CV Scanning Tool

T.H. Brouws, N. van der Laan, D. Vermunt, K. Wendel

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

CV Scanning Tool

by

T.H. Brouws, N. van der Laan, D. Vermunt, K. Wendel

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

at the Delft University of Technology,
to be defended publicly on Tuesday July 4, 2017 at 15:00.

Client:	O. Hektor,	FleXentral
Supervisor:	Dr. C. Lofi,	TU Delft
BEP Coordinators:	Ir. O.W. Visser,	TU Delft
	Dr. H. Wang,	TU Delft

Preface

This final report concludes the development of a CV Scanning Tool, part of the TI3806 - Bachelor Project course at the Delft University of Technology in the Netherlands. The project ran from April 24th 2017 to June 30th 2017, and was carried out for O. Hektor, founder of FleXentral. This report provides the assignment, research, requirements, methodology, design, implementation and results of the project.

During this project we received help and guidance from multiple persons, which we would like to thank:

- Onno Hektor, from FleXentral, for providing and supervising the project, as well as for the nice office environment, group lunches and fun Friday afternoons.
- Christoph Lofi, from the TU Delft, for supervising the project and guiding us during the research phase of the project.
- Otto Visser, from the TU Delft, for coordinating the Bachelor Project course.
- Huijuan Wang, from the TU Delft, for coordinating the Bachelor Project course.

T.H. Brouws, N. van der Laan, D. Vermunt, K. Wendel
Delft, June 2017

Contents

1 Introduction

1.1. Company Description

The project was commissioned by FleXentral. FleXentral is a startup company financed from angel capital that was started as a BEP project in 2016. FleXentral is a platform on which companies can find and manage their flexible workforce. The platform is not yet live, the first trials start in June 2017 and the full platform is planned to go live in the summer of 2017.

1.2. Problem Definition

In the project that was issued by FleXentral (Appendix A) we identified two problems that are important to solve. First we will discuss the problem of creating your own CV that needs to have a high success rate when you apply for a job. Secondly, the problem of extracting information from a CV will be discussed. This is the most technical problem of this project and improves the workflow of the application and one of the use cases of FleXentral.

1.2.1. Creating your own CV

When applying for a job, you need to sell yourself to the company on paper. This is usually done with a curriculum vitae that shows what education you had, your previous work experience and other skills and competences that you have. Based on this, and sometimes without even seeing you in person, the company decides if you fit their profile and their company. This judgement should be based solely on facts and skills but this unfortunately is not always the case. The first impression of a CV is very important because it makes the recruiter or company remember you, and thereby increases your chances at being invited for an interview. Therefore, it is important that the CV is properly designed and features important aspects about you.

However, making such a CV is hard for most people. Not everyone is familiar with PDF generation tools like LaTeX, where you can use pre-designed templates. For this reason, they would rather design their own CV using Microsoft Word. Also, most people do not know what recruiters find important in a CV. Since FleXentral is the mediator between a company and a job-seeker, they have a lot of in-house experience about this.

To solve this problem we need a tool where job-seekers can easily make their own CV based on a design and the important aspects of FleXentral. This tool must be usable by most users and work as easily as possible. If someone has already made their own CV and simply wants to update it, the process to make a new CV with our tool should be as painless as possible. This new CV should higher the chance of finding a new job by being more visually attractive.

Goals

The following goals are important for our application to create your own CV and market FleXentral.

- CV needs to feature elements that are important according to FleXentral
- CV needs to be visually attractive to make a good first impression

- Creating your CV should be easy and straightforward
- Application should have a good user experience to market the FleXentral platform

1.2.2. Extracting information

At the moment of writing this report there are already tools that offer the possibility to design your CV online. However, they lack the possibility to upload an existing CV to extract information from. This lowers the amount of users that will use the tool, because retyping your entire CV requires a significant amount of time. Also, one of the use cases that FleXentral has in mind is making standardised CV for staffing agency. These actors do have CVs of all of their workers but the skills and experiences that they contain are not extracted or stored in a database. Thus automatically extracting this information and save it in a standardised data structure will improve the FleXentral platform. Since this is a very technical problem it is more researched in Chapter 2.

Goals

The following goals are important for extracting information from a CV.

- Extracting personal details such as name, email and phone
- Extract the list of skills and competences that is featured on a CV
- Extract information about the different education and work experience entities listed on the CV.

1.3. Target

Our application mainly targets freelancers who are looking for work. The aim is to allow them to better their chances at finding a job by using our application to create or improve their CV. The second aim is to alert them to the existence of FleXentral, a platform where they can register and apply for jobs.

1.4. Outline

We will analyse our problem more in-depth in Chapter 2, where we will also outline our research. Then, in Chapter 3 the actors and requirements of the project will be described. Next, in Chapter 4 we will go into more detail on our approach, planning and tools used in the project. Further, In Chapter 5 the design and implementation details will be outlined. The formal structure of the application will be shown there, as well as mock-ups of the GUI. A section on ethics will also be part of this chapter. Furthermore, in Chapter 6 we will reflect on our project. Finally, we conclude our report in Chapter 7 by evaluating the results and giving recommendations for future work. Several Appendices and a Bibliography are also included at the end of the report.

2 Research

Our application should minimise the effort done by the user to create their new CV. In order to achieve this, we provide the user with the possibility to upload relevant data about themselves. We offer the user the option to upload their own CV as a starting point, which we then attempt to parse. This means we try to extract the information their CV contains about them. By doing so, we can significantly reduce the amount of effort needed to be done by them. However, parsing a CV is far from trivial, as every CV is different in its structure. Our research focuses on this aspect of our application, as we foresee that our main challenges lay here.

In Section 2.1 we will analyse more concretely what our problem consists of. In Section 2.2 we review recent literature about different ways of approaching the problem. Then in Section 2.3 we will look at how existing commercial as well as open-source tools look to solve this problem. Finally, in Section 2.4 we shall highlight the pros and cons of each approach as well as explain our final approach.

2.1. Problem Analysis

During our research we identified our main research question: 'Given the huge diversity in structure between CVs, how do we extract meaningful information from them?' Every CV is structured in a different way, and depending on the field of expertise, different competences and skills are important. This makes it difficult to find a general approach to parse a CV. However, while the structure of each CV is rather variable, there are a lot of standard ways of displaying information within certain sections. By identifying the different sections and treating each of them as individual problems, we are able to split our problem into smaller and easier sub-problems.

2.2. Literature Study

Parsing documents for their text and extracting meaning from them is a topic that has been around for a while. While scanning a CV for certain information is a basic thing to do for a human, a computer has a tremendous amount of difficulty doing so. The ability of a program to understand human speech is called Natural Language Processing, or NLP for short. Current NLP methods are mostly based on machine learning. In this section we shall discuss how machine learning approaches the problem of NLP, and how this can be applied specifically for varying purposes.

2.2.1. Machine learning

A very common approach these days to the problem of NLP is machine learning. When one uses machine learning to parse CVs the words, sentences or sections are transformed to feature vectors. The machine learning algorithm will then classify these as personal information, educations, work experiences or other sections based on classifiers that it found during training.

To retrieve good results with machine learning, the training is the most important part since this will provide the classifiers. For training there are three different approaches: supervised learning, unsupervised learning and a hybrid approach.

- In supervised learning, the machine learning algorithm is provided with pre-classified data and will train itself on this to find the correct classifiers. This approach is not suitable for our project since there is not enough training data. We only have a small set of CVs, obtained from friends and parents, that is not representative for everyone because it is biased towards younger and highly educated people. Obtaining another training set is hard because CVs are not freely accessible on the Internet.
- In the unsupervised learning approach the training data is not classified. The algorithm will find patterns on its own and will use these as classifiers. This approach in an incremental form is possible because we receive feedback after parsing since the user will update their CV and will change what is wrong. But if we take this approach, our tool will not work for the first few thousand CVs that are provided. We think that this is unacceptable and will result in a low usage of our product.
- With the hybrid approach we will have the same problem that there is not enough training data to start with.

NLP is being applied for varying purposes. Two interesting common NLP tasks are Part-of-Speech (POS) tagging and Named Entity Recognition (NER).In the following subsections we shall briefly discuss both of these tasks and how they can be applied specifically to our problem.

2.2.2. Part-of-speech tagging
One application of NLP is POS-tagging. We found three different major frameworks for POS-tagging: The Stanford Core NLP [1] , TreeTagger [2] and Forg [3]. These different frameworks make use of big vocabulary libraries to tag text with part-of-speech and lemma information. This could then be used by an algorithm that then identifies what kind of information each part represents. Mostly machine algorithms are used to do so for whole sections, which we will not implement. Besides, the structure of a CV is much more useful when it comes to extracting meaning, rather than interpreting the context. However, it is also possible to identify simple things through the tags of a single word or a couple of words, such as locations, names and institutions. We could thus look to use these frameworks for extracting information on a section level. But since there is no need to understand the meaning or underlying structure of every word, this approach gives too much overhead. The same result of identifying simple things can be achieved with named entity recognition without the overhead.

2.2.3. Named entity recognition
Another, and perhaps more relevant approach to our problem, is to extract information that is important to recognise entities like names, persons, organisations, location names and numeric expressions. Identifying references to these entities in text is called ”Named Entity Recognition and Classification (NERC)”.[4] A word, its length and other multiple word-level features can be processed based on different rules to identify the entities. According to Nadeau et al. there are multiple word-level features that give a certain meaning to the word. Some of these features and their possible functions are listed in Table 2.1

Feature	**Used to recognise**
Case	Name and surname
	Company names with mixed case (FleXentral)
Punctuation	Company names with abbreviation
Digit patterns	Phone numbers
	Postal code
	Dates
Morphology	Prefixes, suffixes or common endings for jobs (e.g. manager)

Table 2.1: Usage of different features

Another important technique to recognise entities is using a list lookup.[4] Lookups in the lists are best done using fuzzy-matches with a threshold edit-distance. This allows that words that have the same meaning, but are written incorrectly or with a different spelling, still match the correct entity.

Because we are parsing CVs, we are particularly interested in CV-relevant information, such as names of organisations a person worked at, or certain academic titles such as PhD. Our list to recognise such entities can be found in table Table 2.2.

List	Used to recognise
General	Common abbreviations
	Function words
Entities	Organisation, government and educational instances
	Location (country, state, city)
Entity cues	Typical words in organisations (e.g. B.V.)
	Academic titles (e.g. PhD, BSc)
	Location typical words (e.g. at)

Table 2.2: Usage of different entity lists

2.3. State of the Art

During our research on existing tools we discovered that there are multiple approaches to solve this problem. We shall look at how market leaders on this subject solve this problem and what success rates they achieve. We also did research on open source tools that do CV parsing and tested these.

