UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

Data Labeling Platform

Design project report

Abdelrahman Eshy
Zaid Magzoub
Daan Dieperink
Noah Viste
Sietze Van Der Vinne

Supervisor:
prof. Nacir Bouali

Technical Computer Science Department
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

Data labeling allows AI and machine learning algorithms to build an accurate understanding of real-world environments and conditions, and the data labeling market expects to grow at a compound annual growth rate (CAGR) of 30% by 2027 to a massive US$5.5 billion in value”[1]. Data labeling is adding informative and meaningful labels to different types of data such as images, text files, and videos. This makes it machine compatible in a way that it can understand and analyze the information to give the results accordingly. In many cases, researchers and teachers (who might need data for their course projects) would like to work with data they have acquired or generated, but are rarely labeled or adequately labeled. In this project, a developed web application will get more widely materialized and developed where the teacher/researcher from the UT can initiate a data labeling task for non-staff members to interact with the platform. In this project, our group developed a web application where researchers within the University of Twente can crowdsource the process of labeling datasets for their academic research efforts. The web application has been developed using the Django web framework written in Python.

Contents

Chapter 1

Introduction

Labeling data for data-driven projects such as machine learning can be challenging. This project aims to develop a web application that serves as an online platform where dataset labeling can be crowdsourced by other members of the University of Twente. The web application should ease the process of labeling data as much as possible.

Staff members should be able to upload their unlabelled datasets to the platform, creating a new data labeling task. Afterward, other members of the University should be able to provide labels for the data from within the web application.

Finally, the creator of the task should be able to download their dataset, now populated with labels. The labeling process should get done through an efficient interface, so the time cost of labeling the data can be low.

1.1 Motivation

Machine learning has become an essential part of the world as we know it. By leveraging the enormous amounts of data that is available from various means of computer interaction, machine learning algorithms have taken over functions in many different domains. However, not all data is created equally. Many datasets, including those used by researchers at the University, require the addition of labels or other extra information before they can be properly used with statistical algorithms such as supervised learning. This process of labeling the previously unlabeled datasets can be tedious and was previously done by the researchers themselves, which could take up a lot of valuable time. Data labeling is often difficult to automate with machine learning algorithms themselves, since the training of these algorithms requires having access to large amounts of labeled data in the first place.

The Data Labeling Platform as discussed in this report aims to allow students and other members of the University of Twente to perform this task of data labeling, giving the researchers much easier access to the labeled data they need. Moreover, the platform can act as data storage for researchers, so they will be able to upload large datasets to the platform and share them with other researchers without the need of an external data sharing platform; the Data Labeling Platform contains everything necessary to share data with other researchers if that is desired. Optionally, researchers may choose to compensate students for their labeling efforts through the UT Flex payment system, incentivizing them to label large amounts of data.

1.2 Framework

The platform is designed and implemented using Django. Django is a free, open-source, Python-based web framework that follows a modular model–template–views architectural pattern.

In order for the platform to be compatible with the university of Twente it is implemented in accordance with the university login system. As the authentication framework is modular, the current Google implementation can be replaced with the Microsoft login system as needed.

The database is a critical element of the platform, designed and implemented using PostgreSQL in coordination with Django. This platform framework is secured and follows standard privacy practices. Hence, the data cannot be viewed or accessed by another user. Another user can only access the data if the user sets it to public or shares it.

1.3 Goals of the platform

The goal of this platform is to make the labeling of data as easy and as fast as possible. Access to the platform should be by the university members only, considering the platform's privacy and confidentiality to deliver a safe and trusted platform. Additionally the platform should also act as a billboard for the teachers to find new potential labelers. Allowing students to apply to it with a motivation letter.

1.4 Functional overview

This platform is currently only compatible with the university of Twente, as such it requires a university account to access the platform. When a user logs in using their university credentials, they are assigned one of the two roles: the student or the teacher. The teacher has access to all student functionalities as well as all the required administrative functions.

Each user can see all the tasks they have been assigned and the tasks that are open for applications. The user can then send an application with a motivation letter to the teacher that owns that task. The teacher on the other hand can see all the applications and accept or reject them. When any user has a task, they can see all the datasets that have been uploaded by the teacher in charge, read any instructions shown, and start labeling.

The teachers are able to create tasks for datasets that needs to be labeled. Each tasks requires the teacher to fill in all the labels that can be used. All the datasets uploaded in that task will be labeled following those criteria.

The platform tracks the time spent on the platform as well as tracking the amount of uploaded data and the amount of labeled data, which is helpful for hours declaration.

Chapter 2

Domain Analysis

Domain analysis is the procedure of identifying and apprehending domain knowledge about the problem domain to make it reusable when creating new systems.

2.1 Introduction to the Domain

The system domain for this platform concerns the procedure of providing a web application for the university of Twente members for uploading data and labeling the data. This system is for helping students to upload much data as they require and can let other users download part of the uploaded data. Furthermore, the system provides the user to label the data or request to label other users' data.

2.2 General Knowledge of the Domain

The University of Twente has different educational majors and departments for each faculty. Each faculty has it is own domain of proficiency. So, that means that the system requirements might differ for each department. Hence, the system has to determine these differences to dispense to the whole university.

2.3 Client, Users, and interested Parties

The most notable stakeholders of the platform are staff members at the University of Twente that need access to labeled data for their academic endeavors, or wish to share datasets with other members of the University. In addition, students with an interest in helping research, or who wish to earn some money through a UT Flex contract set up by a researcher, are potentially interested in the platform. Some students may also need to have a dataset labeled for their own research conducted during their study. These students can request a maintainer to have their permissions on the platform elevated, so that they are able to create a new labeling task as well, and are able to request help from their peers with labeling the data.

2.4 Software Environment

Since the client had no strict requirements regarding the development environment of choice for the project, we were able to choose the framework that was deemed the most fitting for this project. We found the web development framework Django[1] the most fitting. Which uses the Python programming language[2]. The system is developed to run within a Linux environment. However, the application and its dependencies can be run through Docker containers, allowing it to be easily deployed on different machines, and even on different architectures such as Windows machines. The reasons behind these software choices, and more detailed explanations of the software architecture can be found in section 4.2.

2.5 Procedures of the current situation

The platform in the current situation starts with the user logging into the platform using the university login system. Then, the user can view his profile and upload data on the platform with the option of sharing this data. In addition, The user can label his uploaded data or apply to label other user data. Nonetheless, the non-student user has the feature of requesting other students to label their uploaded data. Applying for labeling other user data consists of sending motivational reasons for labeling, and this reason will be sending the requested user, and he makes the selection. Nevertheless, The users can edit and delete their uploaded data on the platform.

[1] https://www.djangoproject.com/
[2] https://www.python.org/

2.6 Commonalities of UT Software

The platform users also employ other software dispensed at the university of Twente. Hence, the UT sign-in credentials and utilizing a similar design in the platform. These commonalities should make the platform familiar to users.

2.7 Conclusions

Indicating the methodologies and necessity of the supportive system is stated in analyzing the domain for the developed system. Likewise, the development contains the identification of the entire stakeholders' roles.

Chapter 3

Methodology and Requirement Specification

This chapter covers the development strategy, as well as the functional and non-functional requirements that identified for the platform.

3.1 Agile Project Management

Project development is a complex system that requires forethought in order to have a smooth development process. This is especially important when it involves cooperating with a client. The Agile methodology is one of the available solutions which aims to pave the way for a successful project deployment. As Agile follows a strict development cycle, it has limitations with regards to the the team dynamics. Therefore this project follows a variation of the Agile methodology. The core tenets of Agile are still present, however the development cycle is replaced with softer sprint deadlines instead.

Despite the changes made to the Agile development, it was chosen due to the importance of the client-centric approach. Ensuring that the final product is in line with their vision and expectations. It is also beneficial in ensuring that the development does not stagnate completely as the client meetings prompts incremental improvements.

