Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Relatório de Engenharia de Software 2

Trabalho Final

Breno Salles

Docente:

Carlos Cunha

Viseu, 2020

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Relatório de Trabalho Final

Cursos de Licenciatura em:

Engenharia Informática

Trabalho Final

Breno Salles

Docente:

Carlos Cunha

Viseu, 2020

Índice

Índice de Figuras

Índice de Tabelas

1 Introdução

Este trabalho tem como objetivo desenvolver uma aplicação em Java para um simular a gestão de utilizadores numa empresa. Esta aplicação tem como principais objetivos e requisitos métodos para:

- Criar, consultar, alterar e remover utilizadores.
- Criar, consultar, alterar e remover recursos.
- Registar um utilizador para permitir o acesso à aplicação.
- Autenticar um utilizador.

De forma a permitir a qualidade da aplicação é necessário realizar testes como:

- Testes de unidade, para a funcionalidade da aplicação.
- Testes ao serviço REST que irá ser utilizado para *back end*.
- Testes de integração depois de tudo estar a funcionar.
- Desenhar testes de performance, nomeadamente testes de robustez e disponibilidade.

2 Desenvolvimento

Para o desenvolvimento da aplicação pensou-se em algo como está mostrado na Figura 1, dividindo-se em diversos módulos independentes para separar conceitos.

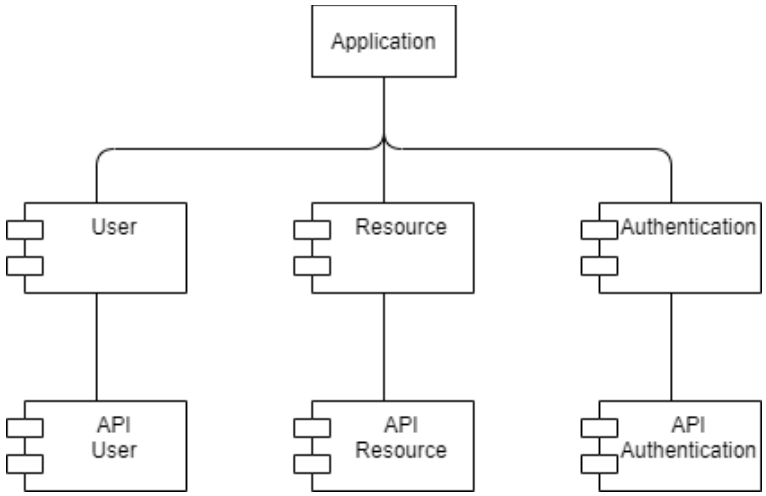

Figura 1 - Módulos da aplicação

2.1 Desenho da aplicação

Design Patterns ou Padrões de Desenho são soluções comuns para problemas em engenharia de software. Um padrão difere de um algoritmo na medida em que, enquanto um algoritmo é uma equivalente a uma receita de culinária em que seguimos todos os passos à risca, um padrão é uma *blueprint* que podemos personalizar para resolver determinados problemas de *design* na nossa aplicação.

No âmbito desta aplicação, foram utilizados dois desenhos para simplificar o processo de resolução do problema, nomeadamente o *Chain of Responsibility* e *Decorator*. O diagrama de classes final pode ser visto na Figura 2.

