CSE2000 Software Project

Fixing e-books with Image Descriptions for Visually Impaired Users

Final Report: From Design to End Product

Technical Writing group 4C

Aratrika Das
Nadine Kuo
Vlad Makarov
Filip Nguyen Duc
Denis Tsvetkov

Client: Ted van der Togt
TU Coach: Dr. Christoph Lofi
Teaching Assistant: Alexandru Bobe

June 2022

Koninklijke Bibliotheek
National Library of the Netherlands

TU Delft
Delft University of Technology

Preface

This report was written by a group of five students studying Computer Science and Engineering at the Delft University of Technology (TU Delft). During our ten-week bachelor course Software Project (CSE2000) from April 2022 to June 2022, we have created a demonstrator for a web-based image annotation platform, thereby aiming to increase accessibility of e-books in the future. This system allows users to create image annotations for e-books, for which they can get automated suggestions to make that process more efficient. This project was an initiative started by the *Future Libraries Lab*, a collaboration between the *National Libaries of the Netherlands (Dutch: Koninklijke Bibliotheek or KB* and the *Web Information Systems (WIS) group* at TU Delft.

Our target audience includes our client Ted van der Togt, course coordinators, but also other readers who are interested in learning more about or project and/or image annotation for fixing e-books. We expect our audience to have general knowledge about software engineering. Readers who would like to learn more about the problem of inaccessibility of e-books will find this in chapter 2. Readers who are more interested in technical aspects behind our system can refer to chapters 4 and 6.

We had the privilege to work with incredibly supportive people. Firstly, we would like to thank our client Ted van der Togt for giving us the opportunity to work on this project. His enthusiasm throughout our project inspired us to make the most out of the final product. We highly enjoyed hearing about his past experiences in the field of accessibility of e-books during our weekly meetings. We would also like to thank his colleagues at *KB* - Marianne Hermans, Johan Van Der Knijff, Willem Jan Faber and Heleen van Manen - for their engagement in our project. Furthermore, we must not forget to thank our TU coach Dr. Christoph Lofi for his supervision of this project. His passion and expertise guided us well during the development phase of the project. Also, we want to thank our teaching assistant Alexandru Bobe, who was our first point of contact. Finally, we are very thankful for Gregorio Pellegrino and his colleagues at *Fondazione LIA* - Chiara DeMartin and Cristina Mussinelli - for showing interest in our project and providing us with feedback based on their expertise in the field of accessibility of digital publications.

Whereas we initially had little knowledge about accessibility of digital media, after ten weeks we can say that this project has definitely peaked our interest in this subject. Our hope is that this report will also inspire you to contribute to current efforts to make digital media - including e-books - accessible to all people, also those who are visually impaired.

Delft, June 2022
Aratrika Das
Nadine Kuo
Vlad Makarov
Filip Nguyen Duc
Denis Tsvetkov

Summary

Images have always been and will continue to be an effective method of conveying a great deal of information, by supporting or replacing a piece of text. However, people who are visually impaired are often denied the information images contain due to lack of meaningful image description. E-books have become a popular alternative to books in today's digital age and an important part of e-books is their visual content. For readers with a disability, e-books, when made accessible, become a way to access information and read stories that previously were not possible. However, only a very small percentage of e-books that are published each year are made accessible. This is due to the fact that making e-books accessible is costly, time consuming and not scalable as it is mainly performed manually. A major task involved in making e-books accessible is adding annotations to images. These annotations are often not detailed enough and do not communicate the necessary information depicted. This is posing a major problem as it hinders inclusion and equality in society.

The objective of this report is to document the development of a demonstrator for an interactive image annotation platform, which combines automated and manual image descriptions to make the process of image annotation efficient. By introducing automation, manual labour can be saved. However, as existing Artificial Intelligence (AI) tools are not accurate enough yet, one cannot solely rely on AI to create image descriptions. Therefore, our system uses human intervention to preserve quality of image descriptions. Similar tools that exist already are POET and a pilot project presented by Gregorio Pellegrino (Fondazione LIA) at the DPUB Summit in 2019. Our system presented in this report is unique in the sense that it not only lets image recognition engines generate suggestions by analysing the *image itself*, but it also incorporates automated text analysis tools that consider the *textual context* of the image.

After the system requirements were elicited, our design of the final product proposed was a cloud-based web service with a graphical user interface called "E-BOOK FIXER". The platform allows users to upload e-books, which they aim to remediate, in the form an of an EPUB file. This EPUB is then processed using tools such as EPUBCheck, checking for conformance against EPUB specifications. Valid EPUBs will be displayed in the editor, where users can navigate through their e-book and select images to annotate. Images are then to be classified by the user from a list of available types. In order to generate suggestions for image descriptions, the user can select an image recognition engine. So far, the platform supports Google Cloud Vision API and Microsoft Azure Computer Vision API. Additionally, the system will let the automatic text analyser Yake generate keywords extracted from the textual context surrounding an image, aiming to enhance the suggestions by Google or Microsoft. With the help of these different automated suggestions users can come up with their own manual annotation, which gets saved as ALT-text for that image. Finally, users will be able to download their updated EPUB file containing all the image annotations created.

The platform was implemented using React on the frontend and Django on the backend. A MongoDB database was used to store the metadata related to the e-books, their images and the annotations created. The actual EPUB files uploaded by users are stored in a separate file system on the server. In order to aid developers to manually inspect structural changes made to EPUB files by the system, each EPUB file is pushed to a remote GitHub repository upon upload and download. For deploying the application, tools such as Docker, Netlify and DigitalOcean have been used. To ensure the usability of the graphical user interface (GUI) of the platform, we have conducted a user evaluation on our prototype with various stakeholders and made improvements according to the results obtained.

Nevertheless, there is more to remediating inaccessible e-books than just image description. In the future, this platform could benefit from a more extensive data processing pipeline, that could for instance convert EPUB2 files to the newer EPUB3 format - serving as a better standard for accessibility. After this conversion step, the pipeline could add further accessibility features to the EPUB file, using automatic tools such as Access-Aide. Furthermore, the process of classifying images can be automated using tools such as Google's AutoML. Another extension would be to have separate workflows for images based on their classification, as different types of AI engines are usually suitable for different types of images. Currently, our system only distinguishes between decorative and non-decorative images in the sense that non-decorative images should not be annotated.

All in all, by having an interactive platform that introduces automated suggestions for image annotations - thereby taking into account the textual context of images - the manual process of annotating images is made more efficient for users. They no longer need to read the text surrounding the image, allowing them to focus on creating more descriptive annotations using the AI suggestions provided. As this system grows into a pilot in the future, we are convinced that it can help in the process of making e-books more accessible for those who suffer from vision impairments.

Contents

1 Introduction

Throughout the world, at least 2.2 billion people suffer from vision impairment[1]. Some of these people can have a hard time interpreting visual contents in order to acquire information. Meanwhile, in our modern society, visual contents embedded in digital media play a prominent role in our daily lives. Can you imagine playing video games without graphics, or reading a textbook without any illustrations? It is clear that visuals help us digest complex information quickly and enhance the experience of digital services. Our research specifically revolves around images in e-books, which can reinforce textual narration of a story by conveying emotions or opinions for example. In order to stimulate accessibility of e-books, images require meaningful annotations via alternative text that describe what is depicted. Producing these image descriptions is be time-consuming and laborious. Thus, to assist this manual process of image annotation, *automated* image descriptions can be used.

The objective of this report is to document the development of a demonstrator for an interactive image annotation platform, which combines automated and manual image descriptions to make the process of image annotation efficient. The resulting solution shall facilitate users in the process of creating image descriptions for e-books in a more efficient way. Automated suggestions for image annotations save manual labour, whereas humans can manually inspect these to ensure quality. The initial phase of our project is a problem analysis, which includes investigation of work by experts in automated image descriptions. Based on this problem analysis, an overview of system requirements will be produced. Afterwards, designs for our solution proposal are presented. These designs will then be used to implement our image annotation platform.

In the next chapter the initial problem analysis phase is described. The system requirements that this project will focus on are described in chapter 3. Chapter 4 will present our system designs and motivations behind choices made. Our strategy for planning our software development process is addressed in chapter 5. Chapter 6 elaborates on technical aspects with regards to the final implementation of the system, which is based on the solution proposal introduced earlier. Results of our prototype evaluation with different stakeholders are discussed in chapter 7. Chapter 8 lays out our final product discussion and future recommendations. Finally, the report will end with the key takeaways in our conclusion.

2 Problem Analysis

This chapter presents our results obtained after having carried out a problem analysis, prior to designing our image annotation system. Background information about the problem of inaccessibility of e-books can be found in section 2.1. Then, section 2.2 gives an overview of terminology related to our project; these will be referred to throughout the report. Our project goals are motivated in section 2.3. Finally, section 2.4 compares our system to two existing products that use automation to make image description efficient.

2.1 The Problem: Inaccessibility of e-books

E-books have become a popular alternative to paper books in today's digital age, mainly due to their flexible layout and portability. An important aspect of e-books is their visual contents, which help readers visualize and perceive information provided by the textual narration of a story. Images can for example enable and facilitate the analysis of complex concepts, but also enhance the experience of digital services. For readers with vision impairments, e-books become a way to access information and "read" stories that previously were not possible - by using audio, braille and flexible fonts for instance. Unfortunately, the majority of e-books have not yet been made accessible for people with print disabilities. Meanwhile, the European Accessibility Act (EAA) is striving to ensure that all new digital publications and services are accessible to everyone within the EU by July 2025. This includes e-books, but also e-commerce websites for instance [2]. Making an e-book accessible involves various steps, which all essentially come down to making structural changes to the so-called EPUB file corresponding to that e-book. A substantial part of making e-books accessible is conveying the information provided by images to the reader. Usually this is done by adding alternative text, or ALT-texts, to images. This description allows screen readers (using text-to-speech) or braille to translate the image. Hence, in this project we will be focusing on improving alternative descriptions for images, which is "sufficient for meeting the requirements of the EPUB Accessibility specification" as confirmed by the EPUB Accessibility 1.0 guidelines [1].

Annotating images is a process that is currently mostly being performed manually and thus leads to issues of cost, scalability, timeliness, and quality. These annotations are often not detailed or meaningful enough to effectively communicate the necessary information depicted, which could be due to the annotator's lack of understanding of the textual, cultural or even historical context of the image. Moreover, there are generally not sufficient resources to hire experts for fulfilling this task. This is posing a major problem as it hinders inclusion and equality in society. Even with the help of volunteers or paid workers through general crowdsourcing, the massive volumes of images that are produced daily still impose severe limits on time and scalability. The development of automated captioning systems has shown their potential to produce image descriptions efficiently on a larger scale[3]. However, these AI systems have their own limitations such as the deficiency of content, reliability problems, and possibly ethical issues. Therefore, human intervention is still required (also referred to as a "human-in-the-loop")[4]. This is why our client Ted van der Togt has requested us to build an interactive image annotation platform that provides *automatic* suggestions for image annotations, but can also accept *manual changes* suggested by users.

2.2 Terminology

In the process of analysing the problem to be solved by our image annotation system, we identified several crucial concepts in the scope of our project.

- **EPUB**: electronic publication file using the ".epub" extension is an e-book file format, which is so widely used now that it is supported by many electronic devices. Essentially, an EPUB file is a large ZIP archive containing all mixed-media e-book contents - such as HTML files containing text and images, CSS stylesheets, metadata about the e-book, tables of content and more.

  In this system, we deal with two types of EPUB files:

  1. **EPUB2**: an older format approved in 2007, which is the default type being uploaded by users of this annotation platform.
  2. **EPUB3**: a newer EPUB version approved in 2011, which serves as a better foundation for accessibility with an improved structure and metadata about accessibility and next to that added support for Fixed Layout, SVG graphics, embedded interactivity, audio and video.

- **ALT-text**: alternative text, also referred to as "alt attribute", which is used within HTML code to describe what is depicted in or represented by an image. An example HTML fragment would be:

[1] `https://www.w3.org/Submission/EPUB-a11y/`

```
<img src="images/fig3.jpg" alt="Beautiful dog">
```

The purpose of our project is to allow users of our system to make these image descriptions more meaningful than the one above, by providing them with automated suggestions.

- **Image description:** textual annotation that describes an image, which can be regarded as synonymous to ALT-text in the context of our project, since we are dealing with EPUB files that contain HTML code. However, image descriptions can take on other forms than merely an ALT-text.

  We distinguish between 3 types of image descriptions [5]:

  1. *Conceptual* descriptions: describe what is depicted in the image
  2. *Non-visual* descriptions: provide (contextual) background information that is not visible (who is depicted or where the image is taken e.g.)
  3. *Perceptual* descriptions: describe what the purpose or type of an image is (icon, decoration, infographic, comic etc.), thereby classifying the image into a certain category

  From recent studies it has become clear that in order to fully satisfy the needs of visually impaired people, automatic image description systems should focus on all three types of image descriptions [6].