3.2 Requirement prioritization

The formulated requirements are prioritized based on the importance of the functionality and its impact on stakeholders. The prioritization is based on the client's declaration, which is essential to the platform as well as the feasibility of the implementation.

3.3 Functional requirements

- As a researcher, I want to have access to the labeled data
- As a researcher, I want to be able to create multiple labeling tasks
- As a researcher, I want multiple labelers to be able to work on a task at the same time
- As a researcher, I want to be able to upload unlabeled data
- As a researcher, I want to be able to download a labeled version of previously uploaded data
- As a researcher, I want to be able to upload labeled data to be used as an example
- As a researcher, I want to be able to set the labeling task as either invite-only or public
- As a researcher, I want to be able to advertise a labeling task so that labelers can apply for it with a motivational letter
- As an admin, I want the website to track the amount of time spent and the number of data points labeled by each user so the labeler can be correctly remunerated
- As an admin, I want the labelers to be able to label data and be unable to create new labeling tasks
- As a labeler, I want the data labeling to have an efficient workflow so that I can label large amounts of data easily
- As a labeler, I want to be able to apply for a labeling task.

3.4 Non-Functional Formulation

- As a user, I want to be able to log in using the university system so that I don't need to create an account

- As an international user, I want the website to support English
- As a researcher, I want labeled text data provided in CSV format so that I can parse the results

3.5 Conclusions

Identifying the platform requirements is part of system engineering, and these techniques are part of it. These techniques help create the expected platform.

Chapter 4

GLOBAL AND ARCHITECTURAL DESIGN

This chapter includes the global and architectural design preferences and provides a general overview of the system's structure.

4.1 Revised Work Process

The platform serves to centralize and improve upon the work procedures of University researchers regarding data labeling. Figure 4.1 shows the revised work processes for staff members as well as labelers.

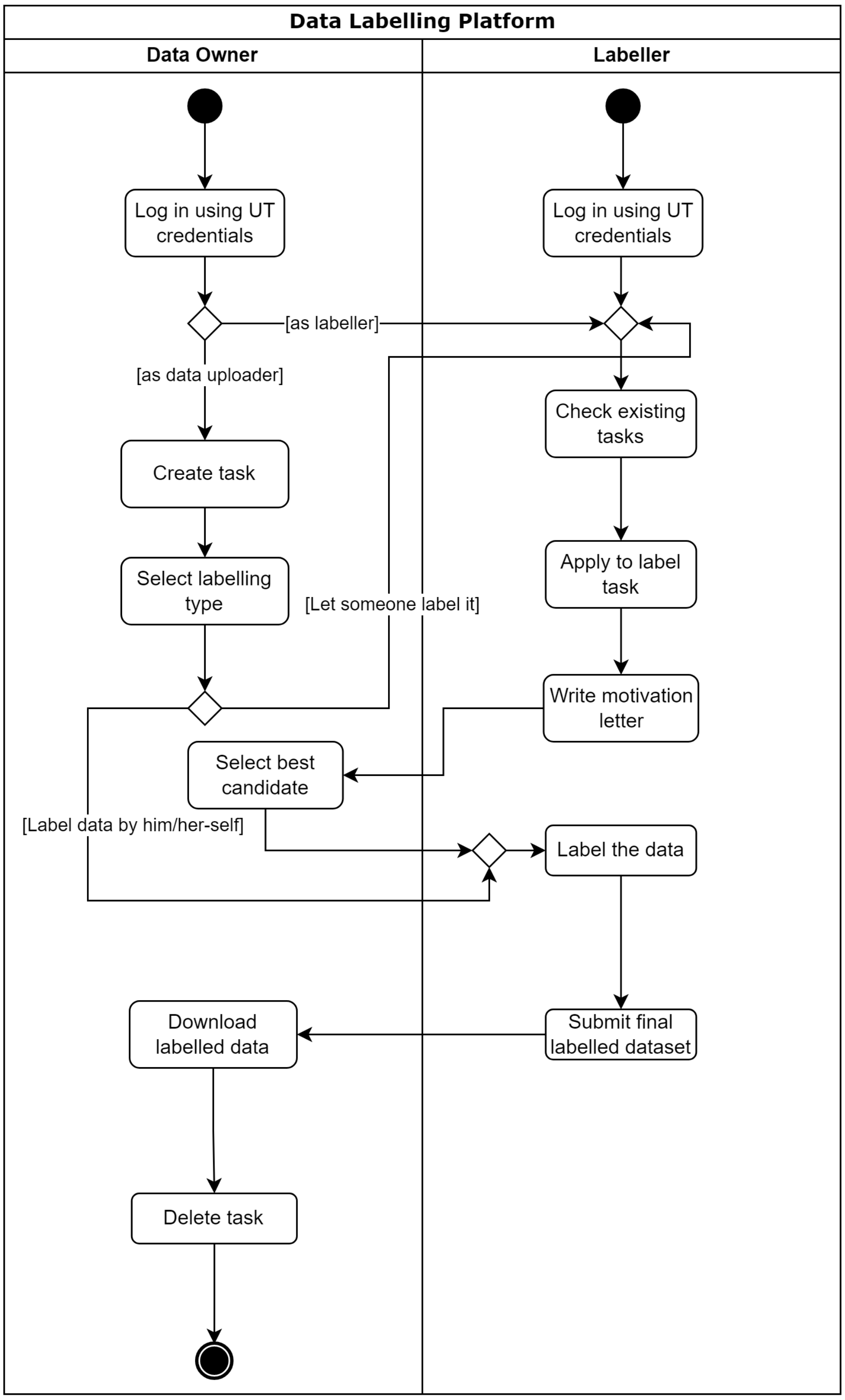


Figure 4.1: The procedure of labeling a dataset using the Data Labeling Platform

4.2 Preliminary Design Choices

Well-advised preliminary design choices are crucial for the project's start. The expertise and skills of the project team are some factors to consider in these choices, with regards to the selected programming languages and frameworks. The following sections will elaborate on the selection of these choices.

4.2.1 Programming Language

This project's primary programming language is Python. Python seemed the most suited for the project since every team member had experience with it, and the language is generally designed to produce understandable code within reasonable timeframes. Additionally, HTML, CSS, and Javascript are the front-end programming languages used for this platform's implementation, as they are required for any standard web application.

4.2.2 Frameworks and Libraries

The web framework chosen for this platform's development is Django. Django is a high-level framework that is well-documented and highly scalable. Due to the many available features and extension libraries, Django facilitates rapid development and practical design. Besides, Django provides various security features by default, such as measures preventing SQL injection, cross-site request forgery and cross-site scripting attacks.

The library Django-Allauth[1] has been chosen to provide the authentication mechanism for authenticating University of Twente members. This library can connect to the Google OAuth system to log users in. Since Google OAuth allows for the restriction of logins to members of a specific organization, the login procedure can easily be restricted to University members only.

[1]https://github.com/pennersr/django-allauth

4.2.3 Architectural Design Choices

System components

Although a Django application can be run as a single, standalone process, adding some other programs to the deployment stack can provide some benefits.

For the database, Django uses an SQLite implementation by default, but we configured it to use PostgreSQL as a backend instead, because it is better at performing complex queries and is more reliable.

The application contains some heavy-duty code related to the parsing of dataset uploads and exporting labeled data. With a standard Django setup, this code cannot run asynchronously and will have to be executed within the request-response cycle of the server. As a result, someone who just uploaded a dataset would have to wait for the entire dataset to be processed while their browser is still waiting for a server response. For large datasets, this could make the users wait for long periods of time while blocking the busy server process from responding at all. Not only would this be annoying, it would also allow users to (accidentally) denial-of-service attack the application by uploading a couple huge datasets. To mitigate this issue, we use the Python task queue celery[2] for the processing of datasets. Using celery, the expensive computations can be delegated to a different process, which frees up valuable resources to the application. If the task queue would ever fill up due to people uploading many gigantic datasets, more celery processes can be created to be able to process multiple tasks concurrently. Celery requires the use of a message broker, and for this purpose we deploy Redis[3], which can also conveniently be used by Django as a cache backend.