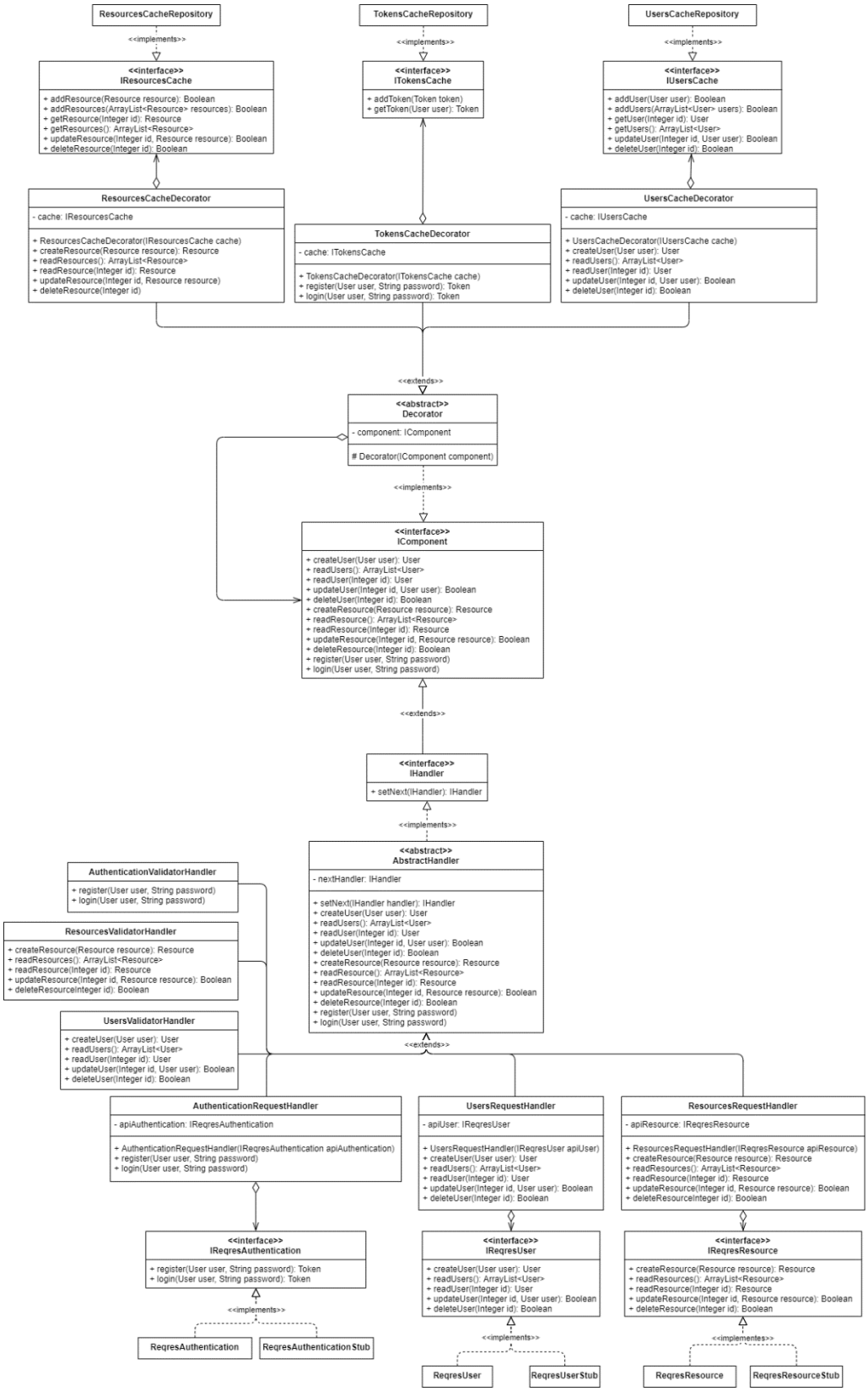

Figura 2 - Diagrama de Classes

2.1.1 Padrão de desenho *Chain of Responsibility*

O padrão *Chain of Responsibility* é um padrão comportamental que deixa passar *requests* ao longo de vários *handlers* (similar ao comportamento de uma *linked list*). Ao receber um *request*, cada *handler* decide se quer processar o *request* ou se vai o passar para o próximo *handler* na corrente.

Quando aplicado ao desenvolvimento da aplicação, este padrão foi utilizado para separar a lógica de cada módulo. Existem dois *handlers* para cada um dos módulos, o primeiro *handler* é sempre responsável pela validação de dados, enquanto o segundo *handler* é responsável pela comunicação com a API. O diagrama de classes associado a este padrão, depois de extraído da Figura 2, encontra-se na Figura 3.

