FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER’S THESIS

Title:	Document management web-app for a small group
Student:	Bc. Daniel Šulik
Supervisor:	Ing. Jaroslav Kuchař, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of winter semester 2021/22

Instructions

Our lives are daily interviewed with documents. An app aimed to manage documents would very likely make a difference. Therefore, create a web application supporting saving and searching saved documents with a focus on a small group.
- Analyze the problem and existing market solutions.
- Design a prototype of a web application.
- Implement it with at least basic operations:
  - authentication,
  - document management,
  - OCR with basic capabilities to extract the content of documents,
  - look up documents depending on the content, upload date, tags, etc.,
  - user interface.
- Create tests for the backend part of the web-app.
- The application should be open-sourced, free to use and available via docker containers.
- Document each component to ease later the app development.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague September 14, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Document management web-app for a small group

Bc. Daniel Šulik

Department of Software Engineering
Supervisor: Ing. Jaroslav Kuchař, Ph.D.

January 5, 2021

Acknowledgements

First and foremost, I'd like to thank my supervisor, Ing. Jaroslav Kuchař, Ph.D., for his time, help and guidance during the work on the thesis. Additionally, I would like to thank, especially to my father, Peter Šulik, for his support not only during the time I was working on the thesis, but also the time I've spent studying at CTU in Prague. Lastly, but not least, I'd like to thank my family, friends and colleagues at my work for their support.

Declaration

In Prague on January 5, 2021 …………………

Citation of this thesis

Šulik, Daniel. *Document management web-app for a small group.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Táto dimplomová práca sa zaoberá návrhom a implementovaním elektronického systému na správu dokumentov, či už pre jedincov alebo pre malé skupiny ľudí. Vytvorená aplikácia je rozdelená do štyroch častí. Tieto časti sú Frontend, Backend, OCR-API a databáza. Frontendová čast je implementovaná pomocou frameworku Angular 10. Backend a OCR-API sú napísané v jazyku Java s vystaveným REST API. Databáza využíva MongoDB technológiu. V poslednom rade, z každého komponentu tejto aplikácie je vytvorený Docker image.

Klíčová slova EDMS, OCR, Java, Docker, Angular, REST API, MongoDB

Abstract

The thesis deals with designing and implementation of electronic document management system intended for individuals or a small group of people. The application itself is divided into four parts: Front-end, Back-end, OCR-API and database. Front-end part is implemented in Angular 10. Back-end and OCR-API are coded in Java with exposed REST API endpoints. The database is using MongoDB technology. Lastly, every of the mentioned components is made into a Docker image available to use.

Keywords EDMS, OCR, Java, Docker, Angular, REST API, MongoDB

Contents

List of Figures

Introduction

We live in very progressive and hectic times. We usually receive a document from a doctor, a bank or any other institution. The document represents a written deal that binds both sides to do something. In a case, there is an accident or problem with agreed deal, we have the mentioned document, which supports our claim and in the worst case, it is possible to use it in legal way. The number of documents we get during our lifetime is huge. However, only small part of them are very important and therefore, we store them somewhere for time of their need(if there is any.), though it's still considerable amount. The moment, we need them, it takes time to find them between other documents. Doing it once is acceptable waste of time, but doing it repeatedly leads to heavy annoyance that may trigger decision to do something about it. There are two common outcomes. The first one is to physically creating system of archiving documents that saves time, while trying to find the correct document. The second one is to use virtual system. For example a software that is able to load documents, index them and in a need, find them.

Big companies deal with the document problem using software called an electronic document management system, in short EDMS. Every EDMS application has its advantages and disadvantages over the other ones. The most essential differences between them are in functionality, pricing, support, but all of them have in common is to store uploaded documents. Later, when a document or documents are required, they are able to locate them quietly fast.

In comparison to big companies, ordinary people or small groups have minimal options to chose from. Usually, they are stuck between free product with limited features and an expensive option with a lot of features, but hard to learn. Therefore, I decided after looking through already existing options to create a prototype on my own that will be free of charge, supporting essential operation with documents and in future extensible by anyone - open-sourced.

The first part of the thesis describes existing solutions and their comparison. The next part will be an analysis of requirements for a new application, followed by design and implementation chapter. Lastly, there is a testing part, which contains existing tests in the application with the addition of manual tests carried out to verify its functionality.

Chapter 1

State-of-the-art

1.1 Electronic documentation management system

Before I start describing current situation of existing products, I would like to give a brief introduction to document management system, although I've already mentioned a word or two in the introduction.

1.1.1 Brief history

An electronic document management system, or EDMS, is a system, which an organization or individual use to store, search, utilize and manage documents or files as they desire.

The history of EDMS starts as far as the 1980s. Before them, people used file cabinets to somehow organize documents in a smart system that would allow them to find them easily. However, with an increased amount of paperwork, file cabinets increased accordingly. More documents mean more required storage and paper. Those cost companies heavily. Moreover, paper documents could be easily destroyed by disasters as fires and floods, or even could be lost to theft. In addition, any modification of already existing document was very time demanding[8].

However, there were people who were thinking modernly, came with a different approach. As mentioned, it all started around the 1980s, when availability of computer technology was more and more common. Companies could use servers to store their organization's data and with the invention of a scanner that allowed the transformation of paper documents to digital one, came the beginning of EMDS. Throughout the year's existence of EDMS came to being and its features increased[8].

1.1.2 EDMS features

EDMS features available in DMS tools may differ from one to another, but the most common features are[9]:

- Storing
- Document analyzation
- Information extraction via OCR
- Structuring and indexing documents
- Locating and retrieving
- Permission granting and gating
- Versioning
- Security and compliance
- Communication and collaboration

1.2 Existing solutions

Currently, there are multiple solutions on the market. Most of them are either expensive or they are not open-source. In a case that they are both free and open-sourced. Usually, their features are minimalistic, sometimes minimized to a point being unacceptable.

I picked a few products that are interesting. However, it may not represent the whole universe of all available products out there. Besides, I'd like to point out that they were not only compared from the view of a user, but also from a Java developer perspective that may intend to extend a current solution. Solutions that I analyzed are the following:

- Paperless
- Mayan EDMS
- OpenKM
- eFileCabinet
- Templafy
- Vienna advantage

1.2.1 Paperless

The paperless project started around 2015, as mentioned on Github[1] website. The creator of the project was also motivated by similar problems and decided to do something about it, as he also experienced situations, where he needed documents, but it was not around[1].

His main goals were to create something that:

- search digital documents
- save storage by using digital documents instead of paper documents
- back up of documents requires no more paper

Even though the application is popular, it is not developed anymore by the creator, because he has another project that takes priority, but he also mentions that in case somebody has a feature requested via pull request, he does not see a problem to incorporate it to current version[1].

Currently, the application coded in the language Python also contains optical character recognition (further as OCR). To be exact, it is using Tesseract, which is nowadays a very popular and free OCR engine coded in C and C++. In addition, there are visible actions to use docker in the project.

Figure 1.1: Paperless logo [1]

A summary of the Paperless project's pros and cons:

Pros

- Simple with basic abilities
- Open-source
- Free to use
- Default security
- OCR for scanning
- Docker for deployment

[1]https://github.com/the-paperless-project/paperless

- Able to upload documents using FTP

Cons

- Written in Python, would be troublesome to expand
- Dying community
- Complicated execution of the application
- No able to upload documents via the application's GUI

1.2.2 Mayan EDMS

Mayan EDMS is a free and open-source project, coded in programming language Python and using web application framework Django. Its initial release dates back to the year 2011. The web application looks very well developed with multiple features using Tesseract as OCR engine, with basic languages. It's deployable via docker image. In addition to this all, it also has extensive documentation. In comparison to the paperless project, it's is still very active in devolopment[2].

MAYAN

Figure 1.2: Mayan EDMS logo [2]

Pros

- Extensive features
- Using docker image
- Open-source
- Free

- Includes basic document operations
- Active community
- Extensive documentation

Cons

- Coded in Python
- Documentation for starting an application is not clear
- Missing Slovak language in OCR

1.2.3 OpenKM

OpenKM is a Java based application that provides web interfaces for managing documents. Its beginnings date back to the year 2005. It is partially open-sourced for the community edition. However, it's missing any JUnit tests, if there were any in the enterprise edition. As you may think, it's more intended for corporations that will pay for the enterprise edition with all its features. Of course, there is also the community edition with minimalistic features and free to use, however after trying a few times to set it up and start it, I gave up. In addition to all its features, it supports more than 35 languages in the user interface(further as UI) and supports multiple relational databases. Lastly, as I mentioned, it's using Java programming language, however the version is Java 8 or 1.8, which is still very popular and highly used[3].

Figure 1.3: OpenKM logo [3]

A summary of the OpenKM project's pros and cons[3]:

Pros

- Java
- Extensive features
- Supports more than 25 languages in UI
- Provides community edition for free

Cons

- Open-sourced only for community edition
- Free only counting community edition
- Open-sourced code missing tests
- Confusing documentation
- OCR using only five languages, even though it's using Tesseract
- Troublesome installation
- Intended for corporations

1.2.4 eFileCabinet

EFileCabinet was founded and is still on the market since 2001[10]. The whole software consists of multiple features like OCR, file versioning, two-factor authentication, cloud storage and so on. However, the chance to use them depends on how much customer is willing to pay. They offer three different plans, starting from expensive to very expensive, where the more expensive plan contains more features[4].

Figure 1.4: eFIleCabinet logo [4]

A summary of the eFileCabinet project's pros and cons:

Pros

- Multiple features
- Popular
- OCR

- Full-text search
- Mobile and desktop access

Cons

- Pricy
- Not open-sourced
- Intended for bigger groups
- Poor description of the application

1.2.5 Templafy

Templafy is another EDMS tool aim at companies rather than individuals. It is not open-source, and surely it's not free. It looks like the price depends on the number of employees in a company as to get the exact amount, user has to request a price offer from Templafy. Templafy supports integration with popular services as Office 365, GSuite, and so on. In contrast to previous tools, they aim more on the creation and modification of documents via templates. Though, they should also support basic document management, though without OCR ability[5].

Templafy T

Figure 1.5: Templafy logo [5]

A summary of the Templafy project's pros and cons:

Pros

- UI user-friendly
- Integration with popular services
- Document creation and management
- Extensive features
- Documentation and guides available

Cons

- No OCR

- Aiming more to generate documents
- Pay for use
- Not open-sourced

1.2.6 Vienna advantage

Vienna advantage is the last tool that I picked from all available out there. As in previous tools, Vienna advantage have in common some features, but still rich with available features. It's open-sourced for community edition by the description from the official website and coded in C#. It contains tools for CRM, BI, HR. There is also a choice to pick from database types that the client would like to use. Moreover, it is possible to upload via a phone app, email, scanner or web service. Additionally, OCR supports 27 languages and there is available indexing of metadata. Lastly, but not least, Vienna advantage supports versioning of documents[6].

Figure 1.6: Vienna advantage logo [6]

A summary of the Vienna advantage project's pros and cons:

Pros

- Open-sourced for the community edition
- Free community edition, with basic features
- Dozens of features in general
- Uploading documents via multiple devices
- OCR supporting 27 languages
- Professional support for 30 days free

Cons

- Coded in C#
- Advanced feature only in the paid version
- Open-sourced only for the community edition
- Installation is a little bit too complex

1.2.7 Summary

In conclusion to all mentioned EDMS, there is none, which would be capable of all three necessary features:

- Free
- Open-source
- Modifiable in Java

The Closest is Mayan EDMS, but it's coded in Python language, thought if I would be experienced with Python and its frameworks, I would surely choose this one. The second option I though of extending was OpenKM, but it had many flaws that I couldn't overcome to accept.

- Open-sourced **only** for community edition(further CE)
- OCR was missing the Slovak language
- After a few tries and multiple hours waiting, I gave up to install it
- Missing tests and poor documentation for CE

Therefore I decided to create my own prototype with the necessary features.

Chapter 2

Analysis

This chapter is going to explain what should a new application contains. To be more specific, I will elaborate on the necessary functional and non-functional requirements. Lastly, but not least, it is necessary to define use-cases and use-case scenarios.

2.1 Requirements

In the requirements, I will describe the necessary functionalities. Usually, in every software development, we talk about two requirements:

- **Functional**
- **Non-functional**

2.1.1 Functional requirements

Functional requirements define the behavior of the function in a system. Whereas a function is the specification of behavior between data inputs and outputs.

An EDMS application for small groups with basic operation requires functionalities for the following groups:

- User
- Documents
- OCR

2.1.1.1 User

When we talk about EDMS for a small group, then it would be expected that there are accounts for every user to avoid breaching of privacy or deletion of documents that belong to somebody else.

In the case of minimalistic EDMS, there should be these functionalities for user:

- Creating a new user account in EDMS
- Resetting password by a user, who forgot or want to change password
- Login into the app to gain access to view documents
- Logout

2.1.1.2 Documents

The sole reason why electronic document management systems exist are documents. Therefore, it is necessary for an EMDS to have document functionalities.

Therefore, there should be functions for:

- Uploading paper document/s to system
- Searching imported documents using full-text search or advanced search
- Deleting uploaded documents
- Adding tags for document

2.1.1.3 OCR

A very important part of EDMS is the integration of an OCR engine that enables the extraction of text from scanned documents or photos of documents. It is a very essential part of EDMS as it creates a bridge between uploaded documents and the ability to search through them using scanned text, thus the software should have functions of:

- Extract text from document/s using OCR engine
- Specify/Change extraction language of the document to increase accuracy

2.1.2 Non-functional requirements

Non-functional requirements, in contrast to functional requirements, defines software attributes as reusability, open-sourced, scalability, security and other attributes. Usually, it serves as a limitation or restrictions on software design. I mentioned that non-functional requirements are constraints of the application or in other words, one could say they are general characteristics. Therefore, a new prototype of EDMS should contain the following requirements:

- Scalable
- Unit test coverage
- Basic security
- Integrable
- Web-application
- Java based back-end
- Open-Sourced
- Free
- Reusable
- Documented

2.1.2.1 Scalable

Even though this should be software for individual or small group of people, it should have a chance to scale in case of need of a customer. Therefore used technology should be compatible with horizontal scaling, if the customer decides so later.

2.1.2.2 Unit test coverage

Created a back-end part of the application has to be covered by unit testing. This will decrease the chances of unexpected failures/errors, whenever the application gets a new change. Moreover, it will ease further code development.

2.1.2.3 Basic security

The application should be able to provide basic security to user data, even though the software is aimed mostly at local environment, hence the access to user data has to be secured by password.

2.1.2.4 Integrable

The back-end part of the software should have available HTTP API for integration with other applications in the future.

2.1.2.5 Web-application

The created application has to be implemented as a web-application accessible via Mozilla Firefox version 83.0+ and Google Chrome version 87.0.4280+.

2.1.2.6 Java based back-end

The back-end application has to be coded in the programming language Java.

2.1.2.7 Open-sourced

The application code has to be available to everyone. Not just the final version of code, but also version control of the code. The aim is to boost up chances of future work on the project with other people.

2.1.2.8 Free

One of many reasons, why I'm creating new software is the price of existing ones. I believe, there should be an available alternative with satisfactory features for individuals or small groups. A synergy of free and open-source application is much stronger than the effect of only one of them alone.

2.1.2.9 Reusable

The designed software app should contain parts/components that would be available to others for reuse in their own software tools.

2.1.2.10 Documented

Code that is not self-explanatory, should be documented. This is mainly aimed at parts of code that could be reused later by other software developers.

Chapter 3

Design

3.1 Use-cases

Use-cases are an inseparable part of designing every application that should be developed. In general, use-case represents a list of actions describing interactions between selected actor, that may or may not exists in the system and system itself. Whereas an actor represents a role played by a user that interacts with the system[11].

In the application was identified only one role, which is:

- User - the one that is using application

Normally, there would also be the role of an Admin to take care of existing users. However, as this will be a prototype with minimal functions, it is enough to have just one role, with all document functions assigned to the role user 3.1.

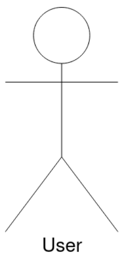


Figure 3.1: Identified user role

For the identified role user, there exist the following use-cases visible on included image, see 3.2.