2.3.1. Existing commercial CV parsers

CV parsers are commonly used in the HR industry to identify qualified candidates for the job automatically. Extracting information like personal information and previous work experience can save a tremendous amount of time. Some market leaders are DaXtra, RChili and Textkernel, who all have several years of experience in the field of CV parsing. However, the accuracy of these tools is still highly dependent on the CV given as input, and they require human supervision to correct mistakes. Testing them against a variety of CVs shows how dependent they are on their input, with an accuracy on correctly identifying previous work experiences up to 25 percent [5]. This shows how correct parsing of a CV can be an extremely difficult task to do. Focusing on becoming better at this than market leaders would be unwise and very likely too ambitious. For this reason, we aim not to do a perfect job at this. Instead, we should look at what information can be easily extracted and allow the user to fill in the missing information about themselves.

2.3.2. Existing open source CV parsers

LikeRr/code4goal-resume-parser

This repository was created for the Code4Goal competition where the contest was to develop a CV parser for companies (`https://github.com/likerRr/code4goal-resume-parser`). After cloning and installing the repository we came to the conclusion that this tool did not work anymore. After displaying that it started with parsing the provided test data in the console, it printed null and stalled, and no output was given. The parsing works with a dictionary and regular expressions. The dictionary is used to recognise the titles that are above the different sections and the regular expressions are used to recognise name, phone and email. [ref https://github.com/likerRr/code4goal-resume-parser]

bjherger/ResumeParser

The CV parser of bjherger is fully written in python version 2.x (`https://github.com/bjherger/ResumeParser`). It extracts basic information and counts if you use certain keywords like LinkedIn, GitHub, Java, Latex etc. When supplying CVs other than the provided test data, the parser gives an error because it does not understand certain information due to the regular expressions that it searches for.

antonydeepak/ResumeParser

The approach of antonydeepak to parse CV is different than the other ones. It uses a hybrid machine-learning and rule-based approach that focuses on semantic rather than syntactic parsing (`https://github.com/antonydeepak/ResumeParser`). To understanding the semantics of a CV, antonydeepak uses GATE, which is an open source language framework (`https://gate.ac.uk`). After a quick test with our own test data we came to the conclusion that it has a hard time understanding Dutch language. Since the majority of our users are either going to supply Dutch or English CVs, we will not use this tool in our project.

While this tool does not perform perfectly, the semantic approach of it is very interesting. Therefore, we did more research on other natural language processing frameworks.

2.4. Research Conclusion

To choose which of the approaches suits our problem best, we should look at the pros and cons of each of them. Machine learning and other training-based algorithms are powerful tools to extract meaning from unstructured text. However, CVs are very predictable in what they can contain. The only varying element between CVs is the way they are structured. For this reason, machine learning seems to be a little out of place. Furthermore, it would require a very big data set of CVs to train on, something we simply are not in possession of. The other more suitable options are Natural Language Processing and Named Entity Recognition. The advantage of these approaches is that they will allow us to identify almost every predefined entity, as well as recognise it through natural language processing. The downside of this is that it is very limited to just recognising these entities without interpreting the context, which would be needed to reach higher precision rates. However, the research pointed out that achieving a high precision on CV parsing currently is very difficult without human supervision. This is especially true for us, as we only have a limited amount of time to create our application. Therefore, we will focus on trying to extract as much information from CVs and let the user validate if it's correct. Natural Language processing combined with Named Entity Recognition is a very suitable approach to our problem of parsing CVs. Finally, to achieve higher accuracy in a different way, the integration with LinkedIn and third-party databases will be given more attention because this data is already more structured and gives better results.

3 Requirements

3.1. System Actors

In our project, we have two main actors. The consumers that use the platform, and FleXentral using the product both as a marketing tool and as part of a bigger platform. We will discuss each of these actors in this section.

3.1.1. Consumer

For the consumer, our application is a tool to help them find a job faster. To do so, we give them the possibility to create a beautiful CV without any knowledge of design or difficult editing programs. We try to make the process of creating this CV as painless as possible by giving multiple options as a starting point. Firstly we offer, in case they already have a CV, to acquire as much data as we can from their previous CV, thus saving the consumer the pain from having to rewrite their CV to create a new one. Secondly, we allow to log in with LinkedIn. This gives us some of the data of the users LinkedIn profile, which we can already substitute in their new CV. LinkedIn also offers a partnership which allows for a lot more data retrieval from profiles. We applied for this partnership but were denied without any feedback. Lastly, we offer the user to start from scratch.

3.1.2. FleXentral

Next to helping consumers find a job, our application should function partly as a marketing tool for FleXentral. People who are looking for a job and decide to build their CV using our application should also get the possibility to directly register on the FleXentral platform. This should be a painless process since all the data required to register is already in the CV they just made. On the FleXentral platform they can apply for actual jobs posted there by companies and staffing agencies, also using the CV they built through our tool.

Lastly the CV building part of the application is something that could be integrated in FleXentral as a kind of create-your-profile service. The data on education, work experience and competences can be used to create the CV, which can then be displayed as a profile page to employers. This should create a uniform way of presenting someones skills and background so employers can easily compare possible employees.

3.2. Global Requirements

To define global requirements for the application we will define them for each actor separately and combine these in a detailed MoSCoW model. MoSCoW is a prioritisation technique that allows stakeholders and developers to agree on requirements.[6]

3.2.1. Consumer

For the consumer, the most important requirement is that he or she can easily create and download a good-looking CV. To meet this requirement, we need an application that allows the consumer to easily

Must Have	Should Have
Parse CV in PDF format	Parse CV in Word format
Use data from LinkedIn	Extract education and work experience
Extract personal info	Fluent editing of CV
Generate a CV PDF	
Create a professional CV	

Table 3.1: The Must Haves and Should Haves for the consumer

Must Have	Should Have
Easily create a user profile on FleXentral	Uniform styling with FleXentral
The data model contains grading	
for skills and competences	The CV can be styled in line with FleXentral styling

Table 3.2: The Must Haves and Should Haves so the application can be used for FleXentral

add sections to a CV, change parts and switch between multiple layout options to suit their preferences. Also, downloading the PDF should be a painless process.

In addition, we want to speed up the process of creating the CV as much as possible. In order to do so, accessing the most basic LinkedIn data is definitely a must. As for parsing of the CVs, parsing some of the data accurately is more important to the consumer than parsing everything. This is because changing falsely classified data is at least as much work as adding the data in the first place. Besides, incorrectly parsing data makes our application less trustworthy.

These requirement are specified further in Table 3.1

3.2.2. FleXentral

To use the product as a marketing tool for FleXentral, the most important requirement is the possibility for the user to create an account on the platform before or after downloading the CV. Also the FleXentral visual front-end design should be implemented to create the sense of the platform and application belonging together.

For the application to be useful for FleXentral as part of their platform, the data structure we use to generate the CVs should in a sense be compliant with the model used by FleXentral. This model is based on skills and competences and a rating given to the user by previous employers. This means our data model should accept such grading of skills and competences. The Must Haves and Should Haves for FleXentral are specified in Table 3.2

3.3. Detailed Requirements

We made a division in requirements based on modules in our application. This detailed MoSCoW model of our requirements can be found in Appendix B, this model includes extra requirements for the data model and also Could Haves and Won't Haves.

4 Project Methodology

In order to ensure a good workflow during the project and good results afterwards, we've devised a project plan and a set of rules that guard code quality. We'll discuss this briefly in this chapter.

4.1. Planning

We've decided to split the project in the following three phases:

- Research, basic working application (phase 1)
- Styling, extra features (phase 2)
- Connection to the FleXentral platform, deployment, documentation, presentation (phase 3)

In this section we'll discuss each phase in more detail and provide a reflection on the outcome.

4.1.1. Phase 1

The first phase consisted of setting up the project, performing research and creating a fully working application that has all the basic features as outlined in our requirements. This distribution of work meant that we were shielded from potential setbacks, as we already had a working application by week 6.

In retrospect we estimated the duration of this phase quite well. We were able to demo a working version of the application by the end of week 6, coinciding with the first SIG code delivery. We benefited a lot from having structured, prioritised requirements, enabling us to only focus on the bare necessities.

4.1.2. Phase 2

The second phase was mostly focused on implementing the styling for both the front-end of the platform and the CV template, while also functioning as a buffer. If everything went well, we would add more features during the second phase. If not, we would focus more on fixing the elementary problems we were experiencing.

The second phase turned out to be more chaotic than anticipated. A lot of work went into implementing the design, while the list of necessary features that still needed work was also quite extensive. We also needed to spend more time on reporting, as the deadline for that was planned a week earlier than the actual end of the project. Looking back, if at all possible we should have allocated more time for this phase.

4.1.3. Phase 3

The last phase was focused on delivering the project, both as a product to the client, and as a report and presentation to the TU. The software would be deployed and usable after this phase. Any final hiccups could be resolved here, but no more new features would be added during this phase.

During this last phase we still needed to spend a lot of time on the report, which was far from finished. This had both advantages and disadvantages; the advantage being that it was very clear what the content needed to be because we were almost finished with the project, the disadvantage being that less time was left for last-minute development and deployment.

Not adding any new features was also impossible, as some essential items were still missing from the application to make it a success. It would have been better to have a longer second phase and to have had a hard deadline at the end where the product had to be done.

The final deployment of the software was done after the deadline for this report, so it could not be reflected on.

4.2. Roles

In order to make sure that every part of the project gets enough attention, we've assigned roles to every team member. Each member has one global role and responsibility for one or two modules. It is important to note that, for the modules, responsibility does not mean that you work alone on that module. It is more comparable to a product owner in a Scrum model, so this person knows what the requirements and priorities are for the module.

The distribution of the roles and modules can be found in Table 4.1, a description for each global responsibility will be given after that.

Team member	Global responsibility	Module responsibility
Daan	Sprints	Front-end
Kasper	Planning	Output
Niek	Deliverables	Data Model
Tom	Code Quality	Input & Data Processing

Table 4.1: Roles for the project

The general task description of the mentioned roles is as follows:

- Code Quality: The team member responsible for code quality will make sure that there are clear guidelines in respect to consistency in code style and test coverage of submitted code.
- Deliverables: Having responsibility for the deliverables means that this team member has a view on the deadlines of the deliverables, and also knows what each deliverable consists of and will make sure that the group starts on time.
- Planning: The team member responsible for the planning will make sure that the project keeps on track.
- Sprints: The last team member is responsible for the sprints, this person will make sure that each week a sprint is planned and executed. This person is also responsible for a clean backlog.

4.3. Process Flow

We're going to work with a Scrum-based approach. Scrum is an agile software development framework that is well suited to our needs, as the nature of our project is that further requirements and challenges are likely to emerge during the process.

We'll perform weekly sprints, which include a planning session at the beginning of the week. A backlog is constructed, from which issues for the coming week will be picked during the planning session. Each team member will estimate duration for solving their issues, filling up their hours for that week.