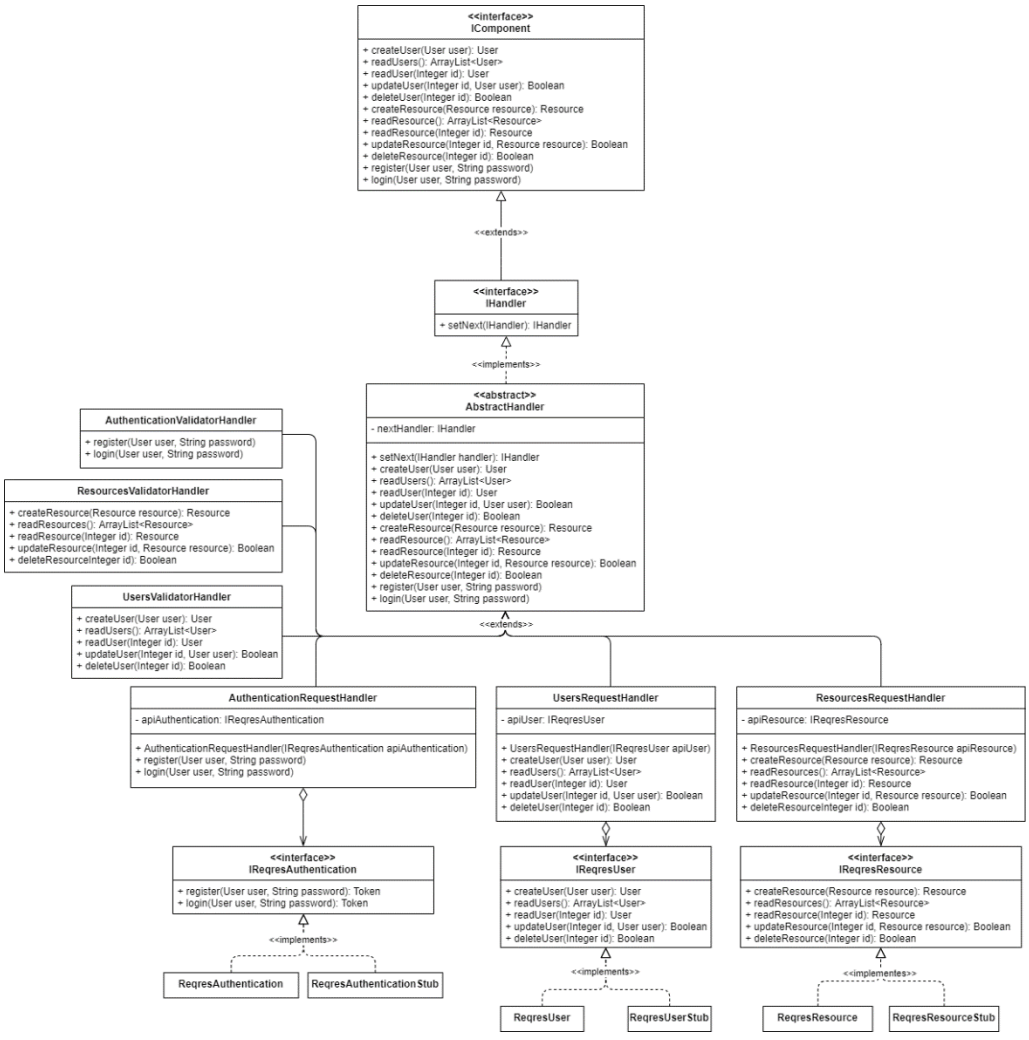

Figura 3 - Diagrama de classes do Chain of Responsibility

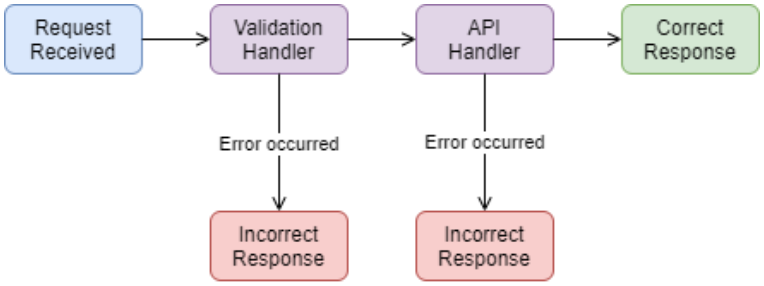

Figura 4 - Fluxo do processo dos handlers

2.1.2 Padrão de desenho *Decorator*

O padrão *Decorator* é um padrão estrutural que permite adicionar novos comportamentos a objetos colocando esses objetos dentro de *wrappers* que contém os comportamentos originais.

Quando aplicado ao desenvolvimento da aplicação, este padrão foi utilizado para adicionar comportamento aos *handlers* que comunicam com a API, mais concretamente, adiciona um simples serviço *cache* a cada um desses *handlers*. O diagrama de classes associado a este padrão, depois de extraído da Figura 2, encontra-se na Figura 5.