- **Annotators:** Users of the image annotation platform, who will create image descriptions for e-books. These people can be authors or publishers of e-books themselves, volunteers or paid workers. [2]

2.3 Project Goals

The end product of this project is a prototype for a web-based platform that assists users in creating better image annotations, to enrich the experience of reading e-books for visually impaired users eventually. In order to realise this given the time frame of ten weeks, we have set the following three goals:

1. **Investigating existing solutions for image annotation using automation**
   In order to establish the need for a new solution, existing products need to be assessed. By investigating these solutions we can assess there shortfalls as well as their positive points. Section 2.4 presents the results of our research conducted on two such existing solutions.

2. **Designing an accessible platform to make image annotation more efficient**
   The next goal involves creating a design for our platform and a development process to convert that design into a usable product, which will make image description more efficient than current methods in that field. With regards to generating automated image descriptions, the system shall rely on existing services (APIs) as much as possible. Our objective is to incorporate two types of AI to assist users:

   (a) Image recognition systems that generate labels and/or descriptions serving as suggestions for *conceptual* image descriptions;

   (b) Text analysis systems that generate keywords serving as suggestions for *non-visual* image descriptions.

   By adding suggestions based on the textual context of an image, we aim to enhance the suggestions provided by image recognition systems - which are generally not meaningful enough due to lack of context. Moreover, this will allow the user to write meaningful annotations without having to read the text surrounding an image, thereby saving time. Besides the use of AI, our system introduces human intervention by allowing users to manually classify images - serving as *perceptual* description - and annotate these.

3. **Building the image annotation platform based on our design**
   Our final goal will be to implement our design of the image annotation platform based on the requirements of the client. To ensure that the platform is accessible, a prototype evaluation will be conducted on our GUI. It is important to note that the system being built is an experimental setup being used to test whether public domain e-books can be made more accessible using an approach that exceeds earlier solutions. As it will be an open-sourced project eventually, we aim to take into account future extension during implementation.

[2] We expect volunteers to be the largest group of future users of this system.

2.4 Research on Existing Products

Below we present several existing solutions for automating image description that share similarities with our system to be built. Undoubtedly, they come with differences/shortfalls as well, which we aim to complement in our project. Note that these solutions all have specific AI systems *incorporated* in them, which on their own are not solutions to the problem that we are trying to solve, as we want to build a *full workflow* that combines classification, AI suggestions and manual annotations. AI technologies that we have integrated in our system are elaborated on in 6.4.1.

2.4.1 Poet

Poet was developed by DIAGRAM Center in 2013, serving as image description tool. Their main aim was to crowdsource the creation of image descriptions in order to "reduce both the cost for content creators and delivery time for end-users". [3].

Similarities: Poet is also an open-source web application that generates automatic (*conceptual*) annotations by analyzing the image and allows the user to edit these in a graphical user interface. This platform shows the e-book and its contents: images and their textual surroundings. Once users are done annotating images, they can download the e-book again. Furthermore, POET comes with an "Image Coverage Checker", which can generate a report indicating which images already have descriptions - allowing the user to estimate how much time is required to annotate images in their e-book. We strive to implement a similar feature into our system [4], as we believe it greatly contributes to efficiency of image description.

Shortfalls: Poet does not have functionality implemented that deals with image classification (*perceptual* description). However, information about the purpose of an image can be useful to allow for service by classification, as is the case for the product described in 2.4.2. In the scope of our project, images classified as "decoration" will not be described - thereby saving unnecessary resources. Nonetheless, it can be observed that Poet does allow users to indicate whether or not an image should be described in the form of a tick-box. Secondly, whereas Poet merely incorporates AI that looks at the image only, our system also uses AI that considers the surrounding textual context of an image to generate suggestions for *non-visual descriptions*. The latter is crucial in order to create meaningful image descriptions. Finally, while Poet accepts EPUB3 files only [5], our application will support files in either EPUB2 or EPUB3 format.

2.4.2 Pilot Project by Fondazione LIA

Fondazione LIA [6] is a non-profit organization that promotes accessibility of Italian digital media. At the Digital Publishing (DPUB) Summit 2019, their Chief Accessibility Officer Gregorio Pellegrino presented a pilot project initiated by Fondazione LIA in collaboration with the University of Siena. This project served as experiment in using image recognition algorithms in the world of publishing. Specifically, to generate automatic image annotations in EPUB files. [7].

Similarities: His system is able to generate automatic (*conceptual*) image descriptions by using available services such as Google's Cloud Vision AI and Microsoft's Azure Computer Vision API. Secondly, his system has image classification as part of the workflow, similar to ours. Furthermore, at the time of the presentation his research project is still an experimental pilot, not mature for production yet. This is the case with our project as well, for which we are aiming to create a mere prototype, not intended to be actually used by annotators yet.

Shortfalls: The main difference here is that his system is fully automatic in the image annotation process, as he does not mention that the users are allowed to modify it after it is generated, whereas we are aiming for the combination of automated and manual image descriptions. Moreover, we allow users to manually classify images, but Gregorio mentions that his image classification process is automated (using GoogleML). As stated by himself, current AI systems for image classifications are not well-trained yet for reliable results, which is why we opted for manual classification. Also, his system does not seem to have any GUI attached to it, which we will incorporate to allow for user-friendly image annotation. Finally, similar to Poet, the AI systems he used are focused only on the image and not the textual context around it. Thus, our system is unique in the sense that it considers more than just the image itself.

[3] `http://diagramcenter.org/poet.html`
[4] See Could Have requirement 6 under 3.2.1.
[5] Poet also accepts e-books in DAISY3 (Digital Accessible Information System) format, which are less common however.
[6] `https://www.fondazionelia.org/`
[7] `https://www.youtube.com/watch?v=XZpgGNoBQoo&list=PLQAmHah-XQKwswkeLAerx6k09Xhgs29FH&index=9&ab_channel=EDRLab`

3 System Requirements

After having carried out a problem analysis, requirement engineering starts - the core subject of this chapter. That is, understanding what the system should do in order to satisfy all stakeholders' wishes, whom are presented in section 3.1. Afterwards, we explicitly specify descriptions of the services and constraints of the system to be built. This specification of prioritised system requirements is presented in section 3.2.

3.1 Identification Stakeholders

Our stakeholders will be divided over the following categories, which may overlap:

- **Direct stakeholders**: have influence on the requirements of the system;
- **Indirect stakeholders**: are impacted by the system, but will not be in touch with the system;
- **Internal stakeholders**: involved in the development process of the system;
- **External stakeholders**: outside the scope of the development process of the system.

Below, we list the *most relevant* stakeholders we identified per category, as well as what their demands are.

Direct stakeholders:

- **External**
  - Users: annotators
    Annotators can be authors, volunteers, or paid workers who will be using our system to improve image annotations in e-books. Hence, it is crucial that our web interface is user-friendly, meaning it is taking into account their wishes to make the platform easy to navigate through. We expect volunteers to be the main direct users of the annotation platform.
  - Government
    Governing bodies such as the European Union (EU) may impose legislation with regards to accessibility of (digital) media. Hence, our system should adhere to these by taking them into account when specifying system requirements.
- **Internal**
  - Development team
    We as development team are involved in day-to-day business with regards to this project: designing and implementing the image annotation platform. The project should be well-structured and coordinated in order for the developers to work efficiently. Our main aim is to have a working demonstrator at the end of this project, that is taking into account all stakeholders' demands.
  - Client: Ted van der Togt
    Our final product should by all means meet the requirements given to us by our client Ted van der Togt. As a researcher in (accessibility of) digital media at the *Dutch National Library (KB)* [8], he has a clear vision of what steps are necessary for making e-books accessible in an effective way.
  - TU Coach: Dr. Christoph Lofi
    Our coach Dr. Christoph Lofi is a researcher and professor in fields such as semantics, named entity recognition, digital libraries at the *WIS group* at TU Delft [9]. His expertise can guide us as developers in the design and implementation of the image annotation platform.
  - TA: Alexandru Bobe
    Our Teaching Assistant (TA) Alexandru Bobe is currently a third year Computer Science and Engineering student at TU Delft. As he has taken this Software Project (CSE2000) course previous year, he can help us with non-technical matters.

Indirect stakeholders:

- **External**
  - Visually impaired people
    People with visual impairments are the ones that annotators aim to help when it comes to accessing e-books containing images. Thus, our system should ensure that annotators can easily add meaningful image descriptions for them to facilitate their needs.

[8] `https://www.kb.nl/en/over-ons/missie`
[9] `https://www.wis.ewi.tudelft.nl/`

- Researchers
  Other researchers on accessibility of digital media may be impacted by the resulting product (think of interesting outcomes).

- **Internal**
  - *Future Libraries Lab*
    Since this project is part of the *Future Libraries Lab* [10] - a collaborative effort between *KB* and the *WIS* group at TU Delft, it is crucial that our product aligns with their initiative to define new future for libraries using current technology and innovation.

3.2 Requirement Specification

Based on demands of the various stakeholders we identified, we have specified the system requirements in the form of Natural Language Requirements, presented in 3.2.1. These are prioritised using the MoSCoW methodology. This way it will be clear by the end of the project if we have managed to complete the most important requirements. Subsection 3.2.2 highlights several noteworthy requirements.

3.2.1 MoSCoW Prioritization

We have classified all requirements into two types:

- **Non-functional**: define product properties - system will still work if these are not met;
- **Functional**: define product features.

In order to manage our project within the time constraints of ten weeks, we prioritised all functional requirements using the MoSCoW method. This entails we have divided them into four categories:

- **Must haves**: critical for a successful project, and thus should be prioritized;
- **Should haves**: important, but not as time-critical;
- **Could haves**: desirable in the future to improve user experience, but not necessary in the scope of this project;
- **Won't haves**: dropped, may be considered for inclusion in a future project.

Figures 1 - 5 present all system requirements - approved by our client Ted van der Togt and ordered by MoSCoW label.

1.	The system should be in the format of a web application.
2.	The system can work both as a standalone web application and plugged into another existing webpage.
3.	The design of the system's web interface should adhere to web accessibility standards as outlined by W3C.
4.	The system will rely on reusing existing solutions and approaches for image analysis and content analysis wherever possible.
5.	The frontend application shall be written in Javascript ES6.
6.	The backend application shall be written in Python 3.
7.	The system will store application-related data in a database for persistent storage.
8.	The system shall become open source at the end of this project.

Figure 1: The non-functional system requirements

[10] https://delftdesignlabs.org/future-libraries-lab/about/

1.	Users will be able to upload an EPUB file to the system from their local storage.
2.	The system will be able to process digital mixed-media content (text and images) from an uploaded EPUB file, by unzipping it and storing the unzipped contents.
3.	The system will be able to extract all images from the uploaded EPUB file.
4.	The system will have an embedded EPUB page viewer which displays all the pages containing images in the e-book page by page, as well as the textual context around the image.
5.	Users can select an image for annotation within their uploaded e-book and get a visual indication of which image is currently under annotation (highlighted e.g.).
6.	The system will have an editor next to the page viewer, which can display image descriptions and visually indicate which image is linked to a description. This description is either: a. the initial, already existing annotation (if any, else blank); b. the manual annotation saved earlier by this user for that image (overrides a).
7.	Users can classify a selected image in the editor by the purpose of the image - serving as perceptual description. (‘decoration’ or ‘comic’ e.g.).
8.	Users can edit descriptions of all images in an uploaded e-book and see these updates in the editor.
9.	The system will be able to let external image recognition systems detect labels or descriptions of the image under annotation - based on the image itself only - serving as suggestions for conceptual image descriptions.
10.	AI suggestions for a certain image are shown in a second field in the editor.
11.	The user can, at any point during the editing process, download the edited e-book, upon which they retrieve an EPUB file with all image annotations created by the user in the corresponding ALT-text fields.
12.	The system can push both the uploaded EPUB file and the final annotated EPUB file to an external version control system (Git e.g.), to be able to visually inspect differences in the format, file structure and code.

Figure 2: The (functional) Must Have system requirements

1.	The system will be able to let external text analysis systems around the image and generate corresponding keywords - serving as suggestions for non-visual image descriptions. These keyword suggestions can help the user make more meaningful image descriptions, as they build on top of AI suggestions that only consider the image.
2.	Users can accept AI suggestions for image annotations, after which this accepted suggestion will be copied to the editing field for manual image annotations.
3.	Users can inspect and navigate through all pages of the uploaded e-book in the editor - also those without images, in the original order of the e-book (by scrolling or using buttons).
4.	The system will add language tags - lang and xml:lang attributes - inside the HTML tags of every HTML/XHTML file in the EPUB.
5.	The editor can show the user a progress bar showing status updates (completed/to be done) about all steps in the current image annotation process, thereby indicating which step the user is currently in.
6.	The editor can show the user an error message whenever a failure occurs during the image annotation process.
7.	The editor has the possibility to display a URL corresponding to a specific book the user has uploaded, which can be used to revisit that book again.
8.	The editor has the possibility to display a URL corresponding to a specific uploaded book and image, which can be used to revisit that image in the book again.
9.	Users will be able to revisit a previously uploaded e-book upon accessing the corresponding URL for that specific e-book. This e-book should include all image descriptions which can be: a. the initial, already existing annotation (if any); b. the manual annotation saved earlier by this user for that image (overrides a).
10.	Users will be able to revisit a previously annotated image in an existing e-book upon accessing the corresponding URL for that specific e-book and image. This e-book should include all image descriptions which can be: a. the initial, already existing annotation (if any); b. the manual annotation saved earlier by this user for that image (overrides a).