In addition to the planning sessions, we'll also hold separate meetings to discuss the report and divide the work. If needed, additional design meetings might also be held. Lastly, we will demo the project to the client from time to time.

Our weeks will be split up as follows: Four days per week will be spent at the FleXentral office in Almere, where the focus will be on research and development. The remaining day will be spent at the TU Delft, also allowing for meetings with the TU Coach. During these 'TU days' we will focus more on the process and deliverables.

During most weeks we will work on multiple modules. A maximum of two people will work on a module concurrently. To ensure quality, the team members working on a module will always review each others work.

4.4. Code Quality

An important aspect of building a good software product is code quality. Having high quality code improves readability, scalability and maintainability. As a result, it's easier to make changes later on in the project, and team members can work on code written by others with little effort. In this section we'll explain how we will ensure code quality for our project.

Firstly, because we're working in a team, version control is an essential tool. It allows team members to work concurrently on the same pieces of code, while also providing a system to merge these changes into one working software product.

Testing is another important aspect of programming, as it helps you ensure your code is working in a correct fashion. It also helps you detect breakage of old code when you make changes. It greatly diminishes the amount of bugs that occur in your code, thus improving the quality.

To automate some aspects of this testing process, we will use a continuous integration solution. This is a platform that automatically builds your software in a clean environment, and runs a series of checks (including tests). Only when all checks pass is that build approved. In order to merge code into the final product, we require that the build succeeds on CI.

Finally, a lot of tools exist to help maintain high code quality. Here are some tools that we used to ensure code quality:

- GitHub - A git-based repository solution, including issue management and code review functionality.
- TravisCI - An online continuous integration solution
- ESLint (with AirBnB stylesheet) - A tool checking adherence to a specific codestyle

4.5. Definition of Done

To have clear criteria of when a specific feature is done, we have written a Definition of Done. All the following points should hold for a feature to be considered done:

- All JS code must adhere to the AirBnB code style.
  (https://github.com/airbnb/javascript)
- Code must be well commented.
- The code must be reviewed and tested by another team member.
- The CI build of the branch containing the code must succeed.
- The code must be merged into the master branch.

5
Design & Implementation

This chapter will explain the design of the application and the different design and implementation choices that were made. Firstly, the global structure of the application will be explained using multiple diagrams. The next section will explain multiple implementation decisions that were made during the project, such as frameworks and design patterns. In the section on the GUI we present our own mock-ups and the final implemented design of FleXentral. After that we will elaborate on the implementation of the parser. Lastly, we will briefly discuss the ethical implications of our application.

5.1. Global Structure

This section will explain the global structure of our web-based application. At the start of the project two diagrams were made: a global module diagram and a user state diagram. The user state diagram will be discussed first because this was our vision of how a user should use our application. The next part will discuss the module diagram that makes a clear distinction between the front- and back-end with the data flow of the user. During the first phase we improved the initial design of the front-end and this is displayed in the class diagram.

5.1.1. Module diagram

The module diagram gives a global overview of the application. The arrows in the diagram represent the data flow.

The input module will handle different kinds of input, in the form of a LinkedIn connection or uploading a file. It will then send this input to the processing module on the back-end. This module is responsible for parsing the CV and extracting useful information. This information is then sent back to the front-end where the user can verify the parsed information in the verification module. This verified data is then added to the data model, and after this step the user can add more information to their CV and change the styling in the CV editor module. To generate the final CV in PDF format and offer it as a download to the user, the compilation module is used on the back-end and will provide the front-end with a download request.

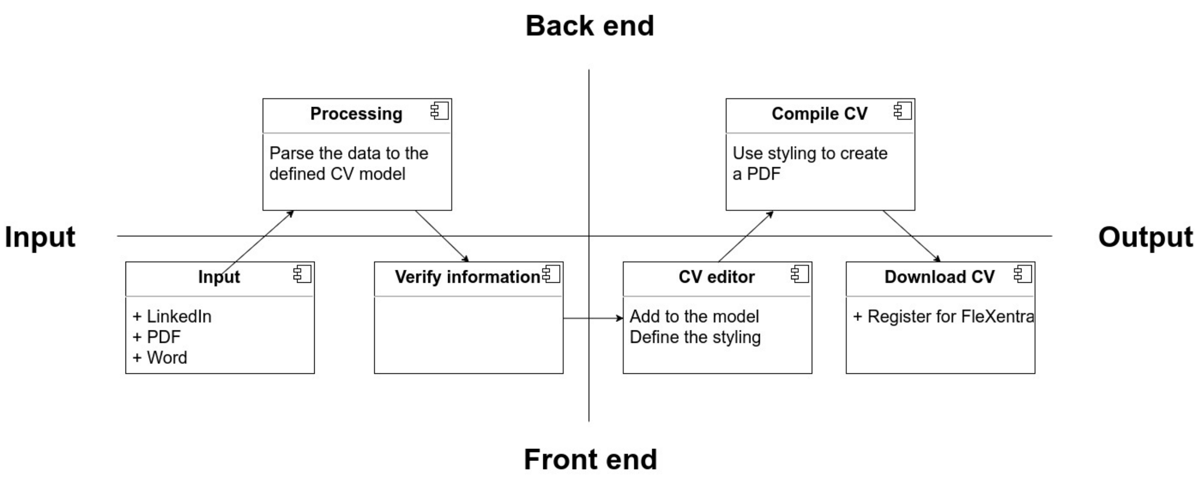


Figure 5.1: Module diagram of the front- and back-end modules

5.1.2. User state diagram

The user state diagram visualises how the user can use the application. They will first arrive at the Home Page where different options will be given: connecting with LinkedIn, uploading an old CV, or starting from scratch. No matter which option is picked, all paths arrive at the Verification Page were the user can edit, add, or remove data. After that step, the user will arrive at the Builder Page where the CV is displayed, along with several options to edit the data or change the styling. Finally, the user can download their CV and they can be forwarded to FleXentral to create an account.

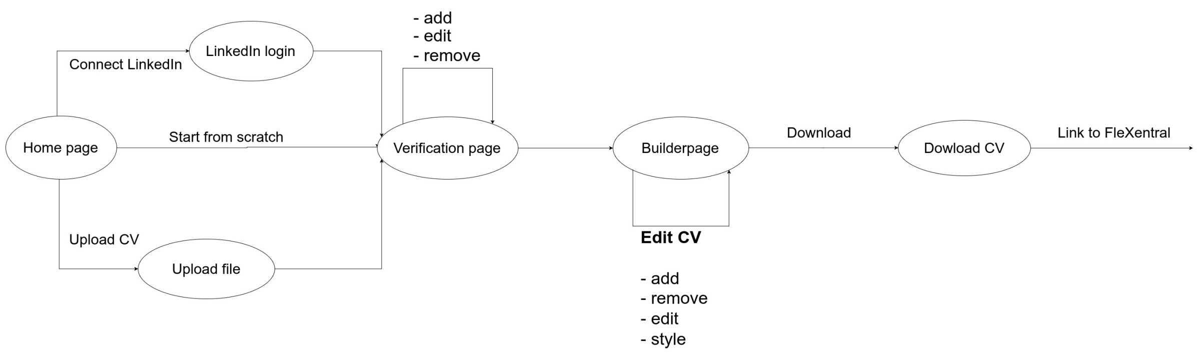


Figure 5.2: User state diagram of the application

5.1.3. Class diagram

The class diagram displays all of the front-end component classes and how they exist in relation to each other. When a class is a child of a parent, the parent will render the child. The methods of each class are either calls to the back-end, routes, or actions dispatched to Redux (Redux will be explained more in-depth in section 5.2.2). In this diagram we can see that the builder class contains multiple children. It contains the menu, which is a complicated class on its own, and the canvas. These classes need to communicate with each other to display the corresponding menu when the user clicks on the canvas. Therefore, the canvas has a method to set the active component in the Redux store. The menu will render according to the Redux store and therefore will always open the corresponding section in the menu.

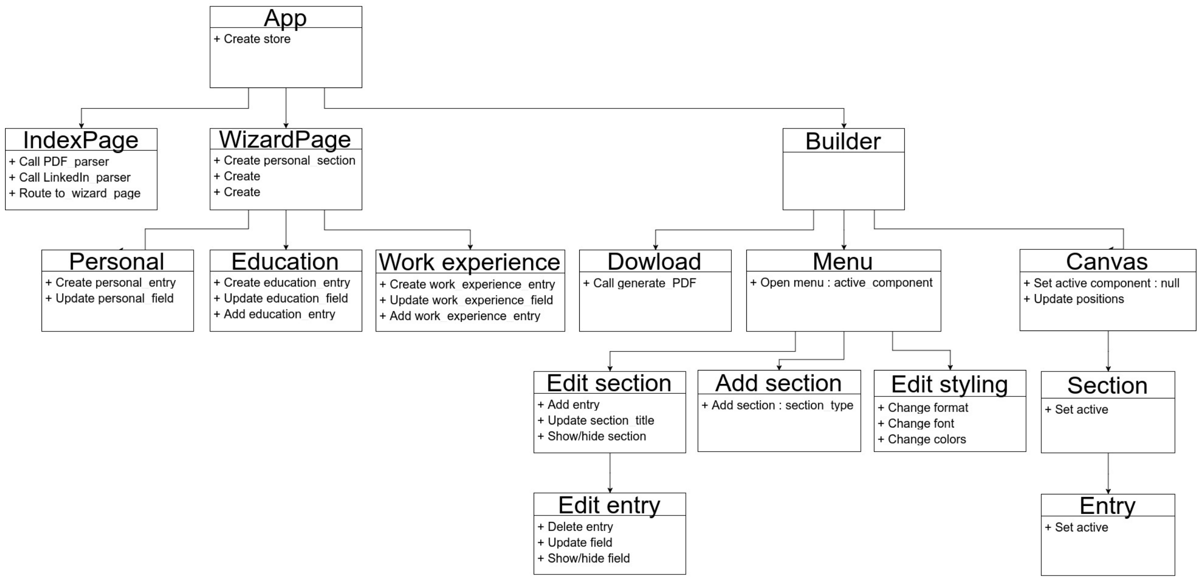


Figure 5.3: Class diagram of the application

5.2. Implementation

In this section we will discuss the different implementation choices that we made regarding frameworks and data storage.

5.2.1. Frameworks

For developing web applications there are a lot of frameworks to choose from. We don't need a framework that captures both the front- and back-end because the requirements for each are different in complexity. Therefore, we made the decision to use a different framework but with the requirement that the language is 100% JavaScript. This section will first discuss the front-end framework, and then the back-end framework.

Front-end

Because there are a lot of options for a front-end framework, we first determined our requirements and we compared different options that match our requirements.