Django, PostgreSQL, celery and Redis all have to run separately to serve the application. To ease the deployment of all these components, we use Docker Compose[4] to run every part in its own container and to allow starting and stopping all containers at the same time, while working across different host platforms. A diagram of this architecture has been provided at figure 4.2.

[2]https://github.com/celery/celery
[3]https://redis.io/
[4]https://docs.docker.com/compose/

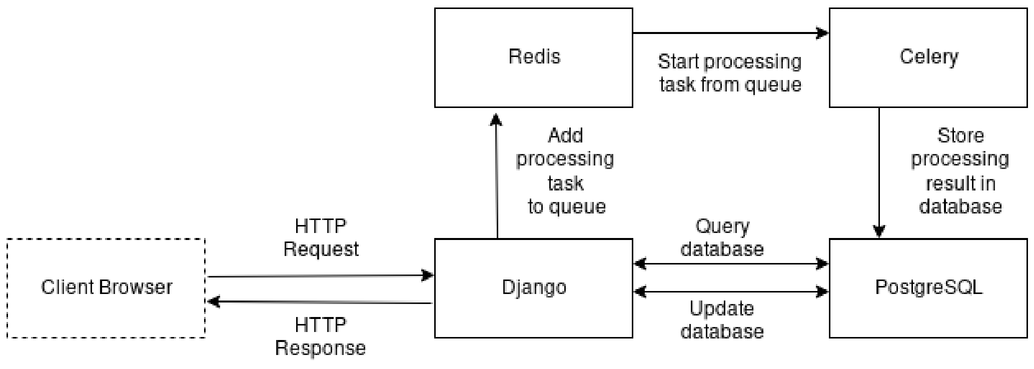

Redis
Start processing task from queue
Celery
Add processing task to queue
Store processing result in database
HTTP Request
Query database
Client Browser
Django
PostgreSQL
HTTP Response
Update database

Figure 4.2: Overview of the system components and their interactions

Code Structure

The Django framework makes use of the Model-View-Template paradigm, which is a variation on the common Model-View-Controller design pattern. The result is that Django programs are naturally split up into multiple Python files, each with a distinct and clear-cut purpose. Because all Django programs tend to be structured in the same way, programmers familiar with the framework will be able to understand the code structure of the project.

The 'models.py' file defines all necessary classes that will be converted to database tables by the Django object-relational mapper[5], and are used to store data required for the functioning of the application. The 'views.py' file contains the logic that is executed whenever an HTTP request if sent by a user and handles permissions checks as well as database updates. Finally, the view function uses one of the templates stored in the application's 'templates' directory to render an appropriate HTML response. These templates contain HTML as well as the Django template language to enable dynamic displaying of page content. There is also the 'urls.py' file, which determines which view is called based on the url in the client's HTTP request. The 'forms.py' file contains classes that are used to autogenerate HTML forms, such as the form used for creating new tasks.

All the aforementioned python files and their functions are used by any Django application, and so this structure is not unique to the Data Labeling Platform. However, the project uses celery (see the previous subsection) for asynchronous task execution, and all code to be executed by celery is located in 'tasks.py'. This setup is normal for Django projects

[5]https://docs.djangoproject.com/en/4.1/topics/db/models/

that use celery.

Although Django provides a relatively straightforward way to structure every project, there are still some choices to be made regarding the writing of views. Namely, views can be written as Python classes or as regular Python functions (also called generic views)[6]. For this project we chose to use a mixture of class-based and function-based views depending on the complexity of the views in question. Ultimately, this resulted in the use of class-based views for every page except for the task creation page (see section 5.2.2) and the labeling page (section 5.2.6). Since these views required a large amount of unique functionality, programming them using generic views makes them more simple to understand.

[6]https://docs.djangoproject.com/en/4.1/topics/class-based-views/intro/

Chapter 5

DETAILED DESIGN

This section explains the provided technicalities of the platform and its descriptions. Likewise, the design options are identified and presented on a more low level, including their justification.

5.1 System Description

5.1.1 Database Scheme

Django generates database tables based on Model classes defined in the application. Figure 5.1 contains a diagram of the Django models, as generated by the Django plugin django-extensions[1]. It uses the Django field names in lieu of SQL syntax, since the fields are automatically translated into SQL columns by the Django object-relational mapper.

5.1.2 Dataset Processing

The imported and exported datasets need to be processed in the back-end of the web-application. The imported dataset needs to be a CSV file containing the text that needs to be labelled. The exported file is also in CSV format which contains a labelled sentence per row. The datasets can be quite large, this is why the processing of these datasets takes place in the background of the application. This is done with the help of Celery [2] processes. Because of this the server is able to maintain fast response times.
The Celery process that takes care of the imported datasets cuts the whole text into sentences. Subsequently, the sentences are cut up in

[1]https://django-extensions.readthedocs.io/en/latest/graph_models.html
[2]https://docs.celeryq.dev/en/stable/index.html