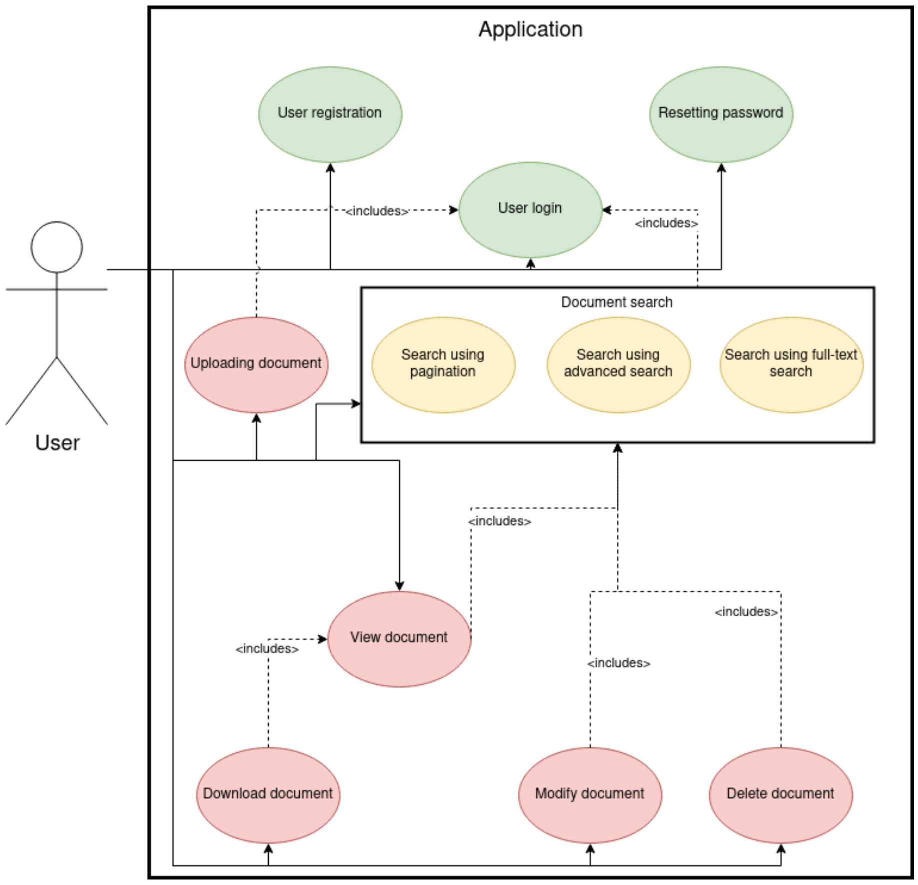


Figure 3.2: Assigned use-cases to the role user

3.1.1 User registration

Basic user operation is his registration to the system.

3.1.2 Resetting password

An essential use-case is resetting the password for the user in case he forgets his name or password. Also could be used, when the user just wants to change the password.

3.1.3 User login

A user logs into the application.

3.1.4 Uploading document

A user uploads document or documents to the application.

3.1.5 Delete document

A user has an option to delete an unwanted document from the system.

3.1.6 View document

A user can view document or document details inside the application.

3.1.7 Modify document settings

In case a user sets an incorrect scanning setting for a document, he should have a chance to modify them without uploading the whole document again.

3.1.8 Search document using pagination

A default searching option that uses pagination.

3.1.9 Search document using full-text search

A document can be found by a user using full-text search.

3.1.10 Search document using advanced search

A document can be found by a user using advanced search with parameters.

3.1.10.1 Download uploaded document

Uploaded documents have to be stored and downloadable, if a user wants to download them.

3.2 Use-case scenarios

Use-case scenarios are a very important part of designing an application. It helps to demonstrate and visualize user or even developer, how use-case action should work, what other actions it contains, what happens in case of invalid data and so on.

3.2.1 User registration

A user that would like to have access to the application has to register first. Upon loading the app, the user will see only choices to login, reset password or register. After clicking on the link register, the user will be redirected to page register with four text fields to fill: username, e-mail, password, same password again. When the user is finished writing input data, he can click to submit to register. If not successful, he will get notification about failed registration and to change invalid data, but if he was successful, he will receive informative alert that the account was registered and a confirmation e-mail was sent to the user. By clicking on a link located inside the confirmation e-mail, user will be redirected to the application and the account will be activated. For better visualisation see C.1 diagram.

3.2.2 Resetting password

In a case a user forgot his password or username, he can request a password change by click on the link ”Forgot Username/Password” under the login option. The user will be redirected to the page for resetting password, where e-mail address is required. After filling e-mail address, the user has to submit by click on a button reset. If the e-mail address was valid, he will get a notification about password reset e-mail being sent to him. If it was not valid, he will stay on the page with chance to change it. After clicking on a link found inside password reset e-mail, he will be redirected to page to change his password. Filling new password, new password again and clicking to submit it, the password will be changed. If he an filled invalid password or both passwords did no match, he has to correct it. For better visualisation see C.1 diagram.

3.2.3 User login

To access the application, a user has to login. On the first visit of the application, the user is redirected to a login page, where he is required to fill username and password. After clicking a login button, the user will be redirected to the documents page if the credentials were correct. If not he will stay on the login page to fix credentials. For better visualisation see C.3 diagram.

3.2.4 Documents

All of the following scenarios belongs to possible operations that could be done with documents. In all of these scenarios a user is required to have an access to the application, before he can proceed further. To do so, a user has to log in and will be redirected to the documents page. If log in was not successful, he has to try again. After being redirected to the page documents, the user will able to see search options at the top and table with documents under it.

3.2.4.1 Uploading document

A user located at the page documents, has to click on the navigation tab import that will redirect him to an import documents page. By clicking on a file selection bar, a window with files will be shown to the user for selection. Next to the selected files, there will be changeable settings for scanning selected files. Settings will contain quality scanning option, option to scan immediately, option for multi-paged files, option for document language.

When the user is satisfied with the selection of files and settings, he can move on to submit and upload files/documents. For better visualisation see 3.3 diagram.

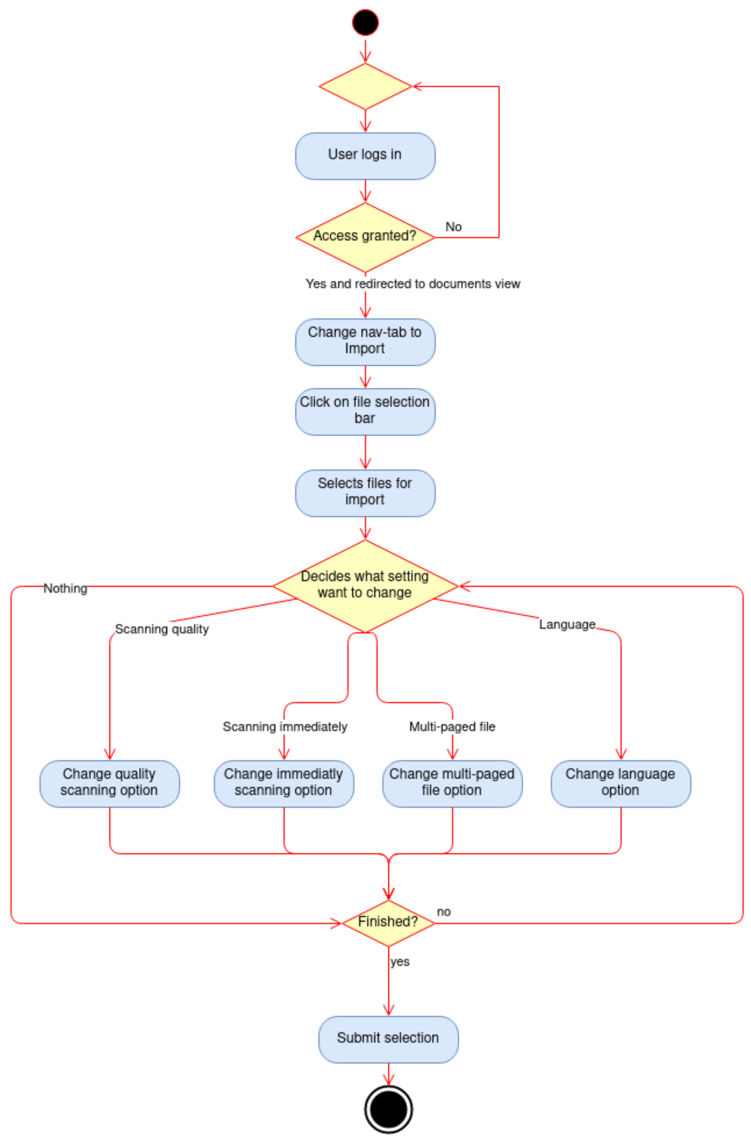


Figure 3.3: Simplified activity diagram of uploading document

3.2.4.2 Search document using pagination

Searching for document/s using pagination is the most simplest approach. A user located on page documents can use pagination arrows at the bottom of the table containing documents. Click on an arrow to the right, page move one index further/deeper and shows a next page. Using an arrow to the left the table shows a previous page. Using a right arrow pointing to the bar next to it will navigate table to the last page, respectively using a left arrow pointing to the bar will navigate table at the beginning of the table. For better visualisation see C.4 diagram.

3.2.4.3 Search document using full-text search

Searching for document/s using full-text search is also a simple approach, but it is much more efficient. A user located on page documents can search using full-text search by selecting full-text search option in top search card, left corner. Afterwards, the user can write text right next to the selection box to the input text field. Search will be started, when the user clicks on a button search. For better visualisation see C.5 diagram.

3.2.4.4 Search document using advanced search

Searching for a document using advanced search a user has to decide, what specific values does the document hast to contain. Firstly, the user has to pick an attribute of the document he will be looking for. A selection of attributes is at the top card, left corner, where is by default selected full-text search. Other options are:

- Text
- State
- Language
- Tag
- Creation date
- Update date
- Shared

By selecting the mentioned options, UI may change the input value format. For example, in the case of the text option, the user can fill a string value inside input text field, but for the state option input text field will be changed to a select box. Anyway, if the user wants to add an already filled search option value to searching, he has to click on the add button. The value will be saved to search parameters. To add more values, the user just needs to repeat previous steps. All values from one search option are disjunctional, but values between options are conjunctional. If all search values are selected, it's only needed to click the search button.

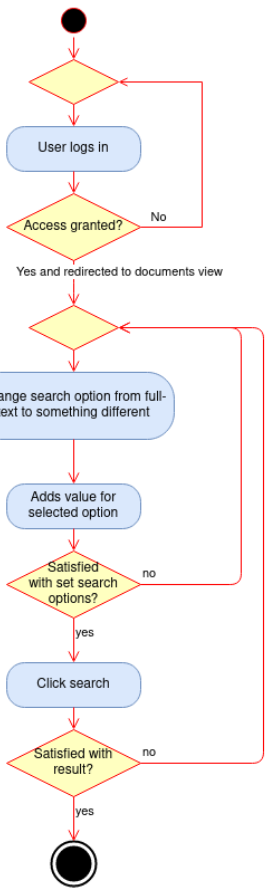

User logs in
Access granted?
No
Yes and redirected to documents view
Change search option from full-text to something different
Adds value for selected option
Satisfied with set search options?
no
yes
Click search
Satisfied with result?
no
yes

Figure 3.4: Simplified activity diagram of using the advanced search to find a document

3.2.4.5 Delete document

When the user finds a desirable document for removing, he has to click on a garbage icon located at the right side of the document row. The user will be prompt, if he is sure to delete it. After confirming deletion, the document will be gone. For better visualisation see 3.5 diagram.

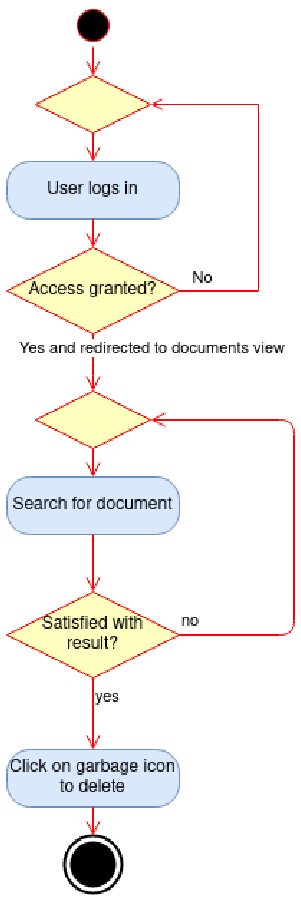


Figure 3.5: Simplified activity diagram of deleting a document

3.2.4.6 View document

If the user wants to view document, first he has to use already mentioned search options:

- full-text search
- pagination
- advanced search

When the user finds a desirable document for viewing and row data are not satisfactory, click on the row will expand and show details of the selected document.

For better visualisation see 3.6 diagram.

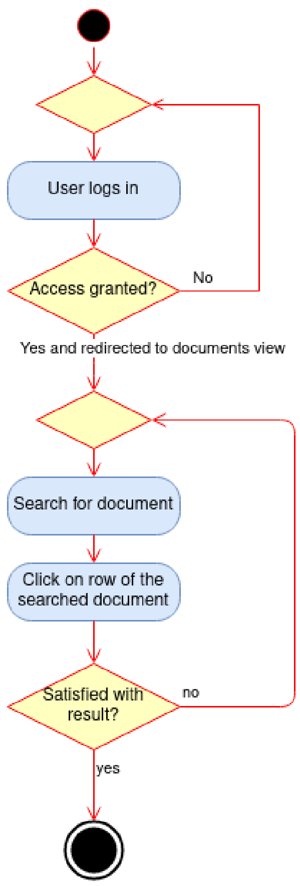


Figure 3.6: Simplified activity diagram of viewing a document

3.2.4.7 Modify document settings

When the user finds a desirable document for modification, a click on the pencil at the side next to the garbage icon will do. After click, a new window will be shown to the user with fields that are non-modifiable as a document's id and filename. Following not modifiable fields will be fields that are changeable:

- Document scanning state
- Document type
- Share document
- Document language
- Scan immediately
- Multi-paged file

- Scanning quality

Below mentioned fields is located change to add new tags for document or remove existing ones by clicking on them. After the user is finished with changes, he needs to submit changes by clicking the button update. For better visualisation see 3.7 diagram.

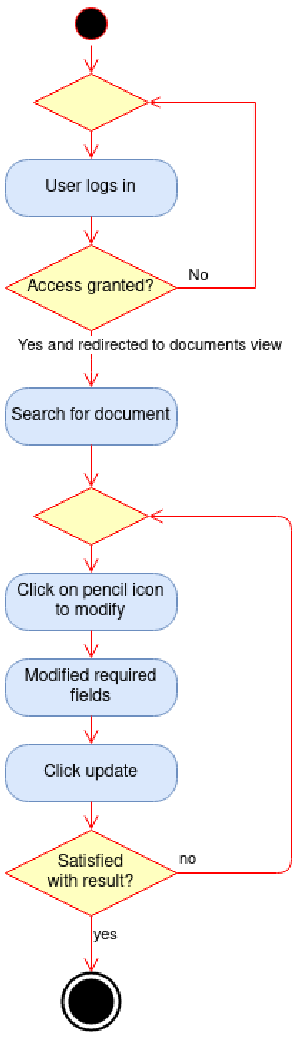


Figure 3.7: Simplified activity diagram of modifying document

3.2.4.8 Download uploaded document

When the user finds a desirable document for downloading its image, he has to click on the row and it will expand and show details of the selected document. In the expanded row is shown thumbnail or preview of the uploaded image of the document. It will be shown, if it's image. Clicking on the preview or the hyperlink below it will trigger pop-up screen to download the file.

For better visualisation see 3.8 diagram.

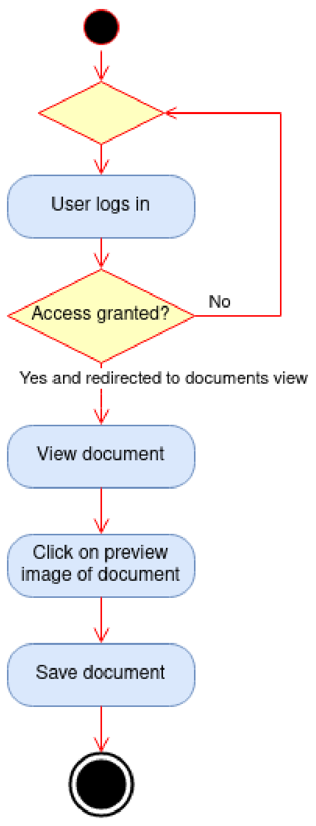

User logs in
Access granted?
No
Yes and redirected to documents view
View document
Click on preview image of document
Save document

Figure 3.8: Simplified activity diagram of downloading uploaded a document

3.3 Architecture of the application

Being aware that the application is going to be a web-application for a small group of people, It would more than satisfactory to use usual architecture - three-layer architecture. It consists of layers:

- **Presentation layer** - Front-end
- **Application layer** - Back-end
- **Data layer** - Database

See Figure 3.9, additional information about the architecture will be described in the following sections.

