

# **Análise de Teste e Software**

Fase 1 - Projeto:UMer

José Dias  
(A78494)

Pedro Silva  
(PG38935)

Tiago Fraga  
(A74092)

14 de Outubro de 2018

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Gramática</b>	<b>4</b>
2.1	Regras . . . . .	4
2.2	Analizador Léxico . . . . .	5
<b>3</b>	<b>Teste do parser</b>	<b>6</b>
<b>4</b>	<b>Conclusão</b>	<b>8</b>

# Lista de Figuras

2.1	Analizador Léxico . . . . .	5
3.1	Login com Sucesso . . . . .	6
3.2	Registrar Condutor . . . . .	7

# Capítulo 1

## Introdução

Neste projeto de Análise e Teste de Software, pretendemos analisar e testar com grande detalhe o software desenvolvido pelos alunos sobre o projeto UMer. Assim, serão usadas técnicas e ferramentas que permitirão aferir da qualidade a aplicação desenvolvida pelos alunos.

Nesta 1ª Fase iremos avaliar a solução apresentada pelos alunos de POO, ou seja, vamos verificar se o software cumpre os requisitos definidos no seu enunciado. Esta solução deve ser executada utilizando uma grande volume de informação, e para tal foi nos disponibilizado um ficheiro de *log* que contém o registo de todas as operações realizadas pela UMer durante um certo período de tempo.

Para tal, foi necessário desenvolver um parser utilizando o sistema *ANTLR*, que é um gerador de parsers para Java, e integrar o parser no software da UMer de modo a executar todas as operações em batch.

## Capítulo 2

# Gramática

### 2.1 Regras

Para esta gramática foram definidas duas regras de produção, log e evento. O log, que foi fornecido no ficheiro inicial, apenas verifica a existência de eventos ou *end of file* (EOF). O evento é dividido em várias categorias, sendo elas: **login**, **logout**, **viajar**, **recusar viagem**, **solicitar**, **registar empresa**, **registar condutor**, **registar cliente**, **regista carro**, **registar carrinha**, **registar helicoptero**, e **registar mota**.

Para todos estes eventos, a ideia inicial, seria registar os dados em cada categoria e de seguida chamar os métodos, da classe *UMer*, correspondentes a essa categoria para que essa ação fosse registada com sucesso. Em quase todas as categorias conseguimos realizar o pretendido, em que localmente guardamos os dados necessários para chamar os métodos pretendidos com a exceção do utilizador que foi necessário guardar globalmente.

No entanto existem 3 categorias que não foi possível a utilização desta abordagem sendo elas: **viajar**, **recusar viagem** e **solicitar**. Esta abordagem não resultou nestes eventos pois não existem métodos definidos na classe *UMer* que, com a informação fornecida no ficheiro *log2.txt*, fosse possível a sua utilização. Um exemplo desta falha no evento **viajar** é aquando da primeira tentativa de realizar uma viagem, a base de dados ainda não contém veículos, o que faz com que seja lançada uma *NullPointerException*. Uma maneira de resolver esta situação seria se no método *doTripQueue* existisse uma condição para o caso de o veículo ser nulo.

O problema maior é no evento **solicitar** porque é neste evento que se criam e adicionam as viagens necessárias para os eventos **viajar** e **recusar viagem**, mas como para adicionar viagens é necessário que já existam veículos registados, a grande parte das viagens não são registadas pois aparecem antes do registo do primeiro veículo. Assim como não conseguimos adicionar viagens, também não conseguimos usar o evento **recusar viagem** para nos remover uma viagem.

## 2.2 Analisador Léxico

Nesta parte definimos quais os elementos léxicos usados na especificação das regras de produção da gramática definida. Estes elementos são os símbolos não-terminais, tais como nome, morada, empresa, etc, e os símbolos terminais, tais como LETRA, NUMERO, CHAR, etc.