Figure 3: The (functional) Should Have system requirements

1.	Upon upload of an e-book, the system will check if the EPUB file is valid (against EPUB specifications) and reject the upload if not valid. Validity will be determined by the open-source epubcheck tool.
2.	Upon upload of an e-book, the system will convert any uploaded, valid EPUB2 files to EPUB3.
3.	Upon upload of an e-book, the system will make the EPUB file automatically accessible, as defined by the W3C standard.
4.	After the user is finished editing the uploaded e-book, the system will check the accessibility of the resulting, annotated EPUB file using ACE by DAISY.
5.	While the e-book is being processed after uploading, the user interface displays status updates (success/failure) about each process being run on the e-book. In case of a failure, that process is aborted and the user is provided with the corresponding error messages.
6.	The system can display to the user which images in the uploaded e-book have been annotated already, allowing users to estimate the required time to annotate images in their e-book.
7.	The system can display to the user which images in the uploaded e-book are decorative, and thus should not be annotated.
8.	Users have the option to disable or skip AI image annotation suggestions for their uploaded e-book under annotation.
9.	The system will apply heuristics for estimating the purpose of an image in the given text context.
10.	Annotations will be linked to user IDs, to be able to keep track of which user added which annotation.
11.	The editor can show all past annotations generated automatically or created by users for each image, in an annotation 'history' menu.
12.	Users can see an overview of all e-books uploaded by him/her, containing all image annotations made so far.

Figure 4: The (functional) Could Have system requirements

1.	User authentication.
2.	Different roles for users, having different access rights.

Figure 5: The (functional) Won't Have system requirements

3.2.2 Noteworthy Requirements

It is noteworthy to mention that the following two requirements seem similar at first sight:

- Must Have 9: *The system will be able to let external image recognition systems detect labels or descriptions of the image under annotation - based on the image itself only - serving as suggestions for* conceptual *image descriptions.*
- Should Have 1: *The system will be able to let external text analysis systems around the image and generate corresponding keywords - serving as suggestions for* non-visual *image descriptions. These keyword suggestions can help the user make more meaningful image descriptions, as they build on top of AI suggestions that only consider the image* [11].

However, Must Have 9 describes the automatic generation of labels or descriptions using off-the-shelf services (such as Google's Cloud Vision API) that analyse the image under annotation. This is in contrast to Should Have 1, which describes the automatic generation of keywords extracted from the text *surrounding* the image under annotation.

Another remark is that we, together with our client, initially insisted on incorporating a data processing pipeline in the image annotation platform that would be responsible for several automated tasks, composed by the following four requirements:

- Could Have 1: *Upon upload of an e-book, the system will check if the EPUB file is valid (against EPUB specifications) and reject the upload if not valid. Validity will be determined by the open-source EPUBCheck tool.*
- Could Have 2: *Upon upload of an e-book, the system will convert any uploaded, valid EPUB2 files to EPUB3 using an external automated tool.*
- Could Have 3: *Upon upload of an e-book, the system will make the EPUB file accessible as defined by the W3C standard, using an external automated tool.*
- Could Have 4: *After the user is finished editing the uploaded e-book, the system will check the accessibility of the resulting, annotated EPUB file using ACE by DAISY.*

This data processing pipeline is intended to add accessibility features to e-books, in addition to image description. Whereas these four requirements listed above were labelled as Must Haves at the beginning stage of our project we, together with our client, figured that it would be more realistic to label these as Could Haves. This is due to the fact that these functionalities are not directly related to the core of our demonstrator: making image annotation efficient (using automation and human intervention). One of the automated steps involved in Could Have 3 is expressed as a separate requirement however - namely Should Have 4: *The system will add language tags -* `lang` *and* `xml:lang` *attributes - inside the HTML tags of every HTML/XHTML file in the EPUB.* Hence, Should Have 4 is not referring to external software, meaning the system would apply this step manually.

Finally, it became clear from our requirements elicitation that our client was strongly advocating for Must Have 12: *The system can push both the uploaded EPUB file and the final annotated EPUB file to an external version control system (Git e.g.), to be able to visually inspect differences in the format, file structure and code.* Different from all other system requirements, this requirement is mainly for developing purposes. Thus, this feature will not be paired with any user interface functionalities and thus kept simplistic. Despite this, we have labelled it as Must Have, since we agree it is useful for tracing back structural changes within the original EPUB, which may be caused (unintentionally) by external tools.

[11] As described in Must Have 9.

4 Design of Final Product: Solution Proposal

This chapter addresses our design proposal for our image annotation platform, based on the system requirements. Section 4.1 will give a high-level description of the workflow of our final product. Afterwards, section 4.2 elaborates on the architecture choices for our internal system. Moreover, this chapter motivates our GUI design choices in section 4.3. Finally, section 4.4 dives deeper into ethical implications that may come along in our design process.

4.1 Workflow of the Image Annotation Platform

Our final product (named "E-BOOK FIXER") will be a cloud-based web service with a graphical web user interface. The goal of the product is to be a demonstrator for a user-friendly image annotation platform that combines image recognition and text analysis, to aid users create image descriptions in an efficient manner. Figure 6 shows a UML diagram illustrating how the user will interact with the system, what parts of the system the user will use and what the use cases are.

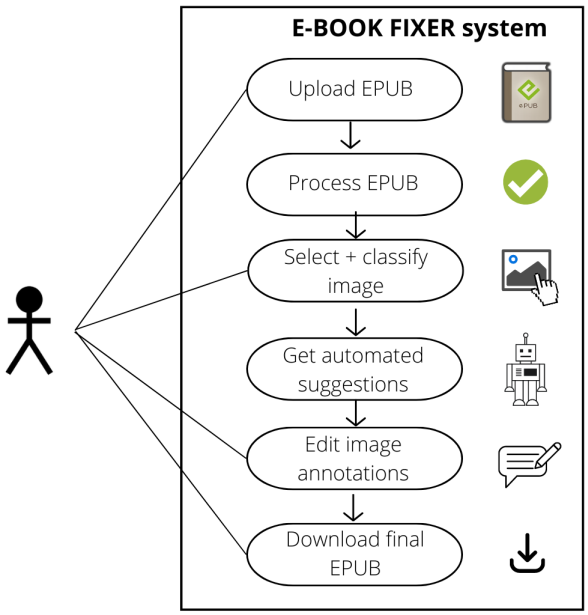


Figure 6: The UML use case diagram of the E-BOOK FIXER system

On the home page, the user will be able to upload their e-book - as an EPUB file - that they want to annotate [12]. We aim to make sure that this page contains enough information about the platform and instructions on how to annotate images. After uploading the e-book, the system will check if the EPUB file is valid (against EPUB specifications) and reject the upload if not valid. If the e-book was valid, the user will be redirected to the editor where the e-book contents - both text and images - are displayed, allowing users to iterate through its contents. After having selected an image for annotation, the user will first have to classify that image, defining its purpose ("Decoration", "Information", "Photo", "Illustration", "Figure", "Symbol", "Art", "Signature", "Text", "Flag", "Comic", "Logo", "Graph" or "Map"). Decorative images will not need to be described. After image classification, different types of automated suggestions can be generated - either related the image itself or its textual context. Based on the automated suggestions, users will be able to create their own image annotations, thereby overriding the possibly existing ALT-text. Finally, the user will be able to download the modified EPUB file containing all the newly created annotations. The editor also allows the user to copy a URL to their e-book or a specific image within their e-book. This URL can be used to revisit that (image within an) e-book or shared with other users to collaborate.

[12] Our system is mainly concerned with EPUB2 files, which are still the majority of e-books, but can also work with EPUB3 files.

4.2 System Architecture

This section explains on what major components will be implemented in order to achieve a complete interactive image annotation platform. Our high-level system architecture pattern - Client-Server - is presented in 4.2.1. Afterwards, 4.2.2 elaborates on what data is exchanged between the server and client.

4.2.1 High-level System Architecture: Client-Server

As high-level system architecture for our platform, we have opted for a Client-Server architecture - as portrayed in Figure 7. Having an explicit separation between client and server allows for different components of the image annotation platform to be separated among the two sides. This allows for maintainability of our system, as developers can easily distinguish between different responsibilities carried out by the two components within the system. Moreover, information can easily be shared among the two through remote connection via a REST API [13] that uses HTTP requests to create, read, update, or delete data (also referred to as CRUD operations).

Figure 7 visualises how the user's browser communicates with the client system to collect the appropriate data to display. To obtain this data, the client will make HTTP requests to the server, which handles data storage and business logic. As illustrated, the server keeps track of image annotations in e-books using its database, whereas the client allows users to see and annotate images inside their e-book through its GUI. Upon receiving a client request, the server will perform CRUD operations on its database entries, after which a JSON response is returned to the client. Besides a database, our system also uses its local file system to store EPUB files [14]. An important requirement that we have identified is that users should not have to install any additional software locally to be able to use our platform. Therefore, our server will be hosted on a cloud and any functionality that requires external software will be handled here. This includes communication with external AI systems that will generate suggestions for image annotations.

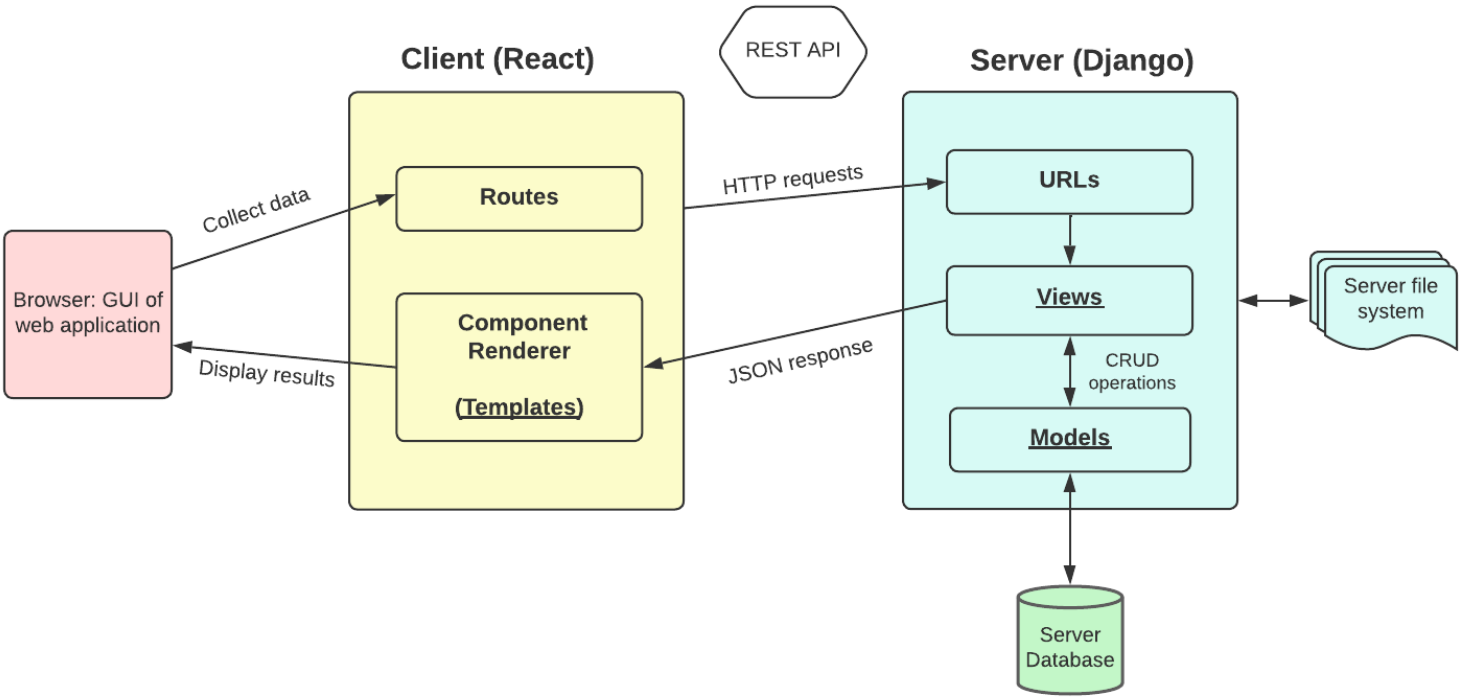


Figure 7: High-level diagram of the Client-Server architecture of the E-BOOK FIXER system

Below we will motivate the frameworks and programming language(s) we have opted for to implement the client and server systems. Besides, our choice for database on the server-side is explained. Please refer to chapter 6 for detailed explanation on how the client and server systems were implemented on a low-level.

