map<String, Map<String, Double>> mockResponse = new HashMap<>();
    Map<String, Double> data = new HashMap<>();
    data.put(key:"USD", value:1.0);
    data.put(key:"EUR", value:0.85);
    mockResponse.put(key:"data", data);

    // Simulando chamada à API
    when(restTemplate.getForObject(anyString(), eq(value:Map.class))).thenReturn(mockResponse);

    // Testando se os dados de câmbio são atualizados corretamente
    currencyAPI.updateCurrencyData();
    assertNotNull(currencyAPI.getLatestCurrencyData());
    assertFalse(currencyAPI.getLatestCurrencyData().isEmpty());
    assertEquals(expected:1.0, currencyAPI.getLatestCurrencyData().get(key:"data").get(key:"USD"));
    assertEquals(expected:0.85, currencyAPI.getLatestCurrencyData().get(key:"data").get(key:"EUR"));
}
```

```java
@Autowired
private MockMvc mockMvc;

@MockBean
private ReservationRepo reservationRepo;

// Write tests here

@Test
public void testCheckReservation() throws Exception {
    when(reservationRepo.findByToken(anyString()))
        .thenReturn(new Reservation(name:"Joao", email:"joao@hotmail.com", phone:123456789, seat:1, busConnectionId:1L));

    mockMvc.perform(get(urlTemplate:"/api/checkReservation")
        .param(name:"reservationToken", ...values:"123456789"))
        .andExpect(status().isOk());
}

@Test
public void testCheckReservationNotFound() throws Exception {
    when(reservationRepo.findByToken(anyString()))
        .thenReturn(value:null);

    mockMvc.perform(get(urlTemplate:"/api/checkReservation")
        .param(name:"reservationToken", ...values:"123456789"))
        .andExpect(status().isNotFound());
}
```

3.3 Functional testing

3.4 Code quality analysis

For code quality analysis first used only Jacoco to see what part of the code was I covering with the tests and after most of the tests were done I used SonarCloud to see more detailed analysis on my tests results and I fixed a lot of things.



The security Hotspot caught on the SonarCloud is due to the exposing of the currency API key (which I don't mind being exposed to).
The 56 maintainability issues are due to package names or public functions mostly and I don't see that as a impactful issue.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines
TQS.Homework.Controller		91%		78%	6	27	3	76
TQS.Homework.Entities		93%		90%	3	49	1	79
TQS.Homework.Component		98%		n/a	1	7	2	18
TQS.Homework.Services.IMPL		97%		92%	2	22	3	47
TQS.Homework		37%		n/a	1	2	2	3
TQS.Homework.Services		100%		91%	1	11	0	24
TQS.Homework.DTO		100%		n/a	0	16	0	31
Total	62 of 1 387	95%	12 of 94	87%	14	134	11	278

For the coverage I have a 93 % coverage which I consider being ok. When I looked up for the parts of the code that were not being covered, I didn't noticed anything of big importance.

There were a total of 61 tests made on this application

3.5 Continuous integration pipeline [optional]

4 References & resources

Project resources

Resource:	**URL/location:**
Git repository	https://github.com/JotaCLS/TQS_110555
Video demo	Video included in git repository
QA dashboard (online)	I runned sonar locally via docker
CI/CD pipeline	[**optional**; if you have th CI pipeline definition in a server, place the URL here]
Deployment ready to use	[**optional**; if you have the solution deployed and running in a server, place the URL here]

Reference materials

https://currencyapi.com – For the currency API