UNIVERSIDADE D
COIMBRA

Mestrado em Engenharia Informática
Integração de Sistemas

Assignment 2
Three-tier Programming with Object-Relational Mapping

Rui Barata
2015238609
rpbarata@student.dei.uc.pt

November 15, 2021

Contents

1 Introduction

Neste projeto foi desenvolvido uma aplicação web para gerir uma companhia de autocarros. O conjunto de operações desta aplicação limita-se essencialmente à compra de bilhetes.
Com a autorização do docente responsável da cadeira, o projeto foi desenvolvido usando a framework Ruby on Rails[1], usando a ORM default do Rails, Active Record[2].
Foi utilizada uma base de dados PostgreSQL e o SideKiq[3] para gerir os processos que executão em background.
Para ajudar a desenvolver a camada de apresentação, foi utilizado Bootstrap 5.
Para o envio de email, foi utilizado o serviço Mailtrap[4]. Mailtrap é um uma ferramenta para testar de forma segura o envio de emails em ambientes de desenvolvimento. Mailtrap irá capturar os emails enviados pelo servidor, através do Action Mailer[5] do Rails, para uma caixa de entrada virtual.

2 Presentation Tier

A camada de apresentação está dividida em três partes:

Login Este ecrã é acedido por qualquer pessoa. Ao fazer login, ou criando uma conta nova, terá acesso aos restantes ecrãs.

Root A root poderá ser acedida por qualquer tipo de utilizador com login feito. É neste ecrã que poderá ver as viagens disponíveis, bem como comprar novos bilhetes. Também consegue aceder ao seu perfil para o editar ou eliminar. Consegue aceder à sua carteira para consultar o saldo que tem, bem como para a carregar. Consegue ainda aceder à lista das viagens que comprou e cancelar uma viagem futura.

Management Apenas conseguem aceder a este ecrã os utilizadores que tenham o cargo de administrador (manager). Aqui é possível listar e criar novas viagens, bem como ver o top de passageiros com mais viagens.

Em todo a aplicação existe uma barra de navegação que permite aceder de forma bastante rápida a todos estes ecrãs.

A Figure 1 mostra o ecrã Root da aplicação.

A parte mais complexa desta camada é a de gerir os acessos e manter a palavra-passe dos utilizadores segura.

[1] https://rubyonrails.org/
[2] https://guides.rubyonrails.org/active_record_basics.html
[3] https://github.com/mperham/sidekiq
[4] https://mailtrap.io/
[5] https://guides.rubyonrails.org/action$_{m}ailer_{b}asics.html$

O armazenamento das palavras-passe foi feito através da classe SecurePassword[6] do Rails. Esta classe fornece métodos para autenticar um utilizador na aplicação, utilizando o método de criptografia BCrypt para proteger a palavra-passe na base de dados. Este mecanismo necessita que a nossa model User tenha um atributo password_digest e adiciona já também algumas validações.

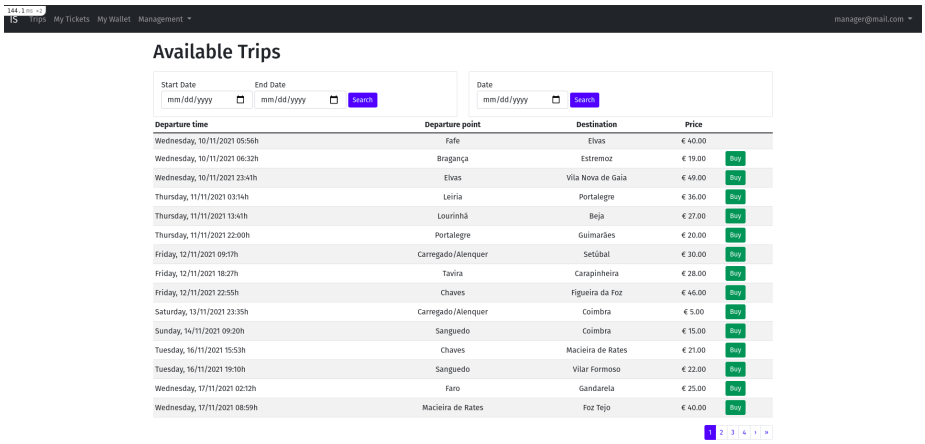

Figure 1: Ecrã Root

3 Business Tier

Para controlar o acesso aos diferentes recursos da aplicação, depois de autenticado, é guardado o id do utilizador na session. A session[7] funciona como uma hash e é disponibilizada pelo Rails a cada utilizador que acede à aplicação. Se o utilizador já tem uma sessão ativa, o Rails usa a session existente. Caso contrário, cria uma nova.
Esta session permite armazenar pequenas quantidades de dados que serão persistidas entre pedidos. A session está apenas disponível nas camadas de apresentação e de negócio e a informação é guardada do lado do cliente.
Para verificar se um user pode aceder a um determinado recurso da nossa aplicação, é verificado se existe algum user na nossa base de dados com o id armazenado na session. Caso contrário, é negado o acesso.

Todas as transações usam a ORM default do Rails, o Active Record. O Active Record facilita a criação e o uso de objetos de negócio cujos dados requerem armazenamento persistente numa base de dados.

[6] https://edgeapi.rubyonrails.org/classes/ActiveModel/SecurePassword/ClassMethods.html
[7] https://guides.rubyonrails.org/security.htmlsessions

Ao realizar estas queries, é devolvido um objeto do tipo ActiveRecord::Relation[8]. É este objeto que é passado à camada de apresentação e que representa um array dos dados obtidos nas queries efectuadas. Para não enviar mais dados do que é necessário apresentar na camada de apresentação, são apenas escolhidos os campos necessários, utilizando o método select[9].

