

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Arquiteturas Aplicacionais & Sistemas Interativos

Gestão de uma Escola de Condução

Engenharia de Aplicações

Grupo 3

*João Vieira (A78468)
Hugo Oliveira (A78565)
Raphael Oliveira (A78848)*

1 de Julho de 2019

Resumo

Este relatório aborda o projeto e todas as decisões tomadas ao longo do seu desenvolvimento no âmbito das Unidades Curriculares de **Arquiteturas Aplicacionais** e **Sistemas Interativos**.

Este projeto consiste no estudo e no desenvolvimento de uma aplicação de gestão de uma **Escola de Condução**. O objetivo passa fornecer um suporte informático de modo facilitar a gestão de informações relacionadas com a escola.

O público alvo da aplicação é constituído essencialmente pelos **Alunos** e por todo o corpo docente, como **Instrutores** e **Secretários**.

Conteúdo

1	Introdução	1
2	Análise do funcionamento de uma escola	2
2.1	Modelo de Domínio	2
2.2	Intervenientes e principais tarefas	3
2.3	Algumas observações	3
3	Requisitos funcionais	4
3.1	Requisitos gerais	4
3.2	Secretário	4
3.3	Instrutor	5
3.4	Aluno	6
4	Modelação da aplicação	7
4.1	Diagrama de Use Cases	7
4.2	Modelo Independente da Plataforma (PIM)	8
4.3	Modelo Específico da Plataforma (PSM)	11
5	Análise dos utilizadores	13
5.1	Aluno	13
5.2	Instrutor	13
5.3	Secretário	13
6	Análise de tarefas	14
7	Mockups da aplicação	16
7.1	Aluno	16
8	Tecnologias e frameworks utilizadas	21
8.1	Camada de dados	21
8.2	Camada de negócio	22
8.3	Camada de interface	23
9	Implementação e algumas considerações	24
9.1	Funcionalidade extra	24
9.2	Serviço externo	25
10	Interface Web final	26
10.1	Técnicas de usabilidade utilizadas	26
10.2	Secretário cancela aula prática de aluno	30
11	Arquitetura adotada e Deployment	31
11.1	Fase de desenvolvimento	31
11.2	Fase de produção	31
11.2.1	Especificações das máquinas utilizadas	32

12 Testes de carga	33
12.1 Tipos de teste	33
12.2 Resultados finais	35
13 Conclusão	39

Lista de Figuras

1	Modelo de Domínio do sistema	3
2	Diagrama de <i>Use Cases</i> da aplicação.	7
3	Modelo Independente da Plataforma	8
4	Diagrama de classes das relações com a classe <i>Register</i>	9
5	Diagrama de classes das relações com a classe <i>Lesson</i>	10
6	Diagrama de classes das relações com a classe <i>Announcement</i>	10
7	Modelo Específico da Plataforma	11
8	Modelo Específico da Plataforma com <i>Servlets</i> e <i>Beans</i>	12
9	Tarefa <i>Marcar Aula</i> realizada por um Aluno.	14
10	Tarefa <i>Marcar Aula</i> realizada por um Secretário.	14
11	Tarefa <i>Cancelar Aula</i> realizada por um Aluno.	14
12	Tarefa <i>Cancelar Aula</i> realizada por um Secretário.	15
13	Tarefa <i>Registar Aviso</i> realizada por um Secretário.	15
14	Página de autenticação do utilizador.	16
15	Página inicial de Aluno depois de autenticado.	16
16	Página relativa a aulas de Aluno.	17
17	Consulta de aulas teóricas realizadas e temas.	17
18	Consulta de horários das próximas aulas teóricas.	18
19	Consulta de aulas práticas realizadas e temas.	18
20	Marcação de aulas práticas.	19
21	Página do perfil do aluno.	20
22	Stack tecnológica da aplicação desenvolvida.	21
23	Processo de marcação de aulas com a funcionalidade extra	24
24	Fluxo dos dados relativos à meteorologia	25
25	Exemplo no lado do cliente dos dados recebido pela API do IPMA	25
26	Exemplo de responsividade da aplicação.	26
27	Fornecer informação sobre o estado da aplicação.	27
28	Fornecer informação sobre o estado da aplicação (menu de navegação).	27
29	Fornecer informação sobre o estado da aplicação (<i>breadcrumb</i>).	27
30	Etapas do processo de marcação de uma aula.	27
31	Confirmação do cancelamento de uma aula prática.	28
32	Aviso geral publicado com sucesso.	28
33	Valor do NIF do aluno não cumpre a pattern necessária.	28
34	Fornecer ajuda sobre certos valores pedidos.	29
35	Impedir o <i>submit</i> de formulários até que os campos sejam preenchidos.	29
36	Aparência da aplicação.	29
37	Tarefa <i>Cancelar Aula</i> realizada por um Secretário.	30
38	<i>Cancelar Aula</i> realizada por um Secretário através da aplicação.	30
39	Arquitetura adotada para o <i>deployment</i> da aplicação	31
40	Pesos de execução para as tasks de um aluno	34
41	Pesos de execução para as tasks de um instrutor	34
42	Pesos de execução para as tasks de um secretário	35
43	<i>Benchmarking</i> com 100 clientes	36
44	<i>Benchmarking</i> com 350 clientes	37

Lista de Tabelas

1	Informações gerais de uma escola de condução.	3
2	Inventário do sistema	32

1 Introdução

A facilidade com que uma pessoa tem em conseguir obter uma ou mais categorias de condução leva, nos dias de hoje, a que o número de clientes inscritos numa escola de condução aumente substancialmente, aumentando ainda a informação sobre os mesmos.

A experiência dos três elementos deste grupo em obtenção de uma carta de condução no passado, permite reconhecer a falta de uma variedade de informações que, geralmente, as escolas não se preocupam em fornecer, levando, de um modo geral, a uma menor satisfação enquanto clientes.

Os pontos negativos passam pela: falta de informação sobre o número de aulas realizadas e respetivas datas, quais os temas abordados nas aulas presenciadas e ainda informações sobre faturas e pagamentos. Outra das desvantagens, passa pela marcação de aulas práticas que, geralmente, é o secretário da escola que é responsável por esta tarefa.

Deste modo, o nosso objetivo consiste no desenvolvimento de uma aplicação web que permita facilitar e satisfazer o processo de obtenção da carta de condução aos alunos das escolas e, ainda, facilitar a sua gestão aos responsáveis.

2 Análise do funcionamento de uma escola

Antes de procedermos ao desenvolvimento da aplicação apresentada anteriormente, decidimos recolher algumas informações acerca do funcionamento geral de uma escola de condução. Para isso, deslocamos-nos a duas escolas de condução, ambas sediadas em Braga: Escola de Condução UM e Escola de Condução Bela Vista.

Apresentamos de seguida, algumas das respostas que foram fornecidas pelos membros de ambas as escolas de condução, após apresentarmos algumas dúvidas sobre o normal funcionamento de uma escola deste tipo:

- Perceber se as aulas teóricas são iguais para qualquer tipo de categoria de condução em que um aluno esteja inscrito → *não*.
- Em caso de indisponibilidade de aluno ou professor a comparecer a uma dada aula, como devem ambos proceder:
 - *No caso do professor, o mesmo comunica a indisponibilidade à secretaria que, posteriormente, trata de passar a mensagem ao aluno;*
 - *No caso do aluno, o mesmo tem um tempo em que pode cancelar a aula (normalmente de 24H). Depois desse prazo, a aula tende a ser contabilizada como lecionada.*
- Saber se é permitido tirar a carta simultaneamente em mais do que uma categoria → *sim*.
- Perceber se um aluno está associado a um instrutor → *por norma o aluno está sempre associado ao mesmo instrutor, salvo poucos casos.*
- Perceber por que dados são avaliadas as aulas lecionadas.
 - Aulas teóricas exigem o comparecimento em 28 aulas.
 - Aulas práticas exigem a realização de 32 aulas e 500km cumpridos.

Ainda foram obtidos outros dados relativos ao funcionamento das escolas:

- A obrigatoriedade de manter o registo médico dos alunos (tendo uma validade de 6 meses);
- Cada aula tem um tema associado;
- As principais entidades na escola são: **Secretário, Instrutor e Aluno.**

2.1 Modelo de Domínio

Com a ajuda das informações recolhidos, foi possível moldar o sistema atual de uma escola. Apresentamos em baixo, o modelo de domínio associado a uma escola de condução.

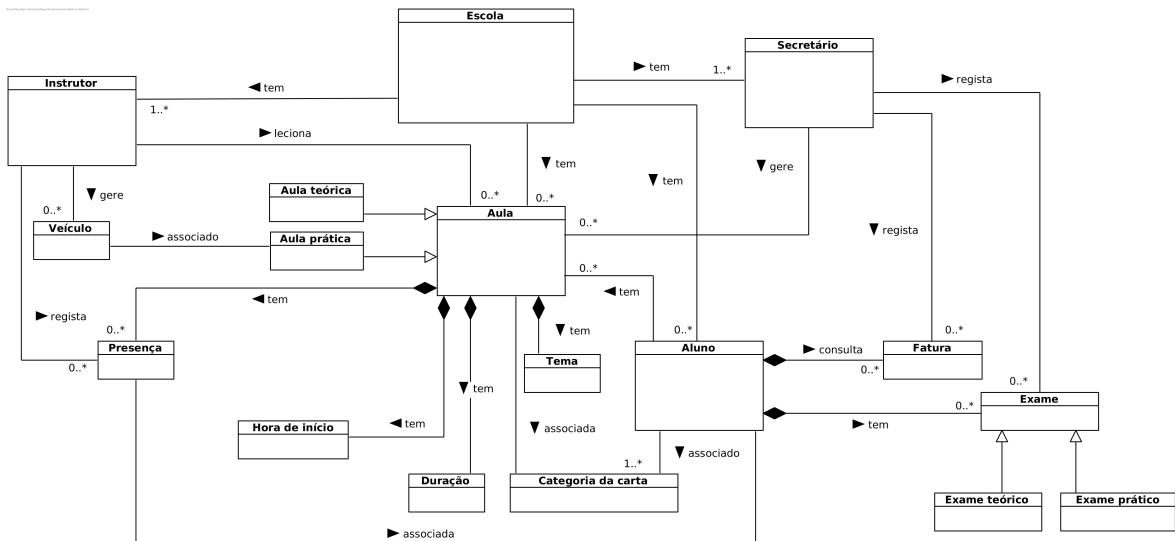


Figura 1: Modelo de Domínio do sistema

2.2 Intervenientes e principais tarefas

Através das informações recolhidas, foi possível analisarmos os intervenientes numa escola de condução e, ainda, as suas principais tarefas.

Intervenientes	Tarefas realizadas
Secretário	Gerir informações dos alunos (registar aluno e editar os seus dados)
	Marcar as aulas dos alunos (registar a hora de início e a sua duração)
	Registar faturas
Instrutor	Lecionar aulas (práticas e teóricas)
	Registrar o tema de uma aula
	Registrar as presenças de uma aula
	Gerir um veículo
Aluno	Assistir a aulas (práticas e teóricas)
	Consultar faturas

Tabela 1: Informações gerais de uma escola de condução.

2.3 Algumas observações

Para além dos dados recolhidos na tabela em cima, foi possível identificar que ambas as escolas carecem de sistemas informáticos capazes de proporcionar uma fácil apresentação dos dados por elas recolhidas. Informações estas como: número de aulas teóricas que um dado aluno frequentou e número de aulas práticas efetuadas, pagamentos efetuados de um aluno, alunos atualmente ativos na escola e, por fim, informações associadas aos veículos da escola.

3 Requisitos funcionais

O principal objetivo desta aplicação, consiste na gestão de uma escola de condução. Neste sentido, apresentamos em baixo, os requisitos funcionais da aplicação, tendo em conta os vários utilizadores existentes.

3.1 Requisitos gerais

Efetuar *login* e *logout* na aplicação

Como Utilizador da aplicação, **quero** poder efetuar *login* e *logout*, indicando a escola de condução, o e-mail e a password.

Consultar e editar dados pessoais

Como Utilizador da aplicação, **quero** poder consultar e editar os meus dados pessoais, como o nome, o contacto telefónico, o e-mail e a password.

3.2 Secretário

Registrar alunos

Como Secretário, **quero** poder registar novos alunos ou efetuar um novo registo para uma categoria nova.

Consultar dados dos alunos

Como Secretário, **quero** poder consultar os alunos registados.

Editar dados de um aluno

Como Secretário, **quero** poder editar os dados referentes a um aluno em específico.

Consultar dados dos instrutores

Como Secretário, **quero** poder consultar os instrutores registados.

Marcar ou Cancelar uma aula

Como Secretário, **quero** poder marcar ou cancelar uma aula.

Consultar aulas

Como Secretário, **quero** poder consultar as aulas de um aluno.

Editar aulas

Como Secretário, **quero** poder editar os dados de uma aula em específico.

Registar faturas

Como Secretário, **quero** poder passar faturas a um aluno.

Registar exames

Como Secretário, **quero** poder registar um novo exame de um aluno.

Lançar avisos gerais

Como Secretário, **quero** poder lançar avisos gerais.

Lançar avisos pessoais

Como Secretário, **quero** poder lançar avisos pessoais.

3.3 Instrutor

Consultar dados dos meus alunos

Como Instrutor, **quero** poder consultar os dados dos alunos associados a mim.

Consultar aulas marcadas

Como Instrutor, **quero** poder consultar as minhas aulas.

Marcar ou Cancelar uma aula

Como Instrutor, **quero** poder marcar ou cancelar uma aula.

Registrar presença

Como Instrutor, **quero** poder registrar as presenças numa dada aula.

Registrar temas lecionados

Como Instrutor, **quero** poder registrar os temas lecionados numa dada aula.

3.4 Aluno

Consultar faturas

Como Aluno, **quero** poder consultar todas as faturas associadas a um dado registo.