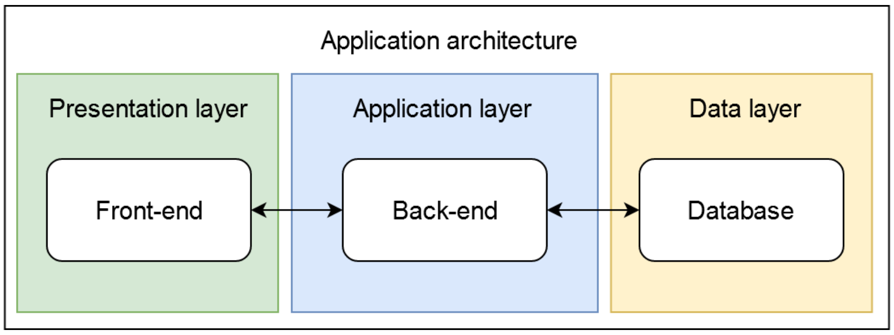


Figure 3.9: General architecture of the application

3.4 Front-end

A presentation layer is the front-end part of the application. Its main component is the user interface(UI) that enables users to interact, see and manipulate with its functions and services.

3.4.1 Pages layout

The first step, before implementing UI is to create a prototype version with pages layout. Using use-case scenarios, there were identified a few pages but only 3 of them most important:

- The login page, see Figure 3.10
- The documents page, see Figure 3.11
- The import page, see Figure 3.12

As you can see in the list above, there the most important pages with layout, how they could look like.

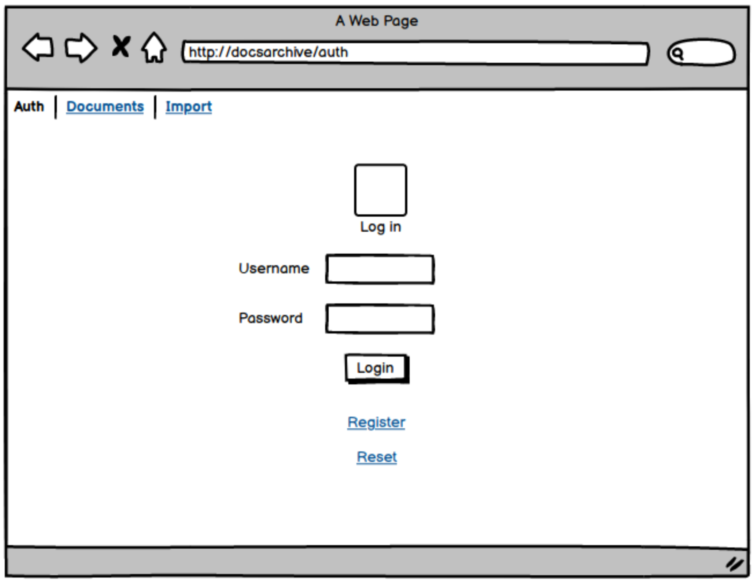
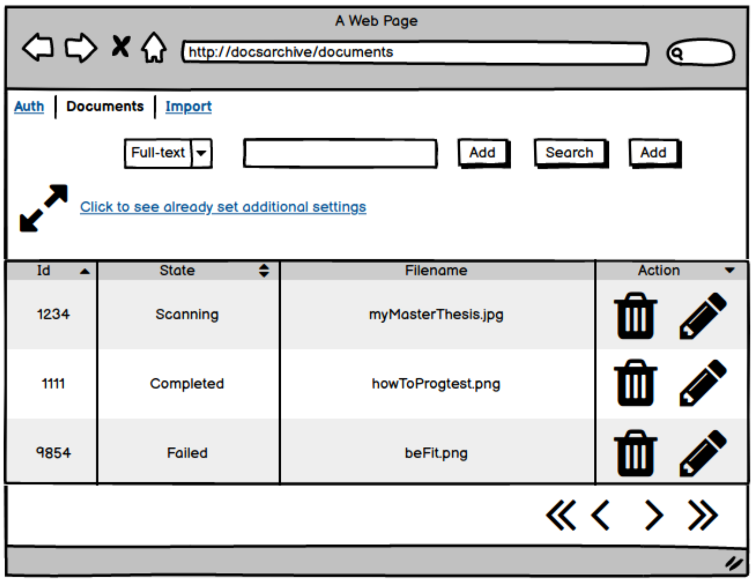


Figure 3.10: Suggestion for the login page


A Web Page
http://docsarchive/documents
Auth
Documents
Import
Full-text
Add
Search
Add
Click to see already set additional settings

Id	State	Filename	Action
1234	Scanning	myMasterThesis.jpg	
1111	Completed	howToProgtest.png	
9854	Failed	beFit.png	

Figure 3.11: Suggestion for the documents page

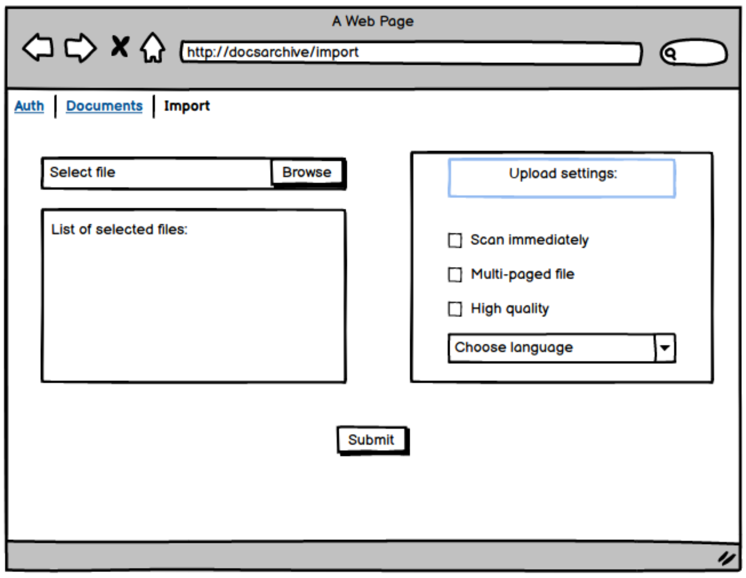


Figure 3.12: Suggestion for the import page

3.4.2 Technologies

After having prepared the layout UI of the application, there is a question about what technology to be used for implementation. There are multiple technologies available, but the most commonly used for front-ends are:

- Angular
- React
- Vue.js

3.4.2.1 Angular

Angular belongs to the most powerful, efficient and open-source JavaScript frameworks. It is developed by Google and it's implemented for development aimed on single page application (SPA)[12].

3.4.2.2 React

React or also known as React framework, was developed by Facebook. Moreover, it has gained popularity in a short period after being released. React is usually used to develop and operate dynamic user interface of web pages that have very high incoming traffic. One of his attributes is the usage of virtual DOM, hence it makes integration into other applications much more straightforward[12].

3.4.2.3 Vue.js

Vue.js was developed in 2016 by one of AngularJS's developers, but even against being younger than other JavaScript frameworks, it has already made its way into the market. Moreover, it offers various features. Its dual integration mode is one of the features that makes developers consider to use it in their single page application[12].

3.4.2.4 Chosen framework

Selecting a front-end framework from all already mentioned is very difficult, as all of them are modern, efficient and popular. To select the final one, I will use the elimination method.

The first one to eliminate would be Vue.js. Although it's gaining popularity and gaining momentum, but it is popular only between developers. Both React and Angular are more often sought, have bigger community and are also backed by big corporations[12].

The decision between Angular and React is as hard as it was with Vue.js, because the differences are somewhat small. However, there is one point that may help. As mentioned in the description of React, it is usually used in apps with high traffic. That cannot be said about this application as it aims for a small group of people, therefore traffic will be relative low or non-existing, which suits Angular much better and makes it the sole winner of the elimination and the version of Angular that will be used is Angular 10.

3.5 Back-end

3.5.1 Technologies

In contrast to the front-end technology, the back-end technology is already predetermined by non-functional requirements, therefore I will be using programming language Java to implement back-end. The version of Java that should be used is at least version 1.8 of its key features as streams, lambdas, optionals and so on. The version 1.8 was released in the year 2014. It is still very popular in projects, stable and as OpenJDK, it is still supported. Later, if it would be needed then it can be rather easily upgraded.

Lastly, there are also other technologies that may be useful in this project and I will introduce them in the following parts.

3.5.1.1 Java

Java is a high-level programming language, but also a platform. Java is mostly recognized by its features:

- Object oriented

- Portable
- Architecture neutral
- Secure
- High performance
- etc.

The greatest benefit of using Java is its portability. It is because of javac compiler compiles source code to bytecodes, which are not native to a normal processor, however with the usage of Java Virtual Machine can bytecodes run on every computer[13].

3.5.1.2 Spring

Spring or fully Spring Framework makes it easier and saves time to create Java applications as it provides nearly everything that a Java developer needs. The Spring Framework is open-sourced and divided into modules. Therefore, developers may choose, which modules they need. Currently, there are about 20 available modules. Not to mention, it is considered secure, flexible and low-cost[14].

3.5.1.3 Swagger

Swagger is an open-source framework for designing and creating documentation and interaction with the API's resources without having any implementation logic done, because it is automatically generated from OpenAPI(formerly known as Swagger) specification[15].

3.5.1.4 Project Lombok

Project Lombok is a very useful library and saves a lot of time and code space in a project. With the usage of annotation in Java classes, Lombok knows, what is left to generate, therefore user does not need to code boilerplate code every time he create a class or modifies an existing one[16]. For example:

- *@Getter* - generates getters for an annotated class
- *@Setter* - generates setters for an annotated class

3.5.1.5 Tesseract

Tesseract, an OCR engine, was originally developed at Hewlett-Packardsince 1985, then it was open-sourced. Later Google took over and is still working on it. Tesseract was code using C++, which makes it a little difficult with Java. However, there are libraries that wrapped C++ Tesseract and made it easier to use it. Therefore, Tesseract could be a great help in EMDS application[17].

3.5.1.6 Docker

Docker is an open platform used for developing, shipping and running applications. It helps you separate applications from infrastructure, so it can be delivered quicker. In addition, Docker provides an ability to package and run an application in a loosely isolated environment named as a container[18].

3.5.2 Back-end structure

Being aware of what technologies are going to be used in the back-end, it would be great to have prepared suggestions for back-end structure.

Considering the non-functional requirements of being integrable and reusable, I would suggest the back-end being separated into two parts. Visualisation available in Figure 3.13.

- **OCR-API** - A Java application integrated with an OCR engine Tesseract to take care of document scanning logic, accessible via REST API endpoint.

- **Back-end** - A Java application taking care of communication between the front-end, OCR-API and a database.

Separating Java application into two parts, as suggested above, will fulfill the non-functional requirements of being integrable and reusable. To push it one step further, it will be an undeniable improvement for reusability, if every component could be dockerized.

3.5.2.1 OCR-API

OCR-API is a component that will be integrated with an OCR engine. To be more specific, it will be Tesseract. Besides, it should be able to have exposed REST API endpoints for receiving new images/documents that should be scanned. Moreover, it has to be able to return the extracted text from scanned documents.

Considering the mentioned points, it should consist of three layers that have separated concerns:

- **REST API layer** - exposing endpoints for new work assignments and communication

- **Business layer** - takes care of a coordination for text extraction and other computations

- **Data layer** - data that are necessary to persist

3.5.2.2 Back-end

As mentioned, the back-end will be an intermediary that takes care of:

- Front-end communication
- Sending and retrieving scanned documents from OCR-API
- Enforcing authorized access to give documents at least basic security
- Persistence of data to a database

Therefore, it will have a similar structure to OCR-API with a different specification for each layer.

- **REST API layer** - exposing endpoints for communication with Front-end
- **Business layer** - will take care of communication with OCR-API and it will be checking, if there are documents that needs to be scanned via OCR-API.
- **Data layer** - persisting data to a database

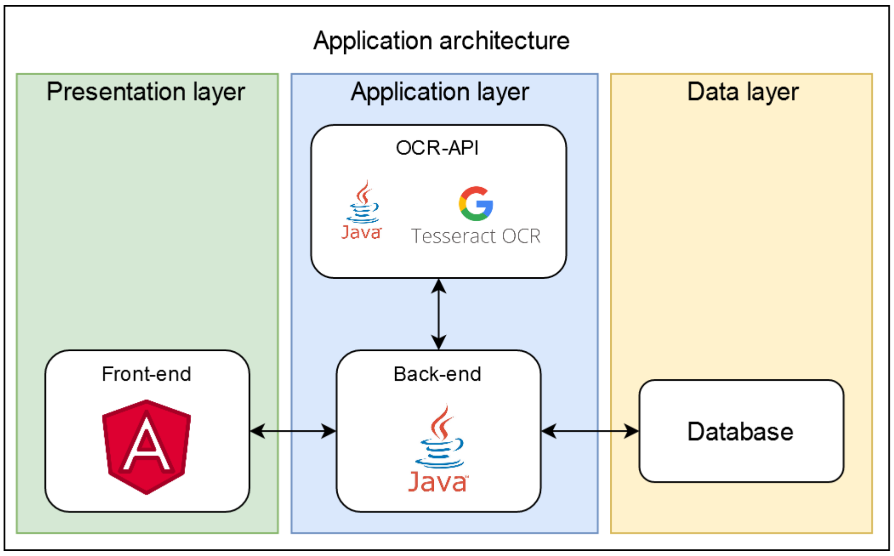


Figure 3.13: Possible structure of the new application

3.5.3 Database

To persist data, most of the existing solution use relational databases as Oracle, SQL Server, MySQL and others. In the constrast to them, I decided to go with NoSQL database called MongoDB that support many interesting features. The most important ones were search operations in text, storing data as JSON and dynamic schema design. The following parts will describe MongoDB in the futher details.

MongoDB is a NoSQL database is non-tabular and stores data differently than a relational database. To be specific, MongoDB is a document-oriented database with scalability and flexibility. Moreover, MongoDB offers far more features, but from all of the following are most important for this project:

- Dynamic schema design - changes in the back-end, does not need to be also fixed in a database
- Supported by Spring - avoids time demanding the creation of repositories
- Data representation in JSON - ideal for saving text after scanning and also retrieving
- Supports text search

According to MongoDB, it also provides ACID transactions for documents. Considering all the mentioned features, it is all I need for development of a prototype of the application, without loosing time to take care of the database and creating object's tables and columns in it[19].

The database was the only component of the application architecture not being defined in this chapter. As it is already resolved, the applications final architecture can be seen in Figure 3.14, where are filled all base technologies that will be used for corresponding components.

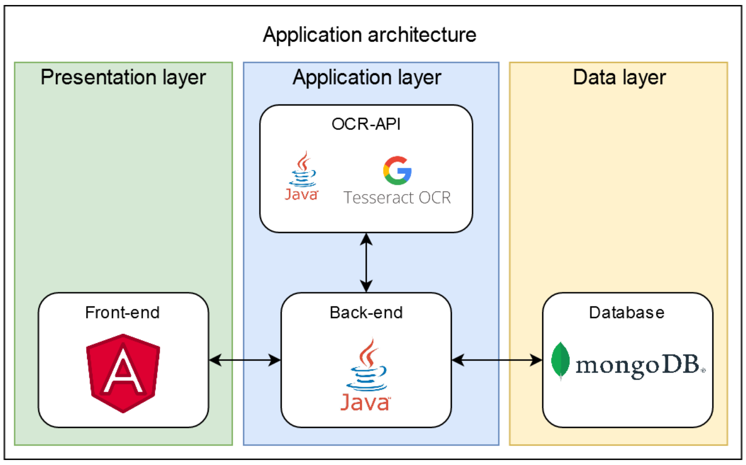


Figure 3.14: Final architecture of the application

Chapter 4

Implementation

Accordingly to the design chapter, I proceed further with the implementation of the application. I took into account that Front-end will be using Angular framework, Back-end will use Java and it should be separated into two components to achieve better reusability. Additionally, a database will be realized by MongoDB.

4.1 OCR-API

OCR-API was implemented as it was suggested in the design chapter. It is separated from Back-end as it enables others to use OCR as a prepared component alone, in a case they only need OCR with API. I will explain more about implementation in the following subsections designed for each component.

OCR-API consists of three layers as suggester in the design:

- REST API layer
- Business layer
- Data layer

4.1.1 REST API layer

The REST API layer consists of controllers that expose API endpoints to others to interact with OCR-API. OCR-API consist of two controllers:

- *ImgDocumentController*
- *PDFController*

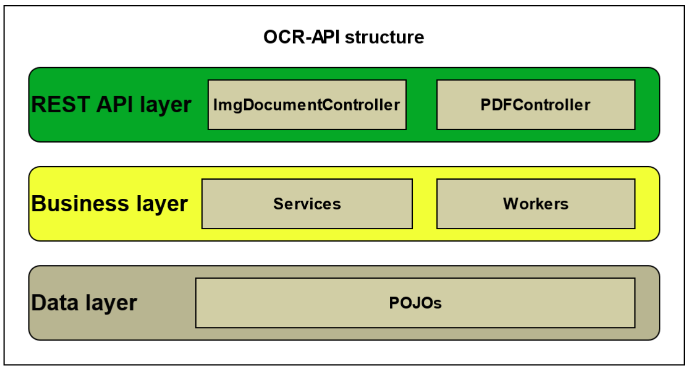


Figure 4.1: Structure of OCR-API

4.1.1.1 ImgDocumentController

ImgDocumentController is intended for work with file formats:

- *JPG*
- *PNG*
- *TIFF*

The *JPG*, *PNG* and *TIFF* are supported only in *ImgDocumentController*, because these are only formats that are supported by Tesseract.