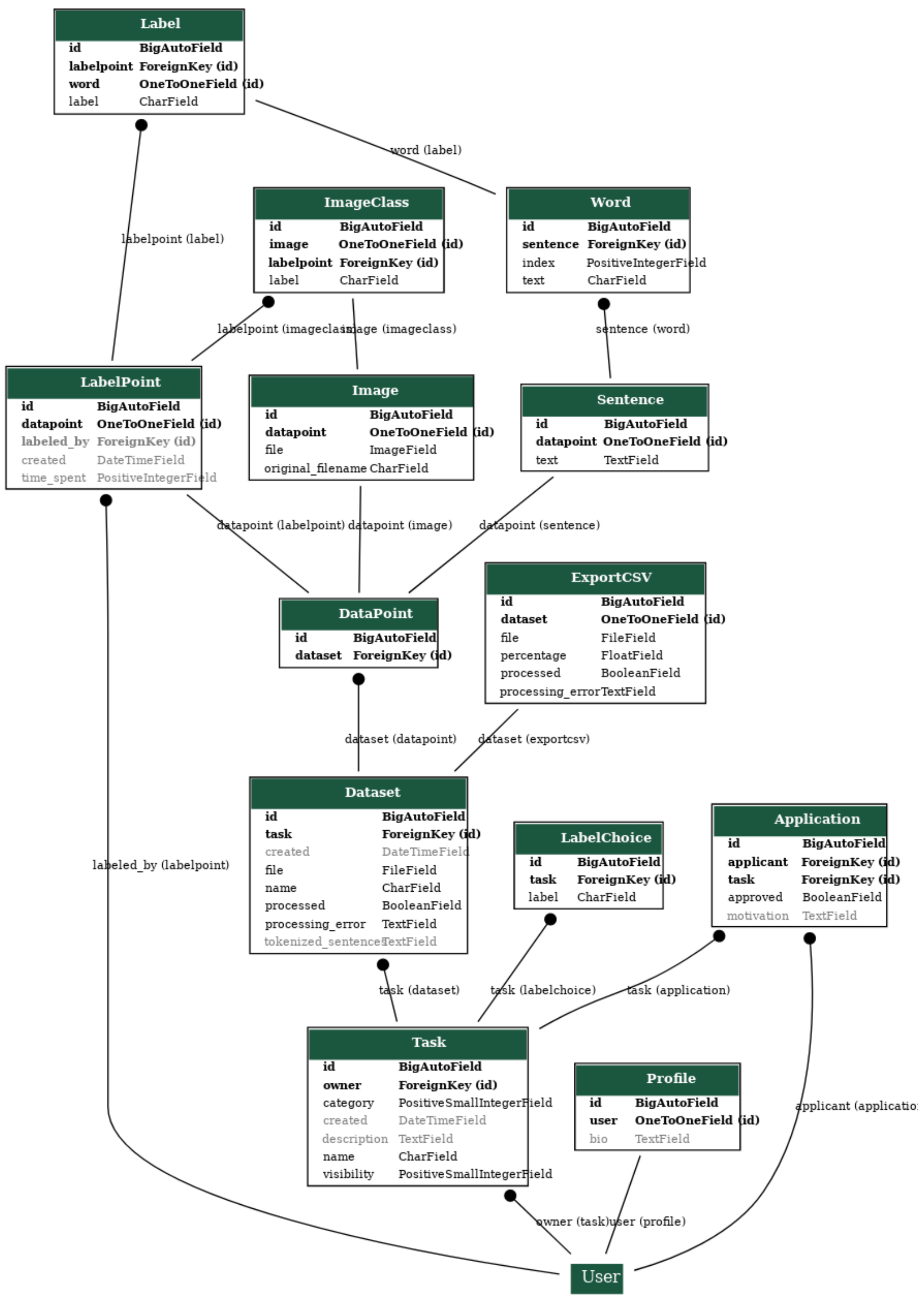


Figure 5.1: Schematic showing the various Django models created for the application, their attributes and relations between them. The 'User' model refers to the default Django user model.

words and punctuation. A sentence is stored in the database as a ”Datapoint” belonging to a dataset. Words and punctuation are stored as ”Words” belonging to a datapoint. After this the dataset is ready to be labelled.
After labelling is done or partially done the labelled datapoints can be exported to a CSV file. This is also done in a separate process. From the selected dataset all labelled datapoints are selected and written into a CSV file with one labelled sentence per row. When the process is done the CSV file can be downloaded.

5.1.3 Authorization Checks

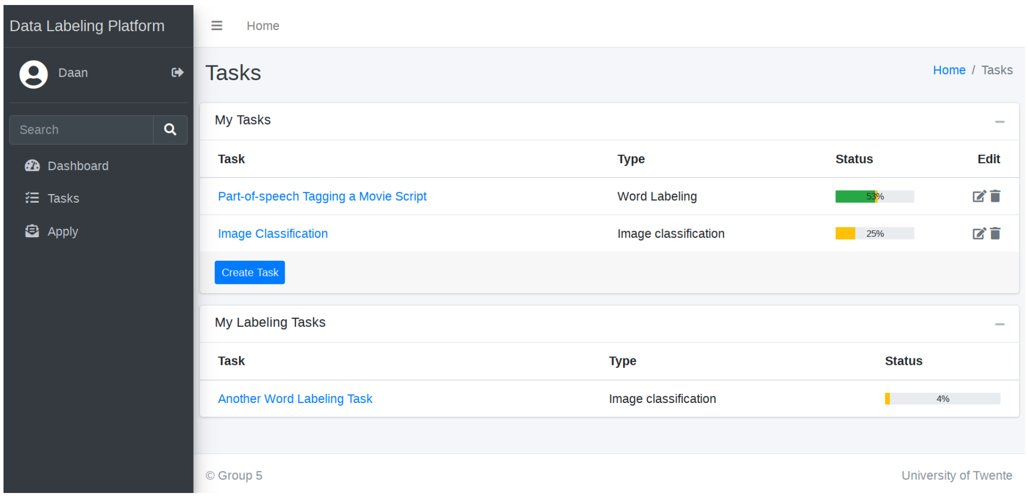
To ensure that users can only access the part of the application that they are supposed to, we added programmatic checks to every page that block out unauthorized users.

We did this by adding test functions to every Django view in our application, which allow us to determine which users are allowed to view the page. For example, users that visit a dataset overview page (see section 5.2.4) will need to have ’labeling access’ to the task to which the dataset belongs. Having labeling access means that either they are the task owner, the task owner has approved the user’s application, the owner has manually invited the user, or the task is set to allow anyone to label. Some other pages, such as the task editing page, are accessible only to the task owner, and creating new tasks is only possible for university staff members.

5.2 Front-end components

The users are not restricted in the number of datasets and tasks they can create. Therefore the front-end is designed with dashboards in mind. The design contains many redundant elements, making it possible to always show the relevant information so the user does not have to go back and forth between the pages. Additionally, the website is designed such that the colors red, green, and yellow are only to be used for showing progress or for the edit/delete buttons. By isolating those colors, it becomes trivial for the user to efficiently gauge the overall progress.

5.2.1 Task list



Tasks

My Tasks

Task	Type	Status	Edit
Part-of-speech Tagging a Movie Script	Word Labeling	53%	
Image Classification	Image classification	25%	

My Labeling Tasks

Task	Type	Status
Another Word Labeling Task	Image classification	4%

Figure 5.2: List of tasks

The tasks are divided into two categories that are both displayed to the user on the task list (see figure 5.2). First and foremost are the tasks the user submitted for labeling, these are shown at the top so that the user has easy access to the tasks that they prioritize. Each task has a progress bar that adapts to the progress of the individual datasets in order to provide a simple yet detailed overview of the progress. The users can thus at a quick glance estimate how the tasks are progressing.

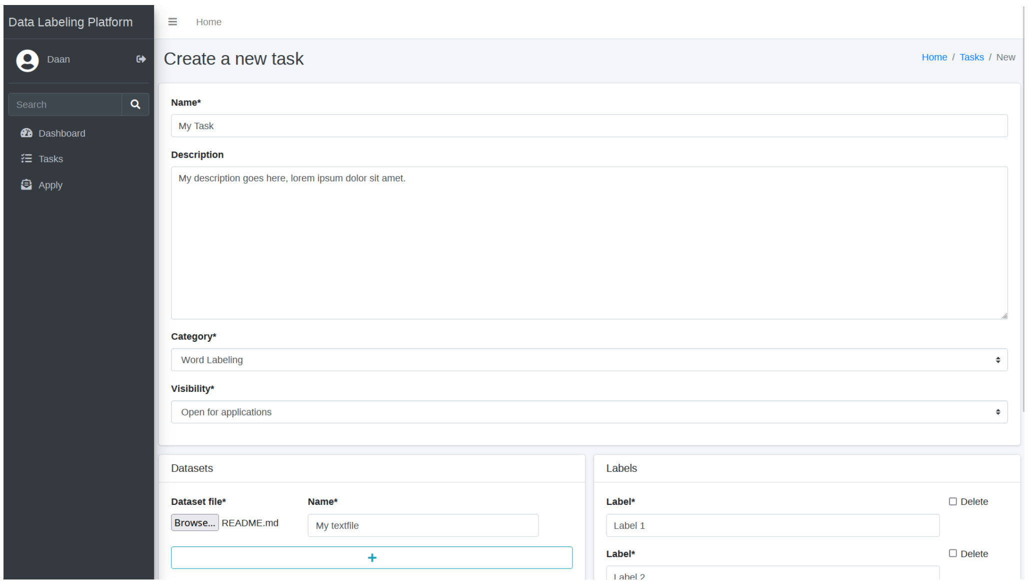
Secondly are the tasks the user is allowed to label. It is displayed at the top for users that are not allowed to create labeling tasks. This section follows the same design decision as above by prioritizing easy access.

5.2.2 Task creation page

The task creation page (see figure 5.3) serves a dual purpose as both the page to create new tasks and also to edit existing tasks. The design is kept simple and straightforward by using basic fields for the user to fill in. The task creation is heavily involved with the back-end, therefore the back-end is responsible for generating the forms.

5.2.3 Task overview

The task overview page (see figure 5.4) follows the dashboard structure. It shows basic information about the task. Details about the datasets is



Create a new task

Name*
My Task

Description
My description goes here, lorem ipsum dolor sit amet.