[13] Representational State Transfer Application Program Interface
[14] Refer to section 6.3 for detailed explanation on how the server stores data.

Client System: React

The client system of our image annotation platform will be written in JavaScript, HTML and CSS - the backbone of frontend web development. We have opted to use the well-known JavaScript web-framework/library React (or ReactJs) [15], as it comes with several features that can help us build a user-friendly interface for our project:

- Easy to learn: as frontend applications are commonly implemented in JavaScript and React, learning and finding resources are made trivial;
- Component-based: React components can be seen as independent, reusable JavaScript classes or functions, which makes the management of data and user interactions a lot more organised. [16]
- Fast and responsive: React creates a copy of the Document Object Model (DOM) tree (serving as "cache" or "virtual DOM"), to avoid having to render the actual DOM tree on every change in the HTML code.
- Extensible: React comes with a large ecosystem for available extensions and supported packages, making our application easily extensible.

Server System: Django

The server system will be written in Python, which seemed the convenient option for us considering our feasibility study showed us that many external tools our system may rely on come with Python packages. Our backend will be developed using the web-framework Django [17]. There are several aspects to Django that are suitable for our project:

- Well-documented: as Django is widely used, the developers of this framework have provided extensive documentation of all available features;
- Fast development and deployment of systems: Django comes with a built-in REST framework, allowing us as developers to instantly create APIs to allow for communication between client and server;
- MVT design pattern: Django comes with a so-called MVT (Model-View-Template) design pattern, which defines our low-level system architecture. This pattern makes our system modular and thus easy to maintain. [18]

Data Storage System: MongoDB

The server has a data storage system that keeps track of information related to e-books, images and their annotations - as becomes clear from Figure 7. There are various advantages to persisting image annotations and their metadata in a database. Firstly, the system will operate more cost-efficiently, as we store image annotations and their metadata generated by expensive external AI algorithms. Automatically generated labels/descriptions in combination with their confidence level could even be used as training data for machine learning algorithms. Furthermore, server-side storage is beneficial for end users, as they can save their progress and come back to it later on any device. Finally, collaboratively annotating e-books is something that can easily be achieved with server-side storage, as data doesn't have to be stored on a specific user's machine.

Our choice for database is MongoDB [19] - a non-relational (or "NoSQL") database coming with several features that our system can benefit from:

- Document-oriented: MongoDB is known to store data in collections of JSON-like documents, instead of tables as in traditional relational databases. This allows for flexibility - as documents may have varying structures - and support for unstructured data that cannot not always be expressed as tables.
- Schema-less models: Documents do not require a predefined schema, as the data structure may be modified at any point during development. This is especially useful in combination with our chosen development methodology: Scrum - which is based on iterative increments to our software. [20]
- Simplified and faster querying: due to MongoDB's document model, data access rarely requires joins. This is in contrast to relational databases, where tables are normalised.

Section 6.3 presents our final database architecture, containing models for e-books, images and annotations.

[15] `https://reactjs.org/`
[16] Section 6.1 breaks down on what exact components we have used in our system.
[17] `https://www.djangoproject.com/`
[18] In our system, the "Template" component is handled by the React client - as is indicated by the underlined concepts in figure 7. See section 6.2 for more explanation about this MVT design pattern.
[19] `https://www.mongodb.com/`
[20] Chapter 5 elaborates on our software development methodology Scrum.

4.2.2 Data Flow Client-Server

This section will dive deeper into what data is exchanged during the process of annotating images in e-books, between different components in our system: the client, the server and its data storage system. This will be illustrated step by step, referring to the diagram shown in figure 8 on the next page.

Upon entering the website, the users have two options: to upload an e-book - as EPUB file - **(1)** or enter the URL specific to an existing one **(2)**.

1. In the case that the user uploads an EPUB file, it is sent to the server, which will validate the uploaded e-book and may add additional accessibility features to the EPUB file **(3)** [21]. Here, the server will extract the contents of the EPUB and store them in its file system. Then, the pipeline checks that the uploaded e-book is valid and may perform structural changes on the EPUB file to allow for more accessibility. Additionally, the server will push both the originally uploaded EPUB file and the resulting EPUB3 file after the data processing pipeline is finished to an external version control system, so that the structural changes made by the scripts can be manually inspected. [22] Finally, the server returns to the client the metadata corresponding to the newly created e-book database entry.

2. In case the user chooses to access a URL specific to an e-book [23] - after having closed his browser or to access a book that was shared with him, we simply return the e-book associated with it (if it exists), including all annotations made so far.

In either case, the user is then redirected to the editor page, in which the e-book is visible in an embedded EPUB viewer **(4)**. The rendering of e-book pages and extraction of images for display is dealt with by the client. Upon selecting an image for annotation, users first need to classify it (5). This classification is sent to the server, to be stored in its database.

After classification, the user has the option to generate AI suggestions **(6)** for that image, by choosing one of the available image recognition engines that produce *conceptual* image descriptions. The server will also generate AI suggestions from a text analysis tool that generates *non-visual* image descriptions. In order to retrieve any type of AI description to display to the user, the client sends a request to the server, which is responsible for sending the corresponding image to the AI, saving the generated description to its database and finally returning the AI suggestion back to the client - to be displayed to the user.

Following generation of AI suggestions, the user can manually improve or completely override the existing ALT-text (if any) **(7)**, using the generated descriptions. Upon the user saving his changes, the new image description is sent to the server so that it can update the metadata in its database. Then, the user can move on to the next (or previous) image and follow the same workflow we just described from **(5)**.

At any point during the editing process, the user is able to press a "Download" button **(8)** to request the modified EPUB file from the server. Subsequently, the server injects all image annotations stored in its database into the corresponding HTML files of the e-book and then compiles it into a new EPUB file. This final EPUB file is, again, pushed to an external version control system for manual inspection purposes. Finally, the new e-book version is returned to the client - ready to be downloaded into the user's device.

[21] This is the data processing pipeline discussed earlier in section 3.2.
[22] See Must Have 12 in section 3.2.
[23] In fact, this URL is also used by the client to request the processed EPUB file, while waiting for the pipeline to finish. This processed e-book will then be displayed in the user's browser (if valid).

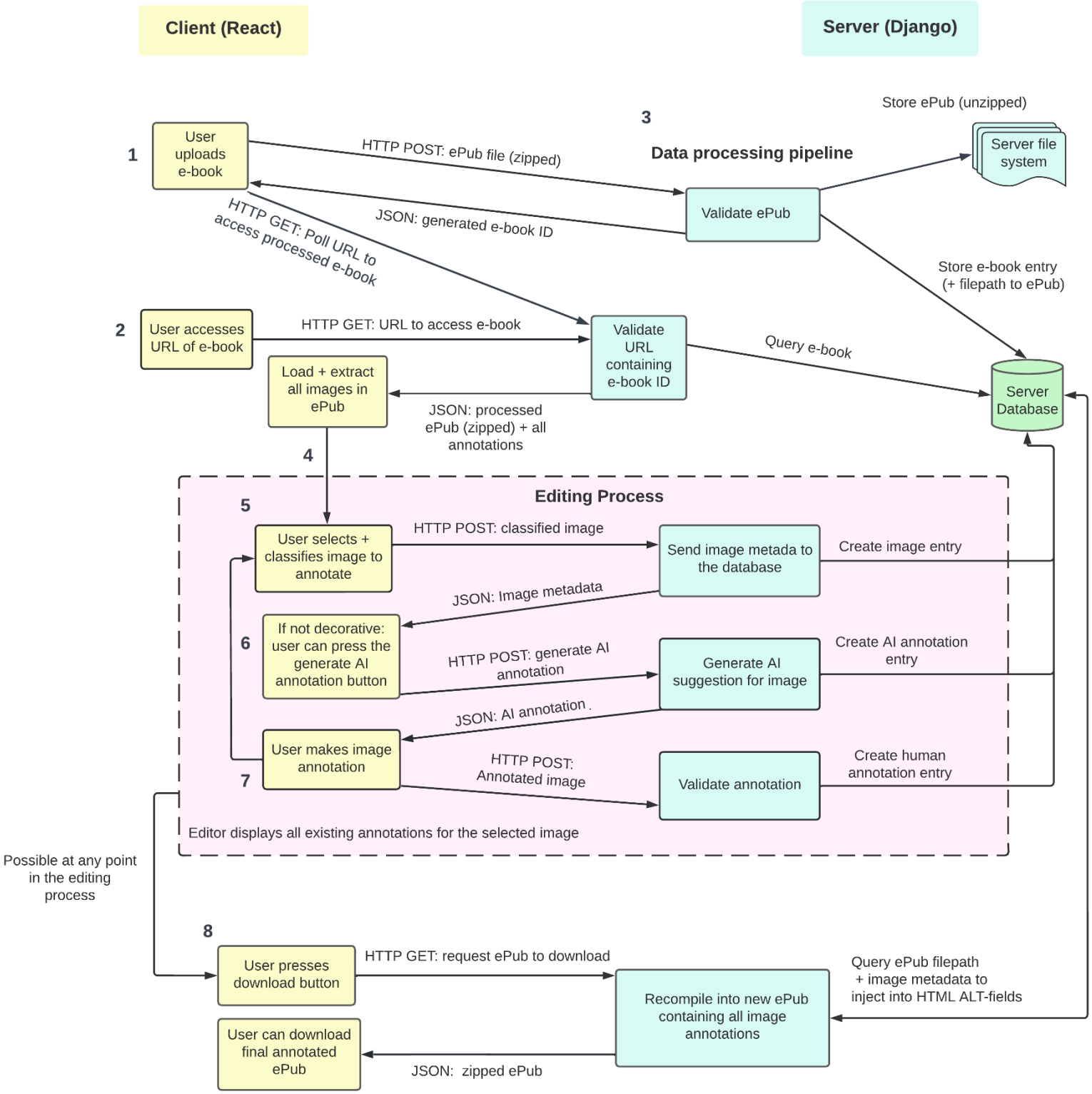


Figure 8: The data flow between the client and the server, in the form of HTTP requests and JSON responses

4.3 Graphical User Interface Design

Our web system for image annotation consists of multiple views that the user can work with, to access functionality in different parts of the system. This section will go over the two main graphical components of our system: The Home View and The Editor View, discussed in subsections 4.3.1 and 4.3.2 respectively. Thereby, we motivate our design choices based on web accessibility standards as outlined by W3C [24].

Overall, we as a team agreed it was crucial that there is sufficient contrast between foreground and background in our GUI designs, ensuring all text is readable for users on the website. Another important aspect for us is easy navigation throughout the page. Moreover, by making use of icons and textual labels rather than merely relying on colours in order to convey information, we take into account people who are colour-blind.

4.3.1 The Home View

The first GUI component that the user encounters upon visiting our website is the home page as displayed in figure 9, which has the purpose of allowing users to upload their e-book files to annotate, and providing users relevant information about the purpose and usage of the annotation platform. Note that this page is scroll-able, preventing the user's screen from becoming too cluttered.

On the very top of the page, the logo and slogan of our web platform take up the centre of the homepage, clearly depicting the identity of our project and grabbing the user's attention upon accessing the page. The slogan summarises the main objective of the annotation platform in a catchy fashion.

Right next to the logo, several clickable icons are depicted, allowing the user to easily navigate to other pages, indicated by their associated label ("Help", "Settings", "Uploaded e-books"). These are additional functionalities that the system may be extended with in the future, which we believe can greatly improve user experience.

The upper purple dashed box below the logo allows the user to upload their e-book, leading them to the editor automatically. By giving the upload box a vibrant colour, it will be easily identifiable when entering the website. Also, users that are familiar with the platform would most likely want to start annotating images in their e-book right away, which is why we put this functionality at the top.

Below this upload box, the most crucial information about the image annotation platform can be found, aiming to guide new users who are trying to understand how the interactive system works and what its main objective is. In the future, more detailed information can be incorporated - for example in a sub-menu or further down the page. Throughout the page, icons are used to support the purpose or message of text, helping the user to navigate through the page more easily in a playful manner. At the very bottom, a complete workflow is displayed to the user, explaining what steps will be taken during the image annotation process in the editor. Bold fonts are used to make the most crucial information stand out and icons are used to help the user process information rapidly.

[24] `https://www.w3.org/WAI/tips/designing/`, see also non-functional requirement 3 in section 3.2.