Additionally, a user has an option to upload files/documents via an asynchronous or synchronous POST request. In the first case, the user sends a file to process and OCR-API saves the file physically and returns to the user object *DocumentAsyncStatus* that contains:

- Current state of the uploaded file/document that can be *processing* or *scanned*
- A status link to check the current state of the file
- A link to a result of the scanned file

In a case, where a returned object has state *processing*, a user has to keep asking OCR-API on the status link, if the extraction of text is finished. When it is finished, the user uses the link to the result of the scanned file, where he receives the extracted text. There is a diagram visualising the described process in Fig. 4.2.

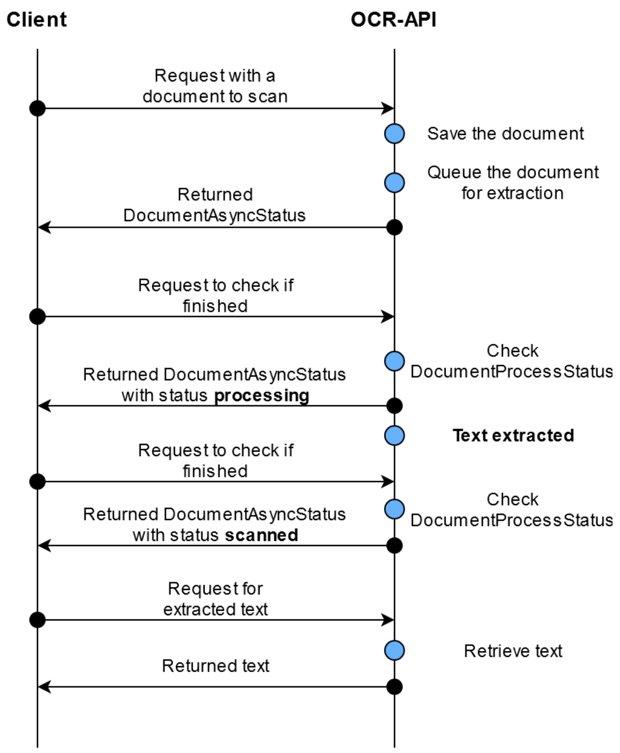

Client
OCR-API
Request with a document to scan
Save the document
Queue the document for extraction
Returned DocumentAsyncStatus
Request to check if finished
Check DocumentProcessStatus
Returned DocumentAsyncStatus with status **processing**
Text extracted
Request to check if finished
Check DocumentProcessStatus
Returned DocumentAsyncStatus with status **scanned**
Request for extracted text
Retrieve text
Returned text

Figure 4.2: Example of an asynchronous communication

The second option is for a user to use a synchronous request. In comparison to the previous method, the user has to just send a request with a document and wait throughout the whole process of text extraction from the document. In the case of sending more than one document or sending a document with multiple pages, it may take quite a while. As well, as in the first case, there is visualization in Figure 4.3. Even though it looks much more straightforward, it is not as efficient as asynchronous communication.

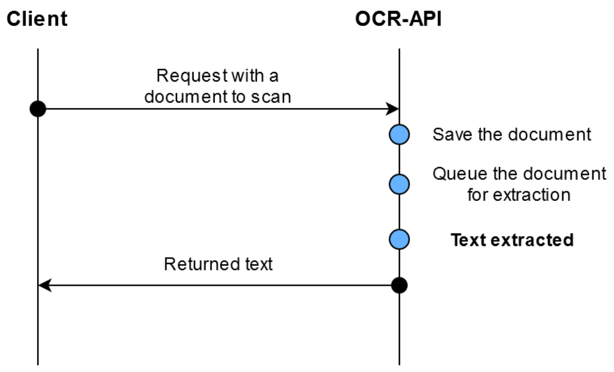
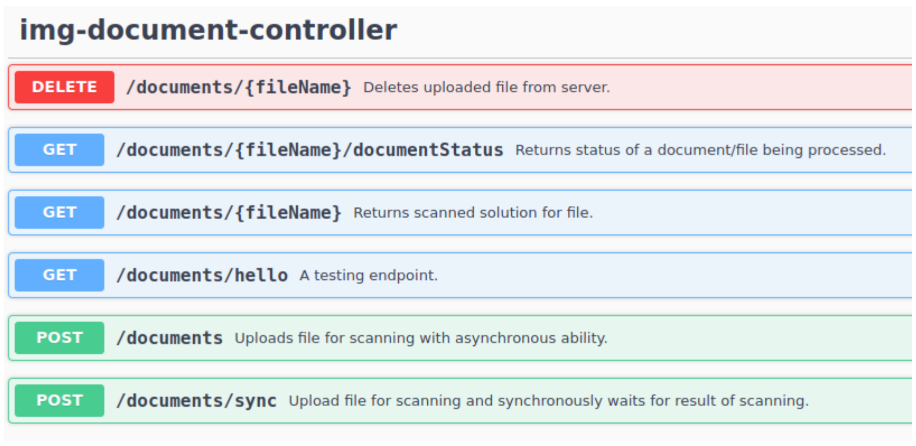

Client
OCR-API
Request with a document to scan
Save the document
Queue the document for extraction
Text extracted
Returned text

Figure 4.3: Example of a synchronous communication

The *ImgDocumentController* has additional four exposed methods:

- Get document status - to get the current status of the document
- Get scanning result
- Delete uploaded file
- Get test method to check availability of OCR-API

For better understanding, I recommend to see Figure 4.4 that was generated via Swagger.



img-document-controller

DELETE /documents/{fileName} Deletes uploaded file from server.

GET /documents/{fileName}/documentStatus Returns status of a document/file being processed.

GET /documents/{fileName} Returns scanned solution for file.

GET /documents/hello A testing endpoint.

POST /documents Uploads file for scanning with asynchronous ability.

POST /documents/sync Upload file for scanning and synchronously waits for result of scanning.

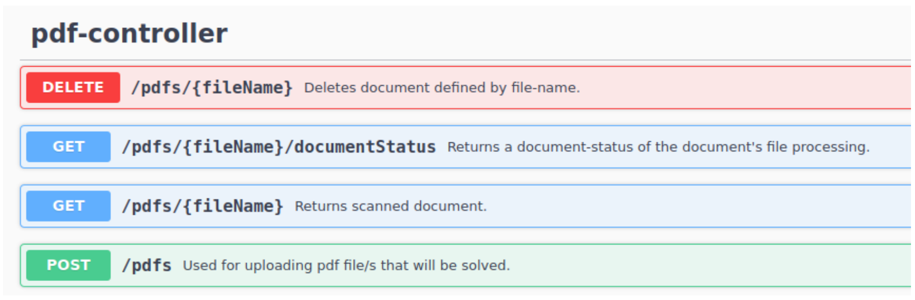
Figure 4.4: Image and document controller visualization using Swagger

4.1.1.2 PDFController

PDFController is intended as the name suggests for *PDF* files. The reason, why it is separated from *ImgDocumentController*, it is because a *PDF* file format is not supported by Tesseract and therefore, it is needed to do preprocessing on the *PDF* files before they can be sent straight to *OcrService* that takes care of the extraction of text from the document. Therefore, I separated controllers to two different ones.

PDFController has similar exposed methods, but it is missing a synchronous request method for scanning PDF files. The reason behind is, that usually *PDF* files contain more than one page and logically, more pages mean more time spent for text extraction that led me to remove asynchronous communication for *PDF* files scanning.

As in *ImgDocumentController*, *PDFController* also have Figure 4.5 showing exposed methods generated by Swagger that are very similar to *ImgDocumentController*.



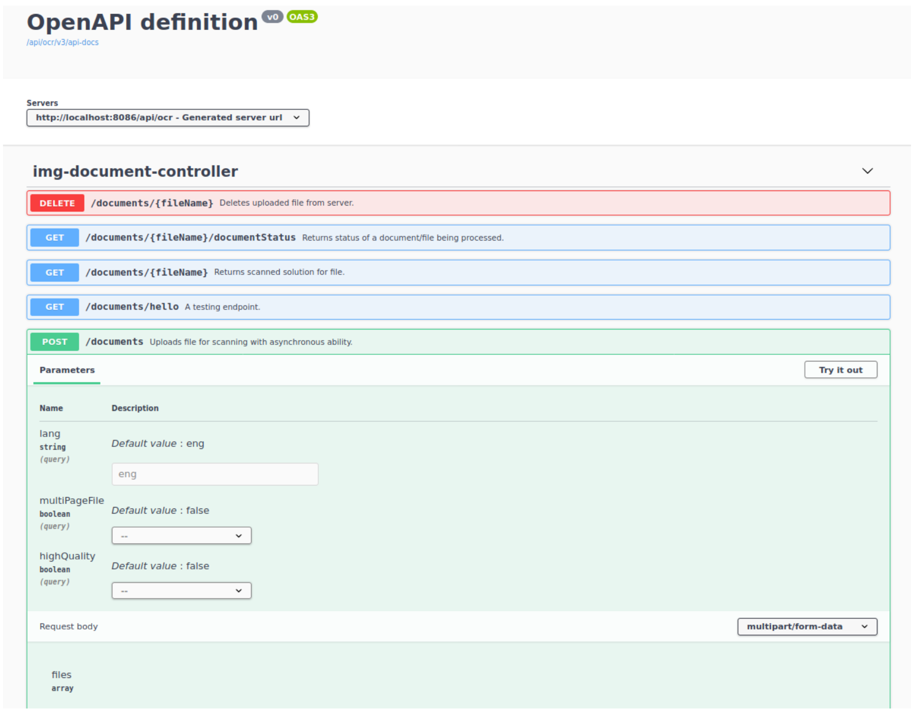
pdf-controller

DELETE	`/pdfs/{fileName}`	Deletes document defined by file-name.
GET	`/pdfs/{fileName}/documentStatus`	Returns a document-status of the document's file processing.
GET	`/pdfs/{fileName}`	Returns scanned document.
POST	`/pdfs`	Used for uploading pdf file/s that will be solved.

Figure 4.5: PDF controller visualization using Swagger

4.1.1.3 Swagger-UI

For all named controllers, there is also available automatically created API documentation that is provided using Swagger-UI. As a bonus to the API documentation, there is also an option for a user to interact with API. Visualization of how it looks can be seen in Figure 4.6.



OpenAPI definition v0 OAS3
/api/ocr/v3/api-docs

Servers
http://localhost:8086/api/ocr - Generated server url

img-document-controller

DELETE	`/documents/{fileName}`	Deletes uploaded file from server.
GET	`/documents/{fileName}/documentStatus`	Returns status of a document/file being processed.
GET	`/documents/{fileName}`	Returns scanned solution for file.
GET	`/documents/hello`	A testing endpoint.
POST	`/documents`	Uploads file for scanning with asynchronous ability.

Parameters   Try it out

Name	Description
lang string *(query)*	*Default value* : eng eng
multiPageFile boolean *(query)*	*Default value* : false --
highQuality boolean *(query)*	*Default value* : false --

Request body   multipart/form-data

files
array

Figure 4.6: Swagger-UI for OCR-API

4.1.2 Business layer

The business layer in OCR-API is a layer that contains all the work logic behind OCR-API. It consists of services and workers. Services in OCR-API are:

- *DocumentStorageService* - used as virtual storage to keep information about currently processed documents
- *DocumentService* - used for distribution of work and processing received documents from *ImgDocumentController*
- *PDFService* - used for work with *PDF* files, where it reads *PDF* file via PDFBox library then saves it as *PNG* file
- *OCRService* - used for extraction of text from files using OCR engine Tesseract
- *FileStorageService* - used for work with received files - saving, reading, deleting

Moreover, OCR-API also contains workers. To be specific, there are exactly two workers that take care of text extraction from files. Workers found in OCR-API are:

- *DocumentJobWorker* - a worker that has specific job to do, when there is a file assigned to it
- *PDFJobWorker* - a worker that has specific job to do, when there is a *PDF* file assigned to it

In the following subsections will be more explanation about important classes that were mentioned.

4.1.2.1 DocumentStorageService

DocumentStorageService is a virtual storage that contains two concurrent maps:

1. Contains documents
2. Contains current state of document

There are used to access, update or get current state of the document by workers or controllers, if a user wants to know current state of the document.

4.1.2.2 DocumentService

DocumentService is a service that receives supported document formats as *PNG*, *JPG*, *TIFF*. In general, when it is called by *ImgDocumentController*, it will first save document to a physical disk and then it will create worker of a type *DocumentJobWorker* that will be added to task pool. *DocumentJob-Worker* will take care of the extraction, while *DocumentService* will return to the controller that the document is assigned.

4.1.2.3 PDFService

PDFService is very similar to *DocumentService*. However, as mentioned *PDF* format is not supported by Tesseract. Therefore, it creates *PDFJobWorker* to task pool instead of *DocumentJobWorker*. It may seem that there was no need for *PDFJobWorker* as it could be easily done in *DocumentJobWorker*. The reason for separation was to make clear that there are two different approaches and that using *PDF* document is not as reliable as the supported formats.

4.1.2.4 OCRService

OCRService is the most important service that uses wrapped OCR engine Tesseract for extraction of data from a document. Additionally, it also contains a part that adds new languages for use and it is done automatically, if it is required by configuration provided with a document.

However, there are additional steps to do, if a language is not English, Czech or Slovak:

- Extend language checker in controller with the language that is required
- Don't forget to add traindata[2] for the language

4.1.2.5 Workers

Workers are essential in asynchronous communication. They work independently from controllers as they have their own thread pool. Whenever they are created, they are put inside the thread poll with a task. When their turn comes, they will start computing. Let me also put forward that currently, it is allowed only for one active thread. Other tasks have to wait in it for their turn.

DocumentJobWorker and *PDFJobWorker* have many similar parts as to extract text from a document and change the document's status. However, *PDF* is not supported by Tesseract. Therefore, before it can be scanned, *PDFJobWorker* has to first split the *PDF* document into documents with only one page. Afterwards, it needs to convert them one by one to *PNG*

[2]https://github.com/tesseract-ocr/tessdata

format. When the conversion is done, *PDFJobWorker* can proceed with the same routine, but in the end, it has put all pages into one document.

4.1.3 Data layer

The data layer is usually used for objects that are being persisted. In OCR-API, the only data that are persisted are received files. They are saved to file storage to avoid jamming memory with the file's data.

The plain old java objects (further POJOs) that are being used in OCR-API are:

- *Document* - containing document data
- *DocumentAsyncStatus* - document asynchronous status returned to a requester
- *OcrConfig* - settings for OCR, currently used only inside OCR-API
- *DocumentProcessStatus* - state of document is processing, when is being processed and scanned, when it is completed

To see exactly what named POJOs are containing, I recommend to see Figure 4.7.

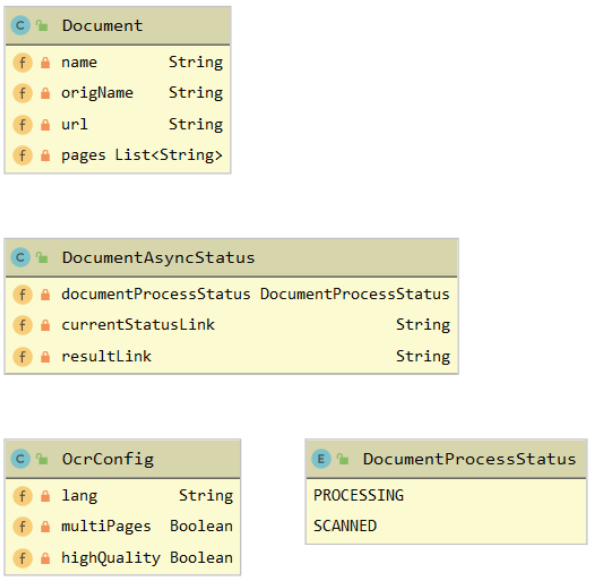


Figure 4.7: POJOs in OCR-API

4.2 Back-end

Back-end, as well as OCR-API, goes with three suggested layers:

- REST API layer
- Business layer
- Data layer

However, there are some differences that I will elaborate about in the following parts, but in Fig. 4.8 can be seen, what those three layers use in comparison to the OCR-API structure.

Before going further, I'd like to point out one important matter. In the implementation in OCR-API, I was using the word "document" in created object names. However, while implementing the data layer in the Back-end, I realized that annotation for entity in JAVA using MongoDB is "@Document". This conflicted with usage from OCR-API, therefore all namings were changed from "document" to "doc" in Back-end.