Consultar aulas disponíveis

Como Aluno, **quero** poder consultar os horários das aulas práticas e teóricas disponíveis.

Consultar aulas

Como Aluno, **quero** poder consultar as aulas práticas marcadas e aulas teóricas em que estive presente.

Marcar e cancelar aula prática

Como Aluno, **quero** poder marcar ou cancelar uma aula.

Consultar avisos

Como Aluno, **quero** poder consultar os avisos gerais e os avisos pessoais.

Marcar aviso pessoal como visto

Como Aluno, **quero** poder marcar os meus avisos pessoais como visto.

4 Modelação da aplicação

Ao longo desta secção serão abordados alguns tópicos subjacentes à modelação UML que nos permitem definir o comportamento e a estrutura da aplicação.

4.1 Diagrama de Use Cases

Através do diagrama de Use Case, é possível ter uma melhor percepção das três entidades da aplicação: (**Secretário**, **Instrutor** e **Aluno**). Percebe-se que todos devem ter a capacidade de se autenticar no sistema e, além disso, tanto o **Secretário** como o **Instrutor**, devem ser capazes de proceder à gestão das diferentes aulas existentes.

Em baixo é apresentado o diagrama de *Use Cases* da aplicação.

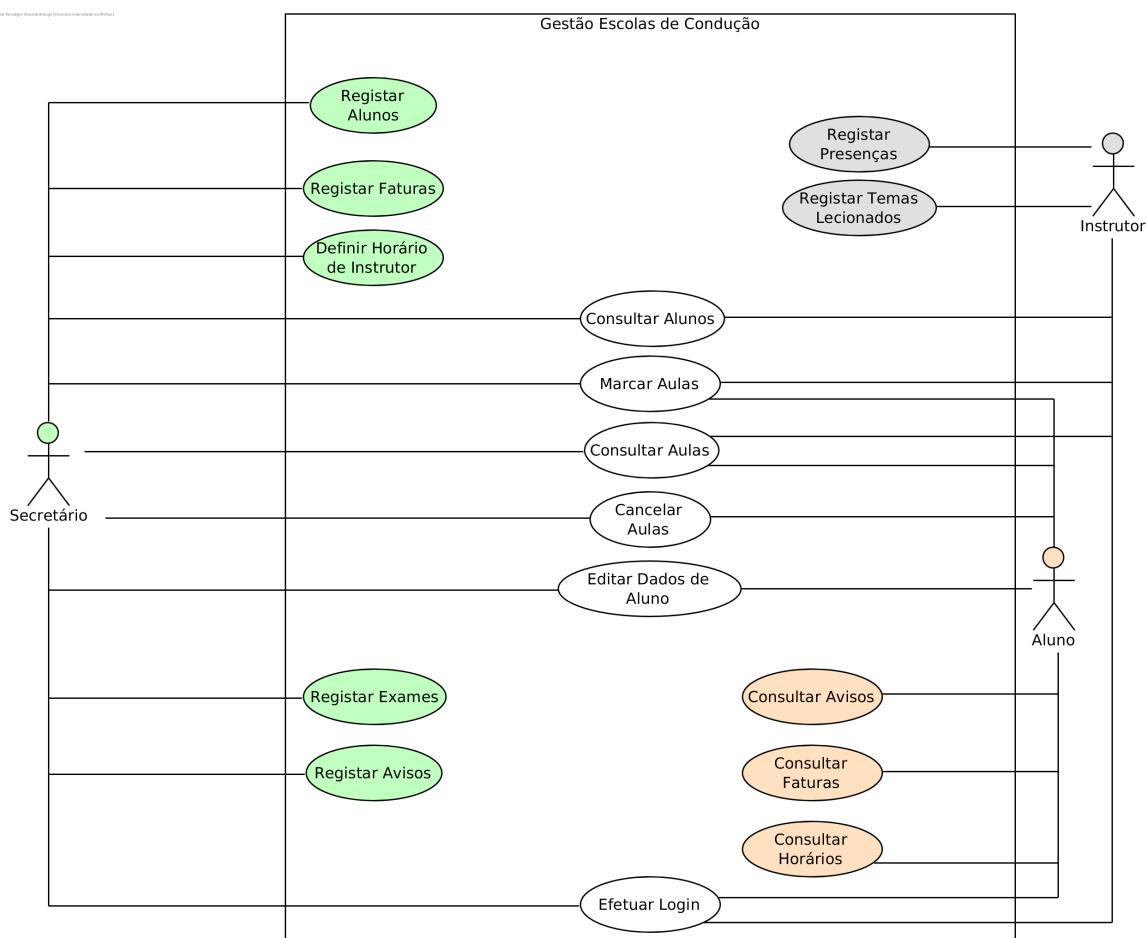


Figura 2: Diagrama de *Use Cases* da aplicação.

4.2 Modelo Independente da Plataforma (PIM)

Com a apresentação deste modelo é possível descrevermos os componentes e os serviços oferecidos pela lógica de negócio. Este modelo é um que abstrai qualquer tipo de tecnologias e/ou frameworks.

De seguida apresentamos o modelo PIM da aplicação desenvolvida.

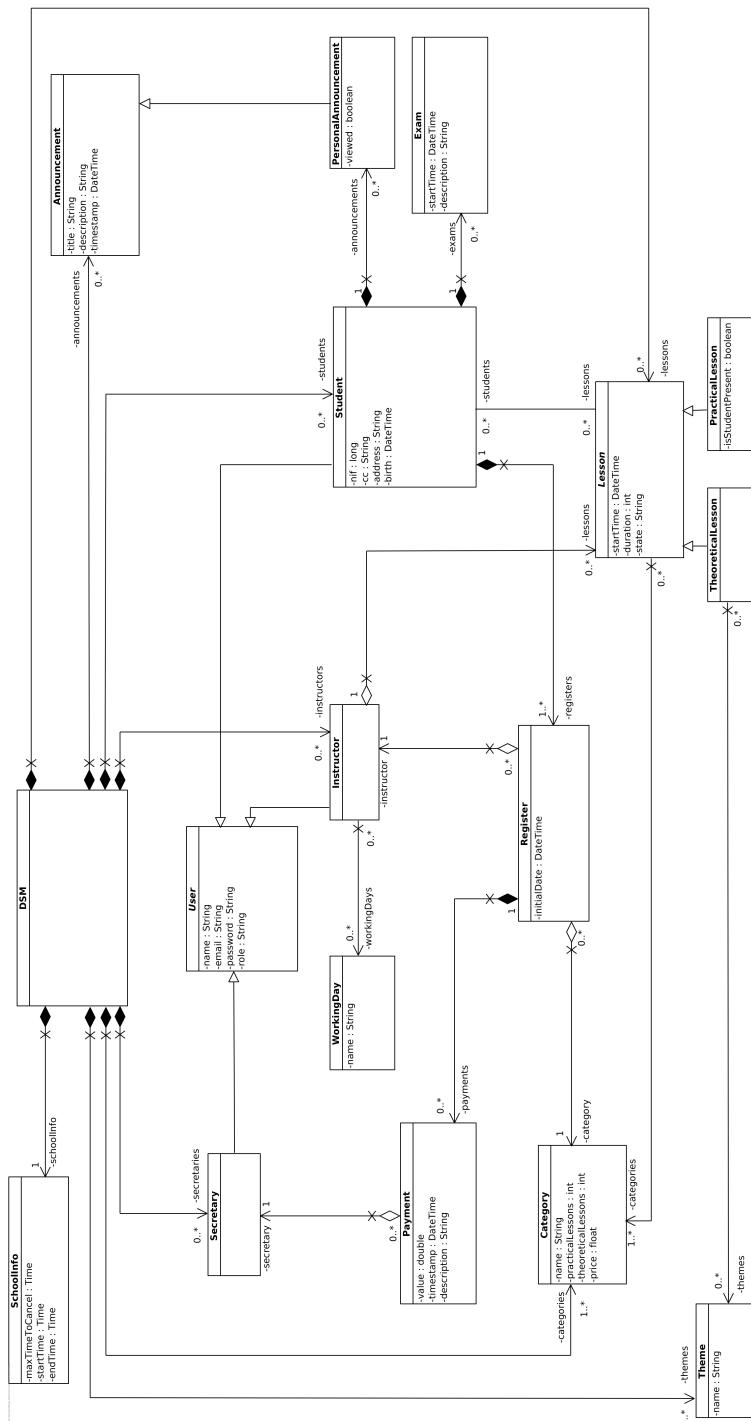


Figura 3: Modelo Independente da Plataforma

Ao longo do desenvolvimento deste diagrama, algumas considerações foram tomadas para permitir uma boa arquitetura da aplicação. Assim, e tendo em conta o diagrama em cima, apresentamos algumas das considerações mais importantes:

- **Register**: representa um dado registo de um aluno na escola de condução. A cada novo registo, deverá ser associado um instrutor e uma categoria de condução;

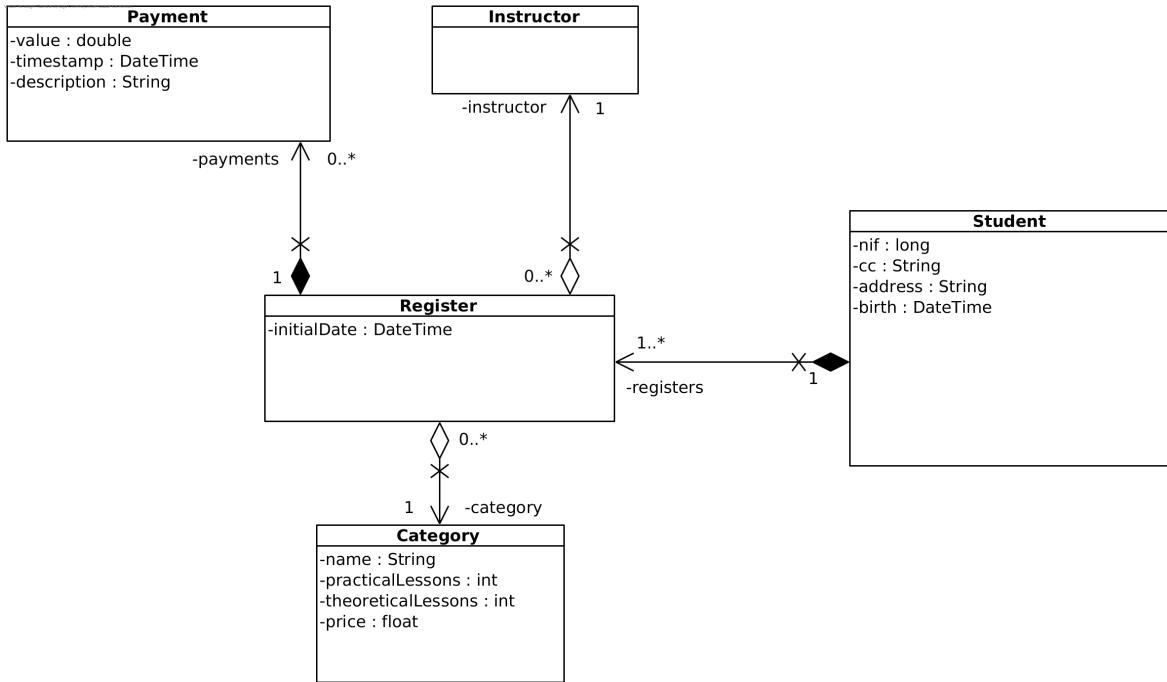
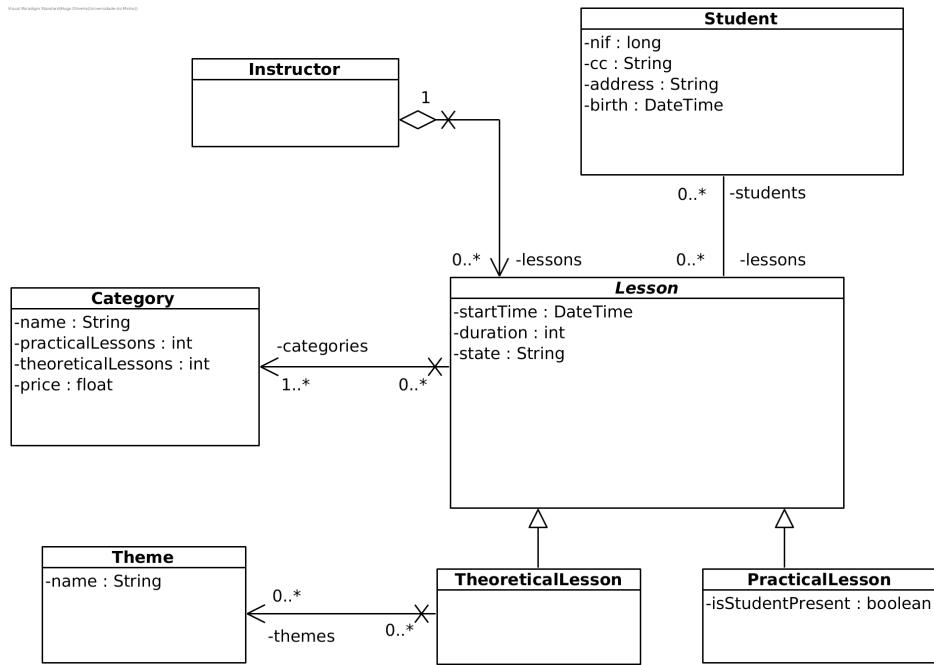


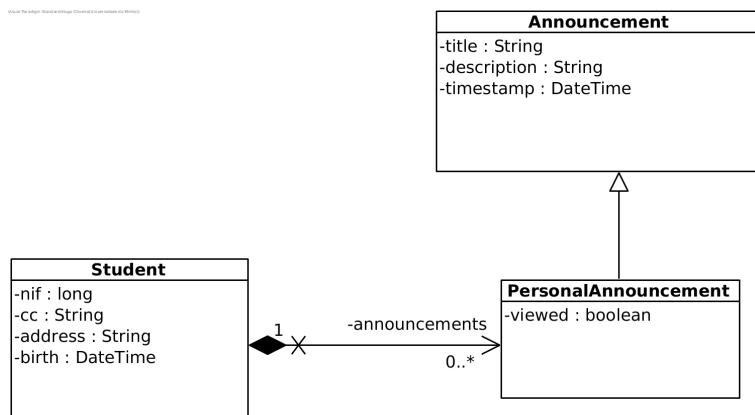
Figura 4: Diagrama de classes das relações com a classe *Register*

- **Lesson**: divide-se em duas subclasses:

- **TheoreticalLesson**: representa uma aula teórica e ao qual deve ter presente todos os temas lecionados. Isto permite que um aluno consiga perceber os temas que aprendeu e quais faltam aprender;
- **PracticalLesson**: representa uma aula prática que inclui o atributo **isStudentPresent** para podermos perceber se o aluno esteve ou não presente na aula;
- o atributo **state** consiste em três estados distintos:
 - * **reserved**: aula que foi marcada por um aluno mas ainda não foi realizada;
 - * **realized**: aula que foi realizada;
 - * **opened**: aula que foi criada e permite que os alunos consultem a mesma; se a aula for do tipo *PracticalLesson* o aluno poderá marcar a mesma para si, caso contrário é uma aula teórica na qual um aluno poderá frequentar.