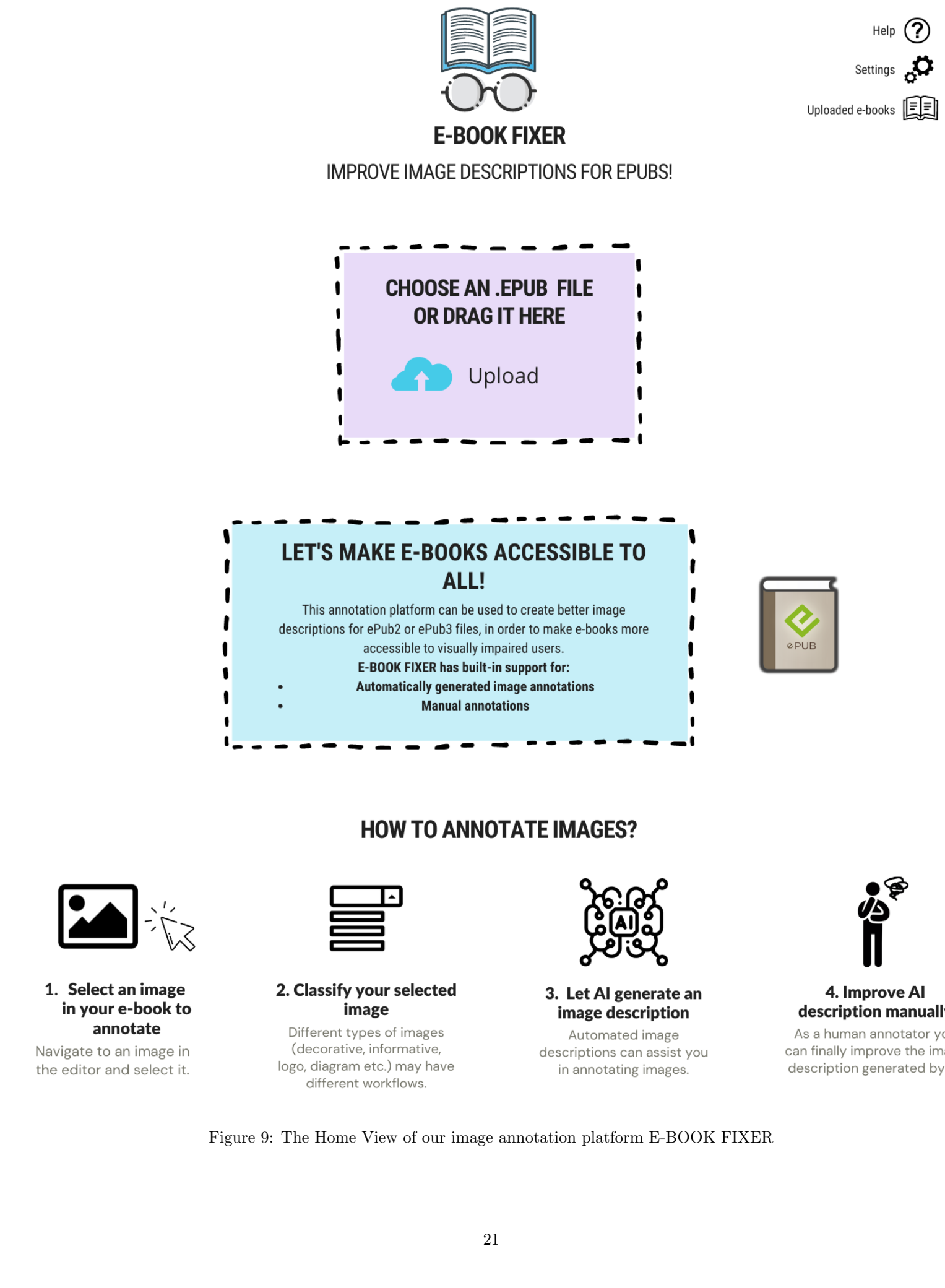


Figure 9: The Home View of our image annotation platform E-BOOK FIXER

4.3.2 The Editor View

After the users have successfully uploaded their e-book, they can proceed to editing it in our editor, for which the design can be seen in figure 10 below. We aimed for the navigation inside the e-book itself to be as effortless and intuitive as possible, so we opted to have the reader display the contents of the book in a scroll-able manner. This way, the user can access the textual context of the image on the page to make a meaningful annotation for the image.

One of the key principles of our design is having coherent pieces of GUI elements, thereby making the the interface intuitive for users. We achieved the latter by using different colours for GUI elements, based on the type of action they are linked to. To illustrate this, a *green* colour signifies an action that involves sending information to the server's data storage, while *blue* involves navigating through currently available information - for example scrolling to the next image in the e-book using the buttons containing "Previous image" and "Next image".

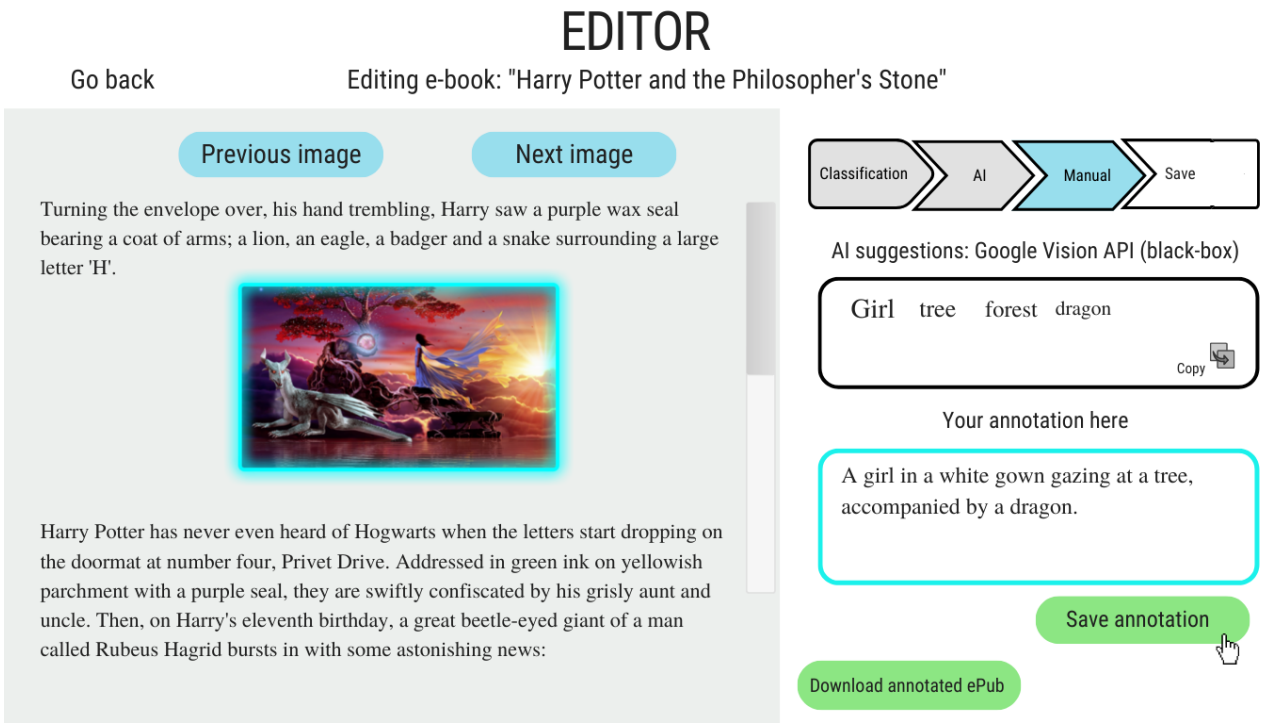


Figure 10: The Editor View of our image annotation platform E-BOOK FIXER

We decided to structure the process of annotating a single image in a step-by-step manner, which is seen in figure 11 below. Pink arrows show transitions between stages and are not part of the interface.

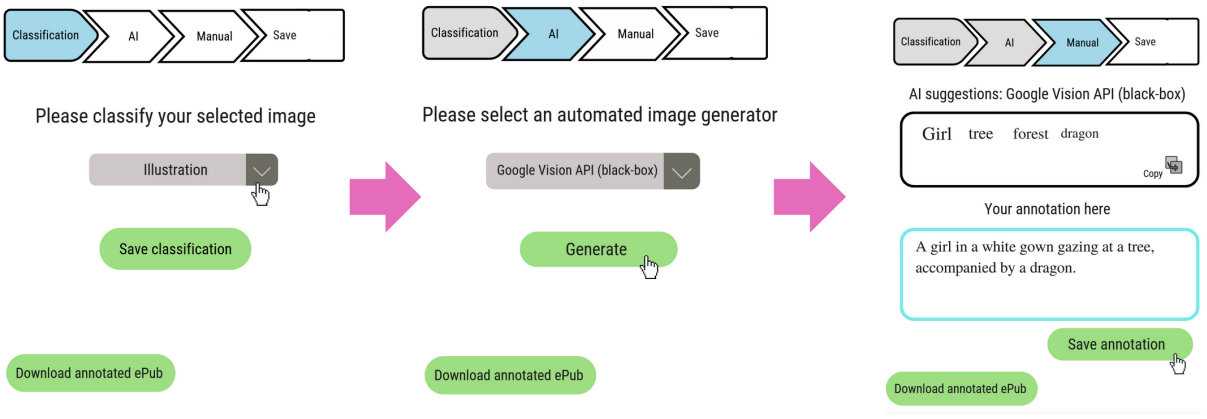


Figure 11: The various steps in describing images in the Editor View, represented by a progress bar

The progress bar on top can give users a visual indication of the current step in the image annotation process. This can be: classifying the image (step 1), selecting an AI engine to generate an automatic description with (step 2), writing the manual description (step 3). In the final stage, users can review and save their work (step 4). These steps are represented by clickable arrows, taking users to their desired steps. The *blue* colour is shown to be the current step the user is in.

Step 2 ("AI") is a core part within our image annotation platform, as this is the stage in which various AI suggestions can be generated by users to save labour. By providing users with a drop-down menu, they can easily select an image recognition engine they wish to retrieve automated image descriptions from - "Google Vision API" for instance. Not only will the system generate suggestions from this selected image recognition engine, but it will also generate keywords from an external text analysis engine. In step 3 ("Manual"), these automatically generated keywords, labels and/or descriptions are then displayed to the user. This is also where they can use the suggestions to annotate their image in an - ideally - more efficient manner.

4.4 Ethical Implications

During the design phase of the image annotation platform, we believe it is crucial to consider ethical issues that may come along. The most crucial ethical aspects in the context of our project are the following two: the use of AI and humans interacting with our system. Those will be discussed in more detail in the subsections below.

As our demonstrator system will not have user management, topics such as privacy and data are not applicable to the scope of our project. Although, we might have to consider copyright laws, as users may upload copyrighted e-books to our system. Once stored there, they might share the link to that e-book with anyone they please, who can then download the pirated e-book from our servers. One way to prevent this legal issue is to moderate the e-books that are uploaded, or only allow uploads from trusted sources - which is resource intensive however.

A common way that Cloud Service Providers (CSPs) deal with complaints relating to the data they store, is by opening a communication channel to people who want to send such complaints. Then, the CSPs have the option to act on them and remove the data that the complaint concerns. According to recent law cases in the EU, CSPs open to the public cannot be held accountable for data the users upload on it. [7] Despite the fact that this is a good sign for us, we may still fall into a different bracket. To clarify this, we do not only store the data, but also modify it and aid the user in modifying it. While we do not sell or do anything with the changed e-books, the sole fact that we are changing them could be cause for concern by a publisher. The fact that our intentions are only for the improvement of the accessibility of the e-books and not their efficient distribution gives us moral support in this issue.

4.4.1 AI

While the AI that we are using does not make any direct decisions about the system and cannot do anything besides recommending image descriptions (if the user chooses to do so), there are still ways that AI can make an unfavourable impressions.

Due to the variable nature of AI, we cannot provide any guarantees that our system will consistently help users save labour when annotating images. It could be the case that humans will be better off ignoring the generated annotations and just do them by themselves. Even the most advanced AI systems for image recognition (Google Vision API e.g.) struggle to describe images in the way humans do [8]. After all, they work by offering the most probable description of an image based upon the prior knowledge on which they have been trained. Books (especially e-books) are an infinitely creative and unpredictable medium, and if that system encounters an image that does not fit with any of its previous knowledge, its predictions become worthless.

One way our AI could influence the decision of a person annotating images in an unfavourable way is if it gives a misleading suggestion for the description of an image. This the extent to which the use of AI in our system can harm our stakeholders however. Moreover, we can draw this concern in terms of a trade-off. Despite the fact that the AI suggestions may be inaccurate, we rely on "humans-in-the-loop" who can manually inspect these automated suggestions (of low quality) and formulate their own improved image descriptions based on these. Hence, it is arguable that the potential benefit outweighs the potential cost here. All in all, we can safely say that attempting to automate the process of image annotation brings us a step closer to improving accessibility of e-books for everyone.