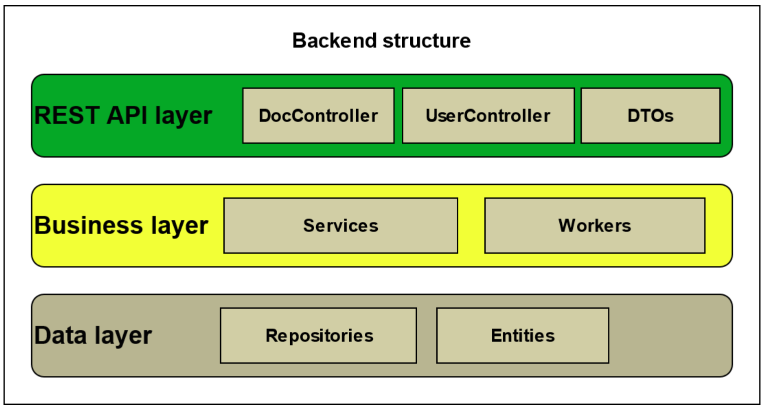


Figure 4.8: Structure of Back-end

4.2.1 REST API layer

Back-end component is consisting of two controllers that are exposing API endpoints for interaction with Back-end. The controllers are:

- *UserController*
- *DocController*

4.2.1.1 Access rights

Before there are described existing controllers, it is necessary to point out, how documents are protected against an unauthorized action, described in the following part.

Uploaded documents are belongings of a user that uploaded them. The user can share them with others, but it only enables everyone else to view them. However, they are not able to modify them in any way.

To enforce access rights, some form of access code was needed. To solve this issue, JSON Web Token(JWT) was used. JWT is an open-source standard (RFC 7519), which defines a compact and self-sufficient approach to securely transmit information between parties as a JSON object. The information is credible, because it is digitally signed[20].

Therefore, when a user wants to access a document, he has to include a JWT that was generated for him in the request for accessing the document. In a case he doesn't have any or JWT validity expired, the user can get a new one by authorizing. A diagram describing the usage of JWT can be seen in Figure 4.9.

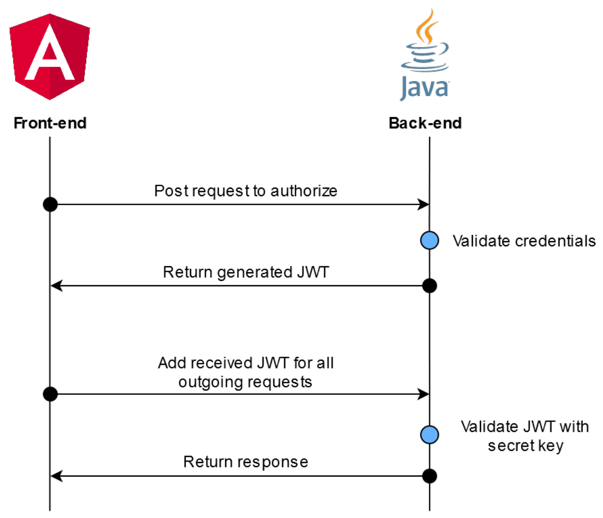

Front-end
Java
Back-end
Post request to authorize
Validate credentials
Return generated JWT
Add received JWT for all outgoing requests
Validate JWT with secret key
Return response

Figure 4.9: Communication using JWT

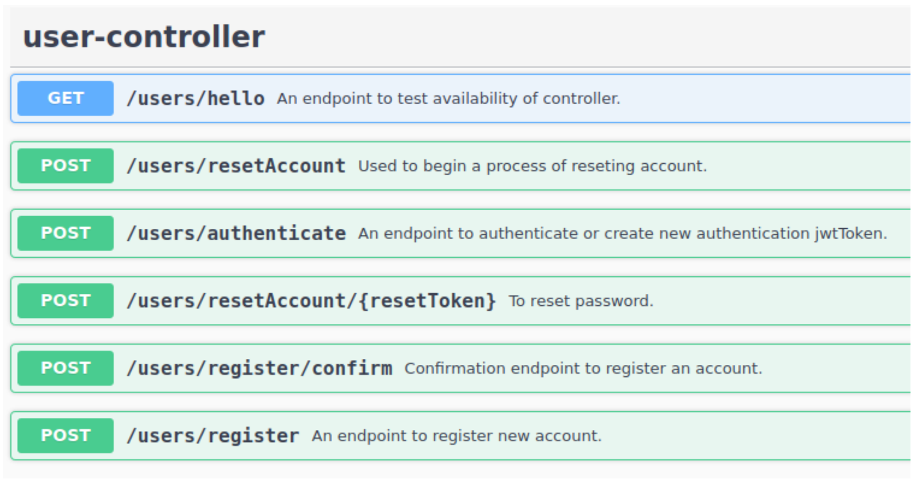
4.2.1.2 UserController

The User-Controller, as the name suggests, is aimed for user operations and therefore, access to them is available to everyone without JWT verification. Exposed operations are the following:

- User registration
- Registration confirmation

- User authentication - grants a new JWT
- Reset account
- Change password after reset

The mentioned operations are visualized in Figure 4.10. *UserController* also contains GET method for testing the availability of Back-end.



user-controller

GET /users/hello An endpoint to test availability of controller.

POST /users/resetAccount Used to begin a process of reseting account.

POST /users/authenticate An endpoint to authenticate or create new authentication jwtToken.

POST /users/resetAccount/{resetToken} To reset password.

POST /users/register/confirm Confirmation endpoint to register an account.

POST /users/register An endpoint to register new account.

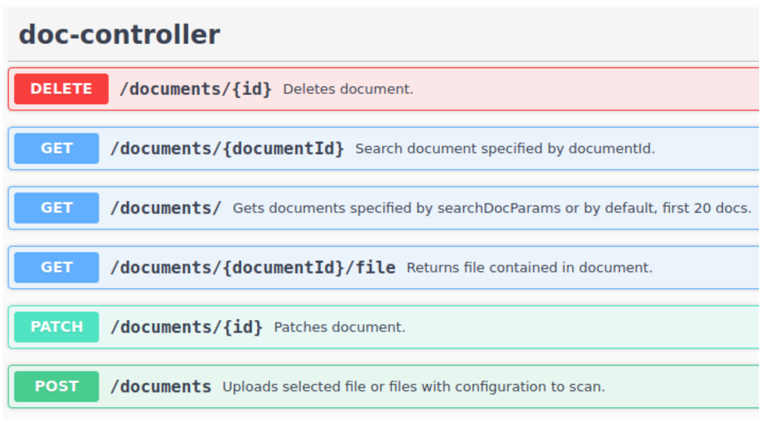
Figure 4.10: User controller visualization using Swagger

4.2.1.3 DocController

DocController is exposing endpoints that control operation with documents. However, every request has to contain JWT that is valid and operation that the requester wants to do, has to be available to the user. For example, user A cannot delete a document that belongs to user B. Exposed endpoints operations are:

- Delete a document
- Get a document
- Get paged documents by default, by specified/advanced parameters or by full-text
- Get file belonging to document
- Patch document
- Upload documents/files

All mentioned API endpoint above, are visible in Fig. 4.12.


doc-controller

DELETE /documents/{id} Deletes document.

GET /documents/{documentId} Search document specified by documentId.

GET /documents/ Gets documents specified by searchDocParams or by default, first 20 docs.

GET /documents/{documentId}/file Returns file contained in document.

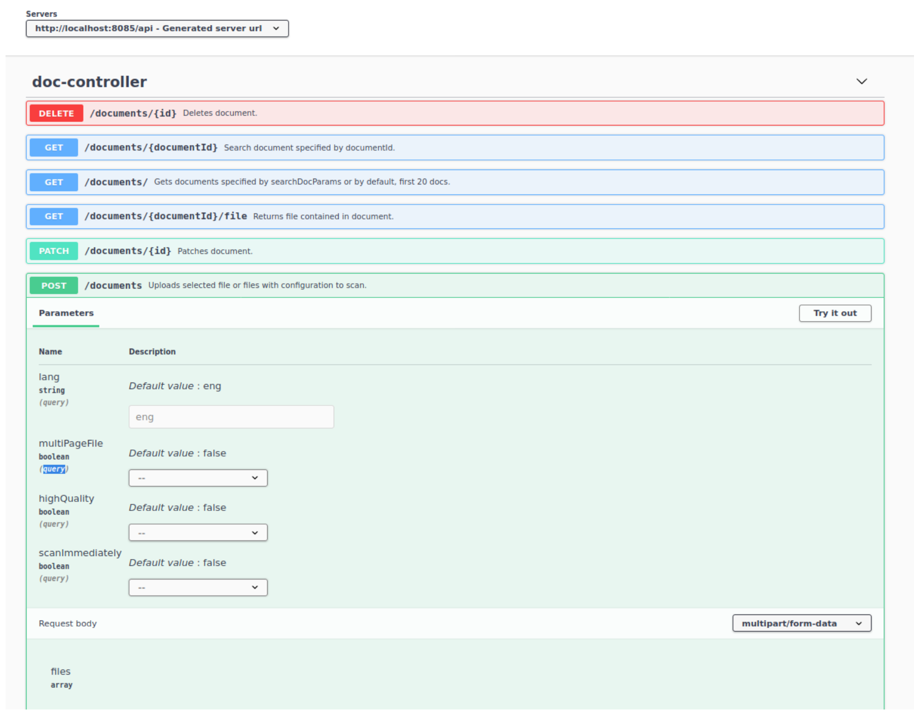
PATCH /documents/{id} Patches document.

POST /documents Uploads selected file or files with configuration to scan.

Figure 4.11: Doc controller visualization using Swagger

4.2.1.4 Swagger-UI

As well as OCR-API, Back-end also contains documentation generated by Swagger and in addition, it also provides a possibility to interact with API. Visualization of how it looks can be seen in Fig. 4.12.


Servers
http://localhost:8085/api - Generated server url

doc-controller

DELETE /documents/{id} Deletes document.

GET /documents/{documentId} Search document specified by documentId.

GET /documents/ Gets documents specified by searchDocParams or by default, first 20 docs.

GET /documents/{documentId}/file Returns file contained in document.

PATCH /documents/{id} Patches document.

POST /documents Uploads selected file or files with configuration to scan.

Parameters — Try it out

Name	Description
lang string (query)	Default value : eng eng
multiPageFile boolean (query)	Default value : false --
highQuality boolean (query)	Default value : false --
scanImmediately boolean (query)	Default value : false --

Request body — multipart/form-data

files
array

Figure 4.12: Swagger-UI for Back-end

4.2.2 DTOs

Lastly, the REST API layer contains Data Transfer Objects(DTOs), which are trimmed data objects containing only necessary parts needed on Front-end. Currently, there are 2 DTOs:

- *UserDto* - returned only after registration to make sure that correct data were set
- *DocDto* - returned every time there is a request for a document or documents

To see what exactly is inside DTOs, see Fig. 4.13.

4.2.2.1 DocDto

DocDto is the most used DTO class. It is send forth and back between communication of Front-end and Back-end.

On the route from Back-end to Front-end is send everything from *Doc* object, except document's file to minimize communication size.

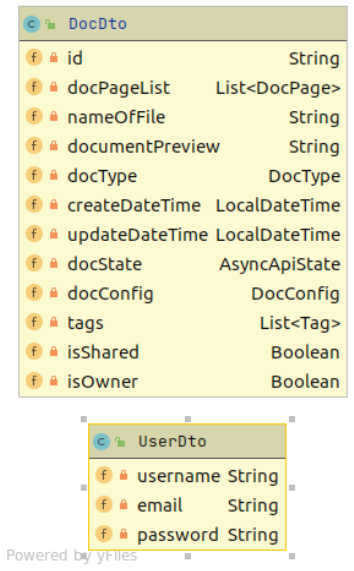
The other way around from Front-end to Back-end is the situation is a lot different. To assure that users won't break the application apart, the following properties are ignored on the conversion:

- Name of the file
- Owner
- Document's file
- Document's preview

4.2.3 Business layer

As well as in OCR-API business layer, a business layer in Back-end contains logic behind Back-end and does required computation. The business layer contains in general two groups:

- Services
- Workers



DocDto	
id	String
docPageList	List<DocPage>
nameOfFile	String
documentPreview	String
docType	DocType
createDateTime	LocalDateTime
updateDateTime	LocalDateTime
docState	AsyncApiState
docConfig	DocConfig
tags	List<Tag>
isShared	Boolean
isOwner	Boolean

UserDto	
username	String
email	String
password	String

Figure 4.13: DTOs used in Back-end

4.2.3.1 Services

Services are classes with methods that work in a specific domain. For example *DocService* works with docs - documents. In Back-end are multiple services. For documents are used following:

- *DocService* - used for work with documents
- *VirtualStorageService* - as the name suggest, virtual storage is virtual storage with concurrent hash map to know, which document are being scanned to avoid adding one document more than once to a scanning queue and also if they failed, including how many times
- *OCRService* - used for wrapping a whole communication with OCR-API
- *RestApiOcr* - prepared specific REST API methods for every necessary endpoint used in OCR-API

While working with users in Back-end component exists following services, where every one of them has specific reason described:

- *ConfirmationTokenService* - used for work with *ConfirmationTokens* generated while creating an account for user
- *ResetTokenService* - used to work with *ResetTokens* intended for user, when they forget their password or username

- *UserService* - used for work with Users
- *EmailService* - used for e-mailing

EmailService has a significant meaning for authorization of a user. To create a new account, the user has to register a new account using an e-mail address. After the registration, the user will receive and a confirmation e-mail to activate the account. Moreover, the activation will be done by *ConfirmationTokenService*. However, there is a catch to *EmailService* for the system to use it. Before the execution of the Back-end, it is necessary to provide an e-mail account with a password. This topic will be described later in a chapter dedicated to the execution of the application.

4.2.3.2 Workers

Workers are a specific part of the business layer in Back-end component. They do their work in a separate thread from the existing logic. In Back-end can be found these two workers:

- *DocOcrChecker* - a job that is currently set to be called repeatedly every 1 minute to check, if there are documents to be processed. If there are, they will be sent to a queue to be processed. A code of the *DocOcrChecker* called every 1 minute can be seen in listing 4.1.
- *OcrApiJobWorker* - a job created by *DocOcrChecker* for every not finished document and put into a queue. The moment it gets the priority, it will use *OCRService* to start communicating with OCR-API to do work that is left to do.

Listing 4.1: Automatically executed code to look for the unscanned documents.

```
// 2min -> 120000milis
@Scheduled(fixedDelay = 60000)
public void checkUnscannedDocs() {
  log.info("Started DocumentOcrChecker!");

  List<Doc> cleaningDocs =
      documentRepository
        .findDocumentsByAsyncApiInfoAsyncApiState(
          AsyncApiState.RESOURCE_TO_CLEAN
        );

  // find processed -> to download
  List<Doc> scannedDocs =
      documentRepository
```

```
            .findDocumentsByAsyncApiInfoAsyncApiState(
              AsyncApiState.SCANNED
              );

    // check status
    List<Doc> processingDocs =
        documentRepository
          .findDocumentsByAsyncApiInfoAsyncApiState(
            AsyncApiState.PROCESSING
          );

    // find to_be_send -> to process not yet
       processed
    List<Doc> waitingToSendDocs =
        documentRepository
          .findDocumentsByAsyncApiInfoAsyncApiState(
            AsyncApiState.WAITING_TO_SEND
          );

    List<Doc> documentsWork = new
       ArrayList<>(scannedDocs);
    documentsWork.addAll(processingDocs);
    documentsWork.addAll(waitingToSendDocs);
    documentsWork.addAll(cleaningDocs);

    // filtering documents already in process
    documentsWork =
        documentsWork.stream()
            .filter(document ->
               !virtualStorageService
              .isDocUsed(document.getId())
            )
            .collect(Collectors.toList());

    // add docs to virtual storage
    documentsWork.forEach(document ->
       virtualStorageService
      .addDoc(document.getId()));

    documentsWork.forEach(
        document1 ->
            taskExecutor.execute(
                new OcrApiJobWorker(
                  beanFactory
```

```
                            .getBean(OCRService.class)
                         docService,
                         documentRepository,
                         virtualStorageService,
                         document1
                         )
                )
        );

        log.info("Done executing works.");
    }
```

4.2.4 Data layer

As it can be seen in Fig. 4.8, the Back-end's data layer consists mainly of 2 blocks:

- Repositories - interfaces that enable Back-end to retrieve, delete and save data to MongoDB
- Entities - objects that can be persisted inside of MongoDB

4.2.5 Repositories

Repositories are the main parts that are used for communication with the database. When working with Java and Spring, one can easily use Spring Data Java Persistence API that gives a user implementation of repositories that contains all CRUD operations - create, read, update and delete. Moreover, the user has a chance to use his own implementation if he desires to. For example, *DocCustomRepository* is customized repository using MongoDB, where *DocCustomerRepositoryImpl* is its implementation. It contains implementation for two methods:

- *findDocsByMultipleArgs* - a method, which is used to search for documents using multiple arguments that the searched documents should contain as a certain tag, status, language and others
- *findDocsByFullText* - a method, which is using a provided string parameter that says, what a document or documents should contain and finds them using a full-text search

Afterwards, *DocRepository* implements *DocCustomRepository*. The existing repositories can be seen in Figure 4.14.