- First off, we want our framework to be mature. This means that it does not have common bugs and that it has a broad community, for example on Stack Overflow. Another advantage of a more mature framework is that it has more plug-ins and examples for common issues.
- Another criterion we value is that the framework is component based. The reason for this is that it allows for a lot of reuse of code because a single component can be used across the application. An advantage of reusing components is that it assures consistency of style of that component, because it is in essence the same component but just reused on a different place.
- Furthermore, because the project has to be finished within 11 weeks, it is important to us that it does not take too much time to master the framework. Preferably we want to be able to start quickly and learn while developing.

We looked at the following frameworks: Angular, Angular 2, Vue, Polymer and React. We decided to use React because the React framework meets all the requirements stated earlier. Another important factor is that one of the members of our team already has positive previous experience with React. The mother project, FleXentral, also uses this as its front-end framework which means that there is a lot of in-house knowledge on using the framework.

Back-end

As mentioned before, the most important design choice we made is that the back-end also needs to be in Javascript. Note that to have a JS back-end framework, it is necessary to use NodeJS as a server.

Because NodeJS only contains the basic functionality for serving, we will not consider it as a framework. The decision was made keeping the following requirements in mind:

- Again, the maturity of the community is an important factor. This way we are sure that there is enough knowledge, and that plug-ins are available so that we don't have to reinvent the wheel.
- Because we want to achieve a clear split between front-end and back-end, we want to use a framework that only provides back-end functionality; it won't have to deal with only front-end logic.

We looked at the following frameworks: Hapi, Sails, Meteor, Kao and Express. We decided to use Express because it's a lightweight framework that is very flexible. It also works well in combination with React.

Parsing

To extract information from files on the back-end, we needed a programming language that could open files, process the information and perform multiple analyses, while doing this in a reasonable amount of time since the user has to wait. We picked python as our parsing language because it has the following benefits:

- The main reason was that python has a powerful library support. There were multiple libraries for parsing PDFs or doing natural language processing which can be implemented very easily.
- Python has a low learning curve. This was an important factor because nobody in the team had any experience with this language.
- It supports very powerful operators and iterators out of the box. This way we can create very powerful loops that can be used to analyse the parsed text.

The back-end will spawn a python process on the server and will wait for the parser to give back information.

5.2.2. Design patterns

To maintain a clear separation between the data and the view, a design pattern was needed. There were two options, MVC or Flux.

MVC

MVC stands for the model-view-controller pattern and is widely used in a lot of projects. The model manages the data of the application, and the view displays the data from the model. Lastly, the controller takes user input from the view, manipulates the model and will update the view. Because of this separation, the view can be easily replaced and the model has better testability. But a big problem with this pattern is that it is common to have a bi-directional data flow. Because a user action in the view will flow to the controller, the controller will update the model and the model will notify the controller that the view needs to be updated. These kind of data flows do not work properly in React.

Flux

In Flux, data flows are unidirectional. The patterns consist of four parts: actions, dispatcher, store and views.

- The actions are simple objects that define an action and their data.
- The dispatcher is the central point and processes actions and calls the functions that the store has registered with certain actions.
- The store is the state of the application and includes all the logic that is needed to process actions.
- The view displays the data.

The data flow of Flux is visualised in Figure 5.4

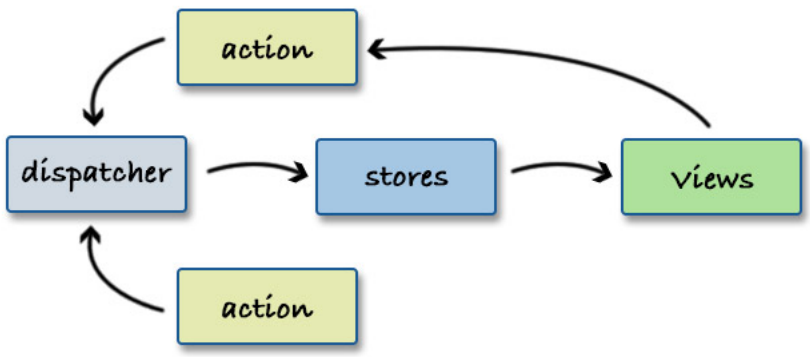


Figure 5.4: Visualisation of the data flow in Flux

Redux

We implemented the Flux design pattern in our application in the form of Redux as a front-end database. This is a package that provides connections for React components to the store. The Redux store provides the application with a state object that is the source of truth for every view. Each view defines what data it needs and what actions are dispatched. After an action is dispatched, the store will be updated and the view will receive the new values from the state object.

The data model that is used in the Redux store of our application is visualised in Figure 5.5. Each part of the state object has its own handler, called a reducer. These reducers take an action and change the state to the new state according to the action. They only have access to their specific part of the state object and thus only change this part. For example, the section object only has access to 'state.sections' and can only change this part of the state object. The section reducer will handle all actions that have a type that corresponds to a section action. Important to note here is that these reducers must be pure functions according to the Redux design principles. This means that functions, given the same input, always return the same output. It must also not produce any side effects or mutate the state object.

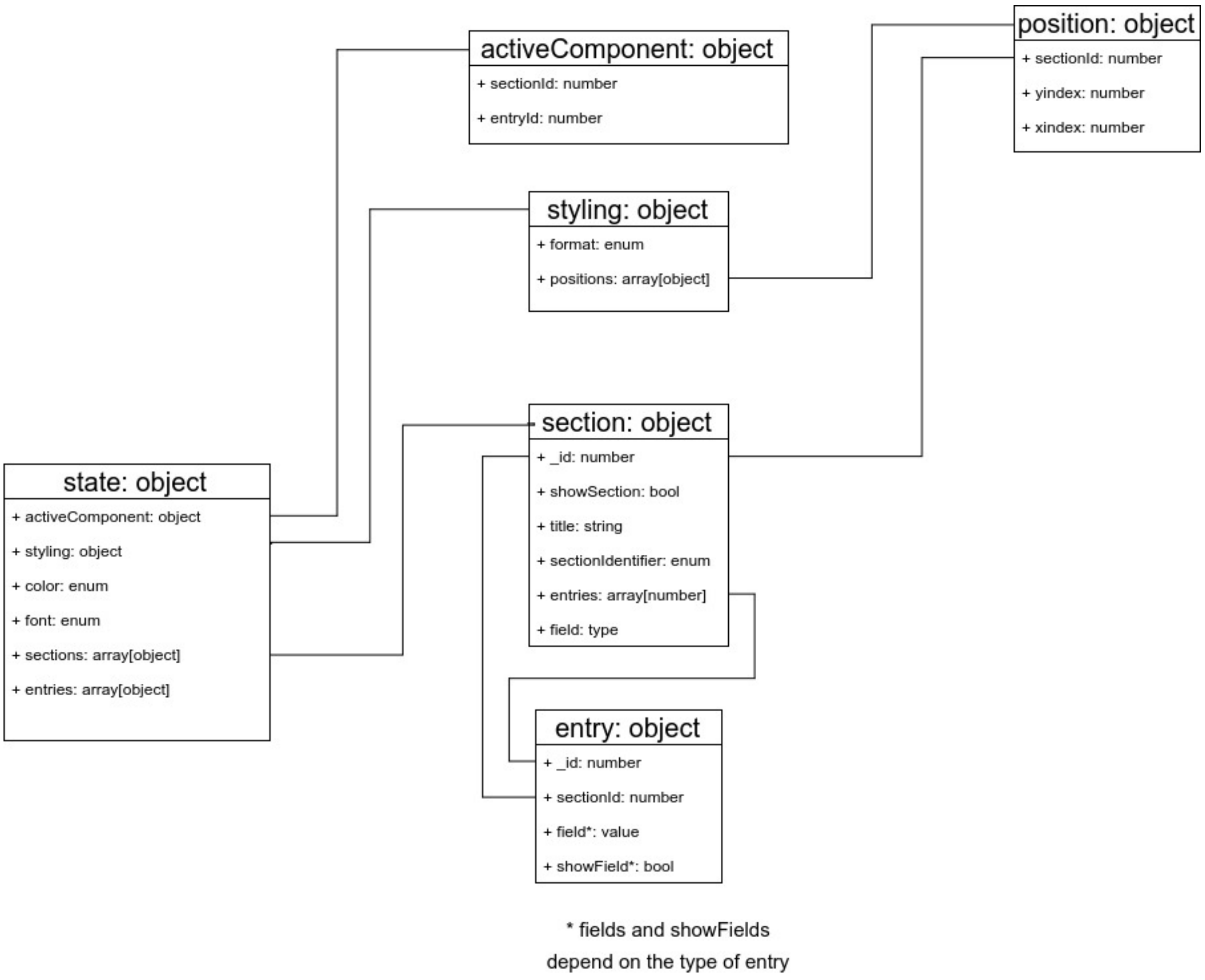


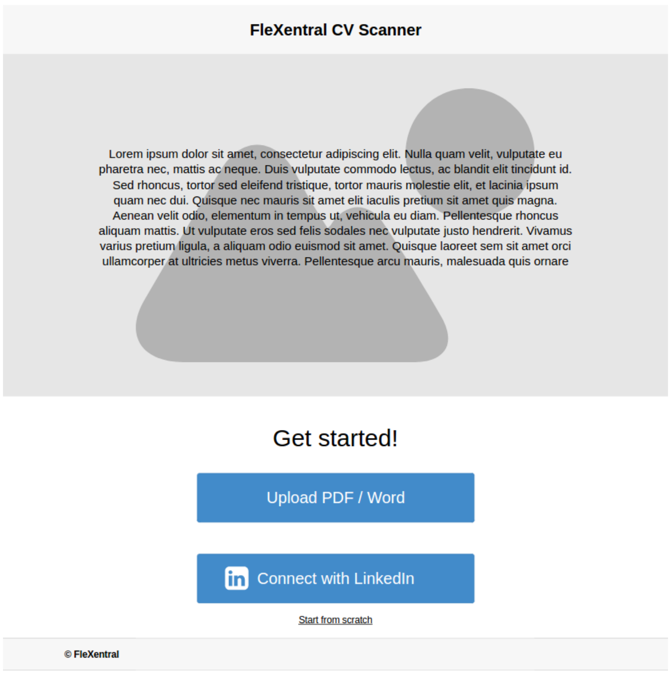
Figure 5.5: Structure of the implemented Redux Store. The state object contains multiple objects that each have their own reducers that handles actions.

5.3. GUI

As explained in the previous parts the application can roughly be split in three parts, namely the index, verification and builder. For the first working version we made our own mock-ups because styling did not matter a lot in this phase. After our first working version we received the final design from the UX designer of FleXentral. We will show both the mock-up and the final design for each part of the application. The final version of the application matches the designs of FleXentral.