 Figura 5: Diagrama de classes das relações com a classe *Lesson*

- **Announcement** e a sub-classe **PersonalAnnouncement**: a distinção destes avisos surge, essencialmente, pois é necessário que o aluno consiga demonstrar que já visualizou os avisos, uma vez que os avisos relacionados consigo requerem especial atenção;


 Figura 6: Diagrama de classes das relações com a classe *Announcement*

4.3 Modelo Específico da Plataforma (PSM)

Depois de definido o diagrama de classes da nossa aplicação, decidimos modelar dois diagrama que permitem refinar o modelo PIM anteriormente apresentado. Isto é, os dois diagramas apresentados de seguida permitem identificar as entidades a serem persistidas e apresentar uma solução arquitetural para a utilização de frameworks e tecnologias específicas que falaremos mais à frente.

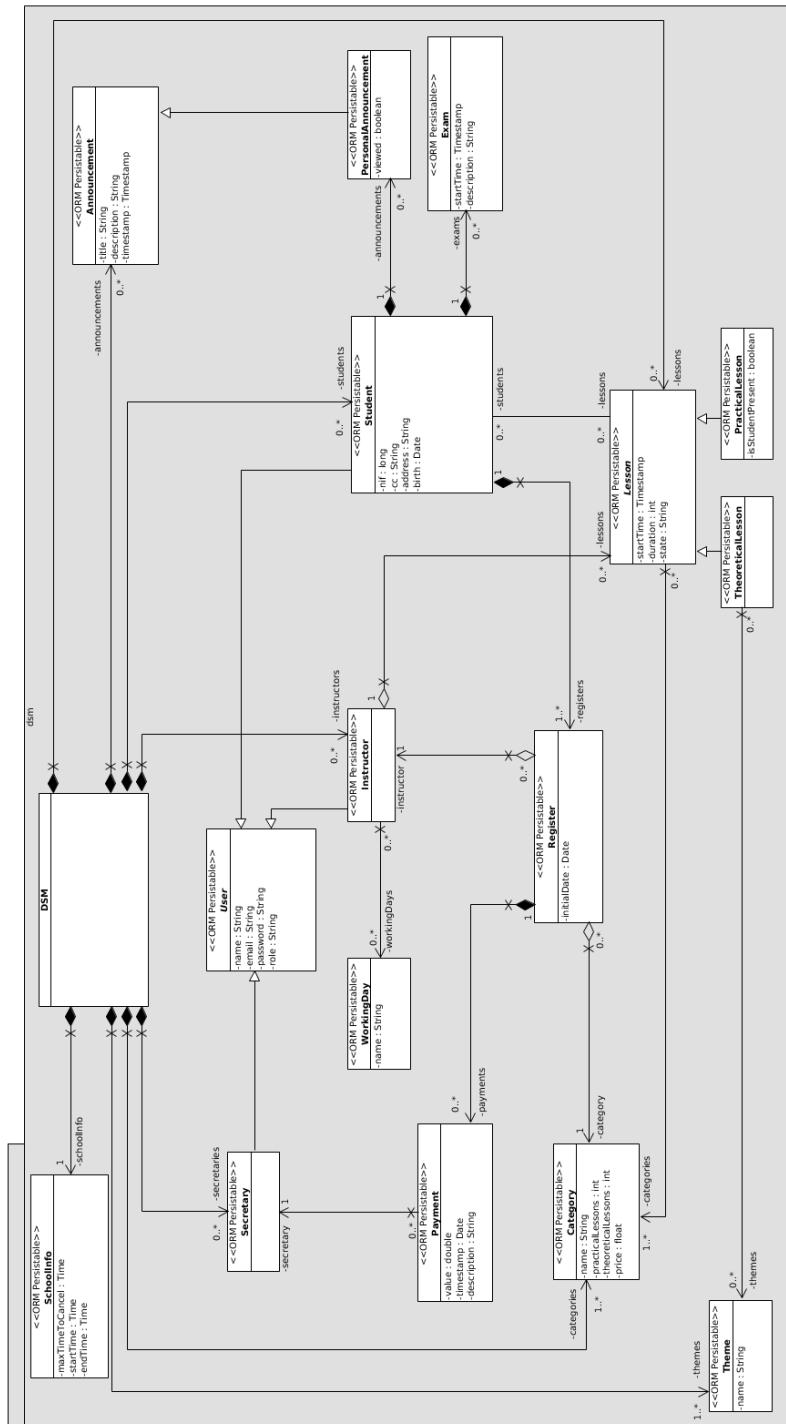


Figura 7: Modelo Específico da Plataforma

Como este projeto consiste no desenvolvimento de uma aplicação web que se foca no paradigma orientado aos objetos com a utilização do **Java EE**, decidimos modelar um novo diagrama que representa a estrutura global dos diferentes componentes existentes numa arquitetura que utilize *Servlets* e *Java Beans*. Neste diagrama são ainda apresentados os *Data Access Objects* que são utilizados para mapear as informações entre os dados relacionais e objetos Java.

Em baixo apresentamos um novo modelo PSM que, dada a sua dimensão, optamos por representar apenas **parte da sua constituição**.

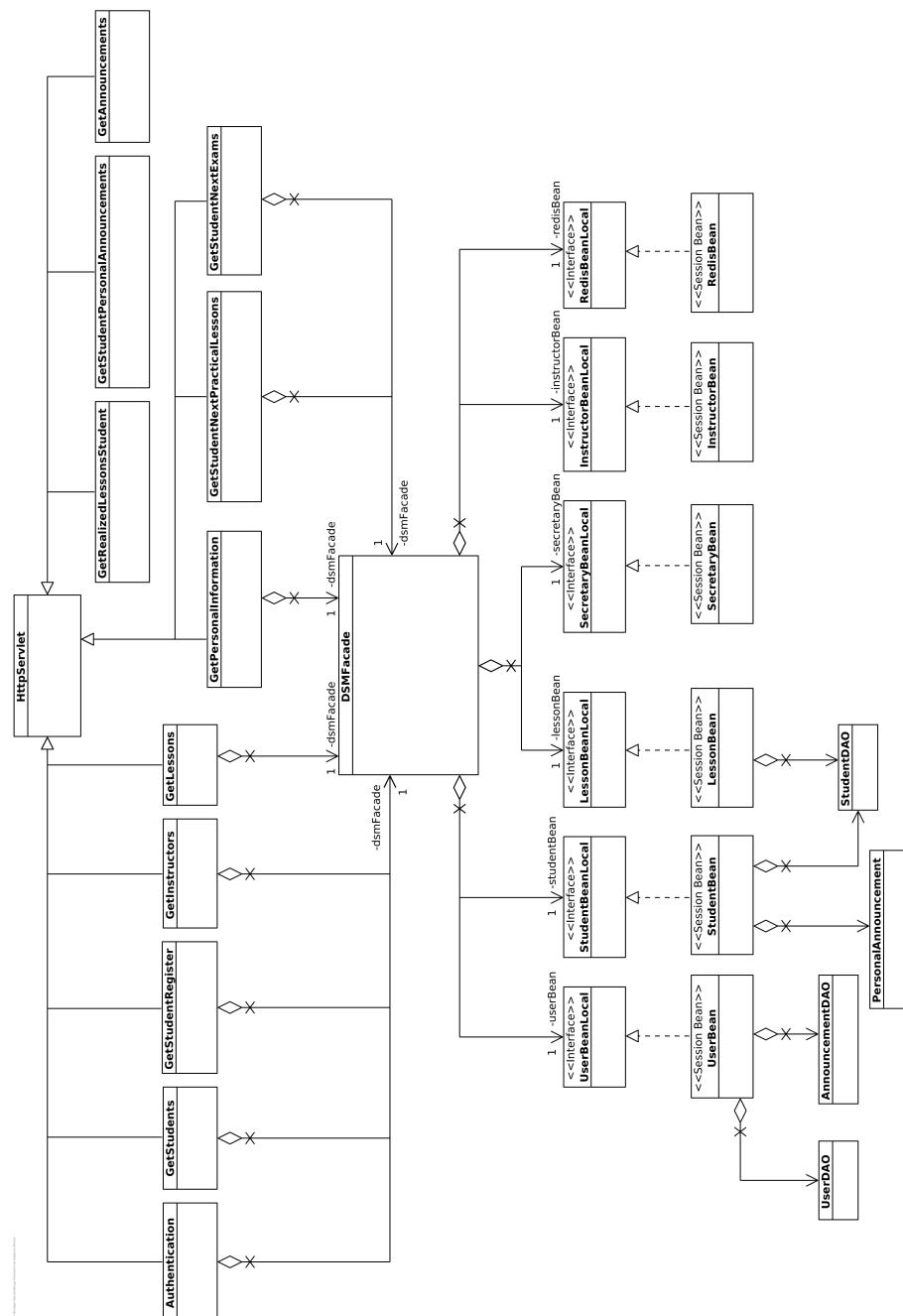


Figura 8: Modelo Específico da Plataforma com Servlets e Beans

5 Análise dos utilizadores

Com base na recolha de toda a informação na modelação da aplicação efetuada, é possível identificarmos os **três utilizadores** que irão utilizar a nossa aplicação.

Uma vez que todos os utilizadores apresentam requisitos maioritariamente distintos, é necessário apresentarmos ecrãs distintos para cada um deles e, para isso, é preciso especificarmos algumas características subjacentes a cada um, para que mais tarde, durante a implementação da interface gráfica da aplicação, possamos oferecer ao cliente uma melhor experiência.

5.1 Aluno

Os alunos de uma escola de condução são numa grande maioria jovens e, por isso, pertencem a uma geração cada vez mais confortável com o uso de tecnologias. Se o público alvo fosse apenas este, podia não ser tão necessário certos cuidados de utilização. Ainda assim é preciso ter em atenção que existem alunos de todas as faixas etárias, o que faz com que a naveabilidade do site e operação de marcação de aulas necessitem de ser facilmente interpretáveis.

5.2 Instrutor

Quanto aos instrutores, e tendo em conta que o mais importante para estes é a gestão das suas aulas e dos seus alunos, é expectável que a interface desenvolvida para o mesmo seja prática, com uma rápida execução e facilidade de interpretação, sendo melhor optar por poucos passos para efetuar uma determinada tarefa, sem grandes animações adjacentes.

5.3 Secretário

A análise feita para os secretários segue um pouco a ideia relatada para os instrutores, ou seja, deve-se permitir que a execução de uma tarefa seja de rápido alcance, visto o secretário ter uma grande componente de gestão de alunos (registo, edição, marcação de exames, pagamentos, etc).

6 Análise de tarefas

Para podermos analisar e definir as etapas de uma dada tarefa na nossa aplicação, recorremos aos diagramas HTA (*Hierarchical Task Analysis*). Em baixo são apresentados alguns dos diagramas desenvolvidos.

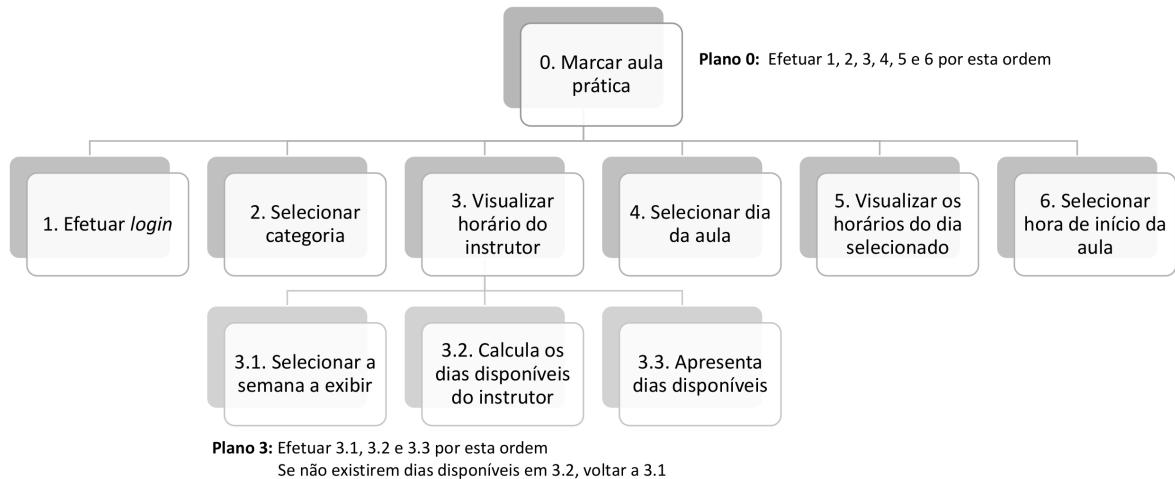


Figura 9: Tarefa *Marcar Aula* realizada por um Aluno.