Used repositories in Back-end:

- *DocCustomerRepository* - a customized repository for searching in MongoDB using multiple search settings
- *DocRepository* - a repository that access collection documents
- *ResetTokenRepository* - a repository that access collection of *ResetTokens*
- *ConfirmationTokenRepository* - a repository that access collection of *ConfirmationTokens*
- *UserRepository* - a repository that access collection of *Users*

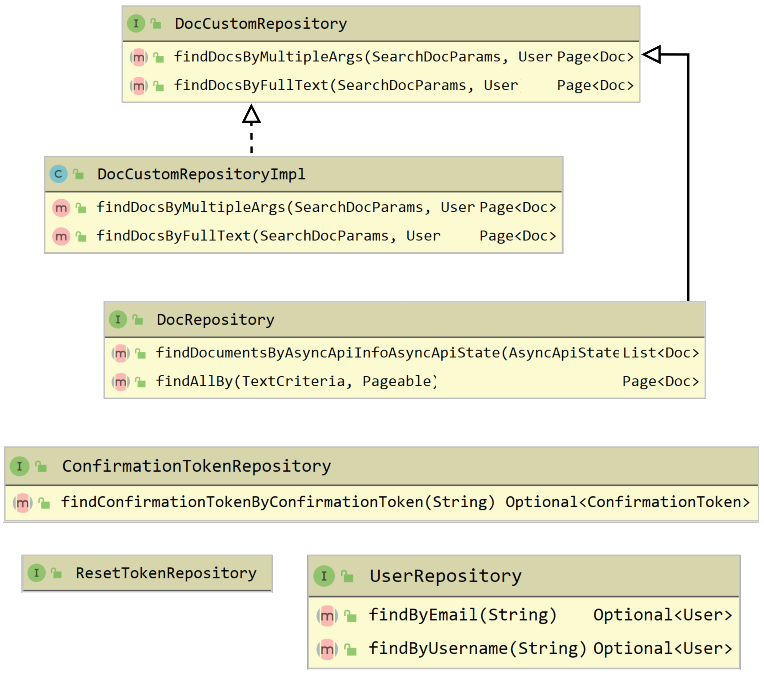

DocCustomRepository
findDocsByMultipleArgs(SearchDocParams, User Page<Doc>
findDocsByFullText(SearchDocParams, User Page<Doc>
DocCustomRepositoryImpl
findDocsByMultipleArgs(SearchDocParams, User Page<Doc>
findDocsByFullText(SearchDocParams, User Page<Doc>
DocRepository
findDocumentsByAsyncApiInfoAsyncApiState(AsyncApiState List<Doc>
findAllBy(TextCriteria, Pageable) Page<Doc>
ConfirmationTokenRepository
findConfirmationTokenByConfirmationToken(String) Optional<ConfirmationToken>
ResetTokenRepository
UserRepository
findByEmail(String) Optional<User>
findByUsername(String) Optional<User>

Figure 4.14: Schema of existing repositories

4.2.6 Entities

Entities are objects that can be persisted in MongoDB and they are also objects that are retrieved from MongoDB via repositories.

Generally, there are two groups in entities that have strong relations.

- User entities
- Doc entities

User entities consist of four classes: *User*, *ConfirmationToken*, *ResetToken* and *UserRole*. Their definition can be seen in Figure 4.15.

Doc entities, in comparison to User entities, consist of seven classes. From these seven, there are two enumerations and two classes that contain the only property of type string. They were created with the foresight that in the future, they may be extended with additional properties. The list of doc entities is the following:

- *Doc* - represents a document object
- *AsyncApiStat* - represents a state of scanning
- *AsyncApiInfo* - a document state from OCR-API
- *DocConfig* - a configuration for scanning
- *DocType*
- *Tag* - currently consist of an description
- *DocPage* - represents a page per file-page

Doc entities specific properties can be seen in Figure 4.15.

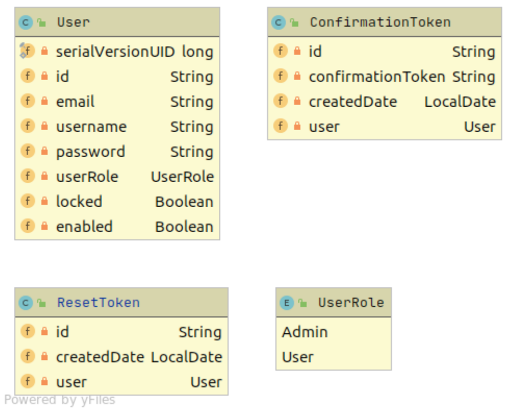


Figure 4.15: Back-end entities with relation to the *User* class

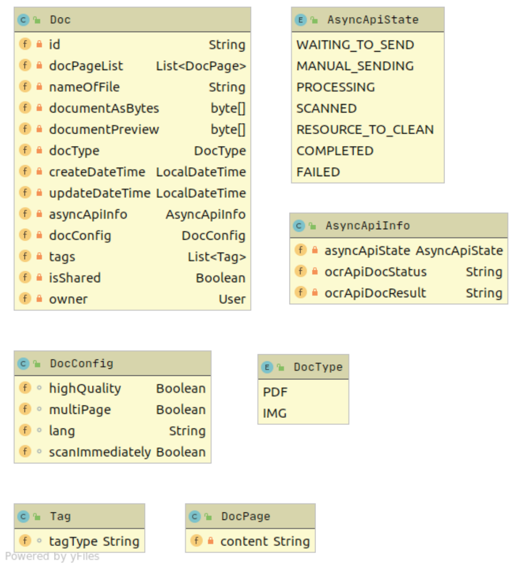


Figure 4.16: Back-end entities with relation to the *Doc* class

4.3 Front-end

An application implemented via Angular framework is built on components. Every component's base is a class to which is bound HTML template with CSS style using directives. Thanks to its logic, a page can be split into multiple components and again merge them together. An image describing Angular architecture can be seen in Figure 4.17.

Front-end logic is hidden inside services that are injected into components using a technique - Dependency Injection.

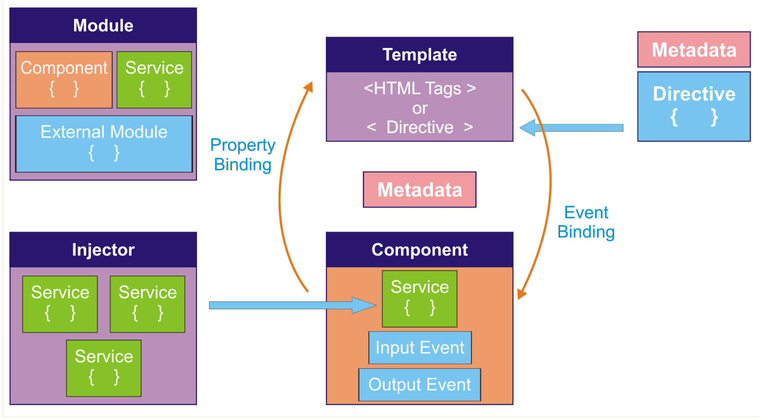


Figure 4.17: Angular architecture [7]

4.3.1 Components of Front-end

Components of Front-end can be separated generally in four groups using designed main pages with one extra group:

- Components of login
- Components of documents
- Components of import
- Rest of components

4.3.1.1 Login page

The login page or also, in other words, the authentication page contains actions as login, register and account reset. A part of the implemented login page can be seen in Figure C.3, located in the appendix. The components that stand behind the design of the login are the following:

- *Login* - a component is displaying login
- *ChangePassword* - a component is displaying page for password change
- *PasswordResetContainer* - a component container for components *ChangePassword* and *PasswordReset*
- *PasswordReset* - a component is displaying option to reset an account
- *Register* - a component is displaying page for registration of a new user
- *Auth* - a component container for user components mentioned above

4.3.1.2 Page for documents

A page for documents is the most robust one out of the three groups. The reason for the robustness is the number of activities available on the page. For example, a user can view documents, search documents using different approaches, modify them and lastly delete them. For the curious readers, there is an available example of the page for documents in Figure D.3 in the appendix. The components that belong to the documents are:

- *AsyncApiInfo* - a component is displaying *AsyncApiInfo*
- *DocConfig* - a component is displaying *DocConfig*
- *DocumentExtendedView* - a component is displaying additional details about document, when a user click on document row

- *DocumentEdit* - a component is displaying window that enables a user to partially modify document
- *DocumentPreview* - a component is displaying a document image thumbnail
- *DocumentTags* - a component is displaying tags of a document
- *Pages* - a component is displaying text-pages of a document
- *Documents* - a component containing tab documents

4.3.1.3 Import page

The last main page is for importing documents. It consists of only one component:

- *ImportDocuments* - a component containing tab for importing documents

Its illustration is located in Figure D.2.

4.3.1.4 Common components

Components that may be used across above mentioned groups belong to this group. The common components are:

- *Header* - a component is displaying header of Front-end
- *Message* - a component is displaying a message after communication with Back-end that may be informative, warning or even error message

4.3.2 Services of Front-end

In simplified manners, a service is an object that contains only computing logic that may be used across all components. Services that are being used in Front-end are:

- *Auth* - a service that is taking care of a user authentication
- *BasicAuthHttpInterceptor* - a service that intercepts every HTTP request made by Front-end and concat to it JWT, if a user is authenticated
- *Document* - a service that takes care of communication between Front-end and Back-end, when there is document request such as modifying, searching or deleting
- *FileUpload* - a service that takes care of file upload to Back-end
- *Message* - a service that shows a message after being triggered

4.4 Docker

The last part of the implementation chapter is the dockerization of created components. To dockerize an application, a user has to fulfill the following steps:

1. Prepare a build and working application.
2. Find a docker image that can be used as a basis for the whole the application.
3. Write a Dockerfile.
4. Build image of the application.

Additionally, to ease development and usage application, I used a useful tool at Dockerhub[3] that connects to a repository of the source-code, builds the application, creates an image automatically and the built image is online available for usage. I used this approach for all three components that are currently visible at URL[4].

Images used for respective components were the following:

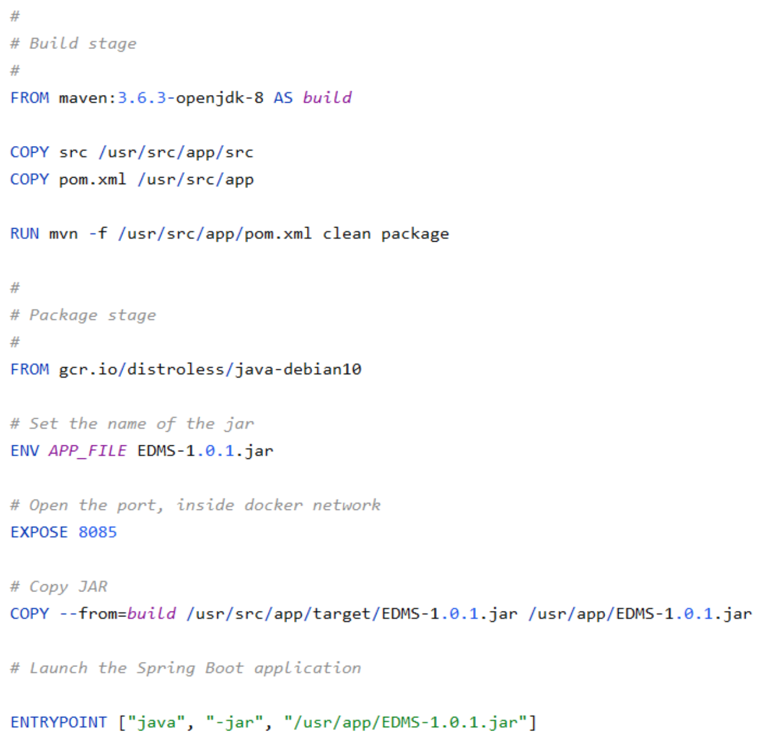
- **Front-end** - nginx:stable-alpine
- **Back-end** - gcr.io/distroless/java-debian10
- **OCR-API** - gcr.io/distroless/java-debian10

An example, how can a Dockerfile look like, can be seen in Figure 4.18. It consists of two stages.

1. **Build stage** - used to build an application from the source code.
2. **Package stage** - stacking the built application onto the provided Docker image.

Using these two stages does, what I described a while ago about building a new image from changed source-code without doing it myself.

[3]`https://hub.docker.com/`
[4]`https://hub.docker.com/u/madgyver`



```
#
# Build stage
#
FROM maven:3.6.3-openjdk-8 AS build

COPY src /usr/src/app/src
COPY pom.xml /usr/src/app

RUN mvn -f /usr/src/app/pom.xml clean package

#
# Package stage
#
FROM gcr.io/distroless/java-debian10

# Set the name of the jar
ENV APP_FILE EDMS-1.0.1.jar

# Open the port, inside docker network
EXPOSE 8085

# Copy JAR
COPY --from=build /usr/src/app/target/EDMS-1.0.1.jar /usr/app/EDMS-1.0.1.jar

# Launch the Spring Boot application

ENTRYPOINT ["java", "-jar", "/usr/app/EDMS-1.0.1.jar"]
```

Figure 4.18: Back-end Dockerfile

Chapter 5

Testing

Testing is the last part of development of every new application. In my case, I used three different approaches:

- Unit testing
- Use-case scenarios testing
- Usability testing

5.1 Unit testing

Unit testing was the first approach to test the application. I used unit testing for Back-end and OCR-API for

- Controllers
- Services

In both cases, the testing was realized via library JUnit that easily enables to do unit testing in Java. These tests are run every-time a new build is being created. In case they fail, a developer has to fix it, so the next build can be created.

5.1.1 OCR-API unit testing

In OCR-API, there are five unit tests. From five of them, three belong to Service testing:

- *DocumentServiceImplTest*
- *PDFServiceImplTest*
- *OCRServiceTPlatformTest*

The remaining two are aimed for controllers:

- *PDFControllerTest*
- *ImgDocumentControllerTest*

5.1.2 Back-end unit testing

In comparison to OCR-API, Back-end testing contains three groups of unit testing. The first testing group takes care of controllers.

- *DocControllerTest*
- *UserControllerTest*

The second testing group looks after services.

- *OCRServiceTesseractTest*
- *ConfirmationTokenServiceImplTest*
- *ResetTokenServiceImplTest*
- *UserServiceImplTest*
- *DocServiceImplTest*
- *EmailServiceImplTest*

Lastly, but not least, is the unit testing group for DTOs consisting only of one class.

- DocDtoConverterTest

5.2 Use-case scenarios testing

After implementing the whole application, I tried to simulate all use-case scenarios to find out any inconsistencies. The found and fixed troubles were:

- Account confirmation of registration not working
- Resetting account password not saving new password
- Advanced search not finding document states and language
- The titles in the documents table were not centered
- Column names were programming naming as *origName* instead of the normal full description *original name*

5.3 Usability testing

In addition to all previous tests, there was also done usability testing. Its main aim was to see, if the created graphical user interface, even though basic, had possible problems that would complicate a user's work with the tool.

5.3.1 Scenario

To users/testers were given a scenario, to at least have a general idea, what can be done in the application.

Scenario:

1. Upload 2 documents with the language set to English.
2. Upload another 2 documents with a different language in comparison to previous task.
3. Add tags to a document A
4. Add different tags to a document B.
5. Download an image of an arbitrary document.
6. Find a document A using added tags.
7. Find a document that has set language to English.
8. Delete an arbitrary document.

The specified scenario is not a limitation of the testing. It is just a means to somehow compare the testing of multiple users, but also parts that didn't belong to the scenario were taken into account.

5.3.2 Testing

In total, there were four people, who tested the applications usability and records of their testing can be found in the USB enclosed to this thesis.

In the following subsections will be described what kinds of problems testers run into.

5.3.2.1 Tester 1

The first tester had experienced complications while trying to import files to one batch of files by adding them one by one. This approach was not considered in implementation as every new selection replace the previous one.

Additionally, he was confused at the beginning, how the advanced searching is working, but eventually, everything worked out.

Lastly, he couldn't download an uploaded document, while it was not in the completed status.

5.3.2.2 Tester 2

For the second tester, the import part worked quietly and without any major problems. However, there was major trouble, when the tester wanted to search via an advanced searching option. The tester's recording has shown his misconception, as he wanted to search, without adding the selected search phrase option to searching parameters. The rest of the testing went smoothly.

5.3.2.3 Tester 3

The third tester had mainly problems with uploading files. He tried to use batches, but added more batches than he needed to. Therefore, he redoes whole files importing, but by adding it file per batch.