5.3.1. Index

The index page is the first page that the user sees. Therefore, it needs to be clear what options there are and what the application does. This can be seen in the design in Figure 5.6 because it features big buttons with clear actions. This page was later changed according to the design of our UX designer to match the overarching styling of FleXentral, displayed in Figure 5.7.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed felis sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet. Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris, malesuada quis ornare

Figure 5.6: Mock-up of the Index page

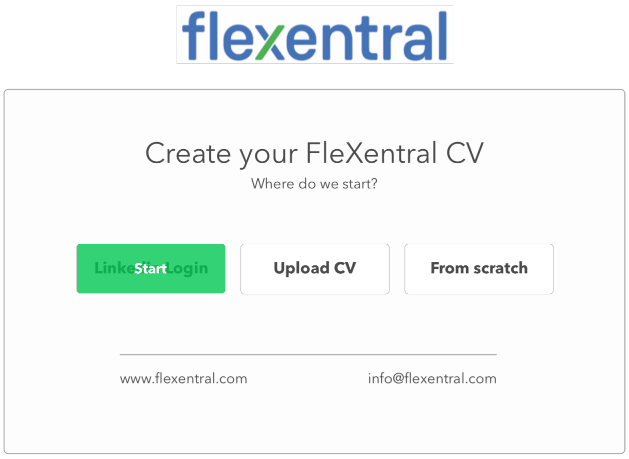


Figure 5.7: Design by FleXentral of the Index page

5.3.2. Verification

In the Verification part of the application the user can check the information that LinkedIn or the CV parser provided, or start from scratch. In the first design every section, such as personal information (Figure 5.8) and work experience (Figure 5.9), had its own page. The UX designer of FleXentral came with the feedback that the progress bar on the verification pages is unclear, it does not display actual progress but only on which page of the verification process the user is. Thus in the new design, green, yellow or red dots mark your progress in the different sections (Figure 5.10). The button to go to the CV building pages will only appear if all dots are green (Figure 5.11).

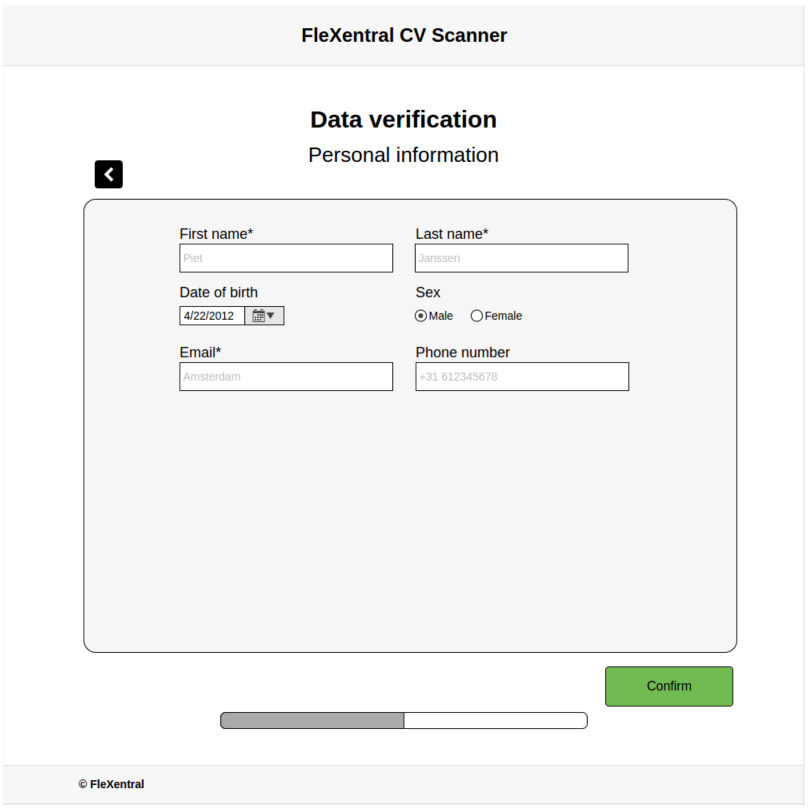


Figure 5.8: Mock-up of the personal information verification page

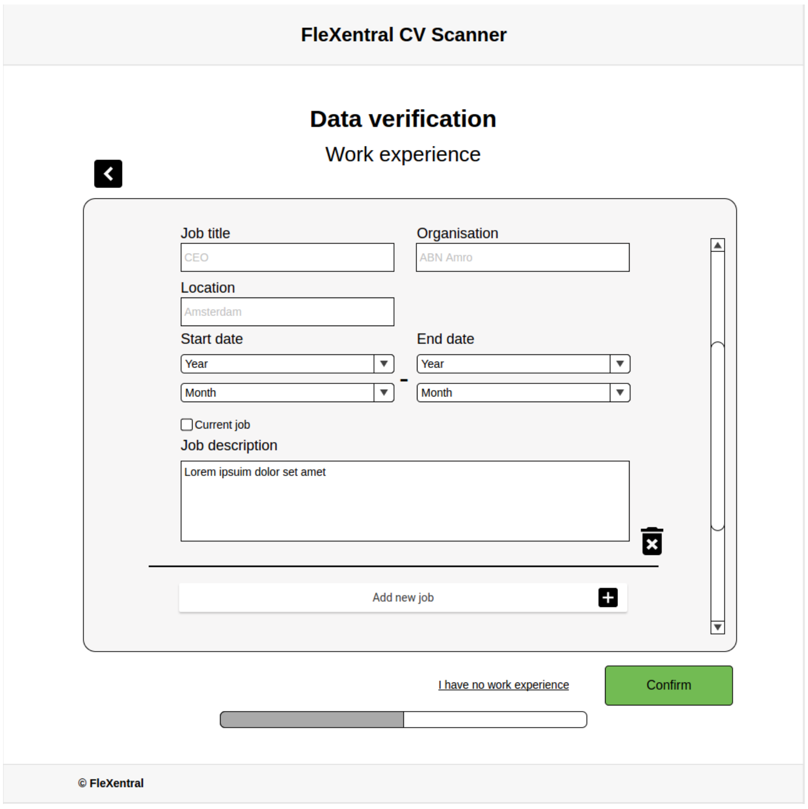


Figure 5.9: Mock-up of the work experience verification page. The education verification page had the same design but different fields

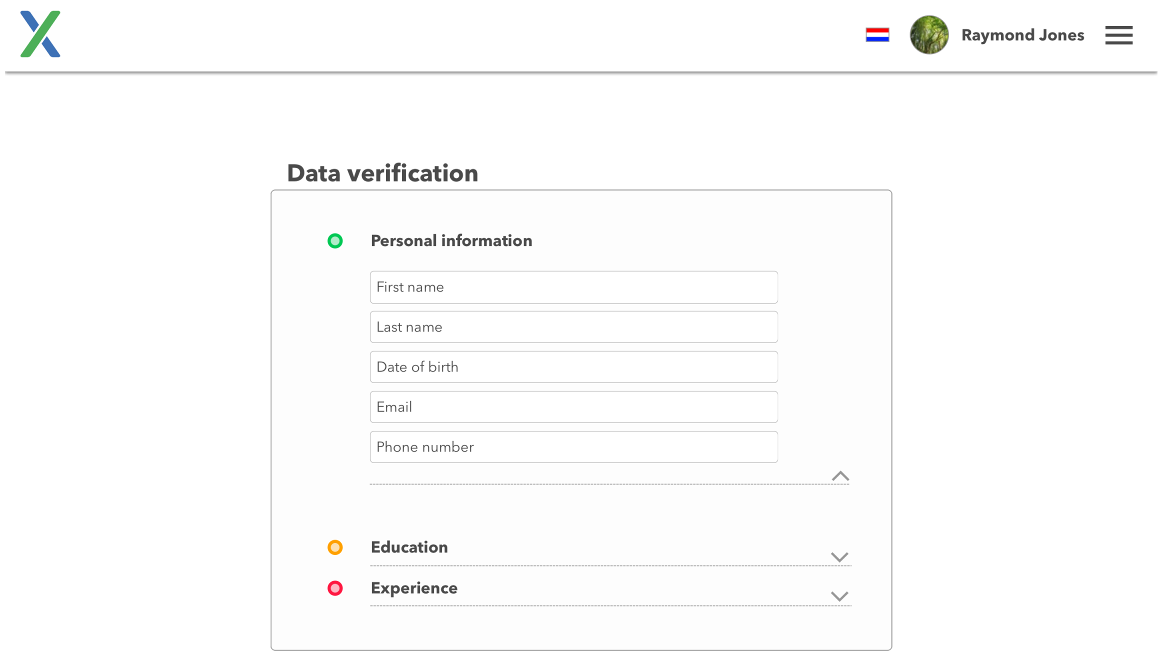


Figure 5.10: Design by FleXentral of the Verification pages. Note the dots that mark the progress

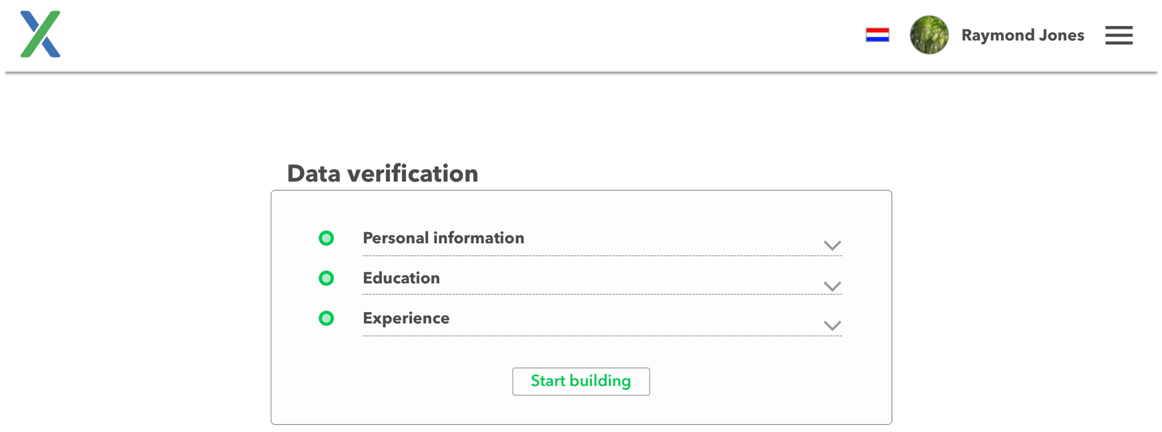


Figure 5.11: Design by FleXentral of the Verification pages. Note the 'Start building' button because all dots are green

5.3.3. Builder

The builder consists of two parts, the menu and the actual CV. We found it most important that the menu was responsive with the CV. So when a user clicks on a section or an entry in that section, the corresponding menu was opened. For sections this is displayed in Figure 5.12 and for entries in Figure 5.13. After our first prototype we received the feedback that it was not clear in which menu the user was, and how the menu navigates. Therefore, the menu was changed to always display all the options without losing the responsiveness (Figure 5.14).