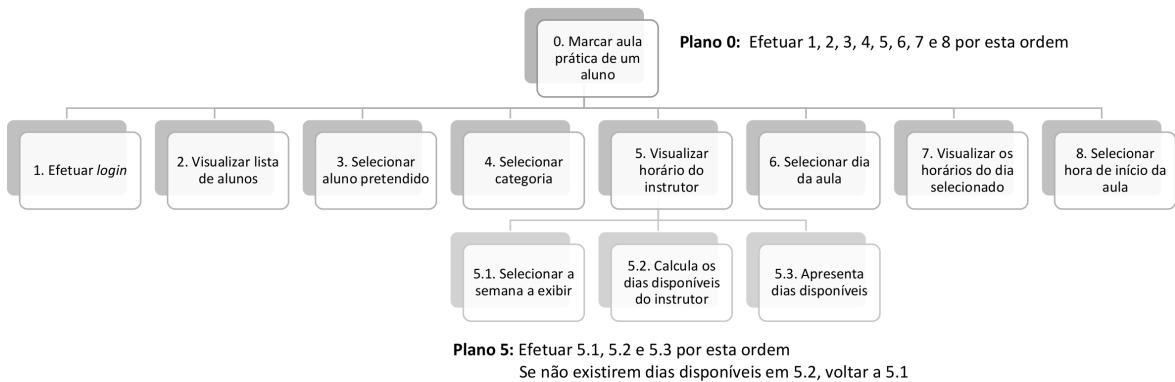


Figura 10: Tarefa *Marcar Aula* realizada por um Secretário.

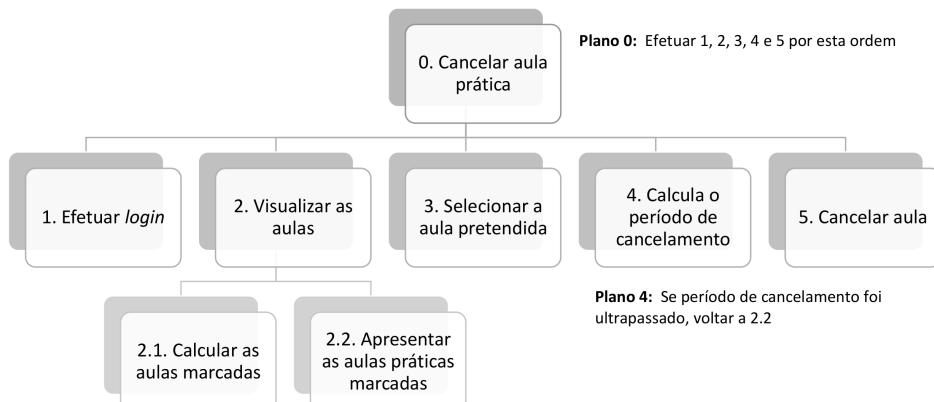


Figura 11: Tarefa *Cancelar Aula* realizada por um Aluno.

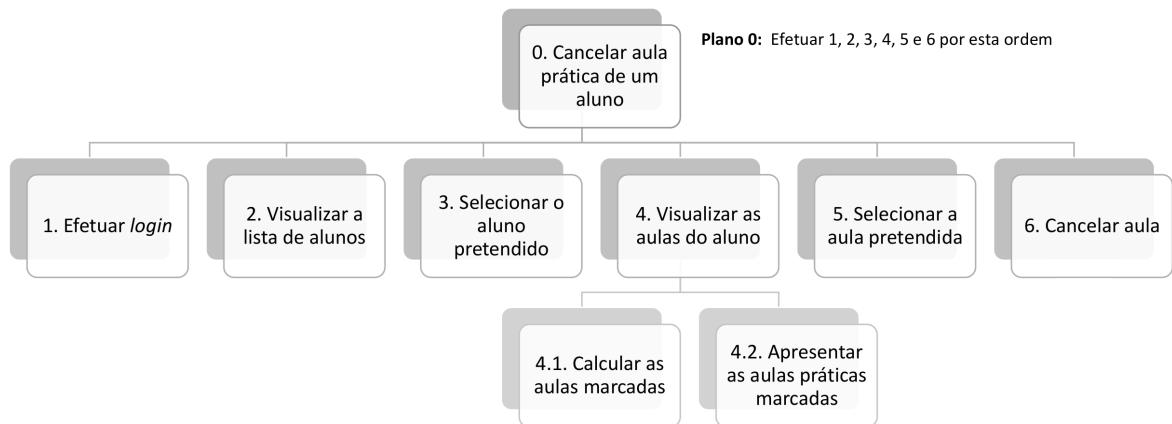


Figura 12: Tarefa *Cancelar Aula* realizada por um Secretário.

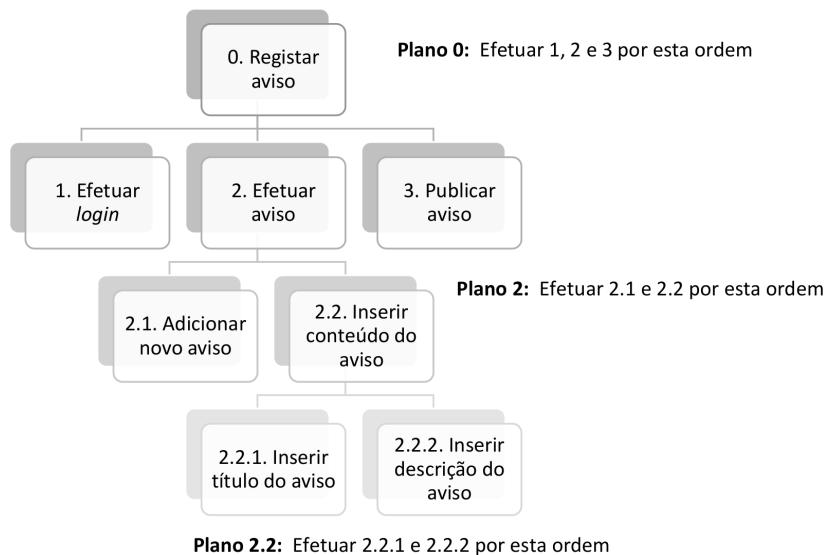


Figura 13: Tarefa *Registrar Aviso* realizada por um Secretário.

7 Mockups da aplicação

De forma a tomar decisões sobre a forma como a interface iria ser desenvolvida, optamos pela definição de *mockups* numa fase inicial do problema.

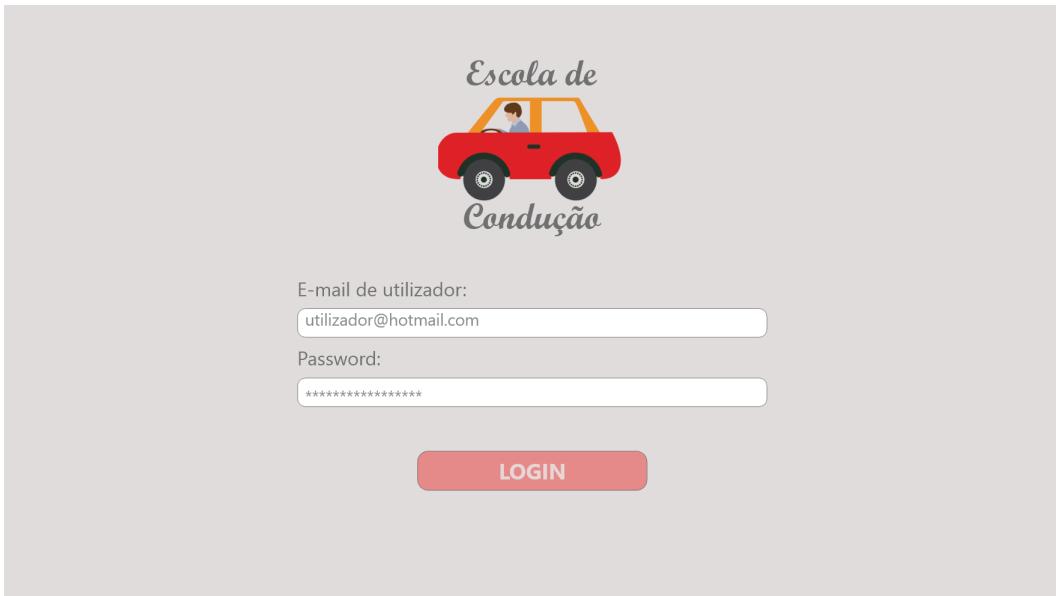


Figura 14: Página de autenticação do utilizador.

De seguida são apresentados alguns *mockups* desenvolvidos para o perfil de aluno na aplicação.

7.1 Aluno

This mockup shows the student dashboard after logging in. At the top, there is a navigation bar with icons for Home, Aulas, and Perfil, and a user profile section showing "utilizador@hotmail.com" and "Logout". The main content area is divided into several sections: "Avisos pessoais recentes" (with items like "Exame prático marcado" and "Aula prática cancelada"), "Avisos gerais recentes" (with items like "Escola fechada" and "Aula cancelada"), and "Próximos eventos" (with items like "Aula Prática - Categoria B" and "Exame prático - Categoria B"). Each item in these lists includes a timestamp and a small eye icon for viewing details.

Figura 15: Página inicial de Aluno depois de autenticado.

The screenshot shows the student's profile page with the following sections:

- Próximas aulas marcadas** (Marked Classes):
 - Aula Prática - 12 / 01 / 2019, 14h30
 - Aula Prática - 14 / 01 / 2019, 17h00
 - Aula Prática - 14 / 01 / 2019, 18h00
- Aulas Teóricas**:
 - Realizadas: 18 / 24
 - [Consultar aulas teóricas realizadas e temas →](#)
 - [Consultar próximas aulas teóricas →](#)
- Aulas Práticas**:
 - Realizadas: 15 / 32
 - [Consultar aulas práticas realizadas e temas →](#)
 - [Marcar aulas práticas →](#)

Figura 16: Página relativa a aulas de Aluno.

The screenshot shows the consultation of theoretical classes with the following sections:

- Próximas aulas marcadas** (Marked Classes):
 - Aula Prática - 12 / 01 / 2019, 14h30
 - Aula Prática - 14 / 01 / 2019, 17h00
 - Aula Prática - 14 / 01 / 2019, 18h00
- Aulas Teóricas**:
 - Realizadas: 18 / 24
 - [Consultar aulas teóricas realizadas e temas →](#)
 - [Consultar próximas aulas teóricas →](#)
- Aulas Práticas**:
 - Realizadas: 15 / 32
 - [Consultar aulas práticas realizadas e temas →](#)
 - [Marcar aulas práticas →](#)
- Table of Past Theoretical Classes** (Data, Tema):

Data	Tema
12/11/2018 14h30	Regra de rotundas
13/11/2018 14h30	Sinais de transito
14/11/2018 14h30	Mecânica
14/11/2018 17h30	Cedência de passagem
17/11/2018 17h30	Luzes
20/11/2018 17h30	Classificação de veículos
21/11/2018 14h30	Manobras
23/11/2018 17h30	Via pública
27/11/2018 14h30	Contraordenações

Figura 17: Consulta de aulas teóricas realizadas e temas.

The interface shows a navigation bar with Home, Aulas (selected), and Perfil. Below the navigation bar, there are two tabs: Cat. A1 and Cat. B. The main content area displays a table titled "Próximas aulas marcadas" (Scheduled next classes) with three entries:

Aula Prática	12 / 01 / 2019 14h30
Aula Prática	14 / 01 / 2019 17h00
Aula Prática	14 / 01 / 2019 18h00

Aulas Teóricas
Realizadas: 18 / 24
[Consultar aulas teóricas realizadas e temas →](#)
[Consultar próximas aulas teóricas →](#)

Aulas Práticas
Realizadas: 15 / 32
[Consultar aulas práticas realizadas e temas →](#)
[Marcar aulas práticas →](#)

To the right, there is a table titled "Já dado?" (Given?) with the following data:

Dia	Hora	Tema	Já dado?
03/01/2019	14h30	Contraordenações	✓
	17h30	Velocidades	✗
04/01/2019	14h30	Segurança ativa/passiva	✗
	17h30	Manobras	✓
05/01/2019	17h30	Mecânica	✓
08/01/2019	14h30	Categorias de veículos	✗
	17h30	Ultrapassagem	✗
09/01/2019	14h30	Regra de rotundas	✓
10/01/2019	14h30	Habilidades	✗

Figura 18: Consulta de horários das próximas aulas teóricas.

The interface shows a navigation bar with Home, Aulas (selected), and Perfil. Below the navigation bar, there are two tabs: Cat. A1 and Cat. B. The main content area displays a table titled "Próximas aulas marcadas" (Scheduled next classes) with three entries:

Aula Prática	12 / 01 / 2019 14h30
Aula Prática	14 / 01 / 2019 17h00
Aula Prática	14 / 01 / 2019 18h00

Aulas Teóricas
Realizadas: 18 / 24
[Consultar aulas teóricas realizadas e temas →](#)
[Consultar próximas aulas teóricas →](#)

Aulas Práticas
Realizadas: 15 / 32
[Consultar aulas práticas realizadas e temas →](#)
[Marcar aulas práticas →](#)

To the right, there is a table titled "Tema" (Topic) with the following data:

Data	Tema
11/01/2019 10h00	Iniciação à condução
12/01/2019 18h00	Mudanças
15/01/2019 13h00	Estacionamento
16/01/2019 11h00	Manobras
17/01/2019 17h00	Prática em rotundas
20/01/2019 15h00	Condução em auto-estrada
20/01/2019 16h00	Estacionamento e manobras
21/01/2019 9h00	Condução em auto-estrada
23/01/2019 14h00	Prática em rotundas

Figura 19: Consulta de aulas práticas realizadas e temas.