In the second part to look for documents, there were no problems, as the tester used just brute force as a viewed file by file. However, there is visible confusion, when he tried to download the file. At first, he wasn't aware, where it is located. Secondary, how to achieve it.

5.3.2.4 Tester 4

The fourth and the last tester met with similar troubles as previous testers. Documents uploading was not as intuitive as it could be and usage of the advanced searching method was too complex.

5.3.3 Summary of testing

The usability testing was fruitful as it helped to found a couple of problems:

- Uploading file/s was too simple and bothered the testers
- A way to download the documents file was not obvious
- Advanced search is too complex
- A user is not able to download document, when scanning is not completed

5.3.3.1 Uploading file

To fix lack of features for uploading file, an additional were added, see E.1.

- A user can drag and drop files
- A user can deselect unwanted files
- A user can select additional files

5.3.3.2 Download file

From testing could be seen confusion, when a user is tasked to download a file belonging to the document. Therefore, there was added a download button in the preview, see E.3.

5.3.4 Advanced search

The users' experiences from usability testing has shown that advanced search needs to change in the future. It needs to be more intuitive and less complex.

5.3.4.1 Document download

A problem with downloading document while being in a different state than completed, was fixed inside the Back-end code.

Conclusion

This thesis's goals were to analyse existing electronic document management systems and design a new prototype that will be open-sourced and free. The designed application should be divided in parts that would make it easier to reuse it as the whole or reuse only specific parts. The requirement for reuse was indirectly pointed at the component OCR-API that could be used by anyone else, who just want OCR exposed via REST API. Finally, it should be implemented and tested to make sure it is working well.

All goals were accomplished. Existing solutions were analyzed and have been taken into account, while trying to create a new prototype. The implemented prototype is open-source and free to use for everyone. It is as well documented and contains JUnit tests.

In addition, the whole application that is implemented, is available not only on Github[5] as source code, but also as executable Docker image on Dockerhub[6]. Moreover, the Dockerhub repository not only contains a docker image of every component, but also builds a new Docker image, when the source codes were changed at Github.

In the future, there are multiple tasks that can be done. Firstly, it is needed to change advanced searching for something more user friendlier. Secondly, create an admin page and implement its logic to Back-end. Thirdly, *DocOcrChecker* needs some tweaks to prevent future overloadings and to enable possibility to change execution interval by admin. Moreover, OCR-API is currently using only basics of Tesseract, but there is much more options to use.

Lastly, but not least, even though there is still room for improvement, it has a solid foundation. If given additional time and dedication, it could stand on equal ground with other document management systems or even surprise them.

[5]`https://github.com/EnjoyB/docs-archive-backend`

[6]`https://hub.docker.com/u/madgyver`

Bibliography

[1] Pitkley. Paperless [online]. May 2020, [Cited 2020-8-14]. Available from: https://github.com/the-paperless-project/paperless

[2] EDMS, M. Mayan EDMS [online]. 2020, [Cited 2020-8-11]. Available from: https://www.mayan-edms.com/

[3] System, O. D. M. Document Management System [online]. 2020, [Cited 2020-8-11]. Available from: https://www.openkm.com/

[4] eFileCabinet. Introducing a New Way to Accomplish Business [online]. [Cited 2020-11-12]. Available from: https://www.efilecabinet.com/how-it-works/

[5] Templafy. Templafy platform [online]. [Cited 2020-11-9]. Available from: https://www.templafy.com/platform/

[6] Vienna-Advantage. An enterprise level commercial open source ERP/CRM [online]. [Cited 2020-11-11]. Available from: https://www.viennaadvantage.com/

[7] Prajapati, A. Angular Architecture [online]. 2020, [Cited 2020-12-11]. Available from: https://www.ngdevelop.tech/angular/architecture/

[8] Biels. A History of Document Management [online]. July 2016, [Cited 2020-12-12]. Available from: http://biels.com/a-history-of-document-management/

[9] Oragui, D. EDMS: A Comprehensive Guide [online]. March 2020, [Cited 2020-12-16]. Available from: https://helpjuice.com/blog/edms

[10] eFileCabinet. About eFileCabinet, Inc. [online]. [Cited 2020-11-8]. Available from: https://www.efilecabinet.com/company/

[11] Specification, O. A. Omg unified modeling language (omg uml), superstructure, v2. 1.2. *Object Management Group*, volume 70, 2007.

[12] Arora, S. K. 10 Best JavaScript Frameworks to Use in 2020 [online]. September 2020, [Cited 2020-11-4]. Available from: `https://hackr.io/blog/best-javascript-frameworks`

[13] Oracle. About the Java Technology [online]. [Cited 2020-11-11]. Available from: `https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html`

[14] Spring. Spring Framework Overview [online]. October 2020, [Cited 2020-11-11]. Available from: `https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html`

[15] SmartBear. OpenAPI specification [online]. November 2020, [Cited 2020-11-10]. Available from: `https://swagger.io/resources/open-api/`

[16] Kimberlin, M. Reducing Boilerplate Code with Project Lombok [online]. January 2020, [Cited 2020-11-11]. Available from: `https://objectcomputing.com/resources/publications/sett/january-2010-reducing-boilerplate-code-with-project-lombok`

[17] Tesseract-ocr. Tesseract OCR [online]. July 2020, [Cited 2020-11-11]. Available from: `https://github.com/tesseract-ocr/tesseract`

[18] Docker. Docker Overview [online]. 2020, [Cited 2020-11-11]. Available from: `https://docs.docker.com/get-started/overview/`

[19] MongoDB. What is MongoDB [online]. 2020, [Cited 2020-11-13]. Available from: `https://www.mongodb.com/what-is-mongodb`

[20] Auth0. Introduction to JSON Web Tokens [online]. [Cited 2020-11-12]. Available from: `https://jwt.io/introduction/`

Appendix A

Acronyms

UI	User interface
GUI	Graphical user interface
DMS	Document management system
EDMS	Electronic DMS
OCR	Optical character recognition
CE	Community edition
OMG	Object management group
UML	Unified modeling language
REST	Representational state transfer
API	Application programming interface
HTTP	Hypertext transfer protocol
FTP	File transfer protocol
OS	Operating system
DTO	Data transfer object

Appendix **B**

Executing application

To start up an application from built images on Dockerhub, it must have a starting script called *docker-compose.yml*. It is located in Back-end's git-repository[7], folder *docker_commands*. Afterward, it is only necessary to start following commands in folder, where is *docker-compose.yml* located.

The created docker-compose.yml was tested and executed inside different OS as Windows 10, Ubuntu and macOS. Therefore, it should be working on other platforms, depending only on the docker settings.

```
#!/bin/bash
#Command to download container images.
docker-compose pull

#Command to start-up application
docker-compose up
```

After starting up the application, components should be available:

- Front-end - `http://localhost:80/`
- Back-end - `http://localhost:8085/`
- OCR-API - `http://localhost:8086/`

For testing purposes, there is created a dummy user with credentials:

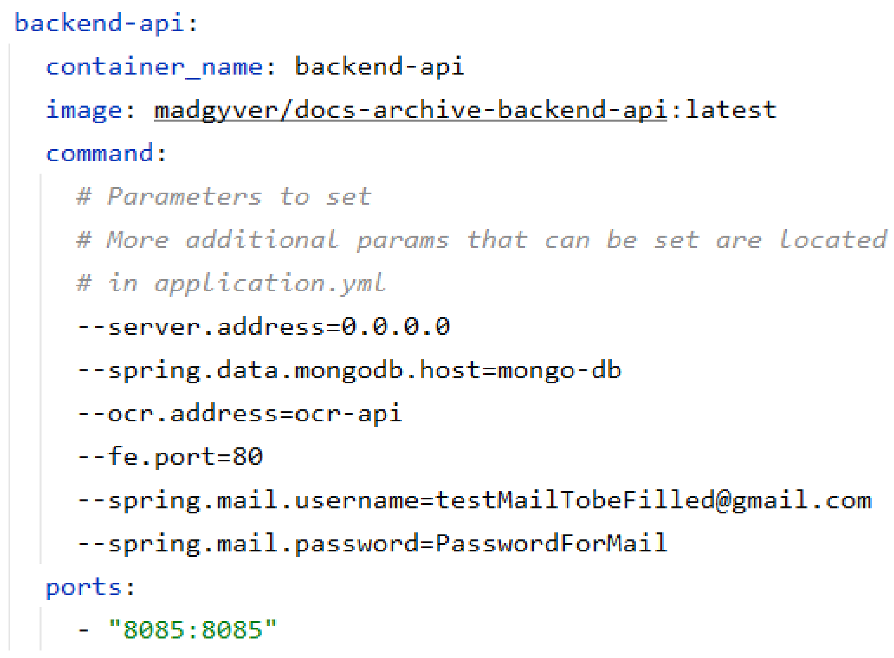
- **name** - tester
- **password** - tester

Different settings may be changed, when creating a container via parameters. In *docker-compose.yml*, there are few parameters that are already set, so others could use them in the future easily. Important parameters to be set are:

[7]https://github.com/EnjoyB/docs-archive-backend/blob/master/docker_commands /docker-compose.yml

- *spring.mail.username*
- *spring.mail.password*

Both parameters are used for registration of users and need to be set for Back-end, see Figure B.1, where are set dummy values for the e-mail and password.



```
backend-api:
  container_name: backend-api
  image: madgyver/docs-archive-backend-api:latest
  command:
    # Parameters to set
    # More additional params that can be set are located
    # in application.yml
    --server.address=0.0.0.0
    --spring.data.mongodb.host=mongo-db
    --ocr.address=ocr-api
    --fe.port=80
    --spring.mail.username=testMailTobeFilled@gmail.com
    --spring.mail.password=PasswordForMail
  ports:
    - "8085:8085"
```

Figure B.1: Docker-compose for Back-end

All other parameters that can be changed, will be found in Github repositories of the respective component. For example, OCR-API needs a path to training sets of the OCR.

- Front-end - does not have any specific parameters to set - `https://github.com/EnjoyB/docs-archive-frontend`
- Back-end - other parameters that can be set are located in application.yml - `https://github.com/EnjoyB/docs-archive-backend`
- OCR-API - other parameters that can be set are located in application.yml - `https://github.com/EnjoyB/docs-archive-ocr-api`

Furthermore, additional information can be found in the specific component repository. However, there is an exception for the ports of Docker images. When the ports are changes through parameters, they are changed only for the application, but it doesn't mean that it will also change the docker network settings. To change also the docker network settings, it is necessary to change *Expose #PORT* in Dockerfile and build it again.

Appendix C

Diagrams

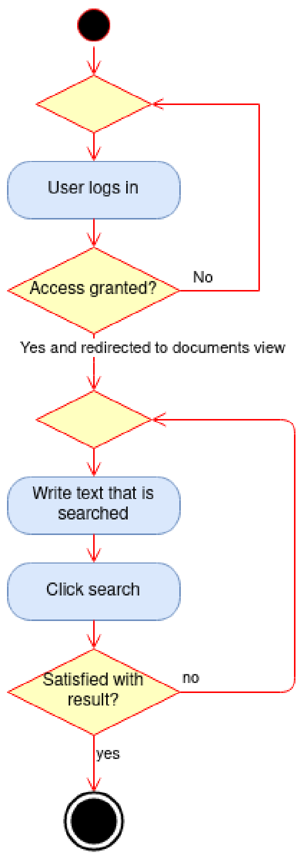


Figure C.5: Simplified activity diagram of using the full-text search to find a document

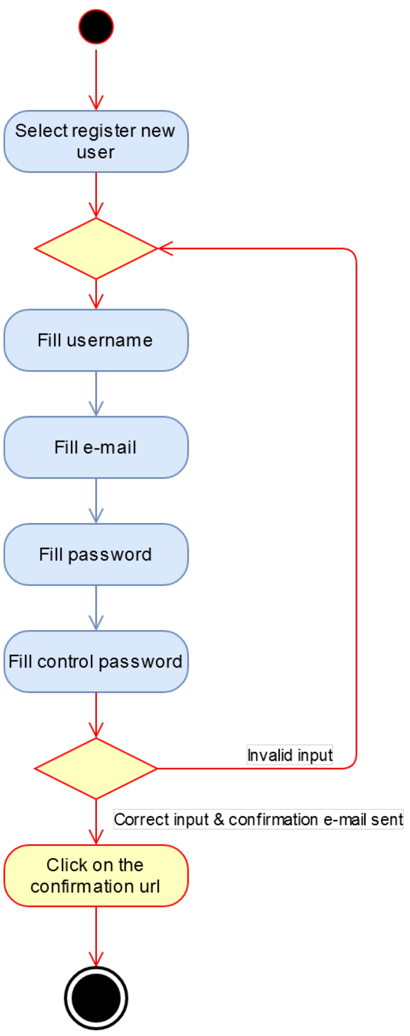


Figure C.1: User registration simplified activity diagram

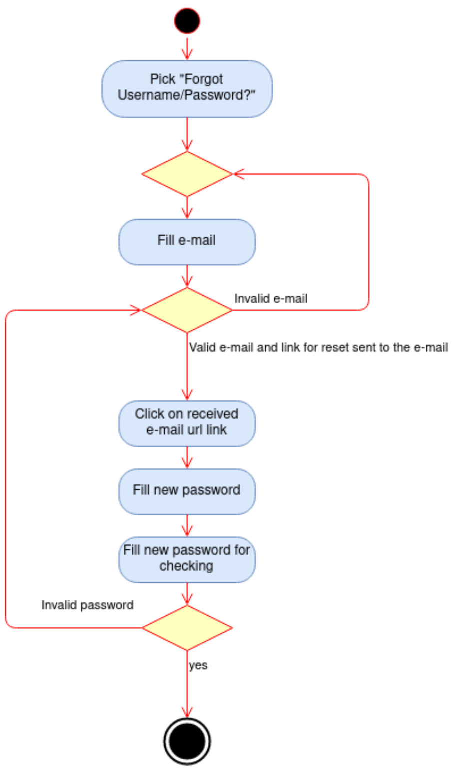

Pick "Forgot Username/Password?"
Fill e-mail
Invalid e-mail
Valid e-mail and link for reset sent to the e-mail
Click on received e-mail url link
Fill new password
Fill new password for checking
Invalid password
yes

Figure C.2: Simplified activity diagram of password resetting

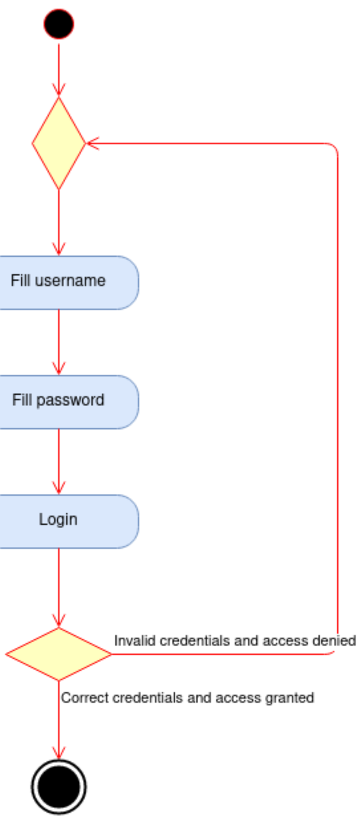

Fill username
Fill password
Login
Invalid credentials and access denied
Correct credentials and access granted

Figure C.3: Simplified activity diagram of user login

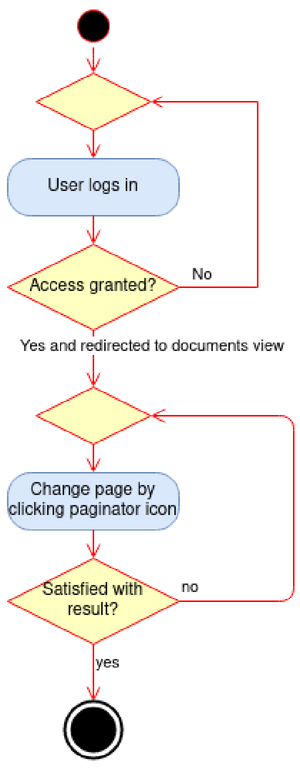

User logs in
Access granted?
No
Yes and redirected to documents view
Change page by clicking paginator icon
Satisfied with result?
no
yes

Figure C.4: Simplified activity diagram of using pagination search to find a document