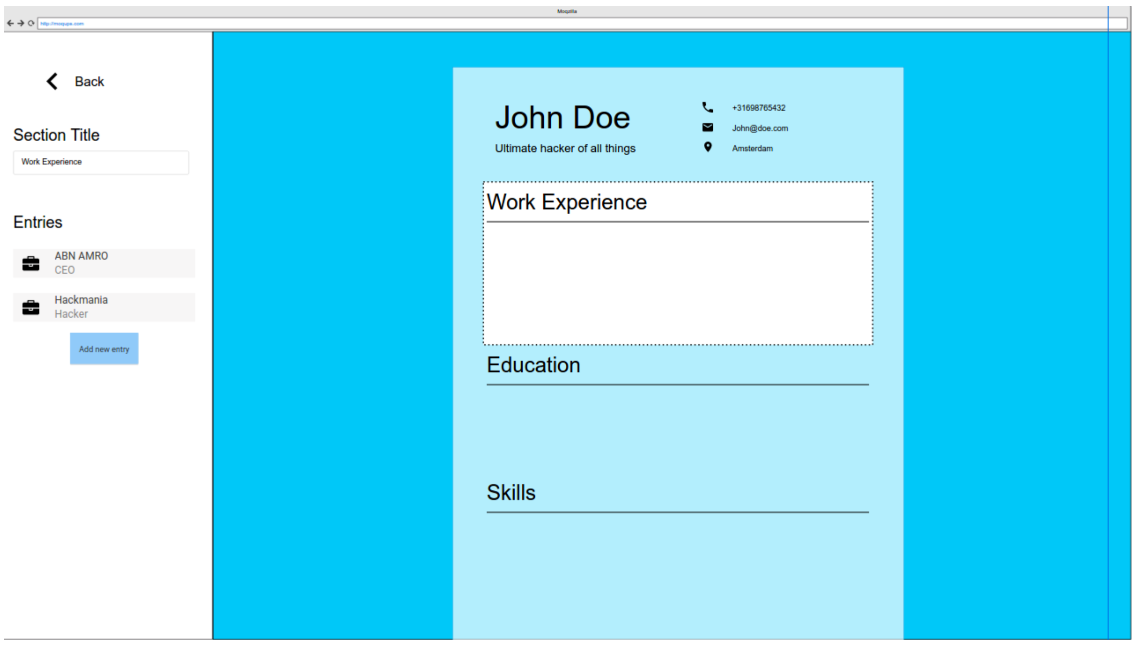
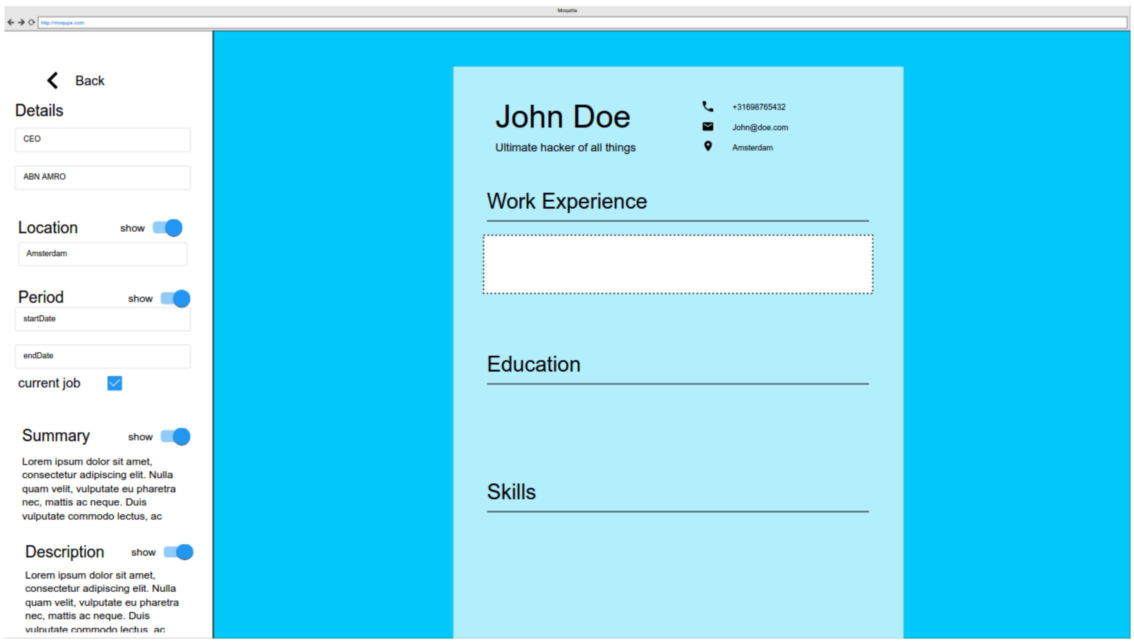


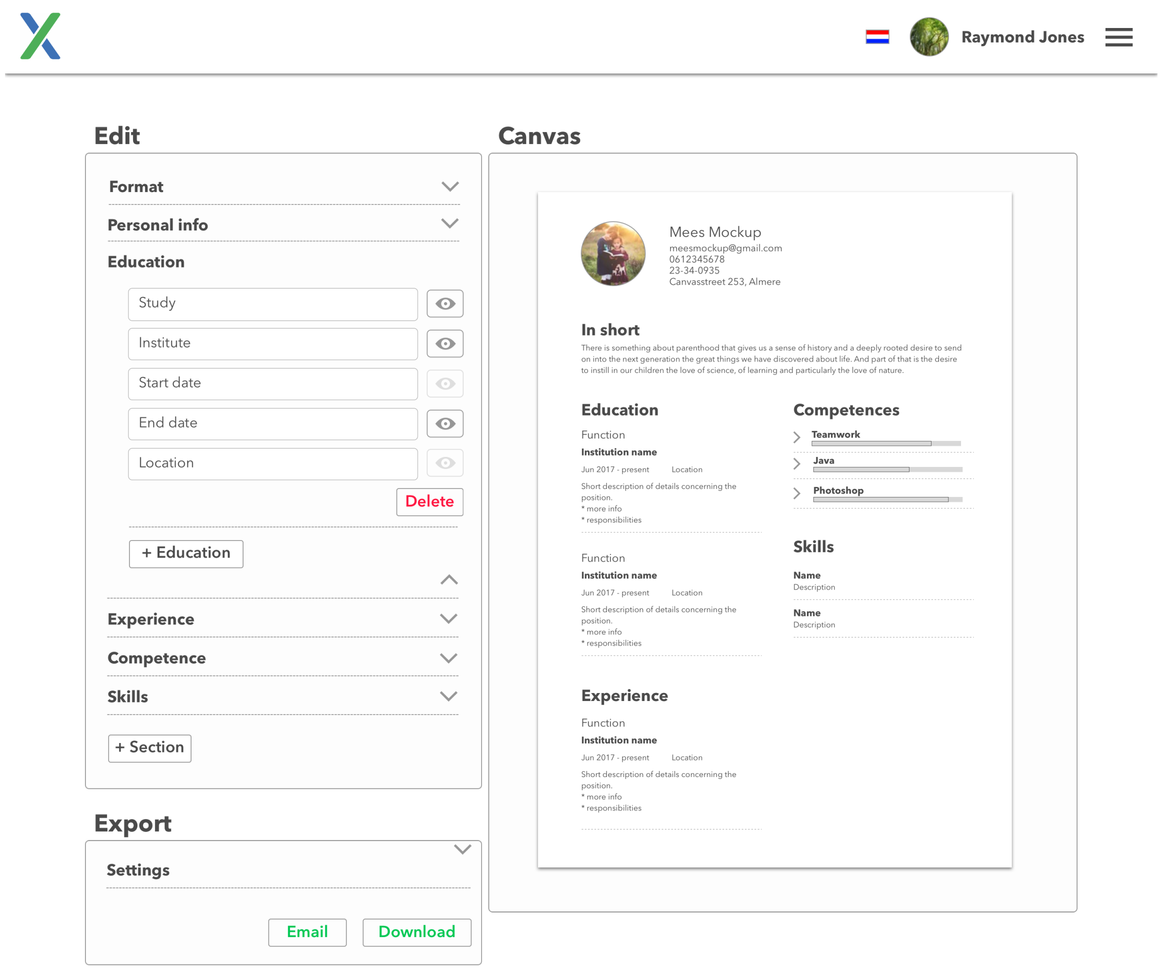
Figure 5.12: Mock-up of the section menu for CVs



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac

Figure 5.13: Mock-up of the entry menu for CVs



Raymond Jones

Edit

Format

Personal info

Education

Study

Institute

Start date

End date

Location

Delete

+ Education

Experience

Competence

Skills

+ Section

Export

Settings

Email

Download

Canvas

Mees Mockup
meesmockup@gmail.com
0612345678
23-34-0935
Canvasstreet 253, Almere

In short

There is something about parenthood that gives us a sense of history and a deeply rooted desire to send on into the next generation the great things we have discovered about life. And part of that is the desire to instill in our children the love of science, of learning and particularly the love of nature.

Education

Function
Institution name
Jun 2017 - present   Location
Short description of details concerning the position.
* more info
* responsibilities

Function
Institution name
Jun 2017 - present   Location
Short description of details concerning the position.
* more info
* responsibilities

Competences

Teamwork

Java

Photoshop

Skills

Name
Description

Name
Description

Experience

Function
Institution name
Jun 2017 - present   Location
Short description of details concerning the position.
* more info
* responsibilities

Figure 5.14: Design by FleXentral of Builder page. The menu now displays all sections that can be changed

5.4. Parser

Parsing the CV submitted by the user and extracting information from it roughly consists of three phases. First, we have to parse the text. Second, we split up the parsed text into the different sections contained inside the CV. And third, we extract information from each section with a section-specific parser. We shall highlight the design and implementation for each of these phases in the remainder of this section.

5.4.1. Parse the text

Parsing the text of the document correctly is of critical importance to achieve success in the following steps. A CV is structured by sections, meaning similar information blocks are grouped with each other. If the parser were to mess up the order of the lines, attempting to split the parsed text into sections is futile. There are several tools available for the parsing of documents, especially for the parsing of PDF documents. Two worth mentioning are PDFminer (https://euske.github.io/pdfminer/ and Apache Tika (https://tika.apache.org/). The former is a pure python library, and is generally rated as one of the best open-source parsers. The latter requires a connection to an Apache server, a process which we experienced taking too long and thereby slowing down our application. For these reasons, we decided to stick with PDFminer to parse the text of the CV.