The figure consists of three vertically stacked screenshots of a web-based application interface, likely a student management system. The top two screenshots show 'Próximas aulas marcadas' (Scheduled Classes) and the bottom one shows 'Aulas disponíveis' (Available Classes). Each screenshot includes a header with navigation icons (Home, Aulas, Perfil), user information (utilizador@hotmail.com, Logout), and category filters (Cat. A1, Cat. B).

Screenshot 1: Próximas aulas marcadas (Top)

- Aulas Práticas:**
 - 12 / 01 / 2019, 14h30
 - 14 / 01 / 2019, 17h00
 - 14 / 01 / 2019, 18h00
- Aulas Teóricas:** Realizadas: 18 / 24
 - [Consultar aulas teóricas realizadas e temas →](#)
 - [Consultar próximas aulas teóricas →](#)
- Aulas Práticas:** Realizadas: 15 / 32
 - [Consultar aulas práticas realizadas →](#)
 - [Marcar aulas práticas →](#)

Screenshot 2: Próximas aulas marcadas (Middle)

This view is identical to the first, showing the same scheduled practical classes and links for theoretical and practical classes.

Screenshot 3: Aulas disponíveis (Bottom)

- Aulas Práticas:** Realizadas: 15 / 32
 - [Consultar aulas práticas realizadas →](#)
 - [Marcar aulas práticas →](#)
- Aulas disponibilizadas:**

Horas	Marcar
9h00 → 10h00	→
10h00 → 11h00	→
12h00 → 13h00	→
14h00 → 15h00	→
15h00 → 16h00	→
17h00 → 18h00	→

Figura 20: Marcação de aulas práticas.

The screenshot shows a student profile page with the following details:

- User Information:** Home icon, Aulas icon, Perfil icon.
- User Profile:** Placeholder image, name "António Silva".
- Course Category:** Categoria(s) do curso: Categoria B.
- Registration Dates:** Inscrição a: 20/10/2018, Validez até: 20/11/2020.
- Attendance:** Aulas teóricas realizadas: 18 / 24, Aulas práticas realizadas: 15 / 32.
- Instructor:** Instrutor atual: Renato Coelho.
- Exams:** Registo de exames: 21 / 12 / 2018 → Exame Teórico, 10 / 02 / 2019 → Exame Prático.
- Payments:** Faturas table showing three entries:

Data limite	Pago?	Descrição	Valor
23/10/2018	✓	Inscrição na categoria B	78,00 €
23/10/2018	✓	Cat. B - pagamento	120,00 €
23/10/2018	✓	Cat. B - pagamento	100,00 €

 Total: 298,00 € / 798,00 €

Figura 21: Página do perfil do aluno.

8 Tecnologias e *frameworks* utilizadas

Durante a fase inicial do projeto, o grupo decidiu que tecnologias seriam utilizadas na fase de desenvolvimento e produção desta aplicação. Em baixo, apresentamos a stack tecnológica da aplicação desenvolvida.

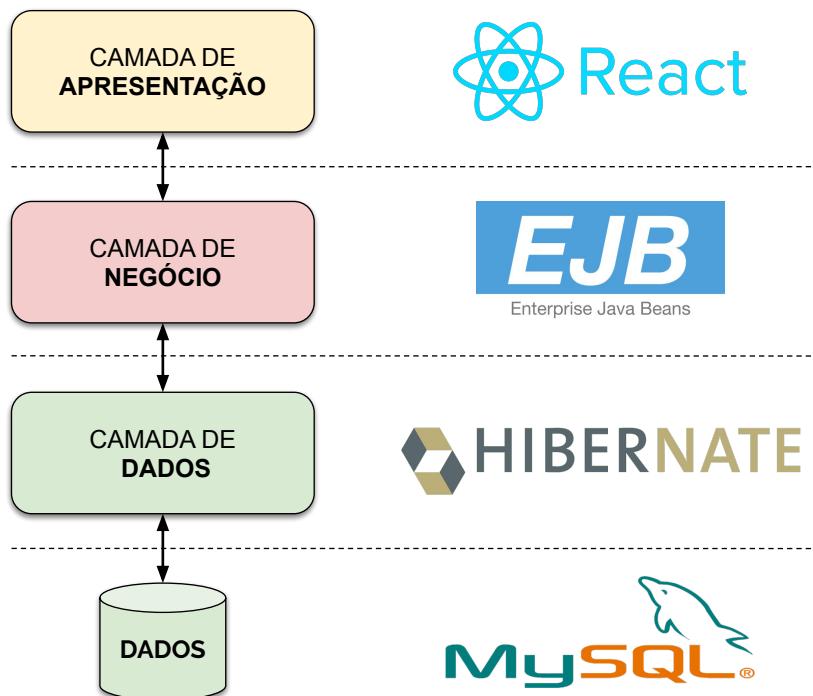


Figura 22: Stack tecnológica da aplicação desenvolvida.

8.1 Camada de dados

Relativamente à camada de dados, optamos pela utilização do **MySQL** para a persistência de dados, no entanto reconhecemos que existem outros motores de gestão de base de dados que poderão fornecer outras funcionalidades ou melhorias de desempenho, como é o caso do *PostgreSQL*. Seria interessante no futuro, verificarmos se obtemos melhores tempos de resposta com a utilização deste SGDB.

Aliado a este Sistema de Gestão de Base de Dados, utilizamos o **Hibernate**, uma framework que permite o mapeamento de objecto Java em dados relacionais.

Para facilitarmos o mapeamento de POJOS em dados relacionais, recorremos à funcionalidade do **Visual Paradigm** que permite a geração de código automaticamente. Este processo facilitou todo o desenvolvimento da aplicação através da criação de DAO's. Deste modo não tivemos de nos preocupar com a escrita de anotações ou a escrita de texto XML.

Para além destas tecnologias, foi ainda utilizado o **Redis**. O Redis é um serviço key-value que permite o armazenamento e procura de informação de uma forma muito rápida.

Apesar de ser utilizado, geralmente, para *caching*, neste projeto o Redis tem a funcionalidade de armazenar os tokens.



das sessões dos utilizadores e, por fim, de armazenar o número de pessoas que estão a ver uma determinada aula prática. No contexto aplicacional, pode afetar o desempenho geral da aplicação, uma vez que é adiciona carga ao serem estabelecidas conexões e ser realizado um pedido ao *Redis* sempre que é necessário verificar se um *token* existe. Por outro lado, o facto de armazenarmos no *Redis* os *tokens* e o número de pessoas a visualizar uma dada aula, permite que consigamos diminuir a carga na camada de dados e, consequentemente, libertar o número de conexões existentes a base de dados.

Assim, a integração do *Redis* parece ser uma óptima escolha e permite que conheçamos novas tecnologias que possam ser úteis no futuro.

8.2 Camada de negócio

No que concerne à camada de negócio, utilizamos o **Java Enterprise Edition**, que nos permitiu abstrair muito dos problemas relacionados com a concorrências de objetos, gestão de conexões, gestão de pedidos *HTTP*, entre outros. Deste modo, foram utilizados os *Enterprise Java Beans* e os *Servlets*, essenciais para a fácil implementação da aplicação.

No que concerne aos *Enterprise Java Beans*, é importante referir que todos os *Beans* presentes na nossa aplicação são *stateless*, pois não existe necessidade de manter estado durante uma dada sessão.

Relativamente ao comportamento da aplicação, é importante mencionarmos que o nosso *backend* foi desenvolvido de forma a fornecermos uma **API REST**. As razões pela qual optamos por esta metodologia são apresentadas em baixo:

- não pretendíamos, de todo, que o conteúdo estático fosse gerado no lado do servidor e enviado para o cliente; isto permite reduzir a carga do servidor aplicacional;
- a integração de tecnologias e frameworks relacionadas com o desenvolvimento *Web*, seria muito difícil ou até mesmo impossível;
- fornecermos uma *API REST* é uma mais valia para, mais tarde, podermos implementar uma interface diferente da *Web* muito facilmente, por exemplo: desenvolvimento de uma aplicação *mobile*;

Deste modo, é clara a decisão em separarmos a responsabilidade de geração de conteúdo *Web* no lado do servidor aplicacional.

Para que seja possível fornecermos uma *API*, é necessário, ao invés de retornarmos conteúdo *HTML*, muito pesado, retornarmos informação num formato menos verboso, como é o caso do *JSON*. Para isso utilizamos a biblioteca **Jackson**¹, que nos permitiu serializar objetos *Java* para *JSON* facilmente.

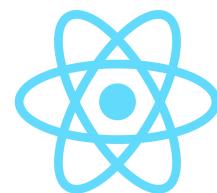
Relativamente às conexões com o serviço *Redis*, foi necessário pensarmos numa forma de fornecermos uma *pool* de conexões. Com isto, é possível que sejam poupanças conexões ao *Redis* e que não seja necessário estabelecer uma nova conexão por cada pedido efetuado por um cliente (devido à verificação do *token* do utilizador). A

¹<https://github.com/FasterXML/jackson>

solução passou pela utilização de um *Enterprise Java Beans* do tipo ***Singleton***, que inicia no arranque da aplicação e mantém-se ativo durante o tempo de vida da mesma. Este *Bean* é único e inclui uma *pool* de conexões ao *Redis* e um conjunto de métodos que permitem a manipulação dos dados. Para a criação da *pool* e acesso ao serviço *Redis*, foi utilizada a biblioteca ***Jedis***².

8.3 Camada de interface

Por fim, na camada de interface optamos pela utilização de ***React js***. Esta opção deveu-se à grande utilização do mesmo nos dias de hoje para construir aplicações Web e desta forma aproveitarmos para aprender a utilizar o mesmo. A juntar a isto, o *React* tem ainda uma grande documentação e suporte *online*, um vasto leque de *frameworks* e bibliotecas que fornecem componentes já feitos e estilização de componentes de acordo com os padrões atuais de desenvolvimento de interfaces Web, e, a juntar a isto, esta tecnologia permite a reutilização de componentes construídos ao longo da aplicação.



Como ajuda no desenvolvimento da interface, utilizamos a biblioteca ***Semantic UI React***³ que fornece muitos tipos de componentes utilizados ao longo da aplicação como *Card*, *Segment*, *Form*, entre muitos outros.



Sendo assim, esta interface consome a ***API REST*** mencionada anteriormente. Assim que o *login* de um utilizador é realizado, o *token* que o *backend* devolve é guardado em memória de forma a ser utilizado em todos os pedidos seguintes.

²<https://github.com/xetorthio/jedis>

³<https://react.semantic-ui.com>

9 Implementação e algumas considerações

Em baixo explicamos com algum detalhe algumas considerações no desenvolvimento do projeto, tendo em conta as tecnologias e *frameworks* mencionadas anteriormente.

9.1 Funcionalidade extra

Ainda que não tenha sido definido nos requisitos da aplicação, o grupo decidiu integrar uma funcionalidade extra que permite uma melhor interação utilizador-aplicação.

Esta funcionalidade, já referida anteriormente, consiste, durante a fase de marcação de uma aula, o utilizador conseguir perceber o número de pessoas a visualizarem uma dada aula num determinado horário, como podemos observar na figura em baixo.

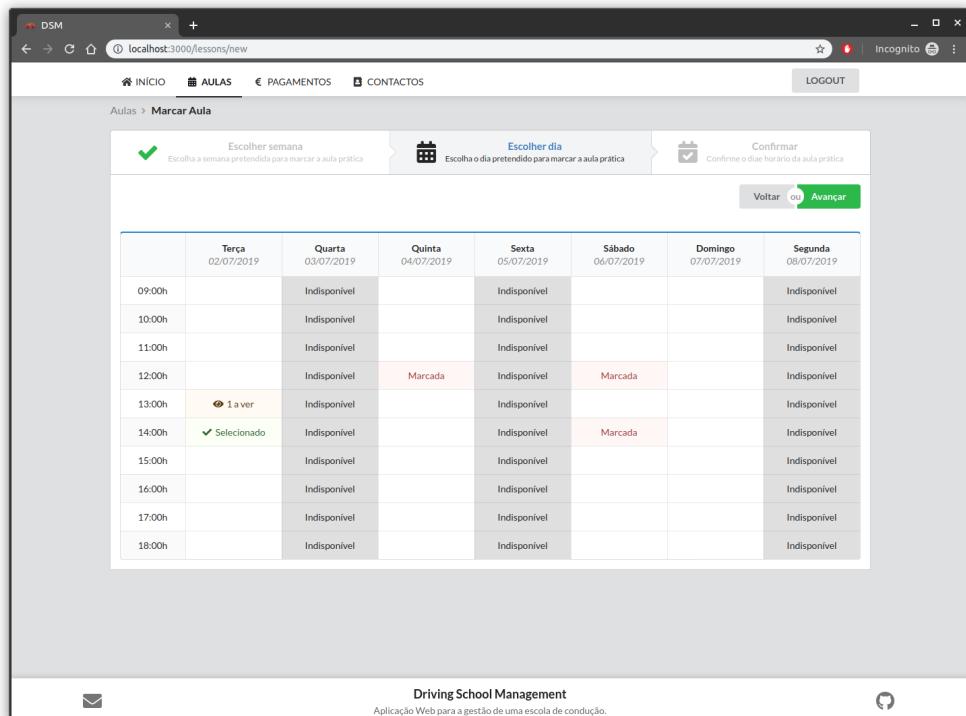


Figura 23: Processo de marcação de aulas com a funcionalidade extra

Esta funcionalidade foi conseguida com a utilização do *Redis*, deste modo tiramos da base de dados toda a carga associada a este processo.

Sempre que um utilizador avança para o passo 3 (ver figura em cima) é incrementado o valor do número de visualizações atuais sobre essa aula. No caso de não existir uma entrada para essa aula esta é criada. Quando o utilizador conclui a marcação ou a cancela, é decrementado o valor.