O objeto ActiveRecord::Relation é guardado numa variável de instância da classe do respectivo controlador.
Para enviar os dados para a camada de apresentação, o Rails responde ao pedido no format html, procurando a respetiva view para apresentar a informação na camada de apresentação. Como por default, este é o formato utilizado pelo Rails, não é necessário indicar o tipo de resposta. Mas caso fosse necessário, é possível responder, por exemplo, com um formato json ou um xml, dependendo do Accept Header enviado pelo cliente.

A tal variável de instância é então partilhada para a view como resposta do pedido feito ao respectivo controlador.
Deste modo, não há qualquer tipo de comunicação entre a camada de apresentação e a camada de dados.

As operações que envolvem transações monetárias, como carregar a wallet do user ou comprar/cancelar um bilhete estão executadas dentro de uma Active Record Transaction[10]. Uma Transaction é um bloco de proteção em que as instruções de SQL só são permanentes se todas puderem ser bem-sucedidas como uma ação atômica. Por exemplo, no caso do cancelamento de um bilhete, esta operação só é de facto realizada se for possível efectuar todo o processo de forma completa, ou seja, devolver o dinheiro à wallet do user e eliminar o bilhete da base de dados. Se algum destes processos falhar por algum motivo, o bilhete não é cancelado.

4 Data tier

O projeto é constituído por 5 identidades: User, Ticket, Wallet, Place e Trip. A relação entre estas está bem definida no diagrama da Figure 2.
Note que o departure_point_id e o destination_point_id na tabela dos Users é uma foreign key para a tabela Place e que a password_digest está encriptada.

Qualquer dado, antes de ser gravado na base de dados é validado.
Estas são as validações que são efectuadas:
User

- username: comprimento máximo de 50 caracteres, se estiver presente

[8] https://api.rubyonrails.org/v6.1.0/classes/ActiveRecord/Relation.html
[9] https://apidock.com/rails/ActiveRecord/QueryMethods/select
[10] https://api.rubyonrails.org/v6.1.4/classes/ActiveRecord/Transactions/ClassMethods.html

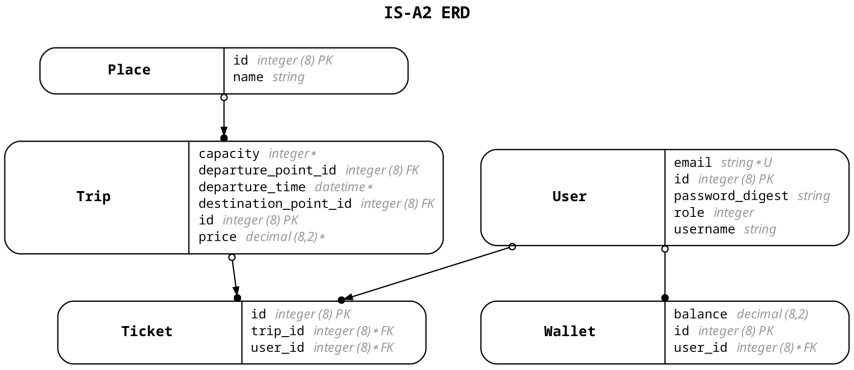

Figure 2: ER Diagram

- password: comprimeito entre 6 e os 40 caracteres
- email: tem de estar sempre presente, tem de ser único e tem de seguir a regex

Trip

- capacity: tem se ser obrigatório, tem de ser um inteiro e maior do que zero
- departure_point: tem de existir sempre
- departure_time: tem de existir sempre
- destination_point: tem de existir sempre
- price: tem de existir sempre e tem de ser um número maior que zero.
- departure_time: tem de ser no futuro

Wallet

- balance: tem de ser maior que zero

5 Project Management and Packaging

O projeto está preparado para correr com o Docker e com Docker Compose. Inicialmente deve-se fazer build dos containers com o comando

```
docker-compose build
```

De seguida, para inicial os containers, deve-se correr o comando

```
docker-compose up
```

Poderá executar os dois comandos de uma vez, com

```
docker-compose up --build
```

Para entrar dentro do container, poderá fazer

```
docker-compose exec rails bash
```

numa outra janela do terminal. A partir daqui, pode-se também aceder a uma consola do Rails, escrevendo na bash

```
rails console
```

Outra alternativa é usar o Visual Studio Code com a extensão Visual Studio Code Remote - Containers. Também são fornecidos os ficheiros necessários a esta configuração.

Ao iniciar o projeto pela primeira vez, são corridas as seeds que populam a base de dados com alguns dados iniciais para ser mais fácil desenvolver e testar a plataforma.
São criados 51 locais que irão fazer parte da lista de locais de partida e de chegada das viagens criadas pelos administradores. Além dos locais, são criadas 50 viagens com os campos gerados aleatoriamente, utilizando a gem Faker[11]. Além disso, é criado um Manager default.

Todo o resto do projeto, segue a estrutura normal de um projeto Ruby on Rails, encontrando-se o código mais importante dentro da pasta app.
Dentro desta pasta, encontram-se três pastas: controllers, views e models que são, respetivamente, onde se encontram o código das camadas de negócio, de apresentação e de dados.
Além disso, dentro da pasta job encontra-se o código relativo a todos os processos que correm em background.

Para criar o primeiro administrador/manager, deve estar a executar uma consola dentro do terminal e correr o seguinte comando

```
rails manager:new
```

seguindo as informação que serão apresentadas.

[11] https://github.com/faker-ruby/faker