# HW1: Relatório do Projeto Individual

*Guilherme da Silva Amorim [107162]*, v2024-04-09

1.1 Visão Geral do Trabalho....1
1.2 Limitações Atuais....1
2.1 Funcionamento Geral e Interações....2
2.2 Arquitetura do Sistema....2
2.3 Especificações da API....3
3.1 Estratégia Geral de Teste....3
3.2 Testes Unitários e de Integração....3
3.3 Testes Funcionais....4
3.4 Análise da Qualidade de Código....4
4.1 Recursos do Projeto....5

## 1 Introdução

### 1.1 Visão Geral do Trabalho

Ele relatório apresenta o projeto individual de TQS, abrangendo tanto os recursos do produto como a estratégia de garantia de qualidade.
*BuzzyTrips* é um sistema de reserva de viagens de autocarro entre grandes cidades.
No desenvolvimento deste projeto, a estratégia optada passou por inicialmente criar o *frontend*, de modo a ter uma ideia inicial, de que entidades e *endpoints* seriam necessários. Após se ter chegado à conclusão da necessidade da criação das entidades *Travel* e *Reservation*, que teriam uma relação direta, uma vez que uma viagem teria várias reservas, essas classes foram desenvolvidas, assim como os respetivos repositórios, serviços e *controllers*.

### 1.2 Limitações Atuais

A principal limitação do sistema é o facto de que ao criar uma reserva, o número de lugares da respetiva viagem não é alterado. Para se corrigir isto, seria necessário desenvolver um uma função que alterasse o número de lugares disponíveis da viagem, tendo em conta o número de passageiros que a reserva apresentasse.
Além disso, uma *feature* que poderia ter sido implementada era a possibilidade de um utilizador poder pesquisar por uma reserva que tivesse feito anteriormente, uma vez que existe o *endpoint* necessário para obter uma reserva pelo *token*, que é apenas usado para apresentar os detalhes da reserva, na página final.

# 2 Especificação do Produto

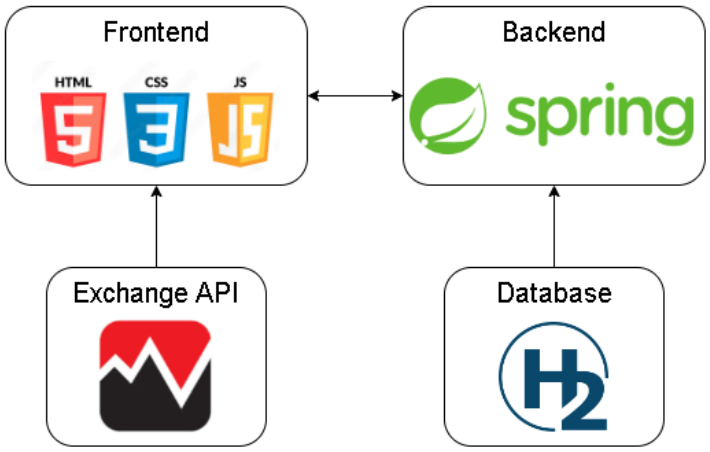
## 2.1 Funcionamento Geral e Interações

O sistema desenvolvido tem como único ator um utilizador que pretende reservar uma viagem de autocarro. Fá-lo através dos seguintes passos:

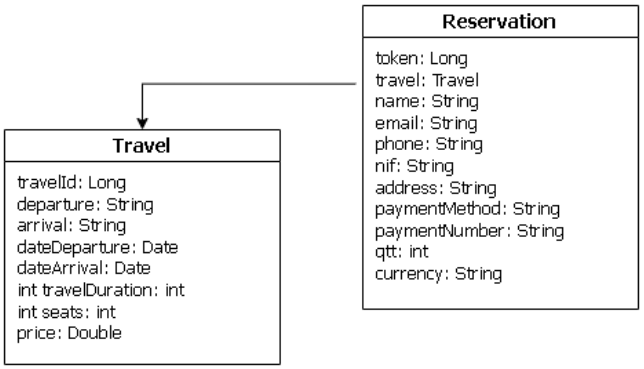
1. Escolher origem e destino da viagem, podendo ou não especificar o dia desejado
2. Escolher o horário desejado, através dos que se encontram listados
3. Preencher um formulário com os dados pessoais e selecionar o número de bilhetes disponíveis, de forma a efetuar a reserva

## 2.2 Arquitetura do Sistema

O Frontend foi desenvolvido em *HTML*. Foi usado *CSS* para personalizar a aparência da aplicação e *Javascript* para ligar a aplicação à *API*.
O *backend*, foi desenvolvido com *Spring Boot*, como recomendado e a base de dados escolhida foi o H2.
Para obter as taxas de câmbio de moeda em tempo real, é chamada a API externa *Exchange Rate*, que é aplicada diretamente no frontend.