4.4.2 Humans

On the topic of humans, the indirect stakeholders of our project include people who are visually impaired and would still like to enjoy e-books containing visual content. Our direct end users will be mostly volunteering annotators, but also people whose job is to help those people by making e-books more accessible - by providing descriptive image annotations. These people make great contributions to society by improving equality and fairness regarding access to information. By annotating images they allow a large number of people to learn from and be entertained by e-books. Helping them make their job more efficient than it is now, will allow them to reach more people and make many lives better. While the mission is a noble one, there are things which we have to be aware of, so that we do not end up detracting from our users' experience.

Though our users will probably not be needing extensive accessibility features in the GUI, that does not mean we should not follow general web accessibility guidelines. Those guidelines include things such as colours and contrast of different elements on the page, making it easier for people who cannot distinguish certain colours to navigate our website. On top of that, the text size and the font choice is important for the readability of our interface. Most of those things can be checked visually, but there are also extremely useful tools like the Lighthouse tool available in Chrome browsers which can test our website on its code as well as its appearance. The code of our website should also adhere to accessibility standards and be easy for humans to interpret, as making our software open source is counter-productive if we do not document and organise our code in a helpful manner.

To conclude, our end goal is to make the life of the end users easier, but we have to be careful with the design decisions we make, as taking a wrong turn can result in the opposite effect than the one we intended.

5 Software Development Planning

Up to this point, our report has mostly touched upon aspects related to features of our *end product*. Contrarily, this chapter dives into the planning of our *development process*. Below is motivated our choice of software development methodology: Scrum. Section 5.1 presents our plan regarding software quality assurance. An overview of various tools that helped in our development process is given in section 5.2. Finally, section 5.3 showcases our general project planning.

Scrum Methodology

We will be using the well-known Scrum methodology to organise our project. This means our process of planning, designing, developing, testing and deploying will be based on iterative and incremental processes. We believe this approach can help us "increase flexibility and produce a system that is responsive to both initial and additional requirements discovered during the ongoing development" [9].

Every sprint, starting on Monday and lasting for one week, a subset of all issues (system features to implement) will be worked on. The scrum leader has the task to coordinate progress within the team and thus is also the chairman of all meetings that week. On Mondays, the team comes together for a sprint meeting to evaluate the previous sprint by documenting what went well, what problems may have been encountered, and what could be improved on in the next sprint. Following this reflection, we will distribute new issues for the next sprint by considering our list of prioritised system requirements. Each functional requirement may be split up into several issues, to be assigned to one or multiple members. Thereby, we ensure fair work distribution by collectively estimating time required for each issue.

5.1 Software Quality Assurance

Software quality assurance is a priority throughout the project, especially considering the fact that this project will become open-source. The assignee of a certain issue has the responsibility of testing their own implementation, as this is part of the "definition of done" for all issues. This may happen after or before implementation. Moreover, each new feature is reviewed by at least two members. This gives us confidence that the piece of code is working adequately and we avoid having to deal with errors later in the development process.

In order to facilitate testability as much as possible in the implementation stage, we are aiming for proper design for testability. For example, reducing coupling by allowing for dependency injection allows external dependencies to be mocked. Furthermore, we run static analysis tools on our code [25], which analyse our code without actually running it, to see if it complies with coding conventions and code quality standards.

In our system, different levels of testing come into play, each serving its own purpose. The lower in the list, the more complicated and expensive testing becomes, but the closer to reality as well.

1. **Unit Tests**: main type of testing used for business logic on the server-side (using PyTest). Isolated pieces of code are tested, while external dependencies are abstracted away using mocking and stubbing.
2. **Integration Tests**: used for testing API calls from the server to external dependencies, thereby making sure the different components together behave as intended. For example, client's requests to the server are tested using Python's RequestFactory.
3. **Manual Tests**: used for accessibility/exploratory testing of the full application. This type of testing is essential for verifying the GUI of our system, including user interactions and visual aspects. Not only did we as developers perform these manual tests [26], we have also conducted user evaluations with various stakeholders on a prototype of our system - as will be discussed in chapter 7. Furthermore, to allow for simple verification of our GUI, we have created a manual testing plan, which can be found under Appendix A.

Since testing is costly, and exhaustive testing is impossible, an indication of our confidence in the code quality is required. Hence, we measure code coverage reached by our unit and integration tests. We strive for at least 80% code coverage on our server-side.

[25] Using eslint and flake8 on the client- and server-side respectively.
[26] Using public domain e-books from `https://www.gutenberg.org/` e.g.

5.2 Developer Tools

Below we list several tools that are crucial to our development process of our image annotation platform.

GitLab

GitLab supports us in terms of version control, code quality checks (Continuous Integration) and issue maintenance. Issues in Gitlab can represent a feature to implement or a bug fix. For each issue that will be worked on, a dedicated branch will have to be created with the number of the issue in the name of the branch. Each issue description follows a template which covers the following aspects:

- Description;
- Related issues;
- Definition of done;
- Possible solutions.

Furthermore, issues come with a weight estimate and relevant predefined labels with regards to priority (must/should/could), status (assigned/implementing/done/review) and type (backend/frontend).

To accept and merge new functionality, we require the related issue to adhere to its "definition of done", and to have two Merge Request (MR) approvals by fellow team members who did not work on that issue. Furthermore, each MR will follow a predefined template which covers:

- Type of issue (feature or bug-fix)
- Related issues;
- Checklist (test coverage, style errors, definition of done etc.);
- MR functionality / details;
- How to test this feature;
- Additions to codebase.

As well as that, meeting agendas, meeting notes, sprint retrospectives and task divisions will stored in our GitLab Wiki page.

GitHub Pages

GitHub Pages allows us to host our code documentation on a webpage, using a custom domain name: *https://revirator.github.io/Ebook-Fixer/*. We believe that hosting our code documentation can guide new developers that may continue this project in the future.

Mattermost

This platform will serve as our main means of communication with our TA and to keep track of the latest announcements concerning the project, by the TA or course organizers.

Google Drive

Our most important project files are stored and shared in a Google Drive folder, which serves as a simple version control and collaboration tool.

Microsoft Teams

Meetings with our client or TU coach will be held in Microsoft Teams, which allows us to easily send invitations.

Discord

Discord will be used as a communication tool within our team, to discuss relevant project-related aspects, vote on decisions to be made, ask each other for help and have remote meetings.

5.3 Project Planning

Below, a rough outline of our general project planning is shown in figure 12. This serves as a general guide over the course of the project, thereby giving us an indication of the feasibility of all tasks to be done in ten weeks. For each task, we came up with time estimates as a group. These estimates were made based on the assumption that each member will spend approximately 32 hours per week on project related tasks (in yellow, blue, pink and green) and 8 hours per week on assignments for the course (in orange and red).
The top rows indicate that each week has a different chair and minute-taker (or note-taker). This allows each member to practice crucial skills related to leadership and communication.

We have meetings with our client, coach and TA throughout the project - to make sure we stay on the same page with them. Around week 4.6 we expect to have implemented a decent prototype to carry out a GUI evaluation on, with various stakeholders. We made sure that there is still sufficient time after this user evaluation to resolve potential issues in our GUI. After week 4.8, we have planned a "buffer sprint" to make final touch-ups. Finally, we believe it is important that we keep on track with documenting our project in this report, as it will be the only officially lasting document of our whole project.

	Week #	**Week 4.1**	**Week 4.2**	**Week 4.3**	**Week 4.4**	**Week 4.5**	**Week 4.6**	**Week 4.7**	**Week 4.8**	**Week 4.9**	**Week 4.10**
	Mon	**18/Apr**	**25/Apr**	**02/May**	**09/May**	**16/May**	**23/May**	**30/May**	**06/Jun**	**13/Jun**	**20/Jun**
	Fri	**22/Apr**	**29/Apr**	**06/May**	**13/May**	**20/May**	**27/May**	**03/Jun**	**10/Jun**	**17/Jun**	**24/Jun**
	Chair	Nadine	Aratrika	Filip	Vlad	Denis	Nadine	Aratika	Filip	Vlad	Nadine
	Minute-taker	Filip	Denis	Vlad	Aratrika	Nadine	Filip	Denis	Vlad	Aratrika	Filip
Planning	Research										
	Interview stakeholder(s)										
	Requirements Priorization										
	Solution proposal										
Design	GUI mockups										
	Architecture										
Implementation	Set-up client app (React)										
	Set-up server app (Django)										
	Set-up entities										
	Set-up (web) database										
	Implement controllers										
	Create HTML templates										
	Add visuals to GUI										
	Implement Must Haves										
	Implement Should Haves										
	Implement Could Haves										
	Test code										
Reflection	Meetings with client										
	Meetings with coach					Midterm					
	Meetings with TA					Meetings					
	Interview stakeholder(s)										
Documentation	Project Plan										
	Final Report										
	Final Presentation								Prep		
	TW assignments	A1 (22-04)	A2&3 (29/04)	A4&5 (6/05)	A6 (13/05)	A7 (16/05)		A9 (30/05)	A10 (9/06)	A11 (13/06)	
						A8 (20/05)				A12 (19/06)	
	Deadlines		**Project Plan**							**Final Report**	**Presentation**
			Thu 28 April							1 week	Date TBA
										before	
										presentation	

Figure 12: General project planning

Moreover, we have created a general task division that can be found under Appendix B.

6 Implementation of Final Product

This chapter will dive into technical aspects that came along during our process of implementing the interactive image annotation platform E-BOOK FIXER. This implementation is based on the solution proposal that was presented in chapter 4. Sections 6.1 and 6.2 will illustrate how the React client and Django server were built respectively. Afterwards, section 6.3 elaborates on the architecture of the server's database. The technologies we have used to let the system generate automated image annotations and deploy our web application are outlined in section 6.4.

6.1 Client System

Using the JavaScript framework React for handling the GUI of our platform is convenient, as it offers a simple way of separating and reusing code in components - as motivated in section 4.2.1 already. This section describes what different components our frontend client system is composed of, as well as what tools and libraries were used to build these.

React Components

Very broadly, our website consists of a Home View and an Editor View. The Home View is where users can upload their EPUBs and find information about the image annotation platform. The Editor View allows users to navigate through their e-book and annotate its images. This view houses most of the interactive parts of our website. To visualise how the Editor View is divided into components, see figure 13.

Figure 13: Examples of components in our frontend system, part of the Editor View as of week 6 of the project

In above figure, blue boxes and accompanying pink text highlight the React components. These components can store and share mutable information via state [27]. The state of a component will be used to make decisions on its appearance and functionality, to be rendered in the user's browser. For example, the navigation buttons - part of the *EditorControls* component - should not let users go past the final image in the e-book. Hence, the right arrow button is hidden once the state storing the current image changes to the final image.

Components are organised in a hierarchy as can be seen in figure 14, where the ones at the top contain the ones below them - also referred to as their "children". As illustration, from figures 13 and 14 it can be seen that the navigation bar (*NavBar* component) contains a button for sharing URLs (*ShareURL* component).