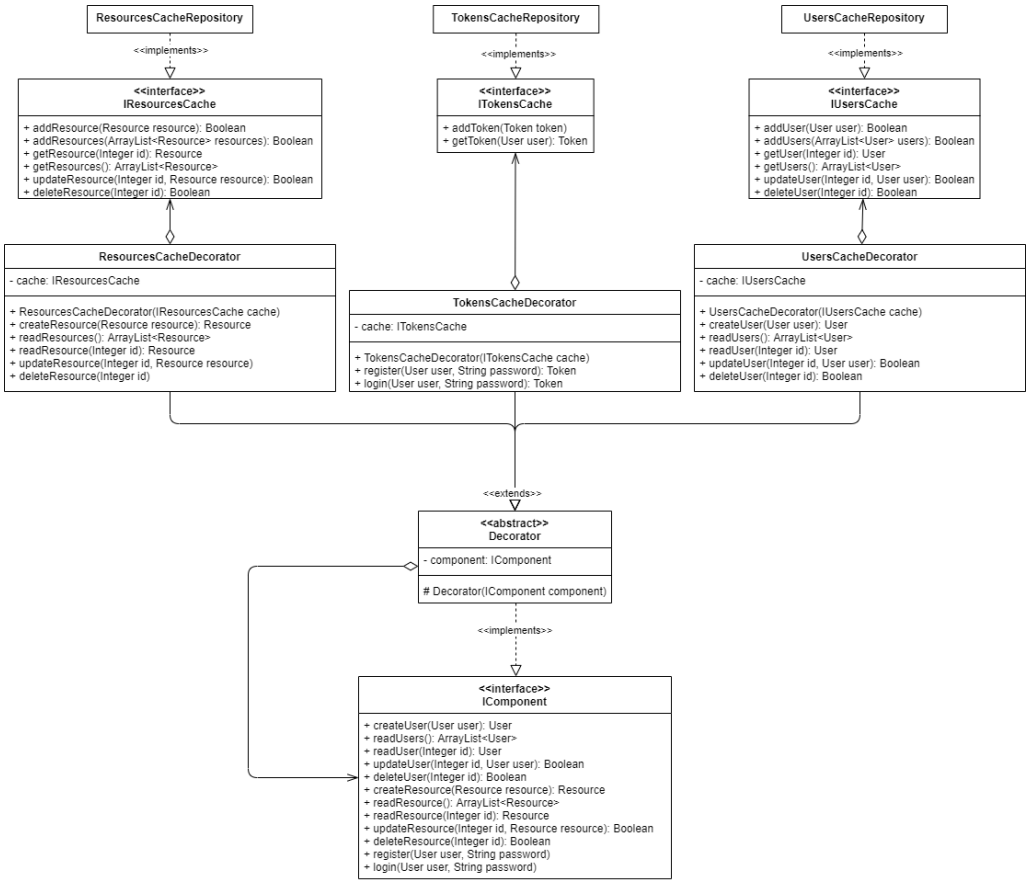

Figura 5 - Diagrama de classes do Decorator

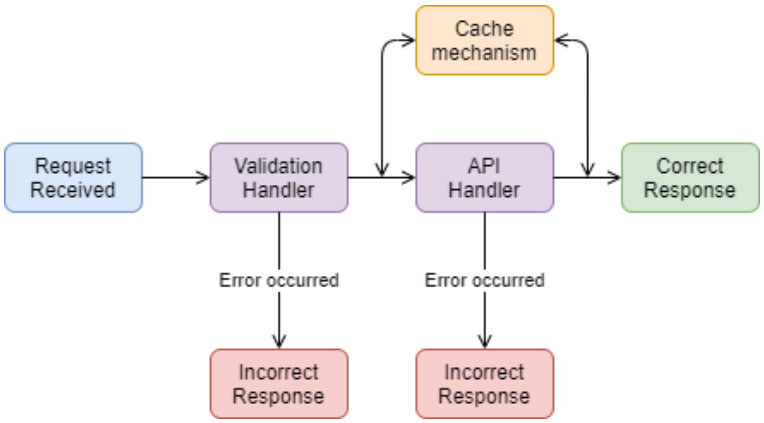

Figura 6 - Fluxo do processo dos handlers com cache

2.2 Testes da aplicação

Os testes de software são um processo de avaliação de um sistema e os seus componentes, com o objetivo de perceber se o mesmo satisfaz os requisitos especificados. Esses testes servem, na maioria dos casos, para encontrar possíveis erros ou falhas de implementação que façam com que o software não esteja de acordo com os requisitos. Neste capítulo vão ser abordados testes de unidade, testes de serviço, testes de integração e testes de performance.

Durante o desenvolvimento deste projeto aplicou-se a metodologia *Test Driven Development* (TDD), isto é, realizou-se primeiro os testes segundo as especificações dos requisitos e depois implementou-se a solução. Isto permite garantir que todas as condições necessárias para as funcionalidades são cobertas, uma vez que, caso não seja, os testes nunca iriam passar.

2.2.1 Testes de unidade

Os testes de unidade são testes cujo propósito é testar uma secção de código (conhecida por unidade) de modo a garantir que o seu comportamento está correto segundo o especificado nos requisitos. Uma vez que, nesta fase da aplicação, ainda não existe o módulo que irá comunicar com a API e estamos a utilizar uma metodologia de desenvolvimento *Incremental Top-Down*, temos de substituir esse módulo por duplos. Essa alteração pode ser vista na Figura 7.

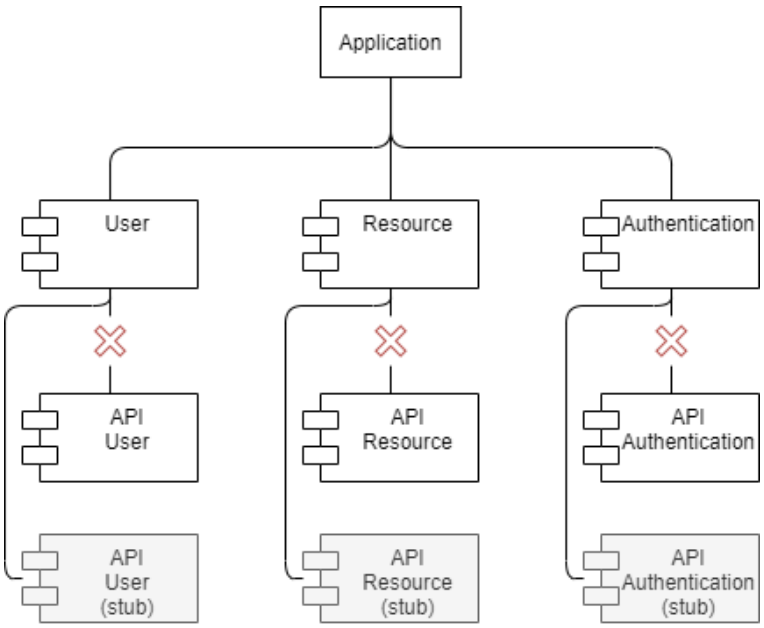

Figura 7 - Duplos (stubs) a substituírem a implementação

Requisito	Descrição	Teste	Dados de Teste	Resultado Expectável
1	Create User (Factory)	createUserWithNullEmail	email == null Everything else correct	Throws InvalidUserException
		createUserWithInvalidEmail	email == invalid regex email Everything else correct	Throws InvalidUserException
		createUserWithNullFirstName	firstName == null Everything else correct	Throws InvalidUserException
		createUserWithLowerFirstName	firstName == 2 Everything else correct	Throws InvalidUserException
		createUserWithHigherFirstName	firstName == 17 Everything else correct	Throws InvalidUserException
		createUserWithNullLastName	lastName == null Everything else correct	Throws InvalidUserException
		createUserWithLowerLastName	lastName == 2 Everything else correct	Throws InvalidUserException
		createUserWithHigherLastName	lastName == 17 Everything else correct	Throws InvalidUserException
		createUserWithNullAvatar	avatar == null Everything else correct	Throws InvalidUserException
		createUserWithLowerAvatar	lastName == 3 Everything else correct	Throws InvalidUserException
		createUserWithHigherAvatar	lastName == 65 Everything else correct	Throws InvalidUserException
		createUserOk	email == "email@email.com" firstName = "firstName" lastName = "lastName" avatar = "avatar"	User
2	Read Users API (stub)	readUsersOk		ArrayList<User>
3	Read User API (stub)	readUserWithNullId	id == null	null
		readUserInvalidId	id == 0	null
		readUserOk	id == 1	User
4	Create User API (stub)	createWithNullUser	user == null	null
		createUserOk	user == Valid User	User
5	Update User API (stub)	updateWithNullId	id == null user == Valid User	FALSE
		updateWithInvalidId	id == 0 user == Valid User	FALSE
		updateWithNullUser	id == 1 user == null	FALSE
		updateUserOk	id == 1 user == Valid User	TRUE
6	Delete User API (stub)	deleteUserWithNullId	id == null	FALSE
		deleteUserInvalidId	id == 0	FALSE
		deleteUserOk	id == 1	TRUE
7	Read User API (cache+stub)	readUserInCacheInvalidId	id == 0	null
		readUserInCacheOk	id == Id of created user	User
8	Create User API (cache+stub)	createUserInCacheOk	user == Valid User	User (Check if cache contains user)
9	Update User API (cache+stub)	updateUserInCacheInvalidId	id == 0 user == Valid User	FALSE
		updateUserInCacheOk	id == Id of a created user user == Valid User	True (Check if cache user was updated)
10	Delete User API (cache+stub)	deleteUserInCacheInvalidId	id == 0	FALSE
		deleteUserInCacheOk	id == Id of a created user	True (Check if cache user was deleted)