Como referido anterior, a base de dados foi moldada tendo em conta as entidades *Travel* e *Reservation*, sendo que estas têm uma relação de *Many to One*.

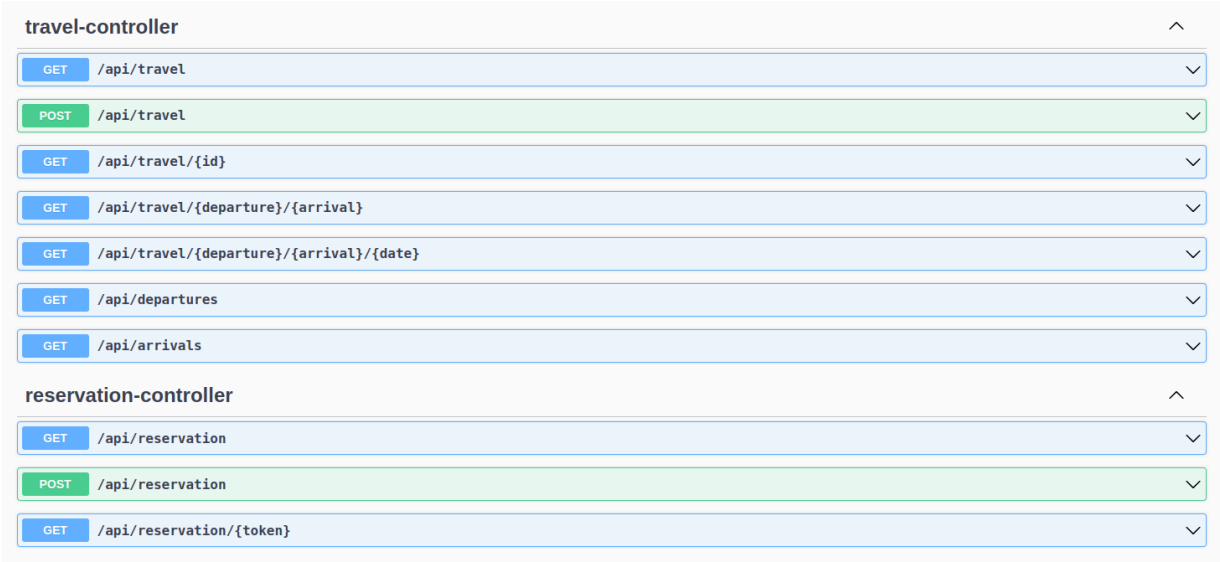


### 2.3 Especificações da API

A API é composta por *endpoints* que permitem obter informação relativo a viagens e reservas:

- Obter viagem por ID;
- Obter viagem por origem e destino (e data)
- Obter cidades de origem e destino
- Obter reserva por *token*
- Obter todas as viagens ou reservas

É também possível, através de POST, criar viagens ou reservas




## 3 Garantia de Qualidade

### 3.1 Estratégia Geral de Teste

A estratégia adotada para testar o produto passou por inicialmente serem criados os testes relacionados aos métodos de *Travel* e *Reservation*. Depois, foram feitos testes de integração e testes à interface, com *Selenium*. No final, foi feita uma análise, usando o SonarQube, onde foram detetados alguns *issues* e métodos que não eram cobertos por testes, que foram depois respetivamente corrigidos e implementados

### 3.2 Testes Unitários e de Integração

Para os testes unitários foi utilizado Junit para testar o *TravelRepository*, *TravelService*, *TravelController*, *ReservationRepository*, *ReservationService* e *ReservationController*, onde foram testados os métodos CRUD de viagens e reservas, de forma isolada. Em seguida, foram realizados testes de integração, de modo a garantir o bom funcionamento das componentes, usando a classe *TestRestTemplate*, do *Spring Boot*.

## 3.3 Testes Funcionais

Para testar a interface, foi usada a extensão *Selenium* no Chrome, onde foi gravado o processo de reserva de uma viagem e depois exportado para *Java* e implementado no projeto.

## 3.4 Análise da Qualidade de Código

Como referido anteriormente, foi usado o *SonarQube* para uma análise ao código. Inicialmente, a cobertura de código era de 87% e foram detetados cerca de 40 *issues*, que tinham algumas semelhanças e, por isso, acabaram por não constituir um grande problema para a sua resolução. Exemplos:

```
    @PostMapping("/reservation")
    public ResponseEntity<Reservation> saveReservation(@RequestBody Reservation reservation) {
        return          new          ResponseEntity<Reservation>(reservationService.save(reservation),
HttpStatus.CREATED);
    }
```

Replace the type specification in this constructor call with the diamond operator ("<>").

```
    @GetMapping("/travel/{departure}/{arrival}")
    public  List<Travel>  getTravelByDepartureAndArrival(@PathVariable("departure")  String  departure,
@PathVariable("arrival") String arrival) throws ParseException {
        return travelService.getTravelsByDepartureAndArrival(departure, arrival);
    }
```