Category*
Word Labeling

Visibility*
Open for applications

Datasets

Dataset file*
Browse... README.md

Name*
My textfile

Labels

Label*
Label 1
☐ Delete

Label*
Label 2
☐ Delete

Figure 5.3: Task creation page

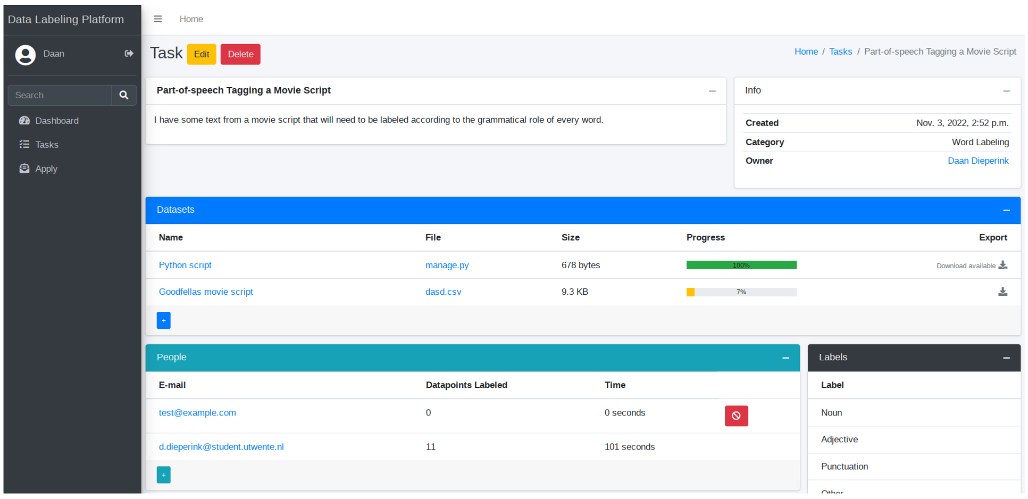
one of the important highlighted sections, containing easy to access links to download both the original and export the labeled file. Thus in the case where there are many datasets it is not needed to open each one individually. The other important section allows the owner to see how much time each labeler has spent, and how many datapoints they have labeled.

5.2.4 Dataset overview

The dataset overview page (see figure 5.5 contains similar information to the task overview page. It provides data that is only relevant to that dataset, such as the processing status and the time spent by users labeling.

5.2.5 Datapoint list

See figure 5.6. The datasets can contain a vast number of datapoints, thus it is important to display it in a condensed yet readable way. Datapoints that have been labeled are shown in green and can be expanded to reveal the associated labels. There is a link that allows to quickly edit the datapoint in the case of a mislabel. The unlabeled datapoints instead of expanding when clicked instead immediately link to the labeling page for that datapoint. It is designed such that every datapoint can be accessed within two mouse clicks.



Part-of-speech Tagging a Movie Script

I have some text from a movie script that will need to be labeled according to the grammatical role of every word.

Info

Created	Nov. 3, 2022, 2:52 p.m.
Category	Word Labeling
Owner	Daan Dieperink

Datasets

Name	File	Size	Progress	Export
Python script	manage.py	678 bytes	100%	Download available
Goodfellas movie script	dasd.csv	9.3 KB	7%	

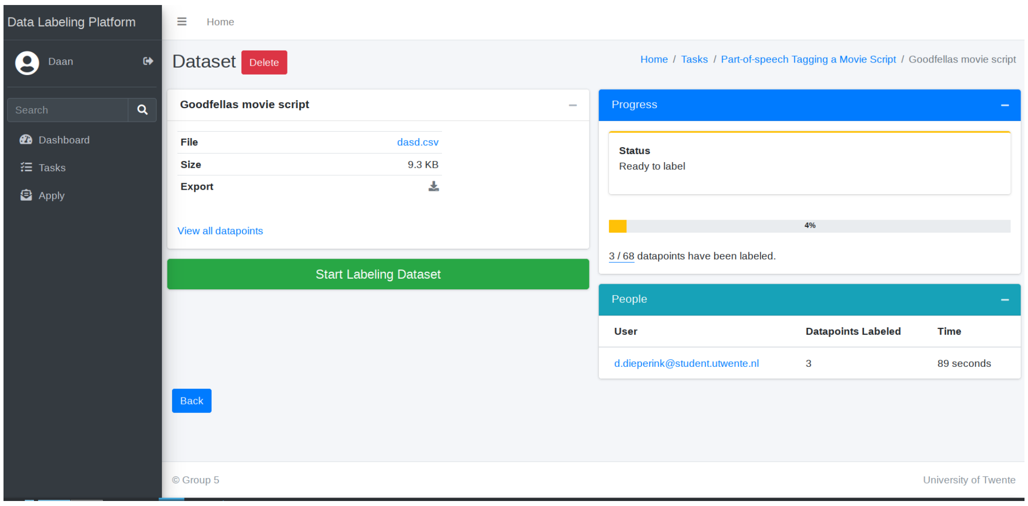
People

E-mail	Datapoints Labeled	Time
test@example.com	0	0 seconds
d.dieperink@student.utwente.nl	11	101 seconds

Labels

Label
Noun
Adjective
Punctuation

Figure 5.4: Task overview page



Goodfellas movie script

File	dasd.csv
Size	9.3 KB
Export	

Progress

Status
Ready to label

4%

3 / 68 datapoints have been labeled.

People

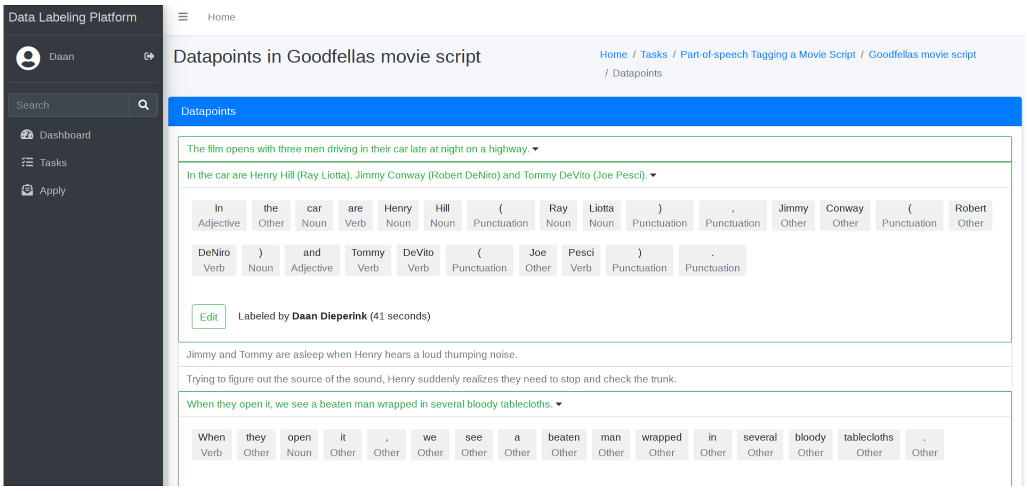
User	Datapoints Labeled	Time
d.dieperink@student.utwente.nl	3	89 seconds

Figure 5.5: Dataset overview page

5.2.6 Labeling page

The word labeling page (see figure 5.7) is the heart of the platform, therefore it is necessary for it to be both intuitive and efficient. Custom javascript is used to allow for dynamic selection using the keyboard. In order to correctly interact with the back-end, the javascript also ensures that the generated form is filled in according to the restrictions.

The user can use the keyboard to change the currently selected section of text using the arrow keys or tab. Alternatively the user could also use the mouse and click on the text. For choosing which label to assign the user can press the first letter of the label to quickly select it, use the arrow keys or use the mouse to select the label from the list. By having multiple ways of labeling it thus supports multiple workflows. For instance


Data Labeling Platform

Daan

Search

Dashboard

Tasks

Apply

Home

Datapoints in Goodfellas movie script

Home / Tasks / Part-of-speech Tagging a Movie Script / Goodfellas movie script / Datapoints

Datapoints

The film opens with three men driving in their car late at night on a highway.