Tabela 1 - Testes de unidade do módulo User

Requisito	Descrição	Teste	Dados de Teste	Resultado Expectável
1	Create Resource (Factory)	createResourceWithNullName	name == null Everything else correct	Throws InvalidResourceException
		createResourceWithLowerName	name == 3 Everything else correct	Throws InvalidResourceException
		createResourceWithHigherName	name == 17 Everything else correct	Throws InvalidResourceException
		createResourceWithNullYear	year == null Everything else correct	Throws InvalidResourceException
		createResourceWithNullColor	color == null Everything else correct	Throws InvalidResourceException
		createResourceWithInvalidColor	color == invalid regex hexadecimal color Everything else correct	Throws InvalidResourceException
		createResourceWithNullPantoneColor	pantoneColor == null Everything else correct	Throws InvalidResourceException
		createResourceOk	name == "Resource" year == 2000 color == "#123123" pantoneColor == "Some Color"	Resource
2	Read Resources API (stub)	readResourcesOk		ArrayList<Resource>
3	Read Resource API (stub)	readResourceWithNullId	id == null	null
		readResourceInvalidId	id == 0	null
		readResourceOk	id == 1	Resource
4	Create Resource API (stub)	createWithNullResource	Resource == null	null
		createResourceOk	Resource == Valid Resource	Resource
5	Update Resource API (stub)	updateWithNullId	id == null Resource == Valid Resource	FALSE
		updateWithInvalidId	id == 0 Resource == Valid Resource	FALSE
		updateWithNullResource	id == 1 Resource == null	FALSE
		updateResourceOk	id == 1 Resource == Valid Resource	TRUE
6	Delete Resource API (stub)	deleteResourceWithNullId	id == null	FALSE
		deleteResourceInvalidId	id == 0	FALSE
		deleteResourceOk	id == 1	TRUE
7	Read Resource API (cache+stub)	readResourceInCacheInvalidId	id == 0	null
		readResourceInCacheOk	id == Id of created Resource	Resource
8	Create Resource API (cache+stub)	createResourceInCacheOk	Resource == Valid Resource	Resource (Check if cache contains Resource)
9	Update Resource API (cache+stub)	updateResourceInCacheInvalidId	id == 0 Resource == Valid Resource	FALSE
		updateResourceInCacheOk	id == Id of a created Resource Resource == Valid Resource	True (Check if cache Resource was updated)
10	Delete Resource API (cache+stub)	deleteResourceInCacheInvalidId	id == 0	FALSE
		deleteResourceInCacheOk	id == Id of a created Resource	True (Check if cache Resource was deleted)

Tabela 2 - Testes de unidade do módulo Resource

Requisito	Descrição	Teste	Dados de Teste	Resultado Espectável
1	Registar um utilizador API (stub)	registerWithNullUser	user == null	null
		registerWithNullPassword	password == null	null
		registerWithLowerPassword	user == Valid User password == 7 chars	null
		registerWithHigherPassword	user == Valid User password == 129 chars	null
		registerOk	user == Valid User password == Valid String	Token
2	Autenticar um utilizador API (stub)	loginWithNullUser	user == null	null
		loginWithNullPassword	user == Valid User password == null	null
		loginWithLowerPassword	user == Valid User password == 7 chars	null
		loginWithHigherPassword	user == Valid User password == 129 chars	null
		loginOk	user == Valid User password == Valid String	Token

Tabela 3 - Testes de unidade do módulo Authentication

2.2.2 Testes de serviço

Os testes de serviço à API REST são divididos em várias ações necessárias para cada teste:

1. Verificar se o HTTP *Status Code* está correto.
2. Verificar se o *payload* em JSON contém todos os campos corretos.
3. Verificar que os *Headers* de resposta estão corretos.

Os testes realizados para cada *endpoint* são:

- Testes positivos básicos.
- Testes positivos em que se testas campos opcionais.
- Testes negativos com dados válidos.
- Testes negativos com dados inválidos.