A chave de entrada no *Redis* é composta pelo *id* do instrutor e pela hora da aula. Isto acontece porque ainda não existe a aula e não existe o conceito de identificador para uma aula inexistente.

Uma vez que um utilizador pode fechar o browser sem retroceder ou concluir a marcação, foi adicionado um *TTL* de 3 minutos à chave introduziada no *Redis*, mas evitar futuros utilizadores inativos/fantasmas.

9.2 Serviço externo

Fomos desafiados a incluir um serviço externo no *backend* da nossa aplicação.

Inicialmente, ponderamos a possibilidade de os utilizadores se autenticarem através de um serviço como a Google, Facebook, etc. Para isso utilizaríamos o mecanismo de autenticação **OAuth 2.0**. No entanto, dado o espaço curto de tempo para a implementação deste projeto e, ao facto de termos de estudar este mecanismo, optamos por utilizar outro tipo de serviço externo.

Assim, optamos por fornecer e integrar na nossa aplicação um serviço de meteorologia. Deste modo, os utilizadores podem consultar a meteorologia para os 5 dias mais atuais. O **Instituto Português do Mar e da Atmosfera** fornece uma API⁴ que fornece até 5 dias com dados relativos à temperatuda máxima e mínima esperada, probabilidade de precipitação e ainda, indicação do estado atual (sol, nublado, etc).

Este serviço podia ser facilmente incluído no *frontend*, e para isso existem *plugin* ou bibliotecas *javascript* que tratam facilmente deste serviço. No entanto, optamos por o incluir no nosso servidor *backend*, enviar para o *frontend* e, só depois, apresentar no cliente. Este esquema é apresentado em baixo.

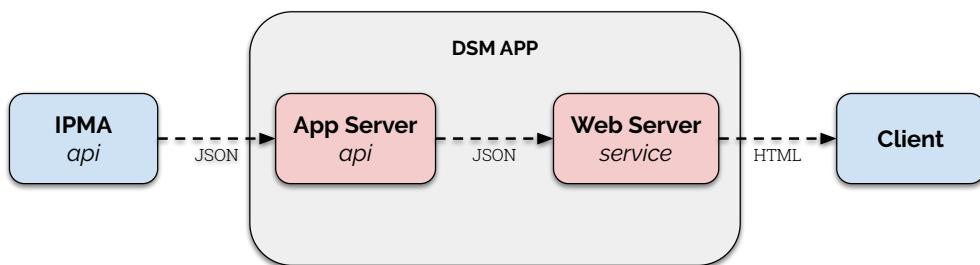


Figura 24: Fluxo dos dados relativos à meteorologia

Em baixo apresentamos um exemplo do que um utilizador visualiza em alguns dos ecrãs da nossa aplicação.

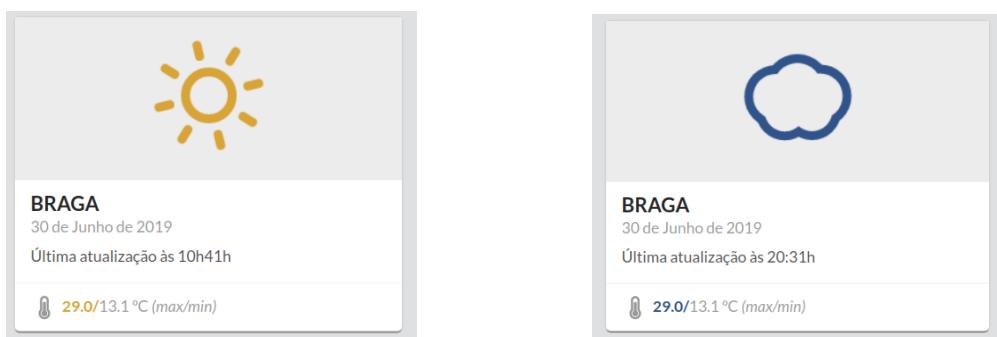


Figura 25: Exemplo no lado do cliente dos dados recebido pela API do IPMA

⁴<https://api.ipma.pt/#services>

10 Interface Web final

A interface desenvolvida para a aplicação tem um grande cuidado em adequar-se ao contexto e aos utilizadores da mesma, tendo ainda um cuidado para que os ecrãs desenvolvidos na mesma sejam responsivos aos vários tamanhos de ecrãs que possa encontrar.

Um exemplo concreto desta responsividade é o facto do menu de navegação se redefinir com base nas alterações do tamanho do ecrã, como apresentado de seguida.

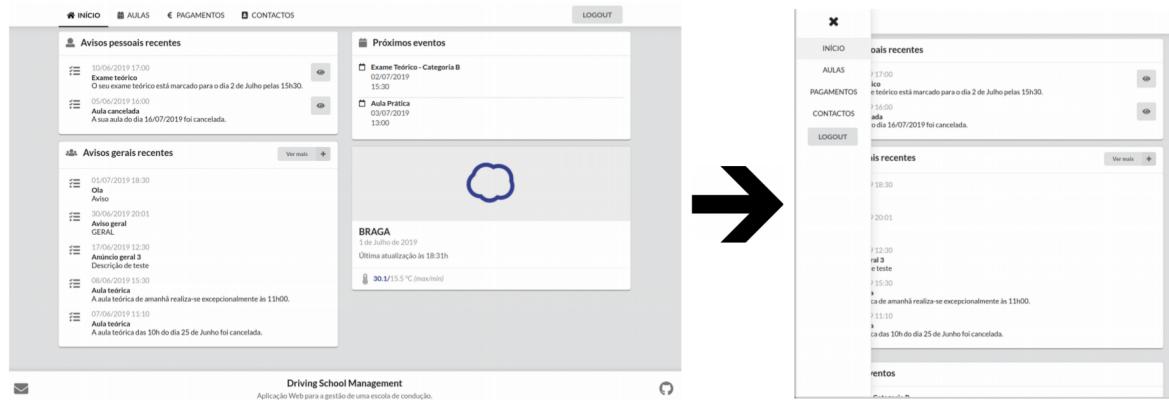


Figura 26: Exemplo de responsividade da aplicação.

De seguida é feita uma avaliação da usabilidade da interface desenvolvida.

10.1 Técnicas de usabilidade utilizadas

Ao longo do desenvolvimento da aplicação Web, foi importante seguir algumas heurísticas para garantirmos uma melhor experiência de utilização da aplicação. Deste modo, o grupo considerou as **Heurísticas de Nielsen**:

- Utilizar diálogo simples e natural;
- Falar a linguagem dos utilizadores;
- Minimizar a carga de memória;
- Ser consistente;
- Fornecer feedback;
- Fornecer saídas claramente assinaladas;
- Fornecer atalhos;
- Fornecer boas mensagens de erro;
- Prevenir erros;

Com isto, pretendemos mostrar de seguida exemplos concretos em que provamos a utilização destas heurísticas de forma a tornar a experiência do utilizador mais positiva.

De forma a **mostrar o estado da aplicação** ao utilizador, optamos por mostrar os momentos em que aplicação está a carregar dados obtidos do *backend*. Sendo assim, utilizamos um **Loader** para mostrar estas situações.

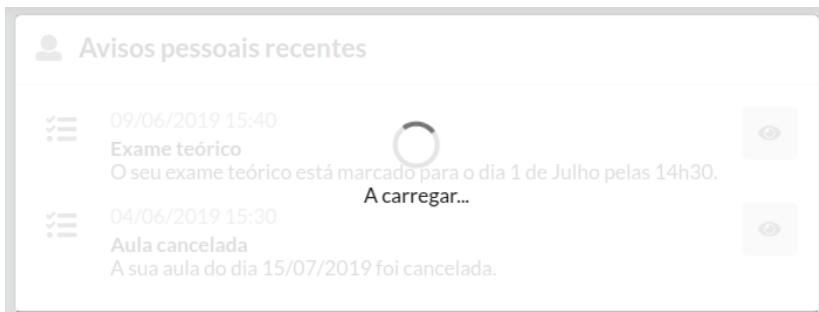


Figura 27: Fornecer informação sobre o estado da aplicação.

Com o intuito de **fornecer atalhos e saídas** ao utilizador, usamos um **Header** que permite uma fácil navegação pelas várias páginas fornecidas, assim como a presença de um botão que permite fazer *logout* a partir de qualquer ecrã. Para além disto, utilizamos o padrão **breadcrumb** para que o utilizador **perceba onde está** e o **percurso realizado** até ao estado atual, permitindo ainda voltar a um ponto anterior.



Figura 28: Fornecer informação sobre o estado da aplicação (menu de navegação).



Figura 29: Fornecer informação sobre o estado da aplicação (*breadcrumb*).

No processo que poderemos considerar o mais complexo e fulcral da aplicação, a marcação de uma aula prática, optamos pela utilização de uma **barra de progresso** com os passos que é suposto o utilizador realizar para concluir o processo. Desta forma, é facilmente perceptível pelo utilizador quais os passos realizados pelo mesmo assim como os que faltam realizar.



Figura 30: Etapas do processo de marcação de uma aula.

No caso de se pretender cancelar uma aula prática de um aluno, optamos por **pedir confirmação da operação a realizar**, tendo em conta a importância da con-

sequência desta ação, pois o aluno se por engano cancela uma aula pode não conseguir mais tarde recuperar a mesma. Sendo assim, é utilizado um **Modal** que pergunta ao utilizador se pretende avançar com a ação selecionada.



Figura 31: Confirmação do cancelamento de uma aula prática.

Foi ainda tido em conta o cuidado de **fornecer feedback** ao utilizador das ações que o mesmo executa. Sendo assim, é mostrada informação sobre o sucesso ou não de determinada operação. Um exemplo concreto de uma mensagem de sucesso pode passar pelo momento em que um secretário da escola de condução consegue publicar um aviso geral com sucesso.



Figura 32: Aviso geral publicado com sucesso.

Já quanto a mensagens de erro, foi tido o cuidado de **fornecer boas mensagens de erro** de forma a que o utilizador perceba o porquê da ação ter falhado. No caso apresentado de seguida, durante o processo de registo de um aluno por parte do secretário, é mostrado o porquê de o NIF do aluno estar incorreto. Para além disto, é ainda tido o cuidado de limitar o tamanho de campos como este ou o tipo de caracteres que se espera (neste caso, 9 dígitos).

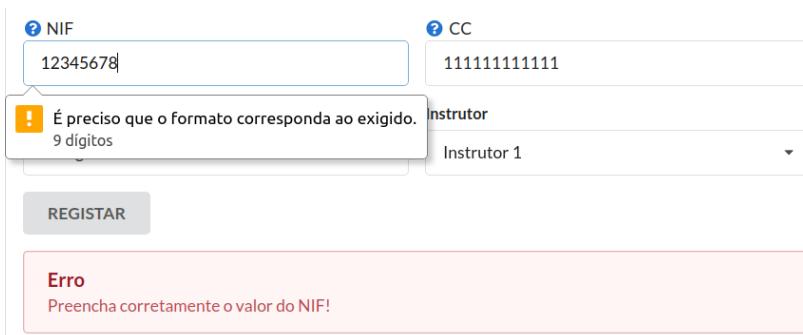


Figura 33: Valor do NIF do aluno não cumpre a pattern necessária.

No seguimento da discussão do NIF do aluno, assim como o CC do mesmo, foi decidido utilizar um componente de **ajuda** junto das *labels* do mesmo, de forma a explicar o significado do valor pedido assim como o formato esperado. Desta forma, com o passar do rato sobre o *icon* de ajuda a informação é transmitida ao utilizador.

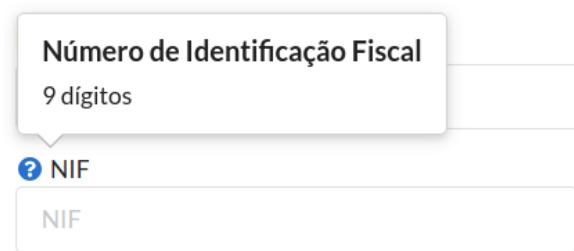


Figura 34: Fornecer ajuda sobre certos valores pedidos.

Ainda assim, e de forma a **prevenir** erros em funcionalidades que envolvem o preenchimento de formulários, não é permitido realizar submissão do mesmo até que todos os campos estejam preenchidos. Desta forma, o botão de *submit* é colocado como *disabled* até que os campos sejam preenchidos.

A screenshot of a form titled "Registrar aviso geral". It has two input fields: "Titolo do aviso" containing "Aviso geral" and "Descrição do aviso" containing "Descrição". Below the fields is a button labeled "PUBLICAR". The "PUBLICAR" button is grayed out, indicating it is disabled.

Figura 35: Impedir o *submit* de formulários até que os campos sejam preenchidos.

Por fim, pretendemos manter uma **consistência** no que diz respeito à imagem e ao tipo de segmentos utilizados na aplicação, assim como tentar utilizar uma **linguagem simples e adequada** ao contexto inserido.

A screenshot of the "Driving School Management" application. The top navigation bar includes links for "ALUNOS", "REGISTRAR AVISO", "CONTACTOS", and "LOGOUT". Below the navigation is a breadcrumb trail: "Alunos > João Vieira". There are three buttons: "Clique para editar", "Marcar Aula", and "Gerir". The main content area displays "Dados de João Vieira" with the following fields:

- Nome: João Vieira
- Data de nascimento: 03/05/1997
- E-mail: joao@gmail.com
- Morada: Braga
- NIF: 333333333 (with a question mark icon)
- CC: 333333333333 (with a question mark icon)

A "GUARDAR" button is located at the bottom of the form.

Figura 36: Aparência da aplicação.