A definição destes elementos léxicos pode ser visualizada na Figura 2.1.

```
matricula : NUMERO MENOS LETRA LETRA MENOS NUMERO ;
localizacao : NUMERO PONTO NUMERO VIRGULA NUMERO PONTO NUMERO ;
nome : (palavra)+;
morada : ((palavra_numero|CHAR|MENOS|PONTO)+)? VIRGULA NUMERO VIRGULA cod_postal;
cod_postal : NUMERO MENOS NUMERO (palavra_numero|MENOS|CHAR|'('|')')+ ;
nascimento : NUMERO MENOS NUMERO MENOS NUMERO ;
MAIL : [a-zA-Z_.-?0-9]+ '@' [a-zA-Z]+ '.' [a-zA-Z]+ '.'? [a-zA-Z]* ;
empresa : (palavra_numero)+;
palavra : (LETRA)+;
palavra_numero : (LETRA|NUMERO)+ ;
password : (LETRA|CHAR|NUMERO|PONTO|VIRGULA|PONTO_VIRGULA|MENOS)+;
LETRA : [a-zA-ZçÇ];
CHAR: [_'+?%$&*#@!/];
PONTO : [.];
VIRGULA : [,];
PONTO_VIRGULA : [;];
MENOS : [-];
NUMERO : [0-9]+;
WS : [ \t\r\n]+ -> skip ;
```

Figura 2.1: Analisador Léxico

## Capítulo 3

# Teste do parser

Para testar a solução que criamos para este problema decidimos usar dois métodos.

Um método de verificação foi utilizar *prints* na definição da gramática, no *parser*, para nos ser mais fácil verificar se os dados que estão no ficheiro *log* estavam a ser inseridos.

Outro método de verificação foi utilizar a interface gráfica, já inserida na aplicação, para realizar logins, logouts e tentar registar veículos e empresas que já tenham sido registadas.

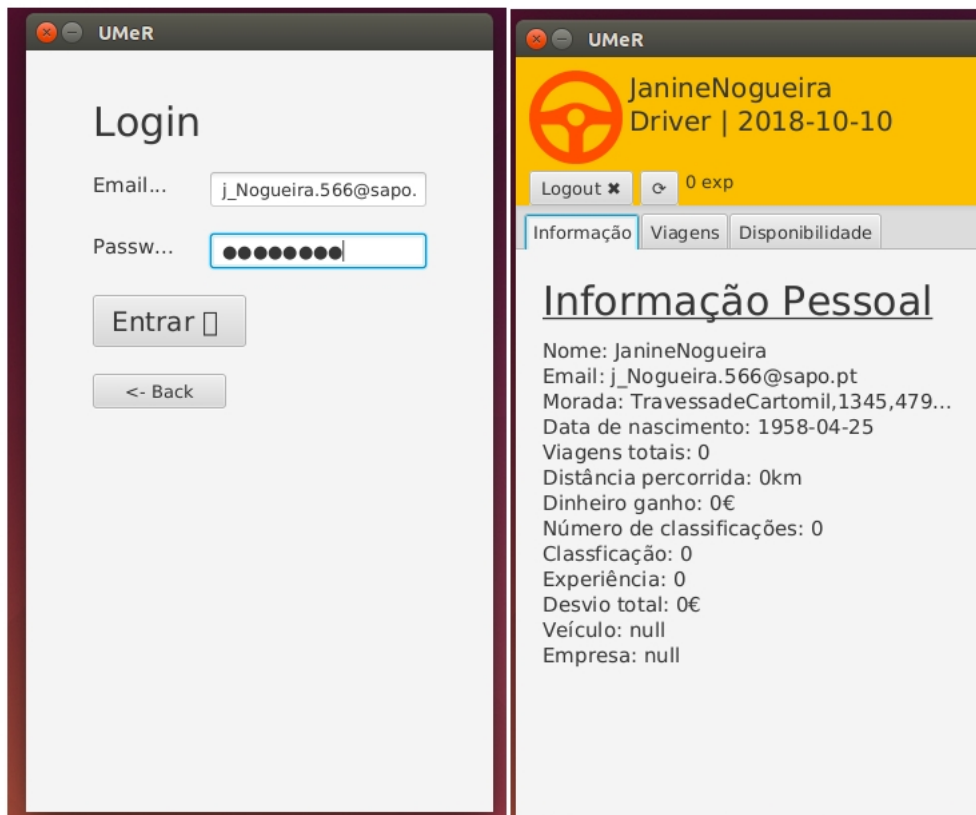
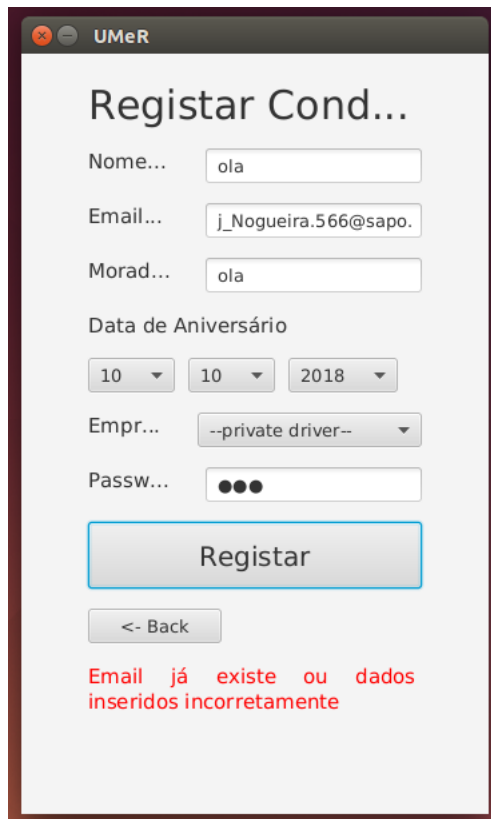