Requisito	Descrição	Teste	Dados de Teste	Resultado Espectável
1	GET /users	getUsersOk	(Parametized Test) url == {"https://reqres.in/api/users", "https://reqres.in/api/users?page=1", "https://reqres.in/api/users?page=2"}	Response.statusCode == 200 Response.contentType == "application/json" Response.body == Valid Json JSONObject.page != null JSONObject.per_page != null JSONObject.total != null JSONObject.total_pages != null JSONObject.data != null JSONObject.page > 0 JSONObject.per_page >= 0 JSONObject.total >= 0 JSONObject.total_pages > 0 JSONObject.per_page == JSONObject.data.size() Every JSONObject.data should have: id (>0), email (valid email), first_name (length>0), last_name (length>0), avatar (valid website)
		getUsersInvalidParam	(Parametized Test) url == {"https://reqres.in/api/users?page=", "https://reqres.in/api/users?page=0"}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
2	GET /users/{id}	getUserOk	(Parametized Test) url == {"https://reqres.in/api/users/1", "https://reqres.in/api/users/2"}	Response.statusCode == 200 Response.contentType == "application/json" Response.body == Valid Json JSONObject.data != null Every JSONObject.data should have: id (>0), email (valid email), first_name (length>0), last_name (length>0), avatar (valid website)
		getUserInvalidId	(Parametized Test) url == {"https://reqres.in/api/users/0", "https://reqres.in/api/users/ola"}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
3	POST /users	createUserOk	url == "https://reqres.in/api/users"	Response.statusCode == 201 Response.contentType == "application/json" Response.body == Valid Json Every JSONObject should have: id (>0), email (valid email), first_name (length>0), last_name (length>0), avatar (valid website)
		createUserInvalidUrl	(Parametized Test) url = {"https://reqres.in/api/users/ola", "https://reqres.in/api/users/1"}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
		createUserWithInvalidJson	(Parametized Test) url == "https://reqres.in/api/users" Json == {invalid Json, missing key/value}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
		createUserWithExistingEmail	url == "https://reqres.in/api/users" Json.email == "michael.lawson@reqres.in"	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
4	PUT /users/{id}	updateUserOk	url == "https://reqres.in/api/users/1"	Response.statusCode == 200 Response.contentType == "application/json" Response.body == Valid Json Every JSONObject should have: id (>0), email (valid email), first_name (length>0), last_name (length>0), avatar (valid website)
		updateUserInvalidUrl	(Parametized Test) url = {"https://reqres.in/api/users/0", "https://reqres.in/api/users/500", "https://reqres.in/api/users"}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
		updateUserWithInvalidJson	(Parametized Test) url == "https://reqres.in/api/users" Json == {invalid Json, missing key/value}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
5	DELETE /users/{id}	deleteUserOk	url == "https://reqres.in/api/users/1"	Response.statusCode == 204
		deleteUserInvalidUrl	(Parametized Test) url = {"https://reqres.in/api/users/0", "https://reqres.in/api/users/500", "https://reqres.in/api/users"}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0

Tabela 4 - Testes de serviço do endpoint users

Requisito	Descrição	Teste	Dados de Teste	Resultado Espectável
1	GET /resources	getresourcesOk	(Parametized Test) url == {"https://reqres.in/api/resources", "https://reqres.in/api/resources?page=1", "https://reqres.in/api/resources?page=2"}	Response.statusCode == 200 Response.contentType == "application/json" Response.body == Valid Json JSONObject.page != null JSONObject.per_page != null JSONObject.total != null JSONObject.total_pages != null JSONObject.data != null JSONObject.page > 0 JSONObject.per_page >= 0 JSONObject.total >= 0 JSONObject.total_pages > 0 JSONObject.per_page == JSONObject.data.size() Every JSONObject.data should have: id (>0), name (length>0l), year (>0), color (length>0), pantone_value (length>0)
		getresourcesInvalidParam	(Parametized Test) url == {"https://reqres.in/api/resources?page=", "https://reqres.in/api/resources?page=0"}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
2	GET /resources/{id}	getresourceOk	(Parametized Test) url == {"https://reqres.in/api/resources/1", "https://reqres.in/api/resources/2"}	Response.statusCode == 200 Response.contentType == "application/json" Response.body == Valid Json JSONObject.data != null Every JSONObject.data should have: id (>0), email (valid email), first_name (length>0), last_name (length>0), avatar (valid website)
		getresourceInvalidId	(Parametized Test) url == {"https://reqres.in/api/resources/0", "https://reqres.in/api/resources/ola"}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
3	POST /resources	createresourceOk	url == "https://reqres.in/api/resources"	Response.statusCode == 201 Response.contentType == "application/json" Response.body == Valid Json Every JSONObject should have: id (>0), name (length>0l), year (>0), color (length>0), pantone_value (length>0)
		createresourceInvalidUrl	(Parametized Test) url = {"https://reqres.in/api/resources/ola", "https://reqres.in/api/resources/1"}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
		createresourceWithInvalidJson	(Parametized Test) url == "https://reqres.in/api/resources" Json == {invalid Json, missing key/value}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
4	PUT /resources/{id}	updateresourceOk	url == "https://reqres.in/api/resources/1"	Response.statusCode == 200 Response.contentType == "application/json" Response.body == Valid Json Every JSONObject should have: id (>0), name (length>0l), year (>0), color (length>0), pantone_value (length>0)
		updateresourceInvalidUrl	(Parametized Test) url = {"https://reqres.in/api/resources/0", "https://reqres.in/api/resources/500", "https://reqres.in/api/resources"}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
		updateresourceWithInvalidJson	(Parametized Test) url == "https://reqres.in/api/resources" Json == {invalid Json, missing key/value}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
5	DELETE /resources/{id}	deleteresourceOk	url == "https://reqres.in/api/resources/1"	Response.statusCode == 204
		deleteresourceInvalidUrl	(Parametized Test) url = {"https://reqres.in/api/resources/0", "https://reqres.in/api/resources/500", "https://reqres.in/api/resources"}	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0