10.2 Secretário cancela aula prática de aluno

Consideramos importante nesta fase de descrição da interface da aplicação, mostrar o exemplo de uma tarefa que segue o diagrama de tarefas apresentado anteriormente no relatório, neste caso, a ação de cancelar uma aula prática de um aluno por parte de um secretário:

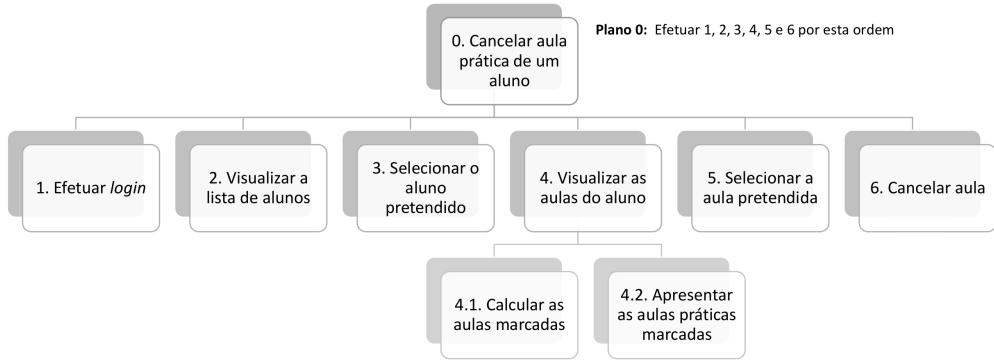


Figura 37: Tarefa *Cancelar Aula* realizada por um Secretário.

Assim, quando um secretário faz *login* na aplicação é-lhe apresentada a lista dos alunos inscritos na escola. O mesmo escolhe o aluno pretendido e ao clicar em **Ver perfil** acede ao perfil do aluno. De seguida, e no segmento que contém funcionalidades relativas à gestão do aluno encontra-se a opção **Próximas aulas**. Na página apresentada depois contêm informação sobre as aulas marcadas do aluno e as que o secretário pode cancelar. Ao clicar em **Cancelar** na aula pretendida o utilizador é questionado sobre se pretende mesmo realizar aquela operação (como mostrado em cima). Em caso afirmativo, a aula será cancelada. Os passos gerais são apresentados de seguida.

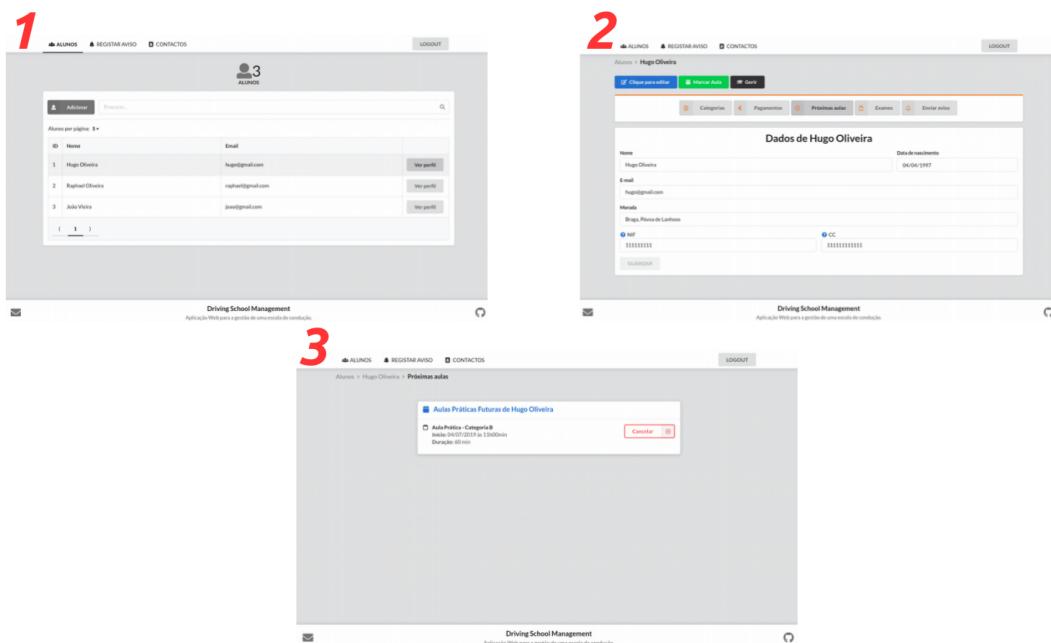


Figura 38: *Cancelar Aula* realizada por um Secretário através da aplicação.

11 Arquitetura adotada e Deployment

No que concerne à arquitetura adotada o grupo decidiu colocar apenas um servidor web a fornecer o conteúdo estático. Para este efeito optamos pelo *Nginx*.

Para podermos suportar mais carga no sistema, decidimos adicionar dois servidores aplicacionais *Wildfly*. Para fazermos o balançamento da carga optamos pela utilização do *HAProxy* que permite um balanceamento *roundrobin* e a deteção de falhas dos servidores aplicacionais.

Relativamente ao sistema de gestão de base de dados *MySQL*, decidimos correr o mesmo numa máquina à parte para poder utilizar os seus recursos na totalidade.

Por fim, adicionamos uma máquina a correr o *Redis*.

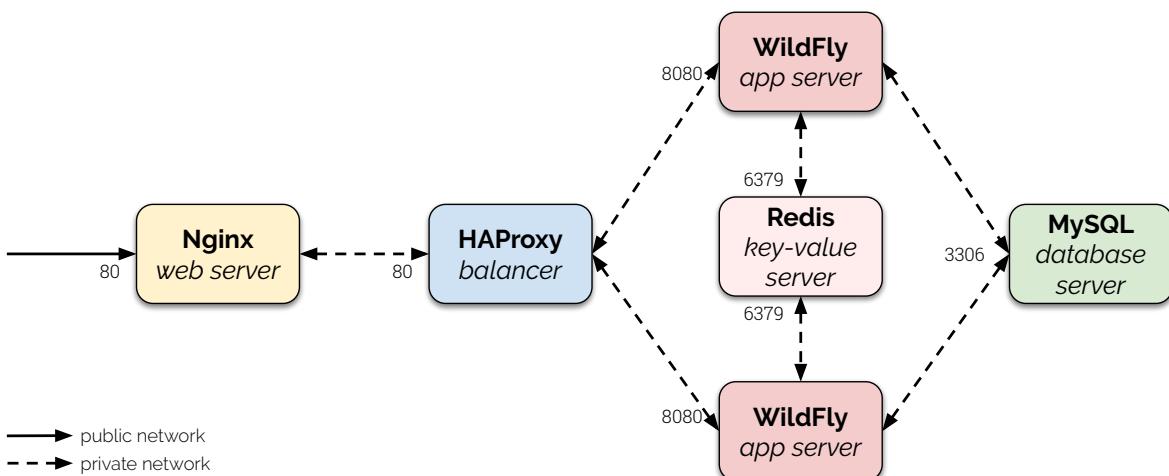


Figura 39: Arquitetura adotada para o deployment da aplicação

11.1 Fase de desenvolvimento

Durante a fase de desenvolvimento foi utilizado o computador pessoal de cada elemento do grupo.

Relativamente ao desenvolvimento do *backend*, cada computador tinha instalado um serviço de base de dados relacional e um servidor aplicacional. Juntamente com estes serviços, foi adicionado um **container** com o *Redis*. Para a gestão deste *container* utilizamos o **Docker**.

No que concerne ao desenvolvimento do *frontend*, utilizamos o próprio serviço do *Create React App* que permite fornecer conteúdo estático através do *localhost* na porta 3000, deste modo foi possível irmos fazendo *debug* à medida que íamos desenvolvendo o projeto.

11.2 Fase de produção

Para a fase de produção o grupo recorreu às máquinas virtuais da **Google Cloud**. Desta forma foi possível obtermos máquinas com melhor performance, ao invés dos containers ou máquinas virtuais nas máquinas pessoais.

Para facilitarmos todo o *deployment* da aplicação, o grupo recorreu à ferramenta **Ansible** que permite a criação de tasks que podem ser facilmente replicadas pelos vários servidores existentes. Isto permite agilizar todo o processo de configuração e gestão das máquinas.

Outra vantagem na utilização do *Ansible* é que nos foi possível destruir e criar máquinas facilmente e colocar o serviço novamente em pé, no espaço de um único comando:



```
ansible-playbook playbook.yml -u <google cloud user>
```

11.2.1 Especificações das máquinas utilizadas

Relativamente às máquinas virtuais da *Google Cloud* o grupo decidiu não escolher as mesmas especificações para todas elas. Existem serviço que requerem mais *RAM* e *CPU* que outros, como é o caso dos servidores aplicacionais e a base de dados.

Na tabela em baixo, apresentamos as especificações de cada máquina. De notar que:

- **n1-standard-2**: simboliza 2 vCPUs e 7.5 GBytes de memória *RAM*;
- **n1-standard-4**: simboliza 4 vCPUs e 15 GBytes de memória *RAM*

	Especificações Google Cloud	Endereço IP (Rede Interna)
Nginx	n1-standard-4	10.128.0.2
HAProxy	n1-standard-2	10.128.0.3
WildFly 1	n1-standard-4	10.128.0.4
WildFly 2	n1-standard-4	10.128.0.5
Redis	n1-standard-2	10.128.0.6
MySQL	n1-standard-4	10.128.0.7

Tabela 2: Inventário do sistema

12 Testes de carga

No que concerne ao *benchmarking* da nossa aplicação, começamos por avaliar quais os melhores métodos e ferramentas para podermos colocar carga no sistema e podermos avaliar os resultados fornecidos. Nesse sentido, surgiram duas ferramentas muito utilizadas para medir a *performance* de um sistema ou aplicação. As duas ferramentas são **Locust** e **Apache JMeter**.

É preciso perceber que a escolha da ferramenta ideal irá depender sempre dos utilizadores e, neste contexto, uma boa ferramenta é aquela que permite um utilizador facilmente **criar, correr e analisar** testes de carga.

O Apache JMeter é uma ferramenta que oferece uma interface gráfica para a criação, execução e análise de testes. Já o Locust, apenas oferece uma interface para a iniciar a execução e para análise de testes, sendo que toda a criação de testes requerem que o utilizador tenha conhecimentos de programação em *Python*. Apesar de ambas as ferramentas permitirem testes simulando vários utilizadores, é preciso perceber que o Apache JMeter utiliza uma *thread* por cada "cliente" e, como todos sabemos, a criação de *threads* é um processo muito custoso em termos de processamento e memória, uma desvantagem muito importante. Em contrapartida, o Locust utiliza um paradigma orientado aos eventos permitindo que todos os pedidos pelos utilizadores sejam assíncronos, não existindo o custo de criação de *threads*.

Conforme o que foi explicado em cima e com base na análise de *reviews online*, consideramos que os testes de carga devem ser realizados com recurso ao Locust.

Nas próximas secções iremos explicar que tipos de testes foram realizados e que permitiram resolvemos alguns problemas que foram encontrados ao longo do *benchmarking*.



É importante mencionarmos que os nossos testes de carga foram efetuados diretamente ao **balanceador** de carga. Isto deve-se ao facto de ser desnecessário adicionarmos mais um ponto de congestionamento (o servidor web), além de mais, não teríamos acesso à nossa *API REST* fornecida pelos servidores aplicacionais.

Para que pudessemos eliminar possíveis problemas de latência (comunicação *ISP - Google Cloud*), decidimos incluir na nossa arquitetura uma nova máquina para *benchmarks*. Isto permitiu reduzirmos não só o tempo de resposta de uma query como também, utilizarmos uma máquina de teste mais potente que as presentes por cada um dos elementos do grupo.

12.1 Tipos de teste

O Locust permite que consigamos fazer testes com **diferentes utilizadores** e ainda que, para cada tipo de utilizador, fornecer um **peso na criação de utilizadores** (relativo ao número de clientes de cada tipo a correr em simultâneo) e um **peso nas tarefas**. Isto é, a nossa aplicação em produção terá mais utilizadores do tipo *Student* do que dos tipos *Secretary* ou *Instructor*, deste modo, será importante que nos nossos testes sejam criados mais utilizadores do tipo *Student* do que dos restantes tipos.

Além disso, dado que existem requisitos que serão mais utilizados que outros, é interessante que alteremos a probabilidade de esse requisito ser chamado. Para isso, recorremos à funcionalidade dos pesos (*weight*).

Relativamente ao aluno, percebemos que este irá consultar ativamente os avisos gerais e pessoais, as suas aulas práticas e as aulas teóricas disponíveis. Por outro lado, irá consultar os seus dados pessoais menos vezes bem como marcar aulas menos vezes. Em baixo apresentamos um esquema com os **pesos de execução** e as tasks efetuadas por um aluno nos testes de carga. Note que, o *Login* e *Logout* são executados no início e no fim de cada teste de carga por cada utilizador.

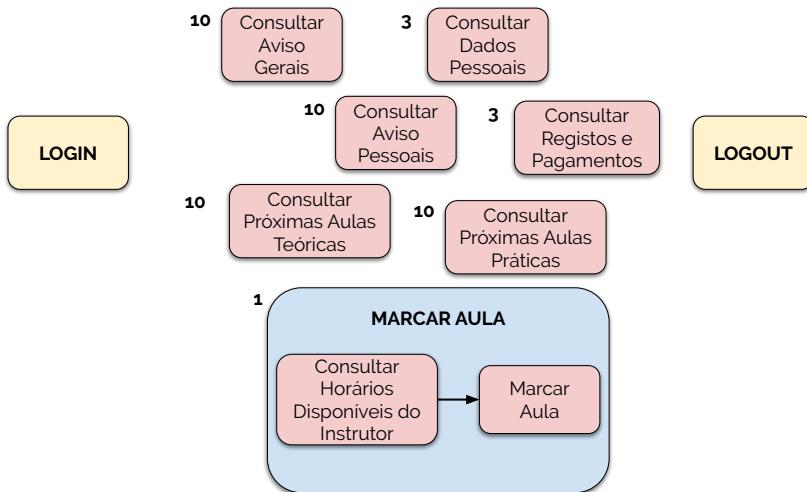


Figura 40: Pesos de execução para as tasks de um aluno

Relativamente ao instrutor, percebemos que este irá consultar ativamente os avisos gerais, os alunos a ele associado, as suas aulas teóricas e/ou práticas, consultar as aulas práticas futuras marcadas pelos alunos e aula teórica em aberto e, por fim, marcar algumas aulas práticas. Em baixo apresentamos os pesos associados às tasks efetuadas por um instrutor.