[27] `https://reactjs.org/docs/state-and-lifecycle.html`

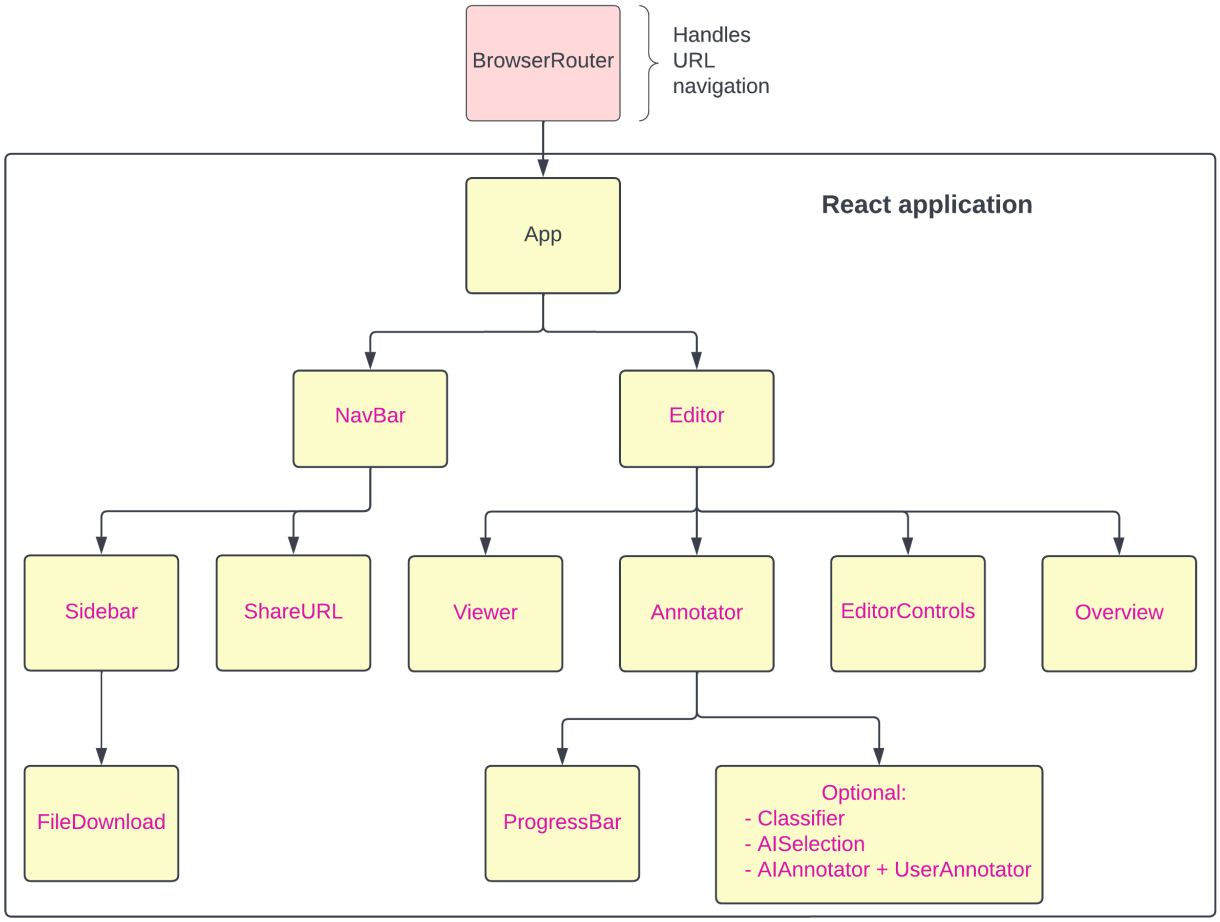


Figure 14: Example of a component hierarchy in the Editor View of our frontend system

The central component that is responsible for image annotation is the *Editor* component, represented as "right child" of the main *App* component in figure 14 [28]. Below is listed the components that can be found inside it:

- *Viewer*: displays the EPUB contents to allow users to navigate through their e-book
- *Annotator*: contains the following "child" components that represent the complete image annotation workflow:
  - *ProgressBar*: indicates which step of the annotation process the user is in currently.
  - *Classifier*: allows the user select the type of the current image.
  - *AISelection*: allows the user to pick an image recognition engine to generate suggestions (or skip the annotation process).
  - *AIAnnotator*: displays the labels/descriptions generated by the selected AI tool, together with keywords extracted from the text surrounding the current image.
  - *UserAnnotator*: allows for manual image descriptions (based on AI suggestions).
- *EditorControls*: allows the user to navigate between images using (arrow) buttons.
- *Overview*: expandable into a window that shows all the images in the book.

[28] Note that this *Editor* component is part of the Editor View, but it not labelled in figure 13 to prevent the image from becoming too cluttered. The same applies to the *Annotator and Overview* components.

Tools & Libraries

When working with JavaScript and React, the usage of the vast package repository NPM [29] allows for accessing plenty of external tools and libraries. Below we list the most essential ones used to make the implementation of our frontend easier:

- **EPUB.js** [30]: open-source library for rapidly loading and displaying EPUB contents in the editor. Also, it gives access to metadata about images in e-books.
- **React-Router**: industry standard for implementing navigation using URLs. Using this tool avoids the need for a server to feed the client the appropriate view. Instead, different URLs can be programmed, linked to their corresponding component.
- **Jotai**: state management library for sharing state globally.
- **JSdoc**: developer tool for automatically generating code documentation.
- **Sass**: powerful tool providing additional functionality on top of CSS, thereby drastically reducing the amount of code duplication.

6.2 Server System

Similar to how our client system was composed of separate React components, our server system - built using the Python web-framework Django - is made up of several sub-applications. Having such modular pieces of code allows for code maintainability and thus simple future system extension. This section lays out what Django applications can be found on our server, as well as what tools and libraries were used to build these.

Django sub-applications

Our backend Django system is composed of three different sub-applications, responsible for handling logic related to e-books, images and annotations. These are visualised in figure 15. Below is listed each of these three applications and their functionalities:

- *Ebooks*: handles uploading of e-books - thereby processing the EPUB file, storing it on its file system and saving an e-book entry in its database. This application also deals with injecting image annotations into EPUB files to be downloaded by users, by using metadata stored in the *Annotations* application.
- *Images*: handles bookkeeping of metadata related to images inside an e-book, including their classification and surrounding textual context.
- *Annotations*: keeps track of metadata related to automated/manual image descriptions, including communicating with external AI tools.

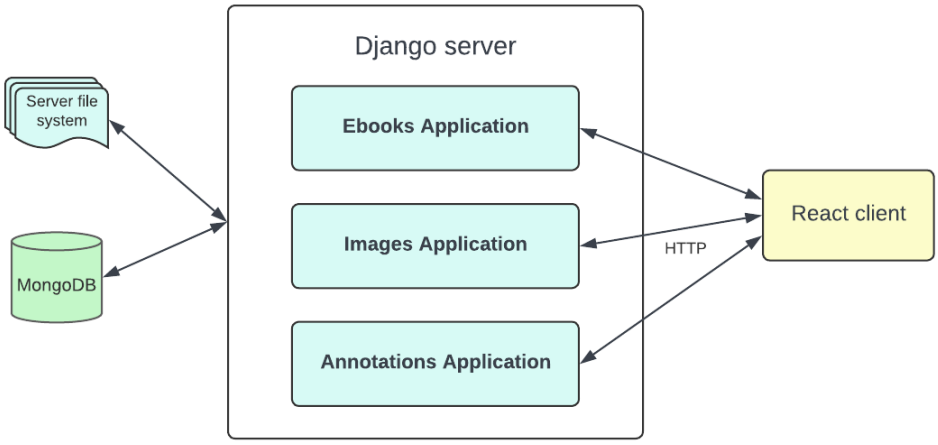


Figure 15: Overview of sub-applications inside the Django server system: *Ebooks, Images, Annotations*

[29] `https://www.npmjs.com/`
[30] `http://futurepress.org/`

All of the above Django applications are built similarly, using the low-level Model-View-Template (MVT) architecture - an example of which is presented in figure 16. Below the main components displayed in this figure are elaborated on:

- **Models**: stores Python objects to represent data of the web-application, which are mapped to (MongoDB) database entries [31] by Django ORM. Each model has its own Serializer class to transform them from Python objects to strings in JSON format and vice versa. So far, the *Ebooks* application contains one model: the *Ebook* model. This model has a FileField referencing the location in the server's file system, storing the actual EPUB.
- **Views**: contain API endpoints which are used to communicate between client and server via HTTP requests. To elaborate, these endpoints contain logic handling CRUD operations on Models. Furthermore, Views are responsible for processing EPUBs, saving their contents to local storage and communicating with external APIs. As can be seen in figure 16, HTTP requests from the React client are forwarded to the appropriate Views via *URLs*. All HTTP requests are managed by the built-in Django REST framework.
- **Templates**: control what information is displayed in the user's browser. Within our system, these Templates are provided by the React client. In fact, React component make requests to the server, after which they will render HTML output depending on the server's response.

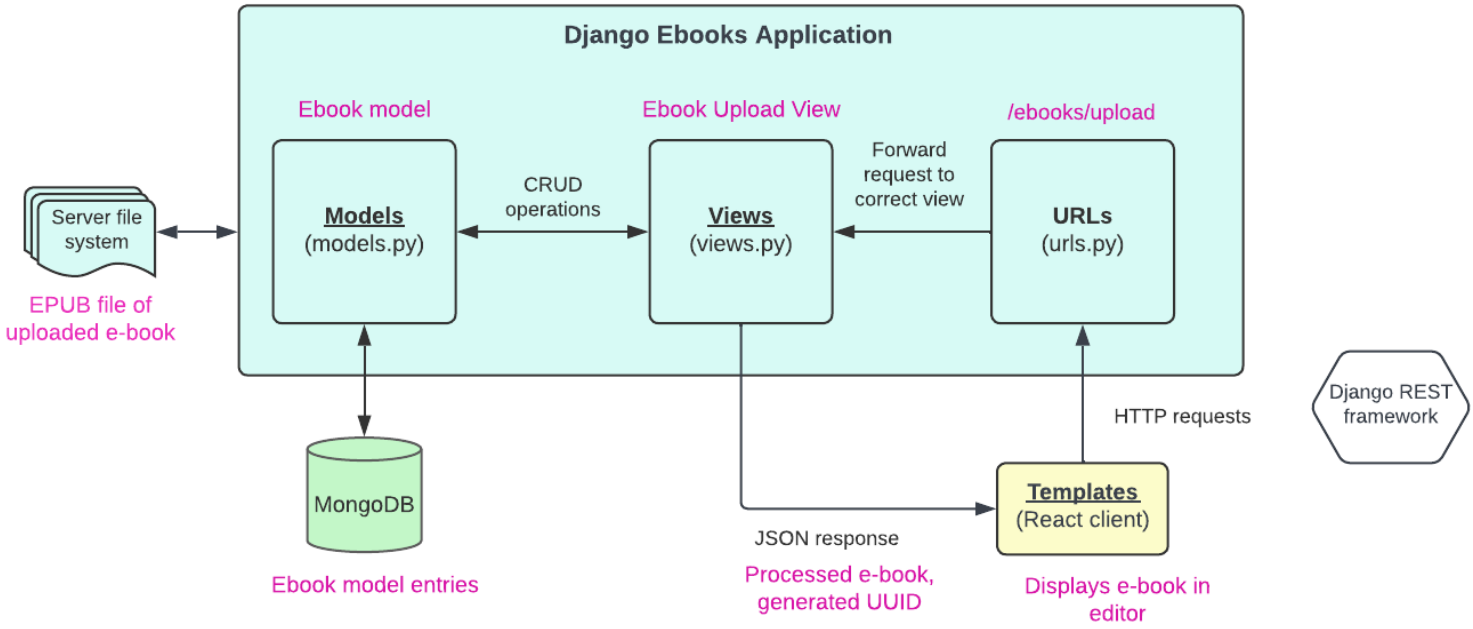


Figure 16: The MVT architecture for the Django *Ebooks* sub-application

Tools & Libraries

Below we outline Python tools and/or libraries used to implement our Django server.

- **epubcheck**: wrapper for the original EPUBCheck that is written in Java. This open-source tool - maintained by DAISY Consortium [32] on behalf of W3C [33] - validates EPUBs against EPUB specifications. The Django sub-application *Ebooks* uses this to check uploaded e-books, as part of its data processing pipeline.
- **BeautifulSoup** [34]: library for extracting information from HTML and/or XML files, used on the server-side for EPUB files. More specifically, it helps extracting the textual context surrounding an image and injecting annotations made by the user into the EPUB.

[31] As presented in the next section.
[32] `https://daisy.org/`
[33] `https://www.w3.org/`
[34] `https://beautiful-soup-4.readthedocs.io/en/latest/`

6.3 Data Storage System

This section will present our data storage architecture, used on the server-side to keep track of information related to e-books, images and their annotations. As stated earlier in 4.2.1, MongoDB allows for flexible modelling on the fly, meaning we did not have a *fixed* design before implementation. Instead, figure 17 below displays the final architecture that resulted from various modifications made, as our system requirements changed. It can be observed that the database will store entries for three types of document models: e-books, images and annotations. Below each of the three models will be elaborated on.

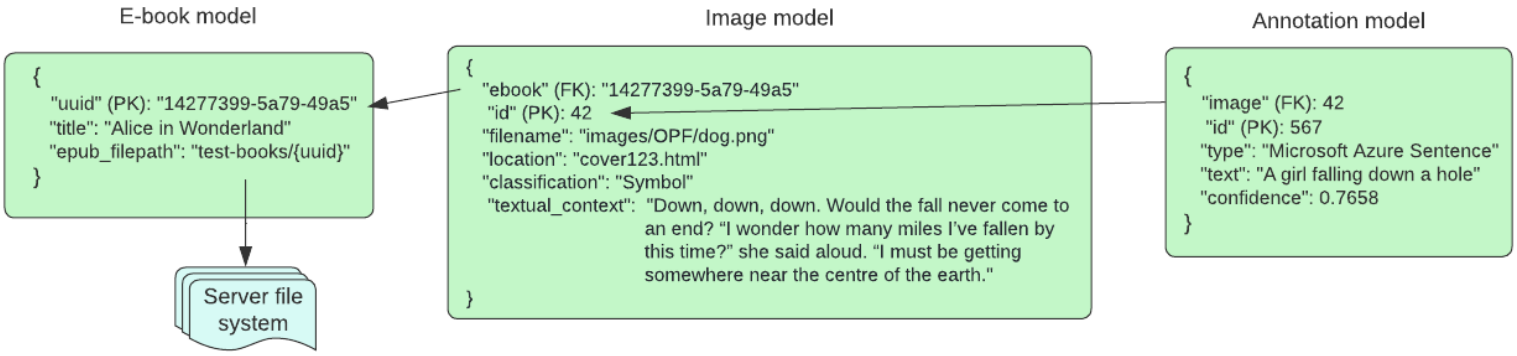


Figure 17: The server's database architecture of the image annotation system

The E-book model is our central database entity. Upon upload of an e-book by the user, the server will add a new e-book entry to its collection. This entry stores relevant information including a uniquely generated UUID for that e-book - serving as primary key (PK). Furthermore, the title of the e-book is stored to allow for simple referencing, as the UUID is not meaningful from a user-perspective. Finally, every e-book entry stores a *file path* to the *actual* EPUB file in the server's file system. Storing this whole file in the database would slow down query performance, since this implies more data is transmitted compared to simply storing a mere file path. Throughout the image annotation process, the server will simply communicate with its database to update information. Only during uploading and downloading of the e-book, the server will access its file system to store or retrieve the corresponding EPUB file.