In the car are Henry Hill (Ray Liotta), Jimmy Conway (Robert DeNiro) and Tommy DeVito (Joe Pesci).

In	the	car	are	Henry	Hill	(	Ray	Liotta	)	,	Jimmy	Conway	(	Robert
Adjective	Other	Noun	Verb	Noun	Noun	Punctuation	Noun	Noun	Punctuation	Punctuation	Other	Other	Punctuation	Other

DeNiro	)	and	Tommy	DeVito	(	Joe	Pesci	)	.
Verb	Noun	Adjective	Verb	Verb	Punctuation	Other	Verb	Punctuation	Punctuation

Edit Labeled by **Daan Dieperink** (41 seconds)

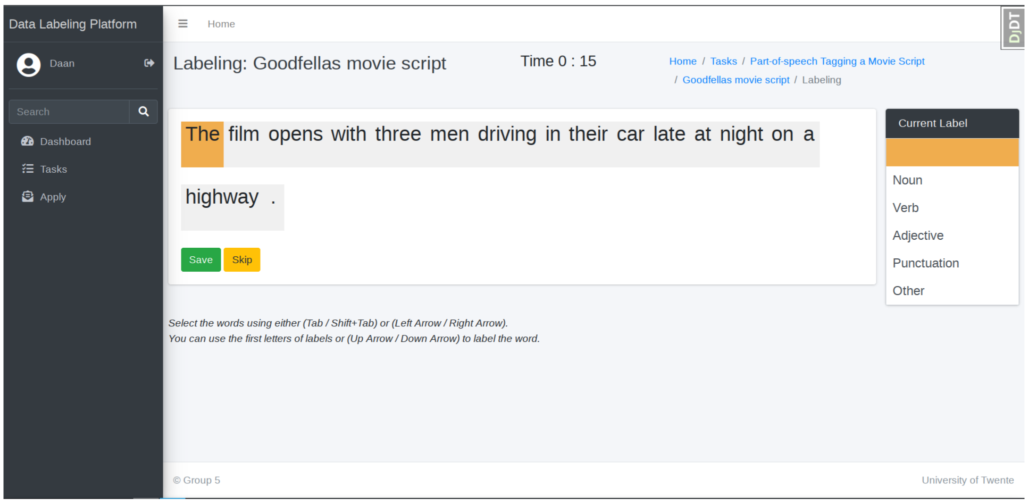
Jimmy and Tommy are asleep when Henry hears a loud thumping noise.

Trying to figure out the source of the sound, Henry suddenly realizes they need to stop and check the trunk.

When they open it, we see a beaten man wrapped in several bloody tablecloths.

When	they	open	it	,	we	see	a	beaten	man	wrapped	in	several	bloody	tablecloths	.
Verb	Other	Noun	Other	Other	Other	Other	Other	Other	Other	Other	Other	Other	Other	Other	Other

Figure 5.6: Datapoint list


Data Labeling Platform

Daan

Search

Dashboard

Tasks

Apply

Home

Labeling: Goodfellas movie script

Time 0 : 15

Home / Tasks / Part-of-speech Tagging a Movie Script / Goodfellas movie script / Labeling

The film opens with three men driving in their car late at night on a highway .

Save Skip

Current Label

Noun

Verb

Adjective

Punctuation

Other

Select the words using either (Tab / Shift+Tab) or (Left Arrow / Right Arrow).
You can use the first letters of labels or (Up Arrow / Down Arrow) to label the word.

© Group 5 University of Twente

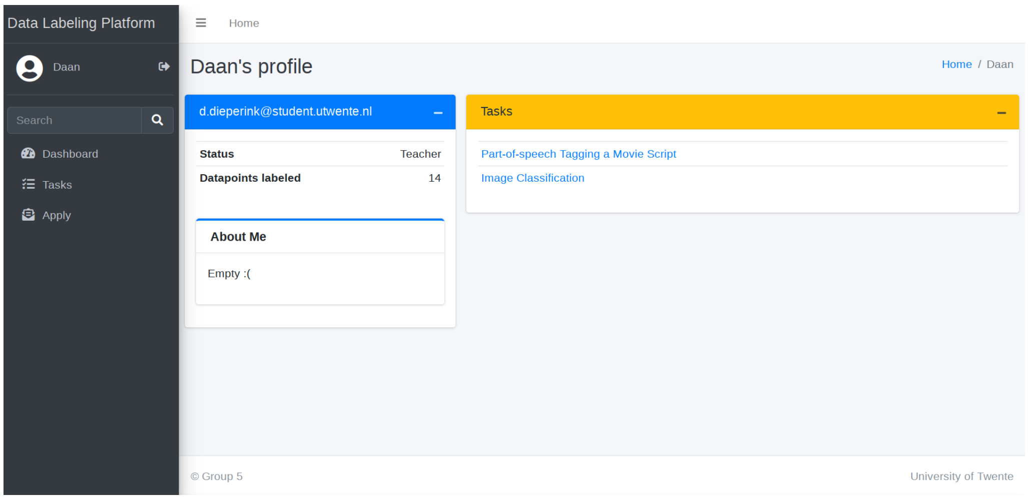
Figure 5.7: Word labeling page

by mixing the mouse and keyboard the user can use the arrows to move the selection and the mouse to click on the label. By enabling this efficiency it has the potential to save hours of work due to the possible scale of the datasets.

5.2.7 Profile page

The profile page (figure 5.8) simply shows the tasks associated with the user and the amount of total labeling done.

It is also designed to serve as a useful tool for evaluating prospective labelers, by being able to see the tasks they have been assigned



Daan's profile

d.dieperink@student.utwente.nl

Status	Teacher
Datapoints labeled	14

About Me

Empty :(

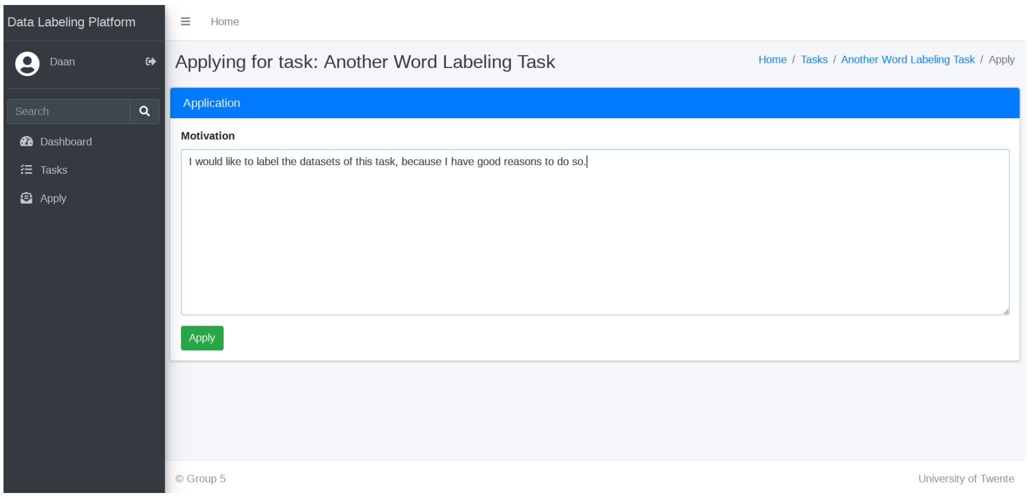
Tasks

Part-of-speech Tagging a Movie Script

Image Classification

Figure 5.8: User profile page

5.2.8 Task application page



Applying for task: Another Word Labeling Task

Application

Motivation

I would like to label the datasets of this task, because I have good reasons to do so.

Figure 5.9: Task application page

The task application page contains a list of all the tasks that the user can send an application to.