Appendix D

GUI after implementation

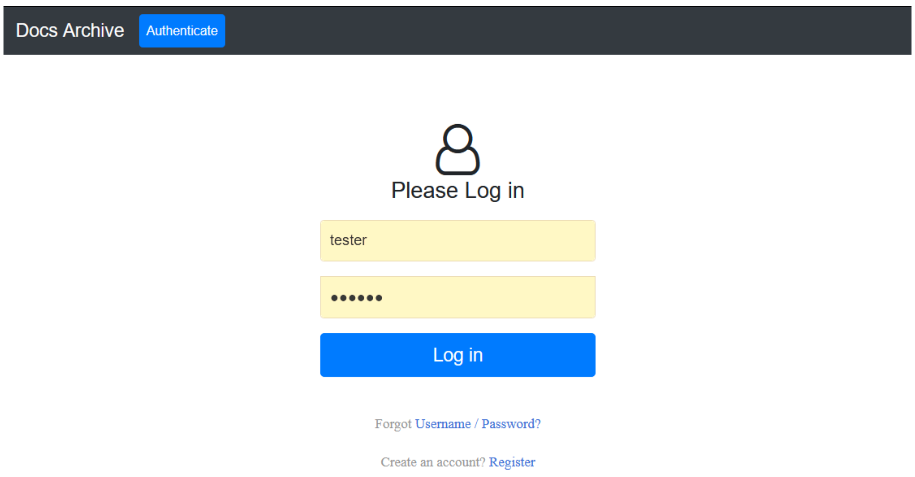
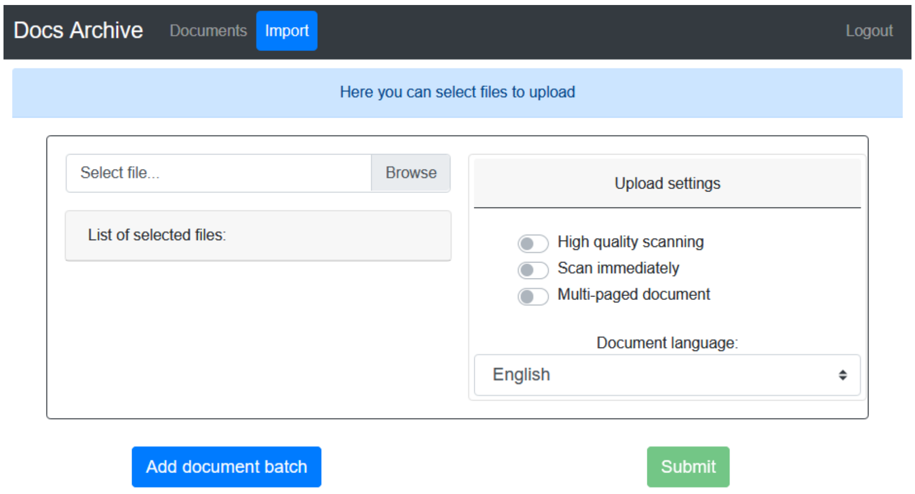


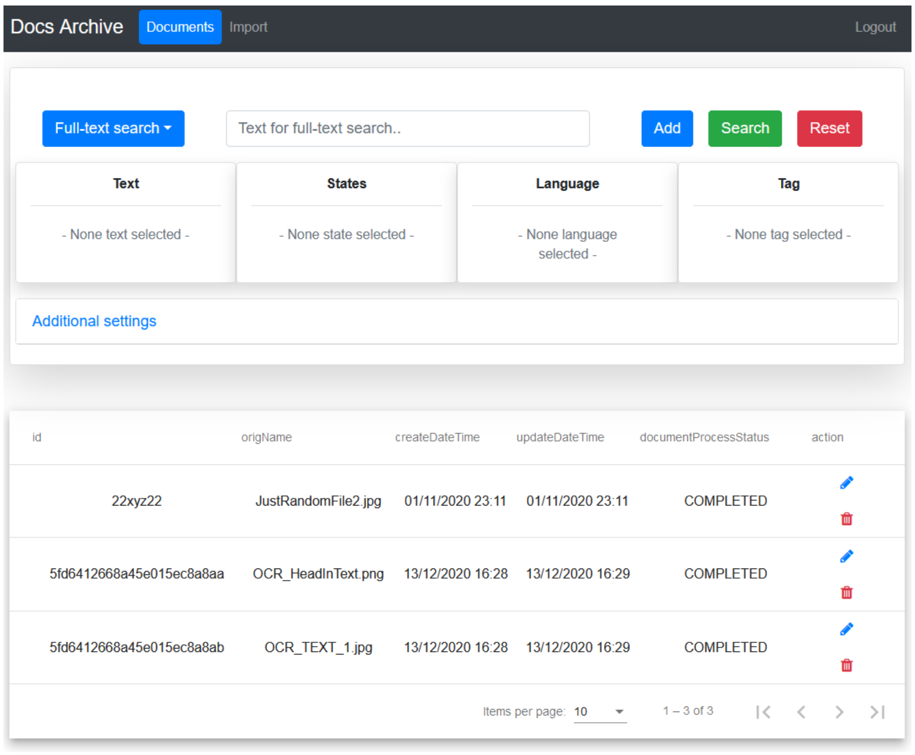
Figure D.1: Login page



High quality scanning
Scan immediately
Multi-paged document

Document language: English

Figure D.2: Import page after implementation.

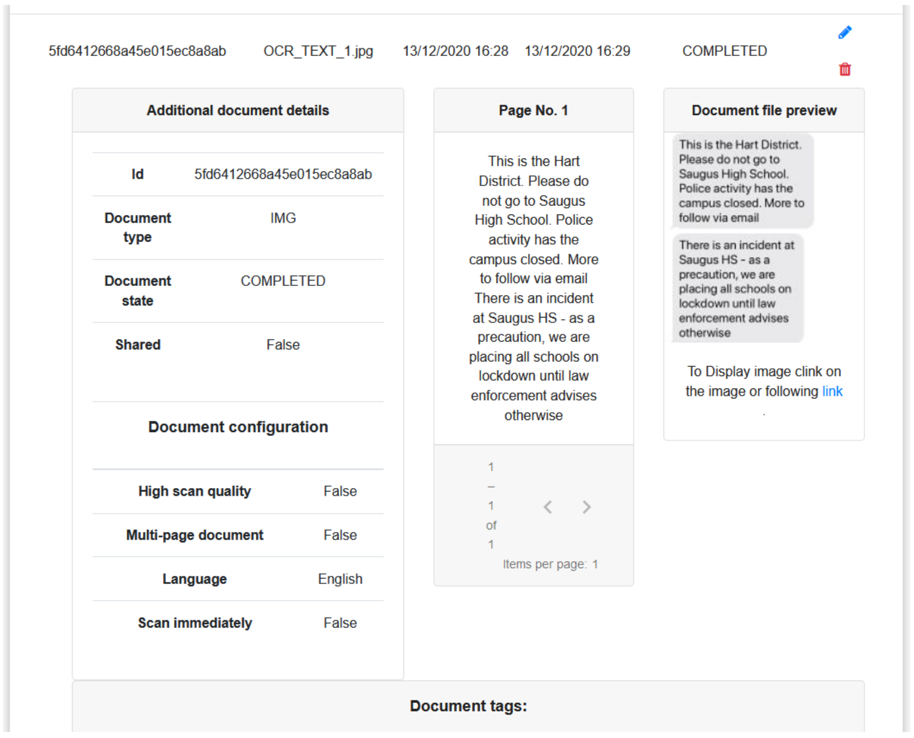


Text	States	Language	Tag
- None text selected -	- None state selected -	- None language selected -	- None tag selected -

id	origName	createDateTime	updateDateTime	documentProcessStatus	action
22xyz22	JustRandomFile2.jpg	01/11/2020 23:11	01/11/2020 23:11	COMPLETED	
5fd6412668a45e015ec8a8aa	OCR_HeadInText.png	13/12/2020 16:28	13/12/2020 16:29	COMPLETED	
5fd6412668a45e015ec8a8ab	OCR_TEXT_1.jpg	13/12/2020 16:28	13/12/2020 16:29	COMPLETED	

Items per page: 10 1 – 3 of 3

Figure D.3: Documents page after implementation.



5fd6412668a45e015ec8a8ab OCR_TEXT_1.jpg 13/12/2020 16:28 13/12/2020 16:29 COMPLETED

Additional document details

Id	5fd6412668a45e015ec8a8ab
Document type	IMG
Document state	COMPLETED
Shared	False

Document configuration

High scan quality	False
Multi-page document	False
Language	English
Scan immediately	False

Page No. 1

This is the Hart District. Please do not go to Saugus High School. Police activity has the campus closed. More to follow via email There is an incident at Saugus HS - as a precaution, we are placing all schools on lockdown until law enforcement advises otherwise

1 – 1 of 1 Items per page: 1

Document file preview

This is the Hart District. Please do not go to Saugus High School. Police activity has the campus closed. More to follow via email

There is an incident at Saugus HS - as a precaution, we are placing all schools on lockdown until law enforcement advises otherwise

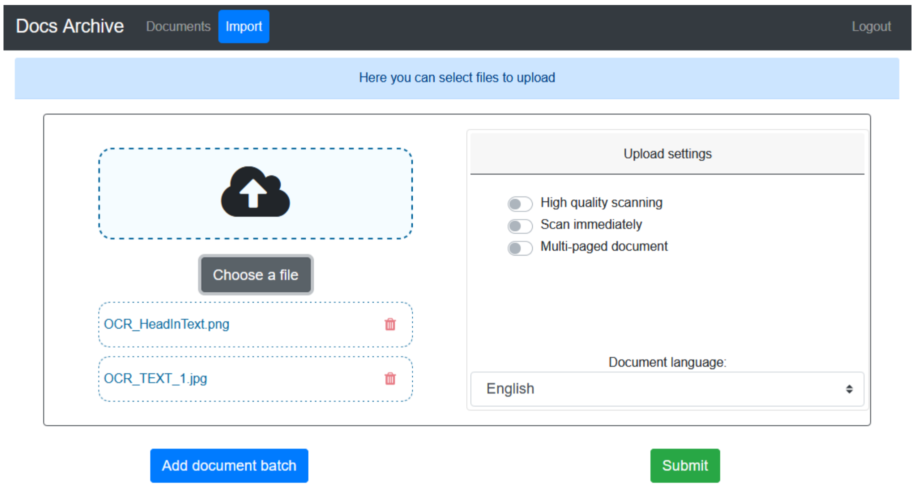
To Display image clink on the image or following link

Document tags:

Figure D.4: Extended view page after implementation.

Appendix E

GUI after testing fixes



Here you can select files to upload

OCR_HeadInText.png

OCR_TEXT_1.jpg

Upload settings

High quality scanning

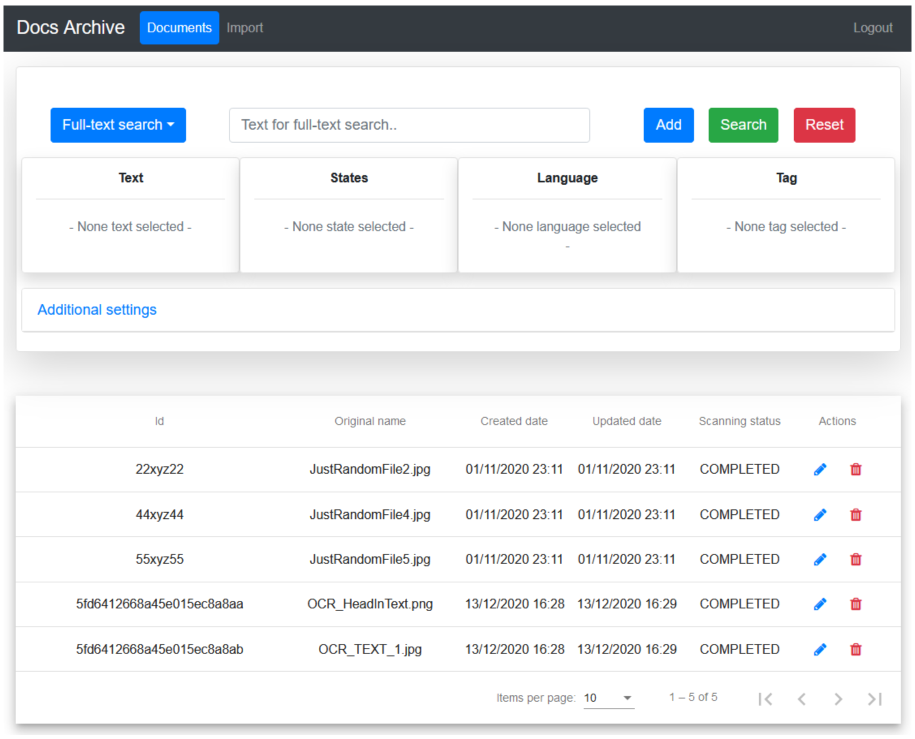
Scan immediately

Multi-paged document

Document language:

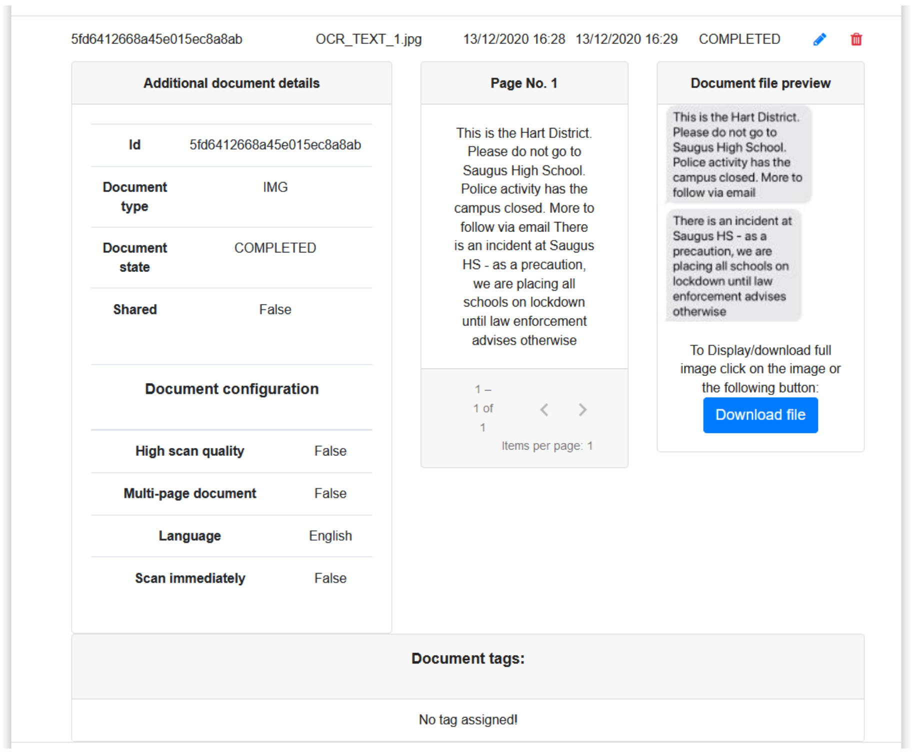
English

Figure E.1: Import page with fixed selection of the files.



Id	Original name	Created date	Updated date	Scanning status	Actions
22xyz22	JustRandomFile2.jpg	01/11/2020 23:11	01/11/2020 23:11	COMPLETED	
44xyz44	JustRandomFile4.jpg	01/11/2020 23:11	01/11/2020 23:11	COMPLETED	
55xyz55	JustRandomFile5.jpg	01/11/2020 23:11	01/11/2020 23:11	COMPLETED	
5fd6412668a45e015ec8a8aa	OCR_HeadInText.png	13/12/2020 16:28	13/12/2020 16:29	COMPLETED	
5fd6412668a45e015ec8a8ab	OCR_TEXT_1.jpg	13/12/2020 16:28	13/12/2020 16:29	COMPLETED	

Figure E.2: Documents page with fixed column names and position.



5fd6412668a45e015ec8a8ab OCR_TEXT_1.jpg 13/12/2020 16:28 13/12/2020 16:29 COMPLETED

Additional document details

Id	5fd6412668a45e015ec8a8ab
Document type	IMG
Document state	COMPLETED
Shared	False

Document configuration

High scan quality	False
Multi-page document	False
Language	English
Scan immediately	False

Page No. 1

This is the Hart District. Please do not go to Saugus High School. Police activity has the campus closed. More to follow via email There is an incident at Saugus HS - as a precaution, we are placing all schools on lockdown until law enforcement advises otherwise

Document file preview

This is the Hart District. Please do not go to Saugus High School. Police activity has the campus closed. More to follow via email

There is an incident at Saugus HS - as a precaution, we are placing all schools on lockdown until law enforcement advises otherwise

To Display/download full image click on the image or the following button:

Document tags:

No tag assigned!

Figure E.3: Extended view page with a newly added button download.

Appendix F

Contents of enclosed USB

```
readme.md ....................... the file with USB contents description
records .................. the directory of records from usability testing
docker-images .......................... the directory with executables
src ....................................... the directory of source codes
    docs-archive ............................. implementation sources
        Front-end ............... implementation sources of the Front-end
        Back-end ................ implementation sources of the Back-end
        OCR-API ................ implementation sources of the OCR-APIs
    thesis .............. the directory of LaTeX source codes of the thesis
text ........................................... the thesis text directory
    thesis.pdf ......................... the thesis text in PDF format
```