Tabela 5 - Testes de serviço do endpoint resource

Requisito	Descrição	Teste	Dados de Teste	Resultado Espectável
1	POST /login	loginOk	url == "https://reqres.in/api/login" email == "eve.holt@reqres.in" password == "cityslicka"	Response.statusCode == 200 Response.contentType == "application/json" Response.body == Valid Json JSONObject.token != null JSONObject.token.length > 0
		loginInvalidUrl	(Parametized Test) url == {"https://reqres.in/api/login/1". "https://reqres.in/api/login/ola"} email == "eve.holt@reqres.in" password == "cityslicka"	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
		loginInvalidJson	(Parametized Test) url == "https://reqres.in/api/login" json = {invalid json, missing key/value)	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
		loginInvalidCredentials	url == "https://reqres.in/api/login" email == "breno@breno.com" password == "cityslicka"	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
2	POST /register	registerOk	url == "https://reqres.in/api/register" email == "eve.holt@reqres.in" password == "cityslicka"	Response.statusCode == 200 Response.contentType == "application/json" Response.body == Valid Json JSONObject.id != null JSONObject.id > 0 JSONObject.token != null JSONObject.token.length > 0
		registerInvalidUrl	(Parametized Test) url == {"https://reqres.in/api/register/1". "https://reqres.in/api/login/ola"} email == "eve.holt@reqres.in" password == "cityslicka"	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0
		registerInvalidJson	(Parametized Test) url == "https://reqres.in/api/register" json = {invalid json, missing key/value)	Response.StatusCode >= 400 && < 500 Response.ContentType = "application/json" Response.body = Valid Json JSONObject.error != null JSONObject.error.length > 0

Tabela 6 - Testes de serviço do endpoint login e register

Alguns testes descritos na Tabela 4, Tabela 5 e Tabela 6 nunca irão funcionar uma vez que estamos a testar um serviço REST que devolve respostas enlatadas, equivalente a um *stub* numa API.

2.2.3 Testes de integração

Os testes de integração são realizados quando diferentes módulos são combinados e testados em conjunto de modo a avaliar o correto funcionamento do sistema como especificado nos requisitos funcionais. Os testes de integração para os módulos de *User*, *Resource* e *Authentication* com os módulos *API User*, *API Resource* e *API Authentication*, como descritos na Figura 1, estão presentes na Tabela 7, Tabela 8 e Tabela 9 respetivamente.

Requisito	Descrição	Teste	Dados de Teste	Resultado Expectável
1	Read Users API	readUsersOk		ArrayList<User>
2	Read User API	readUserWithNullId	id == null	null
		readUserInvalidId	id == 0	null
		readUserOk	id == 1	User
3	Create User API	createWithNullUser	user == null	null
		createUserOk	user == Valid User	User
4	Update User API	updateWithNullId	id == null user == Valid User	FALSE
		updateWithInvalidId	id == 0 user == Valid User	FALSE
		updateWithNullUser	id == 1 user == null	FALSE
		updateUserOk	id == 1 user == Valid User	TRUE
5	Delete User API	deleteUserWithNullId	id == null	FALSE
		deleteUserInvalidId	id == 0	FALSE
		deleteUserOk	id == 1	TRUE
6	Read User API (cache)	readUserInCacheInvalidId	id == 0	null
		readUserInCacheOk	id == Id of created user	User
7	Create User API (cache)	createUserInCacheOk	user == Valid User	User (Check if cache contains user)
8	Update User API (cache)	updateUserInCacheInvalidId	id == 0 user == Valid User	FALSE
		updateUserInCacheOk	id == Id of a created user user == Valid User	True (Check if cache user was updated)
9	Delete User API (cache)	deleteUserInCacheInvalidId	id == 0	FALSE
		deleteUserInCacheOk	id == Id of a created user	True (Check if cache user was deleted)

Tabela 7 - Testes de integração do módulo User com API User

Requisito	Descrição	Teste	Dados de Teste	Resultado Expectável
1	Read Resources API	readResourcesOk		ArrayList<Resource>
2	Read Resource API	readResourceWithNullId	id == null	null
		readResourceInvalidId	id == 0	null
		readResourceOk	id == 1	Resource
3	Create Resource API	createWithNullResource	Resource == null	null
		createResourceOk	Resource == Valid Resource	Resource
4	Update Resource API	updateWithNullId	id == null Resource == Valid Resource	FALSE
		updateWithInvalidId	id == 0 Resource == Valid Resource	FALSE
		updateWithNullResource	id == 1 Resource == null	FALSE
		updateResourceOk	id == 1 Resource == Valid Resource	TRUE
5	Delete Resource API	deleteResourceWithNullId	id == null	FALSE
		deleteResourceInvalidId	id == 0	FALSE
		deleteResourceOk	id == 1	TRUE
6	Read Resource API (cache)	readResourceInCacheInvalidId	id == 0	null
		readResourceInCacheOk	id == Id of created Resource	Resource
7	Create Resource API (cache)	createResourceInCacheOk	Resource == Valid Resource	Resource (Check if cache contains Resource)
8	Update Resource API (cache)	updateResourceInCacheInvalidId	id == 0 Resource == Valid Resource	FALSE
		updateResourceInCacheOk	id == Id of a created Resource Resource == Valid Resource	True (Check if cache Resource was updated)
9	Delete Resource API (cache)	deleteResourceInCacheInvalidId	id == 0	FALSE
		deleteResourceInCacheOk	id == Id of a created Resource	True (Check if cache Resource was deleted)

Tabela 8 - Testes de integração do módulo Resource com API Resource

Requisito	Descrição	Teste	Dados de Teste	Resultado Espectável
1	Registar um utilizador API	registerWithNullUser	user == null	null
		registerWithNullPassword	password == null	null
		registerWithLowerPassword	user == Valid User password == 7 chars	null
		registerWithHigherPassword	user == Valid User password == 129 chars	null
		registerOk	user == Valid User password == Valid String	Token
2	Autenticar um utilizador API	loginWithNullUser	user == null	null
		loginWithNullPassword	user == Valid User password == null	null
		loginWithLowerPassword	user == Valid User password == 7 chars	null
		loginWithHigherPassword	user == Valid User password == 129 chars	null
		loginOk	user == Valid User password == Valid String	Token