5.4.2. Split into sections

After parsing the text, and before we can make any sense of it, we should divide the text into sections. If a person were to look for a specific section, they would identify that section by its header. A section header is typically characterised by its text, such as 'Work Experience' or 'Competences'. Other word-level features, such as having relatively bigger and less occurring font types, are also typical for a header. With this in mind, the person reading the CV is easily able to identify the header they are looking for. The implementation of our program is no different. By comparing a word's font size and the occurrence of that font size to the average font size and font size occurrence respectively, we are already able to successfully identify each header. When the splitting of sections by their headers is done, we should still send each section to their corresponding section-specific parser. This is done by fuzzy list matching the headers to commonly used keywords, as explained in subsection 2.2.3. The amount of words used for headers in practice is so limited that this approach will almost always succeed. A visual explanation of this can be found in Figure 5.15. One special case however, is the parsing of the header of the CV. This is because the header of the CV is often the name of the person the CV belongs to. This header is then typically followed by the personal information section, and is not always introduced by a separate header (e.g. 'Personal Data'). To recognise this, we make use of the Stanford Named Entity Recognition Tagger. This tagger is able to identify a sequence of words as a name of a person very accurately. By tagging the words of the CV header, we can identify whether or not the name is contained in it. If this is the case, we can assume this will introduce the personal information section.

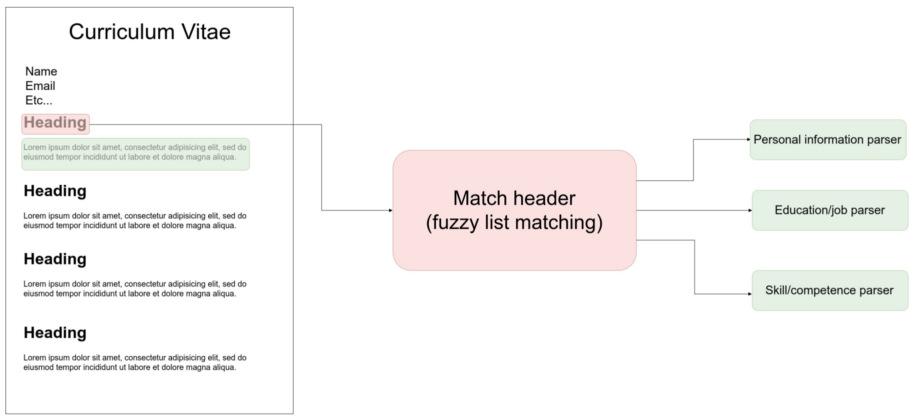

Curriculum Vitae
Name
Email
Etc...
Heading
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Heading
Heading
Heading
Match header
(fuzzy list matching)
Personal information parser
Education/job parser
Skill/competence parser

Figure 5.15: Visualisation of the fuzzy list matching for recognising headers.

5.4.3. Extract information

After dividing the parsed text into sections, we can finally attempt to extract information from each section. Because the way information is presented can be substantially different between two sections, the way the text is parsed is section-specific. On the other hand, some sections are very similar to one another, and may only differ in the keywords they usually contains. With this in mind, we can roughly define three different parsers. We shall shortly walk through the design of each of these section-specific parsers.

Personal Information Parser

Parsing personal information is very unique, in that some fields always follow a specific pattern. Therefore, we are able to gather phone numbers, email addresses and dates of birth by matching them to a regular expression. Retrieving information about a person other than email, date of birth and phone is hard to define by a regular expression. Luckily, it is very common to define the fields before naming them on your CV. An example of this would be: "Address: Hacker Street 42, Hackland". Therefore, we check for each line whether this is the case. If so, we are able to store the rest of the line under its definition. So in our example, we would save "Hacker Street 42, Hackland" as an address. Other than these two methods, we are very limited in our options. Very trivial information, like gender and marital status can of course be done by keyword matching, but other information like address will remain very hard to obtain.

Competences and Skills Parser

After parsing the Personal information, Work Experience and Education sections, in most CVs the remaining unparsed sections are all rather similar to one another. This is because they all sum up certain traits of the owner of the CV, whether it be their skills, their interests or something else that characterises them. However, the way to identify these traits differs slightly per section. For example, in the competences section you may want to match the words against a wide array of pre-defined competences to ensure they are genuine competences. One possible list of competences to match against is the ESCO, a project by the European Commission.[7] In interests however, you will most often want to count every word as a different interest. These are minor differences, but the general idea behind the parsing of each of these sections remains the same.

Job and Education Parser

Parsing the jobs or educations from the CV is the most difficult task to perform out of the three parsing jobs. This is because the information about one job or education, which we call an entry, is grouped within multiple lines instead of one single word or line. Before we can parse an entry, we should divide the section by each entry. To separate one entry from another, we look for something every entry contains. By experience, we know that each entry contains the period in which the owner of the CV participated in that entry. By finding these periods through regular expression matching, we are able to successfully divide the entries.

5.5. Ethical Implications

Our application deals with sensitive data because users enter or upload private data. The most sensitive part of a CV is the personal information section. It contains a full name, address, phone number and sometimes even a citizen service number. To save this kind of information persistently on the server, users must agree to terms and conditions in which they declare that our application can use this data. These agreements have to be made by a lawyer to avoid any problems in the future. To evade this hassle, we made the design choice that no data is saved persistently on the server. Instead we use Redux to save data and no database in the back-end. Because Redux only works on the front-end, and thus the client side, all of the sensitive data stays on the device of the user. The CVs that are uploaded to the server get a randomised name and are removed after parsing. When the user is done and downloaded their new CV, the server also removes these files.

The biggest downside of this choice was that a user has to restart building their CV if the page is reloaded. But because the workflow in our application is to build your CV and then create an account on FleXentral, this was not a breaking problem. Only when the user registers, their data will be saved but then they will have to agree with the legal agreements of FleXentral.

6 Discussion

In this chapter we will reflect on the process and the development, and after this we will reflect on how the requirements we defined in Chapter 3 and Appendix B are met in the final product. We will also discuss the SIG feedback and how we used it.

6.1. Process

To reflect on the process during the project we divided this section into three parts: the sprint cycles, the communication, and the teamwork. We will start by reflecting on the sprint cycles.

6.1.1. Sprint cycle

At the beginning of the project we decided to do weekly sprints where we would generate issues and try to estimate how long these would take. During the project we noticed that tasks took longer than expected and we were more keen to get tasks fully done, which resulted in some delay according to the planning. Looking back we should have made smaller tasks with better definitions so that the time estimation was easier and more accurate.

Due to the fact that tasks were rather big, the implemented features also gave substantial code changes in each pull request. This made code reviews harder because it sometimes took long to make sure that all of the changes introduced no bugs and had no side effects. This could have been solved by using the git feature branch approach where small tasks are built separately and are then merged into this feature branch, and only later into the master branch. Luckily these problems never became too large and the master branch always had a working version.

6.1.2. Communication

We did most of our work in the office of FleXentral, at least four days a week. Because we were working closely together the communication in the group was very good. All questions about each others work could be asked instantaneously. Working in the office of FleXentral also allowed for quick communication with the client of the project. This way we could make sure we had the same idea of the product we were making. We also set up a Slack channel for communicating with each other. This was mainly used to make it easy to communicate information that would be lost if done verbally. It also allowed for a communication channel with the coach from the TU Delft.

6.1.3. Teamwork

The team dynamic was good through the entire process of the project. Everyone in the team was forgiving of each others mistakes and gave each other the time to learn the skills they did not have yet. Due to this forgiving environment, nobody held back any questions about the code or techniques we were using. All this resulted in a project without any conflicts within the team, which allowed us to focus on development.

6.2. Development

This part will reflect on the development of the application. There were some problems that we encountered during the project and we will briefly discuss them here.

6.2.1. Learning curve

An issue we had during development was the fact that not everybody had prior experience with the techniques and frameworks we used. This resulted in a lot of early code that was not written in the correct fashion and had to be corrected later. Unfortunately this is inevitable, however the problem could have been worse if we had chosen techniques with a steeper learning curve.

6.2.2. Redux

The implementation of Redux didn't go as smoothly as hoped. This was caused by the fact that we made the decision to use this pattern only after a few weeks. Therefore, a lot of code had to be refactored because each component class saved data in its own state. While the idea of Redux is to have simple, pure fuctions that only change a specific part of the state object, implementing it required a lot of boilerplate code. It can be done in a smarter way with less code, as shown by the lead developer of FleXentral, but this requires more time and skill that we did not have during this project. Due to Redux, we also dropped our database for which we already made models and schemas.

6.2.3. Design

At the start of the project we made our own designs. Therefore, the app functioned in a specific way and the menus were technically designed to function according to this. When we received the final designs from FleXentral, a lot of parts had to function differently and this required some refactoring. Luckily we had React as a framework so we could reuse components in a different way, but this did cost more time than expected. In future projects it would be useful to receive the design in an earlier stage.

6.2.4. Parser

During our research phase, we identified the parsing of CVs to be a currently challenging problem. What we did not expect, was that simply parsing the CV in the correct order would already be too big of a challenge for existing document parsers. With text being parsed in an incorrect order, it becomes even more difficult to analyse the text. Unfortunately, this is beyond our reach and we would have to deal with this. However, in the event where the text is parsed in a correct manner, we were rather surprised at how easy it was to extract information. Simple pattern and fuzzy list word matching are very often sufficient due to the structured nature of a CV. All in all, we are satisfied with the results we achieved. We achieved our goals, which from the beginning were set rather low as we aimed to parse just basic information from the CV.

6.3. Requirements

To reflect on the requirements we will look at the requirements as described in Appendix B. We will reflect one these requirements per module and look at to what extent the requirements are met in the final product.

Looking at the requirements defined for the input module we see that both Must Haves have been implemented. In subsection 6.2.4 it is described how well the PDF parser works. Parsing of Microsoft Word documents proved even more difficult because of the XML format that is uses. We did manage to compile Word documents into PDF and then use the PDF parser to extract data from the CV. We did not come to the point of parsing batch imports to our CV model because it proved to be technically more difficult then anticipated.

As for the data processing we achieved both Must Have requirements. However correctly extracting education and work experience from a CV was more difficult than expected at the start of the project and did not work properly. We didn't get to extracting data from a database because we didn't get to implementing batch imports. We do have support for both Dutch and English CVs, mostly for personal information.

For the data model module we did everything we wanted to do, meaning we implemented all Must and Should Haves.

For the output module we did achieve to generate a new CV, add real-time editing of the final result and to format the CV in style of FleXentral. However at the time of writing this report it is not yet possible to register on FleXentral. When this is achieved later in the project is will also easily generate a matching score with companies on FleXentral. We did achieve to have multiple formats for the CV which we defined as a Could have but it was easily implemented because of the design choices made earlier in the project.

Lastly for the Front-end module we achieved all Must Haves, except for, as mentioned earlier, the registration page for FleXentral. As for the Should Haves we did implement a uniform styling with FleXentral, but we did not yet spent time on search engine optimisation.

6.4. SIG Feedback

On the first feedback of the Software Improvement Group were received a score of 3 out of 5, which means that our application has an average maintainability. The problems with the code were Unit Size and Duplication.