Figura 3.1: Login com Sucesso

Na Figura 3.1 e 3.2 podemos ver exemplos do uso da interface gráfica para a verificação dos dados inseridos através do ficheiro *log*.

A outra parte da verificação, pode ser consultada na execução do *parser* através de *make ats-compile* para a compilação, seguido de *make ats-run*, que envia as mensagens correspondentes aos registos contidos no ficheiro *log* para a batch, sendo assim uma forma de *debug*.



The image shows a graphical user interface window titled "UMeR" with a subtitle "Registrar Cond...". The form contains several input fields: "Nome..." with the value "ola", "Email..." with "j\_Nogueira.566@sapo.", "Morad..." with "ola", and "Data de Aniversário" with three dropdown menus showing "10", "10", and "2018". There is also a dropdown for "Empr..." showing "--private driver--" and a password field "Passw..." with three dots. A large "Registrar" button is centered below the fields. At the bottom left is a "<- Back" button. At the bottom right, a red error message reads: "Email já existe ou dados inseridos incorretamente".

Figura 3.2: Registrar Condutor



## Capítulo 4

# Conclusão

Damos por concluído o primeiro trabalho prático da unidade curricular de Análise e Teste de Software.

Em primeiro lugar, procedemos à construção da gramática com o intuito de verificar se os dados do ficheiro de *input* estavam corretos, de forma a serem introduzidos na plataforma. Depois de construirmos as regras gramaticais que nos satisfaziam em termos de captação de conteúdo, procedemos aos testes dos ficheiros. Percebemos então, que os dados de ambos os ficheiros - *log.txt* e *log2.txt* - estavam corretos e de acordo com as regras.

Em segundo lugar, quando construimos o código *JAVA* concluímos que, segundo a *API* que nos tinha sido fornecida, apenas podíamos executar as operações de registo (condutor, cliente e empresa), bem como as de login e logout. As operações de **solicitar**, **viajar**, e **recusar viagem** não podiam ser inseridas corretamente na plataforma, devido aos problemas enumerados no capítulo 2.1.

Em suma, consideramos que terminamos um trabalho sólido e com fortes bases para ser continuado nos próximos trabalhos práticos. Apesar das dificuldades encontradas, conseguimos definir as melhores estratégias para as superar.