The application itself contains a single text input form for writing the application for a task. It is kept deliberately simple in order to reduce bloat. For anything complicated the platform encourages email by having the user contact information prominently visible.

Chapter 6

Testing the system

This chapter thoroughly examines the testing procedure and plan regarding the platform and provides the testing results. The testing chapter includes testing the heralded platform functionalities and explaining the testing approach. However, the risks and concerns got indicated besides the testing criteria.

The platform design and functionalities changed based on the testing criteria results, which were based on fixing the platform bugs and making them practical and reliable. The testing approach got divided into two sections. The first section is usability testing, and the second is integration testing.

6.1 Usability testing

The testing approach for the data labeling platform got based on user scenarios. Since university members use the platform, this testing got done by letting the university students use the platform, try every functionality, and go through every scenario. The testing scenarios got divided into two parts. The first part was written scenarios by the developers asking the users to go through some functionalities and perform some actions to see if they contained bugs.

The second testing scenario was letting the users try the platform as they wanted and go through every page and each functionality. However, through the usability testing, there were aspects to concentrate upon. Such as that, the system should always inform the user about its current situation via feedback. Moreover, the system clarifies the user's understanding of how to use and operate this platform. Lastly, ensure that the system design is aesthetic and contains helpful documentation for providing user help.

6.2 Integration testing

Integration testing is essential for providing a bug-free platform. Each platform's functionality can operate as expected; however, problems can occur when integrating these functionalities within the platform. To ensure a fully functional system, interaction testing is necessary to address bugs during the later phases of the platform development. The integration testing approach will be usability testing as the user goes through the entire platform and use every functionality.

Chapter 7

Future Planning

Although the platform's development as a part of the design project will cease at the end of the module, it was designed with extensibility in mind. This chapter discusses the future development that can get applied to the platform.

7.1 Use and Support of the System

The platform is designed in a web-hosting compatible way, the University of Twente can therefore host the platform without much difficulty. In order to have a seamless integration it would require some initial support and on-boarding. Over time, there would also need to be maintenance for bugs and other issues appearing during its operation. In order to fully support the university members, the platform needs to be scaled up to provide more efficiency.

7.2 University wide Enrolment

The development of the platform contained the vision of making the platform easy to scale and support all the university faculties. The administrators can add and remove facilities and members easily to the platform. Therefore, the platform is easy to deploy without much effort and in small steps. Administrators can add or remove the users to the platform by filling in their university ID as a username, then add the user can log in with their university credentials.

7.3 Text labeling extension

The current text labeling is based on words, associating one label with each word. It can be further improved to provide other tools such as labeling based on the highlighted part of the text. Other changes to text labeling tasks, such as the support for multiple labels per word, would also be possible.

7.4 Image labeling

The platform can be extended further to allow for tasks other than text labeling. With the modularity of the platform any type of labeling can potentially be added by providing a custom user-facing web interface. Since image classification is another important data labeling task, it has been partially integrated into the platform already. The csv export functionality has not yet been implemented for image labeling, and both the labeling page and datapoint list pages do not yet conform to the rest of the project's graphical design decisions. However, these functions could be implemented in the future.

The manual (see Appendix C) contains detailed instructions on how to extend the platform by implementing additional labeling tasks.

Chapter 8

Evaluation

This chapter discusses the overall evaluation of the project by elaborating on its planning, evaluating the project team, and the final result.

8.1 Planning

The project planning has been strict since the first day. The workload has been distributed throughout the entire module and divided into tasks per member for each week. The team decided to work on setting up weekly deadlines. The progress and the project's planning, including the deadlines, were monitored using Google Drive using a SCRUM board.

There was a weekly scheduled meeting with the client. This meeting is for discussing the progress and requesting client feedback regarding the platform and the platform design with the opportunity to add new requirements.

The team's weekly deadlines were set far before the client's deadlines. So the team can read and rewrite the documents to improve the deliverables' quality and fix all the platform bugs. Consequently, This planning ensured that the work was equally distributed and high-quality hand-in documents.

8.2 Responsibilities

The team members have different interests and expertise, dividing the project into tasks and responsibilities based on their knowledge and interests. These assigned responsibilities got primarily based on the team members' interests.

8.3 Team Evaluation

The project team contains the difference in the work attitude leading to arguments among the team at the beginning of the project. Therefore, the group agreed to have a team evaluation session every four weeks to discuss the differences in attitude and to reach an agreement with the entire team. These sessions solved and prevented these arguments, and it was sufficient.

8.4 Final Result

It ensures that the platform got accomplished by Customer Intimacy—the project resulted in an easy-to-use platform created by collaboration with the client in every platform aspect. However, the client feedback throughout the project process resulted in the creation of a very well-functioning platform that does not contain unnecessary functions because the client approves the entire operations and design of the platform.

8.5 Conclusion

At the beginning of the project, the team building activity got scheduled to aid team spirit. This activity had significantly valuable for the team to meet regularly. Moreover, the project planning ensured strict regarding the platform system, and deliverables were finalized and handed in on time.

The project's purpose was to deliver a university platform that emphasizes its usability and the ease of labeling data and as mentioned before. Besides, the project's goal is to provide a bug-free completed platform that the university can use without trouble.

As stated earlier, the project resulted in a working platform meeting the client and user requirements and expectations and is ready to be hosted by the university.

Appendix A

Source code

A.1 Link to the source code

https://gitlab.utwente.nl/s2014203/m11-design-project

Appendix B

Design Diary

This chapter is about the proposal of the platform regarding the client. The platform proposal got divided into phases in sequential order in this chapter.

B.1 Requirements Proposal

The start of the project was by meeting with the client. This meeting was to determine the goal and scope of the project. The client proposed his requirements for the project. However, the meeting went as a discussion for submitting better requirements and design. Therefore there was room for ideas. However, the client did not have requirements regarding the project's design. So, we proposed the design to be similar to the university platforms because it is a university platform that is easier to use. Nevertheless, The client had to express his preferences and use these requirements as a guide for the project.

B.2 Mock-ups Proposal

The project requirements was used to create a prototype for the client. They presented mock-ups to the client, and the client considered formulating the criteria for development. Moreover, the client proposed new requirements and designs for the project. Afterward, a meeting got held after offering the new provisions concerning the mock-ups and the layout. This meeting was to finalize the questions regarding the mock-ups. Despite that, the client had the opportunity to set more requirements and ideas for improvements regarding the final project.

B.3 System Introduction

The platform was presented to the client halfway through the development of the platform. The client got to see and observe the platform through diagrams and videos. Moreover, the client could interact with the platform on a laptop and experience every feature. Furthermore, the users previewed videos of the system ruining and going through every feature. The interaction with the platform allowed the client to experience his requirements and ask for more improvements and developments.

B.4 Usability Tests

The usability tests got performed with the clients and students from the university. The test scenarios got created so the user could operate on the platform normally and experience every feature. This scenario detected any bugs or parts of the platform that needed development. Therefore, this way of testing was valid and valuable for the daily usage of the platform. Nonetheless, the tester had the opportunity to test and comment on anything that stood out to help the team develop the platform.

B.5 Proposal Presentation

The platform got presented to the client and students of the university. The presentation showed the platform, and it is value and usability of the platform. It also demonstrated the resulting platform to the university members as it would be helpful and valuable.

B.6 Results of meetings

The meeting resulted in valuable and helpful feedback.

The mock-up meeting was crucial for the client to observe what the platform would resemble. The mock-up meeting enabled the client to provide more overall requirements and details regarding the project. Last, the client expressed his preferences regarding the mock-up and design for providing the asked platform.

The platform introduction proved very useful as the client experienced the platform functionality and went through the entire platform, giving him the impression of what the platform would be. However, it enabled the client to provide more specific feedback regarding the platform and more detailed comments on how to improve and make it more efficient.