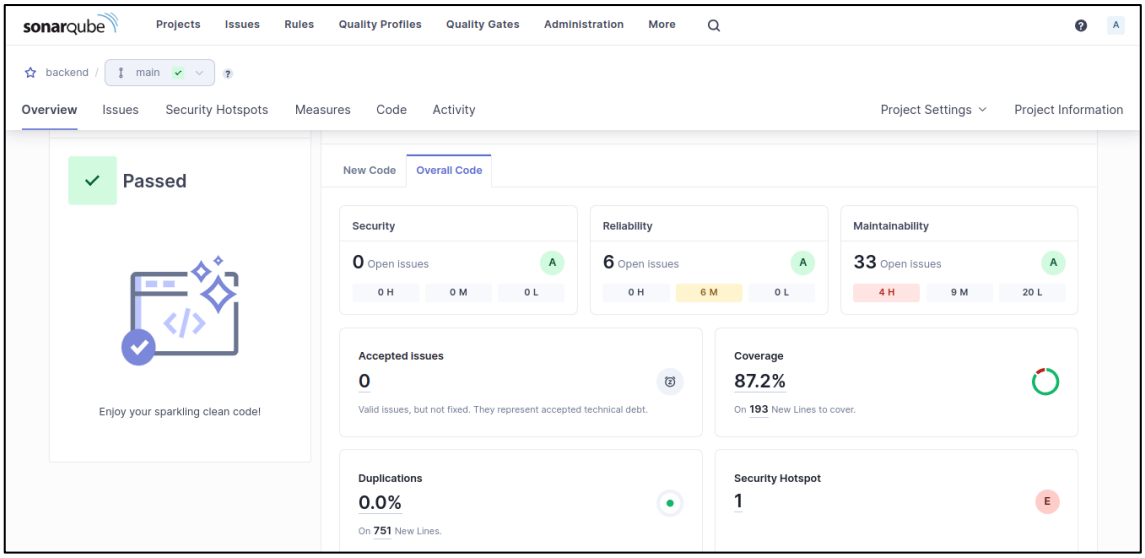
Remove the declaration of thrown exception 'java.text.ParseException', as it cannot be thrown from method's body.

```
    private int MAX_SEATS = 25;
```

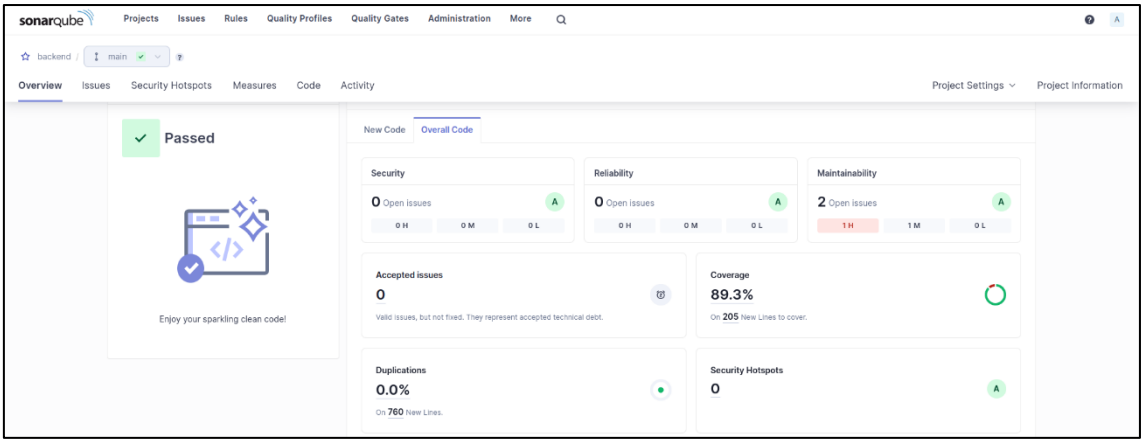
Rename this field "MAX_SEATS" to match the regular expression '^[a-z][a-zA-Z0-9]*$'.

```
@ExtendWith(SeleniumJupiter.class)
public class FrontendTest {
  private WebDriver driver;
  private Map<String, Object> vars;
```

Remove this 'public' modifier.



Após a correção de erros e criação de novos testes, a cobertura subiu para 89%. Mantiveram-se 2 *issues*: o primeiro relacionado ao facto de ser criada uma reserva com mais de 7 parametros e o segundo em relação ao facto de não haver nenhum *assert* em *BackendApplicationTests*




# 4 Referências e Recursos

## 4.1 Recursos do Projeto

| Recurso | URL/Localização: |
|---|---|
| Repositório Git | https://github.com/GuiAmorim03/TQS_107162 |
| Video | /Homeworks/HW1/analysis/videoDemo.webm |