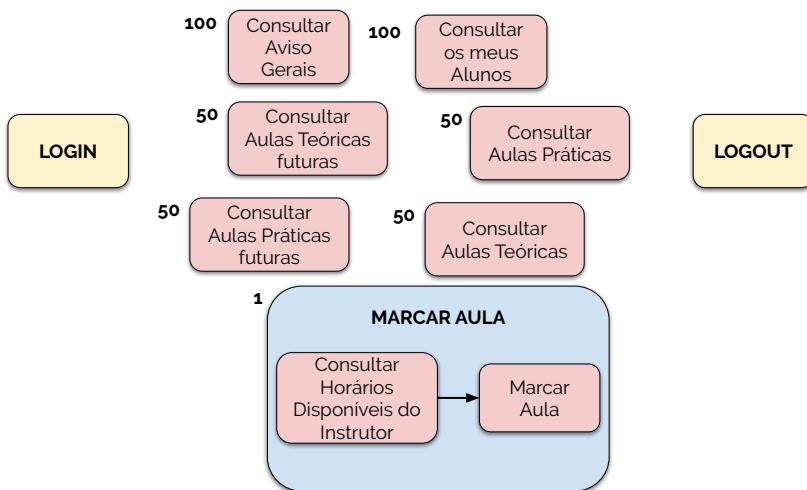


Figura 41: Pesos de execução para as *tasks* de um instrutor

Por fim, quanto ao secretário, e dado que é o utilizador que apresenta menos requisitos, apenas fizemos testes de carga que simulam o registo de exames de um aluno

e a marcação de uma aula prática. Em baixo apresentamos os pesos associadas às tasks de um secretário.

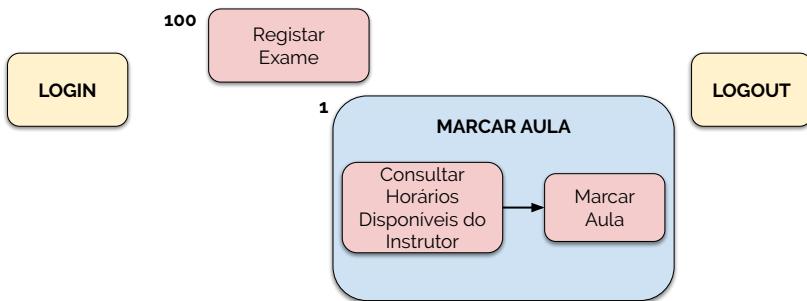


Figura 42: Pesos de execução para as tasks de um secretário

Uma vez que os utilizadores principais e que irão fornecer mais carga ao sistema são os alunos, decidimos dividir a criação de utilizadores da seguinte forma:

- alunos com fator 5;
- instrutores e secretários com fator 1, respetivamente;

Deste modo, para cada 100 utilizadores de teste, $5 * 100 / (7) = 70$ são do tipo aluno, 15 do tipo instrutor e 15 do tipo secretário. Deste modo acrescentamos mais realidade aos nossos testes de *benchmarking*.

Ao longo dos testes, foram realizados testes com 100, 200, 300 e 400 utilizadores simultâneos.

12.2 Resultados finais

Ao longo dos testes, foram detetados alguns problemas na aplicação que acabamos por solucionar:

- o *Enterprise Java Bean Redis*, uma vez que é do tipo *Singleton*, assume por *default* que todos os métodos têm obrigatoriamente um *lock* de escrita associado, pelo que tivemos de mudar para um *lock* do tipo de leitura, pois assim é possível vários clientes utilizarem o mesmo método para consulta e inserção de dados (não existe concorrência);
- como os nossos testes incluem a criação de aulas, ao fim de um determinado tempo de execução do *benchmarking*, reparámos numa degradação na *performance* da nossa aplicação. Isto é, todas as nossas consultas estavam a retornar todos os valores sem um limite associado; deste modo, optamos por introduzir um limite em todos os métodos de consulta do objeto *Lesson*;

O nosso objetivo final não consite em aumentar o número de utilizadores só porque achamos interessante. O nosso objetivo passa por fornecermos um serviço que consiga responder rapidamente ao maior número de clientes por segundo. Para isso, temos de encontrar uma平衡amento entre o número de utilizadores e os valores de *response time* e *request per second*.

De seguida, apresentamos alguns dos *outputs* obtidos consoante a alteração do número de utilizadores.



Figura 43: Benchmarking com 100 clientes

Como podemos observar na figura em cima, a nossa aplicação manteve-se constante durante todo o *benchmarking*, indicando que podemos aumentar a carga no sistema. Os valores de *response time* são os ideais e os esperados pelo grupo.

De notar também que o número de pedidos que o nosso sistema consegue tratar por segundo, é muito semelhante ao número de utilizadores de teste de carga.

Os testes efetuados para 200 utilizadores mostraram indicadores de que podemos acrescentar ainda mais carga ao sistema, pois os valores de *response time* mantiveram-se e a média de *requests per second* foi de 189 pedidos.

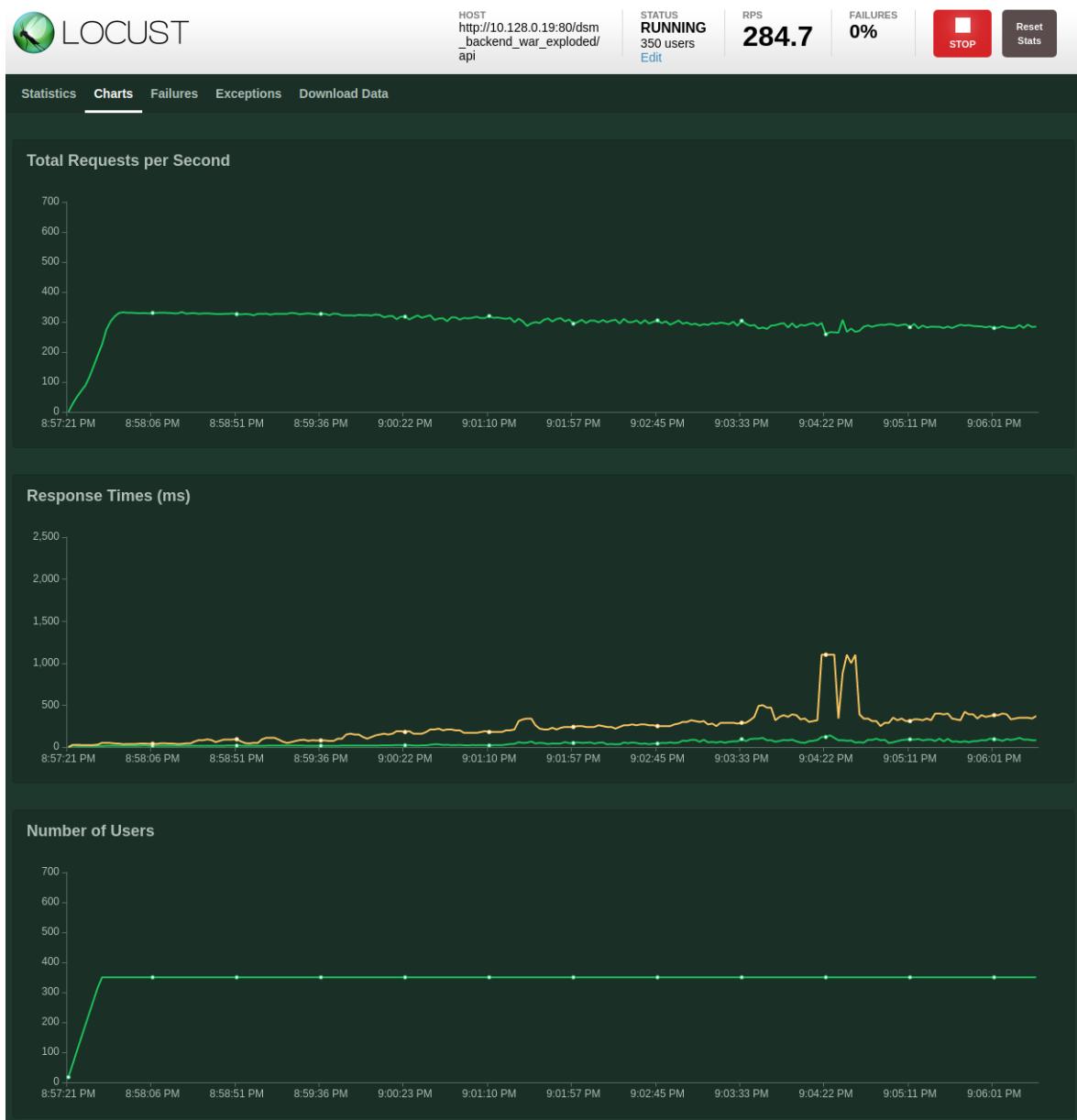


Figura 44: Benchmarking com 350 clientes

Realizamos ainda testes com 300 utilizadores. No entanto só verificamos uma subida nos tempos de *response time* para 30 ms e uma diminuição significativa do número de pedidos, em média de 270 *requests per second*.

Deste modo, efetuamos testes para 350 e 400 utilizadores respetivamente. Os gráficos associados aos testes com 350 utilizadores é apresentado em cima.

Como podemos verificar, à medida que os utilizadores vão aumentando, o número de pedidos por segundo tende a diminuir, indicando que estamos próximo do limiar de carga que pode ser aplicado à nossa aplicação.

Pela análise do primeiro gráfico da figura em cima, percebemos que o número de pedido por segundo vai diminuindo e os tempos de resposta aumentando. Para verificarmos que estamos perto dos limites, corremos os testes com 400 utilizadores. Ao fim de algum tempo, um dos servidores aplicacionais esgotou a memória disponível e parte dos pedidos foram perdidos (dada a existência de *timeouts*).

Com base nas informações em cima e com base no que o grupo pensa ser um bom serviço, concluímos que a nossa aplicação (ou sistema aplicacional) suporta até 300 utilizadores em simultâneo. Valores acima disso, podem levar a *timeouts* e erros da aplicação que, dada a falta de tempo de implementação deste projeto, fomos incapazes de ultrapassar ou fornecer medidas que combatem alguns dos problemas de carga.

De notar ainda que, ao longo destes testes não foram realizadas alterações nos parâmetros de conexões à base de dados (configuráveis no *Hibernate* e no *MySQL*). No *deployment*, definimos um máximo de 150 conexões na base de dados e 25 conexões por servidor aplicacional no ficheiro de configuração *Hibernate*.

Por fim, e como melhoria possível a estes testes de carga, consideramos que estes testes não se devem apenas limitar aos resultados obtidos pelo *Locust*. Seria interessante utilizarmos ferramentas de recolha de métricas dos diferentes serviços ou servidores. Isto permitia percebermos melhor qual o serviço ou servidor que está com mais carga para podermos, em caso de necessidade, tomar medidas quanto à escolha das características dos servidores. Para isso poderíamos utilizar o *Kibana* ou o *Grafana* e poderíamos utilizar o *Elasticsearch* para a recolha das métricas.

13 Conclusão

Ao longo do levantamento de requisitos e da modelação da aplicação, percebemos que tínhamos um projeto de média/grande dimensão e que incluía bastantes entidades. Com base nos dados recolhidos, decidimos não incluir na aplicação algumas funcionalidades que, provavelmente, poderiam fornecer uma melhor gestão de uma escola de condução. Deste modo, decidimos não incluir a gestão de veículos, a gestão de documentos de identidade e partes da gestão relacionada com a inserção de dados, como é o caso de instrutores e secretários. Acreditamos assim, que apesar destas funcionalidades não terem sido implementadas, não desvalorizam de algum modo todo o trabalho realizado nem comprometem a funcionalidade da aplicação.

No decorrer do desenvolvimento da aplicação, foram surgindo vários problemas relativos a exceções de sessões. A origem destas exceções surge, muito provavelmente, dos objetos criados durante a geração de código automático realizada no *Visual Paradigm*. Deste modo, a nossa solução passou por, ao invés de tentarmos lidar com as sessões de forma a reutilizarmos, deixar o servidor aplicacional lidar com este trabalho. Encontramos ainda outros problemas que dizem respeito à inserção de objetos em *run time*. Por vezes, após a inserção de objetos, estes desaparecem e não são apresentados no ecrã. Porém, após a reinicialização do servidor aplicacional, os mesmos aparecem. Não conseguimos encontrar uma solução mas acreditamos que o problema seja da *pool* de sessões existente.

Para além dos problemas descritos em cima, conseguimos identificar várias qualidades no nosso projeto. Desde já, a interface apresentada respeita um conjunto alargado de heurísticas que levam a uma melhor interação com a aplicação, proporcionando uma boa experiência de utilização. A interface é consistente durante os vários ecrãs e apresenta um conjunto de informações intermédias que levam à diminuição de erros por parte do utilizador. A integração do *Redis* no nosso projeto permitiu, não só, que o grupo tenha aprendido uma nova tecnologia, como também permitiu fornecer a funcionalidade de "X alunos a ver esta aula". Esta foi uma funcionalidade na qual estamos orgulhosos de ver realizada. É ainda importante mencionar as tecnologias utilizadas no *frontend* da aplicação. A utilização do *React* permitiu um fácil desenvolvimento da interface *Web* e permitiu ao grupo conhecer tecnologias muito utilizadas atualmente.

Concluímos assim, que dada a limitação de tempo que os elementos do grupo encontraram na realização deste projeto e todas as adversidades encontradas, todos os elementos do grupo vêm este trabalho como concluído e que respeita todos os objetivos propostos.