6.4.1. Unit Size

For unit size we got the feedback that we used a lot of React (.JSX) code to create elements in the menus. These parts could be created programmatically with the different helper methods of React. The menus where these problems occurred are only container components. They are 'dumb' components because they have no own state or methods and only a render method that defines how the component looks. Therefore, we did not change this because we think that JSX is more elegant and should be seen as HTML code here.

6.4.2. Duplication

For the duplication aspect, SIG will look for duplicate code and we got the feedback that some menus use the same layout. This could lead to problems in the future when the layout gets changed but the developer forgets to change one menu. We discussed this point but came to the conclusion that a change was not needed.

The reason that the menus are made this way is because entries in these menus can be very different. If we want to add another field for one specific entry in the future, the layout for that specific menu could change. Therefore each menu needs their own layout in our opinion. Also, an important note is that each input field in the menus are wrapped in a component. So if we want to change all of the input fields in the future, one could simply change this component and the whole application will change.

7 Conclusion

During this 11-week project we developed a working and deployable tool in which users can build their CV, either from an existing CV or from scratch. It's easy to use and requires little effort, thus addressing our initial problem: 'It's difficult to create a professional CV'. The project encompassed a full development cycle, from initial assignment to deployment.

Our main research question was: 'Given the huge diversity in structure between CVs, how do we extract meaningful information from them?'. It turned out to be quite hard to accurately parse a wide range of CVs. We managed to implement parsing of personal information and skills with rather high certainty, but accurately extracting details about different job and education entries remains challenging. We do think that, given more time, we could succeed in parsing more common CV sections, and also achieve a higher success rate.

We've learnt valuable lessons over the course of the project. Firstly, we learnt that good planning is really vital, especially in projects like this that have a strict time frame. This also means devising a realistic set of requirements; we had to drop certain features in the interest of time. We have to note that this is easier said than done, as it is difficult to accurately estimate the workload beforehand.

Secondly, it would have helped if we thought more thoroughly about the technical requirements for the product. We had to change and add some frameworks during the course of the project, which resulted in a significant refactoring overhead.

We have some recommendations for future development of the application, which can be found in the next section.

7.1. Recommendations

The most pressing feature that needs to be implemented is an option for registration with FleXentral. This is part of the core requirements and will make the application useful for FleXentral. This is a small feature but requires good communication with the development team of FleXentral.

Another good feature to implement would be to offer registration and log-in for the application. This would allow users to save CVs, so they can pause working and pick up where they left off later on. A possible implementation could be to let users use their FleXentral account to save and update their CV on our application. This would link the possibility to register for FleXentral and register for our application, but this is something that has to be looked into further as this might have unwanted effects.

As described in subsection 2.2.1, at this point we do not have the proper data for a machine learning approach. However, it would be interesting to re-evaluate this decision in a later phase of the application. Specifically when a bigger and more representative database of CVs is available, either through FleXentral or through registered users.

Lastly, extra content in the CV building part of the application would greatly improve it. It is possible to look at different styles of CV, such as professional or more creative templates. Furthermore it is possible to add more colour and font options, this is a very easy feature to expand, but all extra choices should fit the application so selection must be critical. It is also possible to create multiple extra section types that can be found on CVs, things like extracurricular activities or recommendations.

A
Project Description

In a society where flexible work is becoming more and more common, it is important that, as a flex worker, you have an up-to-date overview of your qualifications. Unfortunately, it is more common than not that a lot of outdated CV's are spread everywhere, and this reduces the chances of getting offered the best jobs for your qualifications.

To solve this problem, we want you to build a standalone platform where users can store and update their CV. In addition to this, a standardized data structure must be developed that works for every professional.

Features we want:

- Uploading a CV which is then automatically scraped and stored in the standard data structure.
- Connection to external sources like LinkedIn to retrieve information automatically.
- Realtime generation of CV pdf from the stored data.
- Creating a (basic) matching algorithm that can match CV's to job profiles (in the same structure) based on the stored information.
- An API to integrate the tool into our existing platform.

The goal of the project is to create a platform where the user can create and update beautiful CV's, with as little effort as possible.

Company description

FleXentral is a one year old Dutch startup financed from angel capital and led by two software veterans: André Bonvanie and Onno Hektor. Our office is situated in Almere and our team consists of six motivated engineers. We are young, energetic and have an ambitious goal of changing the way the flexible workforce is managed.

Auxiliary information

The project team, which is already selected (Daan Vermunt, Niek van der Laan, Kasper Wendel & Tom Brouws), will work part-time at location in Almere. There will be a weekly scrum session, lead by the lead developer of FleXentral, who will also guide them during the project.

B Requirements MoSCoW model

B.1. Modules

B.1.1. Input

Must have

- Read CV in PDF format
- Link with LinkedIn

Should have

- Read CV in Word format
- Batch database imports to Data model

Could have

- Read CV in Google Doc format
- Link with Github
- Link with Stack Overflow
- Read image PDFs

Won't have

- Link with Social Media profiles (e.g. Facebook)

B.1.2. Data processing

Must have

- Extract personal information and competences from CV
- Extract all info given by LinkedIn API

Should have

- Extract education and work experience from CV
- Support Dutch and English CVs
- Extract all info from entire database

Could have

- Linking different input sources
- Linking similar competences for later matching
- Recognise the language of CV

B.1.3. Data Model

Must have

- Personal information
- Previous work experiences
- Education
- Skills/competences with a grade
- Data consistent with FleXentral platform

Should have

- Extracurricular activities/charity
- Level of competence
- Profile picture

Could have

- Personal interest
- Miscellaneous info (Hobbies, languages, etc.)
- Similarity measure between profiles
- User defined priorities/levels/scores on skill

B.1.4. Output

Must have

- Generate a new CV
- Easily create user profile on FleXentral

Should have

- Competence score with companies on FleXentral
- Real-time editing of final result.
- The CV can be in style with FleXentral.

Could have

- API (or data connection) for FleXentral
- Multiple formats for CV
- Select which items of the data model go to new CV
- Prioritise items on new CV

B.1.5. Front-end

Must have

- A landing page
- An input page
- A page to download CV
- A page for controlling the data model
- Registration page/link for FleXentral

Should have

- Search Engine Optimisation
- Uniform styling with FleXentral

Could have

- Login after registration to alter and re-render CV
- A CV strength score
- CV improvements tips (e.g. use shorter info blocks)

C

SIG Feedback (Dutch)

First round

[Analyse]

De code van het systeem scoort 3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Size en Duplication.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt.

In jullie geval ontstaan die lange methodes niet zozeer door de hoeveelheid logica, maar door de hoeveelheid JSX. In sommige gevallen is het volume van de JSX echter vermijdbaar: in EditPersonalSectionMenu.js bestaat de JSX bijvoorbeeld uit steeds hetzelfde stukje. Het is dan beter om de layout programmatisch op te bouwen (met React.createElement). Dat is minder elegant dan met JSX, maar je hebt wel 10x minder code nodig.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de langere methoden ook naar voren als de meest complexe methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

Bij Duplication speelt er een soortgelijk probleem: jullie copy/pasten stukken van de layout tussen verschillende componenten, bijvoorbeeld tussen EditEducationEntry.js en EditWorkExperienceEntry.js. Dit is een klassiek probleem binnen front-end development, en leidt op termijn tot problemen op het moment dat iemand de layout wil aanpassen maar één van de kopieën vergeet. Dit zijn nare constructies om op te lossen, aangezien de layout deels gelijk is maar niet helemaal. Desondanks biedt React genoeg constructies (zoals fragments) om de hoeveelheid duplicatie nog wat te verlagen.

De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Over het algemeen scoort de code dus gemiddeld, hopelijk lukt het om dit niveau nog iets te laten stijgen tijdens de rest van de ontwikkelfase.

Infosheet

General information

Title: CV Scanning Tool
Client: FleXentral B.V.
Presentation Date: 4 July 2017

Description

To find a new job you need a good CV, but creating a professional one can be hard and cumbersome. FleXentral is a startup company that aims to bring companies and flex workers together in a full-circle online platform, and they would like to make this process easier.

The main challenge was the reading and parsing of existing CVs and using that data in a meaningful way to fill the new CV. We performed research on this subject, and our findings were that this is a very difficult program, which could not be solved entirely in the scope of this project. We decided to focus on the CV building aspect of the software, while also allocating some time to implement the parsing of specific CV elements.

We divided the project in three phases, and used a backlog and issue tracking to ensure a smooth process. An unexpected challenge was the chaotic nature and amount of work of the second phase, which we overcame by extending it a bit to meet all demands.

The final product became a tool that can build CVs, either from an old CV or from scratch. The user can easily fill in their information, select the desired segments to put on their CV, and download a PDF of the CV. We tested this application by performing tests both programatically and manually.

The product will go live as a separate entity, linking to the FleXentral platform somewhere in the process. In the future the software could be improved with more options and CV templates, a log-in system, or improved parsing of old CVs.

Project Team

T.H. Brouws, Interests: Software Security, Code Performance, Algorithms
Contributions: Dev Ops, Code Quality, Front-End Development
N. van der Laan, Interests: Algorithm Design, Web Development
Contributions: Front-end Development, CV parsing algorithm
D. Vermunt, Interests: New Programming Languages, Web Development, Maintainable Code
Contributions: Front-End Development, Back-End Development, Styling
K. Wendel, Interests: Web Development, Big Data, New Programming Languages
Contributions: Front-End Development, Implementation of Redux, Refactoring, Styling
All project members contributed to the final report and the final presentation.

Client & Coach

O. Hektor - Owner, FleXentral B.V.
Dr. C. Lofi - Web Information Systems, TU Delft

Contacts

T.H. Brouws - tom@brouws.nl
N. van der Laan - niekvanderlaan@outlook.com
D. Vermunt - daanvermunt@gmail.com
K. Wendel - kasperwendel@gmail.com

The final report for this project can be found at: http://repository.tudelft.nl

Bibliography

[1] *Stanford CoreNLP – Core natural language software,* https://stanfordnlp.github.io/CoreNLP/, accessed: 2017-04-24.

[2] H. . Schmid, *Probabilistic part-of-speech tagging using decision trees,* Proceedings of International Conference on New Methods in Language Processing (1994).

[3] A. Van den Bosch, G. Busser, W. Daelemans, and S. Canisius, *An efficient memory-based morphosyntactic tagger and parser for dutch,* Selected Papers of the 17th Computational Linguistics in the Netherlands Meeting , 99.

[4] D. Nadeau and S. Sekine, *A survey of named entity recognition and classification,* Lingvisticae Investigationes **30**, 3 (2007).

[5] J. Mesens, *Text mining algorithms part iii: Evaluating cv parsers,* (2016), accessed: 2017-05-01.

[6] D. Clegg and R. Barker, *Case method fast-track: A rad approach.* (2014).

[7] *ESCO - European Skills, Competences, Qualifications and Occupations,* https://ec.europa.eu/esco/portal/home, accessed: 2017-06-26.