DLP Manual

Administrator Manual

Running the application

1. Setting up Google OAuth

Follow the steps from the django-allauth documentation for registering an OAuth application. By registering using a UTwente account, the application can be set to 'Internal' under the 'OAuth Consent Screen' settings to restrict logins members of the UTwente organization. As documented by django-allauth, a django setting 'SOCIALACCOUNT_PROVIDERS' (in the ./data_labeling_platform/data_labeling_platform/settings.py file) will need to be adjusted.

2. Running the application using Docker Compose

The entire stack can be run using Docker Compose. The docker compose file at ./docker-compose.yml can be used to run all necessary containers. The application will listen on http://127.0.0.1:8000/ by default. This will run all required components for the application, but database migrations will still need to be applied before the application can be used.

3. Applying database migrations

Django migrations will need to be generated and applied before Django can properly use the PostgreSQL database that is used as part of the docker compose stack. This will need to be done by executing shell commands from within the 'django' docker container. The Docker documentation can be consulted for instructions on opening a bash shell within the container. Once the shell is accessible, the command 'python3 manage.py makemigrations dlp && python3 manage.py migrate' will generate the required database migrations and apply them immediately. These steps create the database tables used to store application data, and should be repeated after changes are made that alter the database scheme required by django.

Deploying the application in a production environment

Some steps should be taken before the application is suited to run in production. You may want to consult the documentation on Django settings.
1. The DEBUG setting must be set to False, and ALLOWED_HOSTS must be configured appropriately.
2. The SOCIALACCOUNT_PROVIDERS must be setup correctly (see Setting up Google OAuth above)
3. For security, the volume containing the application code should be mounted read-only in the docker-compose configuration
4. A django SECRET_KEY value and Postgres database credentials should be generated and kept secret. These can be passed to the application using environment variables. Using an 'env-file' and restricting its permissions could be done here.

5. The application should be run using a production-ready server instead of the default test server. See the documentation on deploying django
6. The front-end server should be configured to use SSL/HTTPS and to redirect unsecure requests to HTTPS.

Word Labeling tasks: Import / Export format

To import a dataset you need to upload it in csv format. Inside the csv file should be the text you want to label. The web application will tokenize the text using NLTK [1] and store it in the database.

To export go to tasks and select your task, then click the export button for the dataset you want to export. The exported file will be a csv file with labeled text in there. One row corresponds to one sentence and every word or punctuation will be followed by its label. Example:

I like food. So, I eat food.

Will be labeled as:

(‘I’, ‘Label1);(‘like’, ‘Label2’);(‘food’, ’Label1’);(‘.’, ‘Label3’)
(‘So’, ‘Label1);(‘,’, Label3);(‘eat’, ’Label1’);(‘food’, ‘Label1’);(‘.’,’Label3’)

With Label1, Label2 and Label3 being random labels.

Maintenance using the administrative interface

An administrator account can optionally be created to allow maintainers full access to change application objects using a web interface. By executing the command 'python3 manage.py createsuperuser' inside the 'django' docker container, a special account can be created that can login on http://127.0.0.1:8000/admin/ for maintenance purposes. Through the admin interface, a superuser account can manually create, edit or delete all application-related objects without having to query the postgresql database itself.
This can be useful, for example, to manually restrict certain accounts from logging in (banning them from the service) or explicitly granting 'staff' permissions to a university student, allowing them to create their own tasks and upload datasets.

Extending the platform

The data labeling platform can be extended and changed to fit the university’s needs. This process requires some knowledge of the Django framework. Fortunately, Django provides extensive documentation and Django projects are often structured in a similar way due to the Model-View-Template architecture employed by the framework. In the following sections, we will provide some exemplary ways of extending the application.

Editing templates

The most straightforward way to change the look of the application's pages is to edit the html templates. For example, the dashboard page can be changed by editing the file 'data_labeling_platform/templates/dlp/index.html' in the project's source code. The template files contain html that is displayed in the front-end but use the Django Template Language to support programmatic changes to the displayed information.

Adding additional labeling tasks

The platform can be extended to allow for many more labeling tasks. For every labeling task that needs to be added, a number of steps should be taken to fully integrate it into the platform.

1. **Create a new task category**

In the file 'data_labeling_platform/dlp/models.py', within the enumeration class Task.Category, a new attribute should be added. The identifier will be used to refer to the category within the code, and its value should be a tuple containing the enumeration integer as well as the human-readable string. For demonstration, the category 'EXAMPLE_TASK' is added below:

File: 'data_labeling_platform/dlp/models.py'

```
[...]
class Task(models.Model):
      """Data labeling task as created by a staff member. Can have multiple
datasets"""
      [...]
      class Category(models.IntegerChoices):
            """Possible choices for the task category"""
            WORD_LABELING = 0, 'Word Labeling'
            IMAGE_CLASSIFICATION = 1, 'Image classification'
            EXAMPLE_TASK = 2, 'Example task'
[...]
```

2. **Create database models for the new labeling task**

In the models.py file, some new model classes should be added so information specific to that labeling task can be saved to the database. This can be as many models as desired, but should have at least a model for saving (unlabeled) data contained in a user-uploaded dataset, and at least one other model for saving labels or other information that is supplied by the labelers on the platform.

3. **Add a function for processing dataset uploads**

In the file ‘data_labeling_platform/dlp/tasks.py’, define a new function that takes as its input parameter a dataset object, and call this function from the *process_dataset* function as such:

```
@shared_task(bind=True)
def process_dataset(self, pk):
      [...]
      if dataset.task.category == Task.Category.WORD_LABELING:
            process_word_labeling_dataset(dataset)
      elif dataset.task.category == Task.Category.IMAGE_CLASSIFICATION:
            process_dataset_image_classiciation(dataset)
      elif dataset.task.category == Task.Category.EXAMPLE_TASK:
            process_my_example_import_function(dataset)
      else:
      [...]
```

4. **Create a template for displaying the datapoint list of the newly created task**

In the template ‘data_labeling_platform/templates/dlp/datapoint_list.html’, provide html for the datapoint list that is used for this labeling task. The django template language should be used for dynamic content.

5. **Create a template and view for the labeling page**

In ‘data_labeling_platrofm/dlp/views.py’, expand the function *labeling_view* for the newly created task category, by calling a new view that takes care of displaying the labeling form as well as saving the labeled datapoints:

```
[...]
def labeling_view(request, pk: int) -> HttpResponse:
      [...]
      if datapoint.dataset.task.category == Task.Category.WORD_LABELING:
            return word_labeling_view(request, datapoint)
      elif datapoint.dataset.task.category == \
Task.Category.IMAGE_CLASSIFICATION:
            return image_classification_view(request, datapoint)
      elif datapoint.dataset.task.category == Task.Category.EXAMPLE_TASK:
            return my_custom_example_labeling_view(request, datapoint)
      [...]
```

6. **Write a function for exporting the labeled data**

Back in ‘data_labeling_platform/dlp/tasks.py’, expand the *process_csv_export* function to account for the newly created task type. Add a check to the if/else statements like in the previous steps that calls a custom function which returns a ContentFile.

After having implemented the aforementioned steps, the new task is integrated into the application.

Integrating email

Django can connect to an SMTP server to send emails, using its builtin email module. After having set up an SMTP connection, the *send_mail* function can be used in views to notify users about important events. For example, an application that a user makes to apply for a task could be forwarded to the email of the task owner.

Changing the authentication backend

Currently, the application uses Google OAuth to authenticate users. However, the University can change the authentication method for another if they please. Since the application does not use any custom user model, the method of adding users can be swapped out for another. Since the university uses SAML authentication on most of its domains, this can also be achieved in the Data Labeling Platform by using a library such as django-saml2-auth. However, the SAML authentication system requires having a dedicated server setup with its own domain name, which is why we chose to stick to the Google authentication method during the initial development phase.

References

[1] Bird, S. (n.d.). *Natural Language Processing with Python. O'Reilly Media*.

https://www.nltk.org/book/