Tabela 9 - Testes de integração do módulo Authentication com API Authentication

2.2.4 Testes de performance

Testes de performance são testes que tentam determinar como um sistema se comporta em termos de responsividade e estabilidade quando submetido a um *workload*. Também pode servir para medir, verificar e validar algumas qualidades do sistema, como escalabilidade, robustez e uso de

recursos. No âmbito deste projeto, foram realizadas apenas três das sete atividades necessárias para a realização de testes de performance:

1. Identificar o ambiente do teste.
2. Identificar os critérios de aceitação.
3. Desenhar e planear os testes.

Todos os passos valores utilizados são apenas *dummy*. O excel está preparado para receber valores completamente dinâmicos e calcular se um dado teste de performance foi aceite ou não.

# CPU (Logic)	# CPU (Virtual)	Amount of RAM (GB)	SSD Size (GB)		OS

Tabela 10 - Test Environment

Test Type	Response Time (ms)	Resource usage (%)	Amount of Time (min)	Throughtput (%)
Stress Testing	100.00	100.00%	120.00	100.00%
Soak Testing	200.00	50.00%	1440.00	100.00%
Spike Testing	150.00	150.00%	120.00	95.00%
Breakpoint Testing	200.00	50.00%	360.00	100.00%

Tabela 11 - Acceptance Criteria

Scenario #	Scenario	Step #	Description
1	Listar todos os resources	1	Aceder à app
		2	Autenticar-se
		3	Invocar o método de listar o resources possíveis
2	Listar todos os users	1	Aceder à app
		2	Autenticar-se
		3	Invocar o método de listar o users possíveis
3	Listar um resource especifico	1	Aceder à app
		2	Autenticar-se
		3	Invocar o método de listar um resource específico
4	Listar um user especifico	1	Aceder à app
		2	Autenticar-se
		3	Invocar o método de listar um user específico
5	Criar um resource	1	Aceder à app
		2	Autenticar-se
		3	Instanciar um objeto do tipo resource
		4	Invocar o método de criar resource
6	Criar um user	1	Aceder à app
		2	Autenticar-se
		3	Instanciar um objeto do tipo user
		4	Invocar o método de criar user
7	Atualizar um resource especifico	1	Aceder à app
		2	Autenticar-se
		3	Instanciar um objeto do tipo resource
		4	Preparar um id de um resource válido
		5	Invocar o método atualizar resource
8	Atualizar um user especifico	1	Aceder à app
		2	Autenticar-se
		3	Instanciar um objeto do tipo user
		4	Preparar um id de um user válido
		5	Invocar o método atualizar user
9	Eliminar um resource especifico	1	Aceder à app
		2	Autenticar-se
		3	Preparar um id de um resource válido
		4	Invocar o método eliminar resource
10	Eliminar um user especifico	1	Aceder à app
		2	Autenticar-se
		3	Preparar um id de um user válido
		4	Invocar o método eliminar user
11	Fazer registo	1	Aceder à app
		2	Escolher um email e password
		3	Invocar o método register
12	Fazer login	1	Aceder à app
		2	Utilizar email e password corretos
		3	Invocar o método login

Tabela 12 - Testing Scenarios

Scenario #	Total /hour	Simultaneous Users	Think Time (s)
1	150	60	6
2	150	60	6
3	75	30	6
4	75	30	6
5	75	15	12
6	75	15	12
7	50	15	12
8	50	15	12
9	25	25	3
10	25	25	3
11	150	60	3
12	150	60	3

Tabela 13 – Workload

Start Date	Start Time	Finish Date	Finish Time	Test Scenario	Test Type	Average Response Time (ms)	90th Response Time (ms)	Average Resource Usage (%)	90th Resource Usage (%)	Valid Acceptance Criteria?	Valid Amount Time?	Valid Test?
12/12/2020	09:00:00	13/12/2020	23:00:00	5	Stress Testing	2	4	2	4	FALSE	TRUE	FALSE
										#N/A	#N/A	#N/A
										#N/A	#N/A	#N/A
										#N/A	#N/A	#N/A
										#N/A	#N/A	#N/A
										#N/A	#N/A	#N/A
										#N/A	#N/A	#N/A
										#N/A	#N/A	#N/A
										#N/A	#N/A	#N/A
										#N/A	#N/A	#N/A

Tabela 14 - Tests

3 Conclusão

Com este relatório, dá-se como concluído o trabalho prático da unidade curricular Engenharia de Software 2.

O que foi implementado na aplicação:

- Métodos para a criação, alteração, leitura e remoção de utilizadores.
- Métodos para a criação, alteração, leitura e remoção de recursos.
- Métodos para a autenticação e registo de utilizadores.

De modo a garantir a qualidade da aplicação, foram realizados diversos testes, nomeadamente:

- Testes de unidade, nos quais recorreu-se a duplos do tipo *stub* para substituir a API que ainda não estaria presente.
- Testes de serviço, de modo a garantir que a API estava a funcionar corretamente.
- Testes de integração, já contando com a implementação da API na aplicação.
- Criação e desenho de testes de performance, nomeadamente testes de robustez e disponibilidade

O que poderia ser melhorado:

- Utilização de JSONSchema para fazer os testes de serviço, uma vez que, caso o JSON comece a ficar muito complexo, testar cada campo um a um torna-se dispendioso.
- Finalizar os testes de performance com a realização dos mesmos.