Similarly, we have another database collection that stores information related to the images in the books - represented by the Image model. Each image entry is linked to the e-book it belongs to (1-to-N relation), as indicated by the foreign key (FK) "ebook". An image entry itself also has a primary key "id", as unique integer identifier for that image in our system. The database keeps track of properties of every image, such as the image filename, the name of the file that contains the image, and the image type. Furthermore, each image stores its textual context within the e-book, which will be fed to automated analysis tools to generate keywords in our system. As the system scales in the future however, this raw textual context can at any point be accessed to apply other algorithms on. The reason why we decided not to embed all Image entities within the related E-book document, is because that E-book document can become inconveniently long depending on the number of images inside that e-book.

Finally, our database also keeps track of image annotations - represented by the Annotation model. Each annotation has a foreign key (FK) "image" that points to the image it belongs to. Also, annotation entries have their own integer primary key (PK) to identify them uniquely. Our system will store different types of annotations, indicated by the "type" field. For example, "HUM" refers to a human annotation, whereas "MS_AZURE_LAB" refers to an automatically generated label retrieved from the Microsoft API. Automated suggestions generally come with metadata such as a confidence level, which we store in the "confidence" field. More metadata may be stored as the system incorporates other AI systems in the future. The textual value of the annotation is stored as well under the "text" field. In the case of automatically generated annotations, this "text" value could be either a label or a sentence.

6.4 External Technologies

This section aims to present external technologies that we integrated into our system, to provide users with automated image descriptions (6.4.1) and to deploy our application (6.4.2). These tools are different from those mentioned earlier in 6.1 and 6.2 in the sense that they are not specific to the client or server.

6.4.1 Technologies used for Automated Image Annotations

Our image annotation platform E-BOOK FIXER relies on two different image recognition engines for providing users with suggestions for *conceptual* image descriptions: Google Cloud Vision API [35] and Microsoft Azure Vision API [36]. The system allows users to select which AI engine to use, thereby providing additional information about each tool. On top of these, the external text analysis system Yake [37] is used to suggest *non-visual* image descriptions, in the form of keywords extracted from the text. Incorporation of these AI systems aims to save manual labour. Below we discuss and compare these technologies by feeding each of them the following image displayed in figure 18:

Figure 18: Image used for generating three types of AI suggestions in E-BOOK FIXER, as displayed in figures 19, 20 and 21

Google Cloud Vision API: Labels

Our system relies on Google Cloud Vision API for automatic image analysis and generation of labels associated to an image to be described. Using this API is convenient, as it comes with pre-trained machine learning models which can detect multiple entities in one image using "Object Localisation" - including products, humans and locations. Our the Django server is responsible for making API calls to Google Vision. This communication is authenticated using a private key, in the format of a locally stored JSON file. The result will contain the generated labels, as well as confidence scores for these labels. This result is displayed as word cloud in the editor of our GUI, which seemed the most intuitive option to us. As can be seen in figure 19, labels with higher confidence are scaled to a larger font size. It can be observed that the generated labels for this image are accurate, but not sufficient to replace human annotations - for which a high-quality description is required.

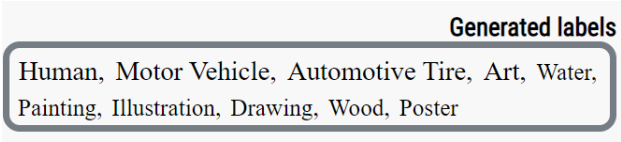


Figure 19: Labels generated by Google Cloud Vision API as displayed in the editor interface of E-BOOK FIXER, associated with the image in figure 18

[35] `https://cloud.google.com/vision`. In terms of monetary costs for using Google's API, the first 1000 units per month for Label Detection are free of charge. After that they charge $1.50 per 1000 units.
[36] `https://azure.microsoft.com/`. Microsoft Azure has a free tier with 5000 images per month.
[37] `https://github.com/LIAAD/yake`

Microsoft Azure Vision API: Labels and Description

The integration of the image recognition engine Microsoft Azure Vision API is similar to that one of Google Cloud Vision API, in the sense that it will perform label detection on images. Also, a private key was again used in to authenticate API calls from the server. However, the main difference between the two is that Microsoft's API returns a human-readable *description* in addition to labels associated to an image. As can be seen in figure 20, the labels are displayed as word cloud, with the generated description shown underneath it in a simple box. Since this description comes closer to an actual human image description, our system comes with the time-saving functionality to "Copy over" the automatic description into the user's editing field. However, it is noticeable that the labels generated by Microsoft Azure are not as descriptive as those generated by Google Vision - that is for this particular image. In fact, different image recognition tools may be suitable for different image types. However, in the scope of our project we do not distinguish between different image classifications, when it comes to which image recognition engine to use. Hence, we simply allow the user to make this choice.

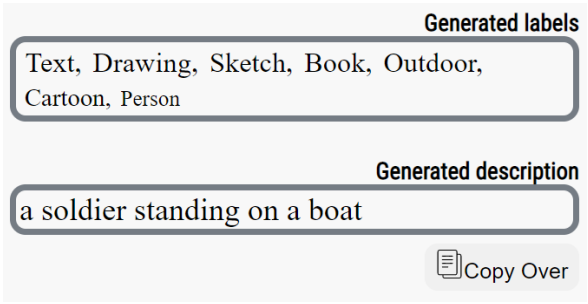


Figure 20: Labels and a description generated by Microsoft Azure Vision API as displayed in the editor interface of E-BOOK FIXER, associated with the image in figure 18

Yake: Context Keywords

The suggestions generated by Google Vision API and Microsoft Azure API (of which *examples* are shown in figures 19 and 20) are generally not descriptive enough to help users create meaningful image descriptions. Hence, our system aims to assist users even more by integrating the automatic text analysis tool Yake (Yet Another Keyword Extractor). Yake is a keyword extractor that uses unsupervised learning by considering statistical features in a given text to extract the associated keywords. Comparable to the two image recognition engines described above, the keywords/key phrases generated by Yake come with a confidence level - proportional to their frequencies within a document. As shown in figure 21, these results are also displayed as a word cloud. Additionally, our system allows users to click on any word or phrase, after which it gets highlighted within the e-book. This helps users quickly trace back where that keyword appeared, thereby possibly analysing the text further to create a meaningful description.

Moreover, it can be noted that the resulting context keywords below can be useful in helping users make image descriptions more detailed, since most of them cannot be inferred from the image itself (especially the proper nouns) - in contrast to the results obtained by Google Vision and Microsoft Azure [38]. Instead, humans would have to read the text surrounding an image themselves, which is labour-intensive. Whereas the input to Google API and Microsoft API was simply the image under annotation, Yake expects a text fragment as input. Selecting this "textual context" of an image was not trivial however. This is because images may be referenced anywhere in the e-book, or nowhere at all. In the scope of our project, we have applied a naive approach: the server extracts the four paragraphs above and below the image to feed to Yake.

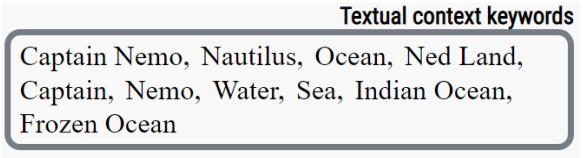


Figure 21: Keywords extracted by Yake as displayed in the editor interface of E-BOOK FIXER, associated with the textual context of the image in figure 18

[38] Whilst experimenting with various images, we have observed this was the case generally.

6.4.2 Technologies used for Deployment

Below, we elaborate on the various technologies we have used to deploy (make available to the public) our web application E-BOOK FIXER: the Docker platform [39] and the external services Netlify [40] and DigitalOcean [41].

During development and deployment it is crucial to store confidential information, such as secret keys, safely. This data should not be added to the version control system or be visible anywhere in the codebase. To this end, all three deployment platforms mentioned below support environment variables, which allow for those keys to be safely used by our services.

Docker

Docker can be used to create a virtual container that stores all of the code and technologies, such as programming languages, external libraries and frameworks, used for the software. This makes the whole process of building, running and sharing an application much more efficient and effortless.

Netlify

Netlify is a web platform used for hosting websites. We chose it mainly because it was easy to set up - all we needed to do to host our own website was to link our GitHub repository that contained all the client-side code.

DigitalOcean

DigitalOcean is a cloud computing vendor that offers an Infrastructure as a Service (IaaS) platform for software developers. We decided to host our server and database here because their platform provides one of the lowest cost services for small scale applications.

[39] `https://www.docker.com/`
[40] `https://www.netlify.com/`
[41] `https://www.digitalocean.com/`

7 User Evaluation of Prototype

After having implemented a prototype for our interactive image annotation platform E-BOOK FIXER, we conducted a user evaluation on the GUI of our system, which this chapter is about. The aim of our evaluation was to assess how accessible and user-friendly the GUI is with different stakeholders and discover what potential modifications that could be made in order to resolve usability issues. Section 7.1 breaks down on our method of evaluation, including the evaluators and procedure. The results that came out of the evaluation - combined with our GUI improvements made - are presented in 7.2.

7.1 Method of Evaluation

During the process of conducting our evaluation, we strive to catch usability issues as early as possible, to keep the costs to resolve potential issues low. We carried out this evaluation after our fourth sprint (in week 4.6). At this point, we agreed our prototype was decent enough to be manually tested and thereby still giving us three more sprints to analyse the results and improve our GUI accordingly.

Evaluators

In our prototype evaluation, we tried to get input from various stakeholders [42], since they all have their own wishes and values that our end product should take into account. On the other hand, we decided to limit the number of evaluators, as it has been proven that increasing the number of participants eventually leads to costs taking over benefits [10]. As can be seen in Table 1 below, we managed to get a professional in image description evaluate our system (Hans Fieggen) - a potential end user of our system. Moreover, two of our evaluators are experts in accessibility of digital publications - the core of our project. Finally, we have asked people who are not familiar with (professional) image annotation to participate. Their input is valuable, as we expect inexperienced volunteers to be the largest group of end users of the platform. Furthermore, having both our client and TU coach participate was important to us, as they have been involved throughout our development process.

Table 1: Demographic information about participants in our user evaluation

Name	**Profession**
Ted van der Togt (client)	Researcher on digital publications and accessibility
Alexandru Bobe (TA)	Technical Assistant at TU Delft
Hans Fieggen	Audiobook narrator (including image description)
Dr. Christoph Lofi (TU coach)	Researcher on Semantics-based Data Engineering and Assistant Professor at TU Delft
Anonymous	(Accessible) digital publishing consultant

Procedure

Our user evaluation was carried as online questionnaire, starting with demographic questions about the evaluators themselves. This is followed by the main "hands-on" part of the evaluation, in which our participants get to evaluate two components of the website:

1. **The Home View**, where user can upload their e-books;
2. **The Editor View**, where users can annotate images.

For each component, we ask the evaluator to interact with the system. Afterwards, the user is provided with a set of statements related to that component, for which they have to indicate to what extend they (dis)agree with it. We created the statements such that they align with web accessibility standards as outlined by W3C [43]. These statements are followed by an open field, which can be used to report on specific issues encountered. Under Appendix C and D respectively, one can find the instructions and survey form that we sent to all evaluators. As two of our participants did not find the timely resources to complete our survey, we carried out a verbal evaluation with them.

[42] See also section 3.1, which lays out the different stakeholders of our project.
[43] See non-functional requirement 3 in section 3.2.

7.2 Evaluation Results and Improvements

This section presents the results to our user evaluation, which proved to be useful in polishing our user interface. This was due to the fact that people having different backgrounds were collectively able to detect various usability issues. Moreover, this section presents the conclusions we have drawn from our results - as well as the corresponding modifications made to our Home View and Editor View.

7.2.1 The Home View

From the results to the set of statements related to our Home View, shown in figure 22 below, it can be seen that no major usability issues were detected by the three evaluators who completed the survey. In this bar chart, the relative frequencies at which each answering options was chosen is displayed.

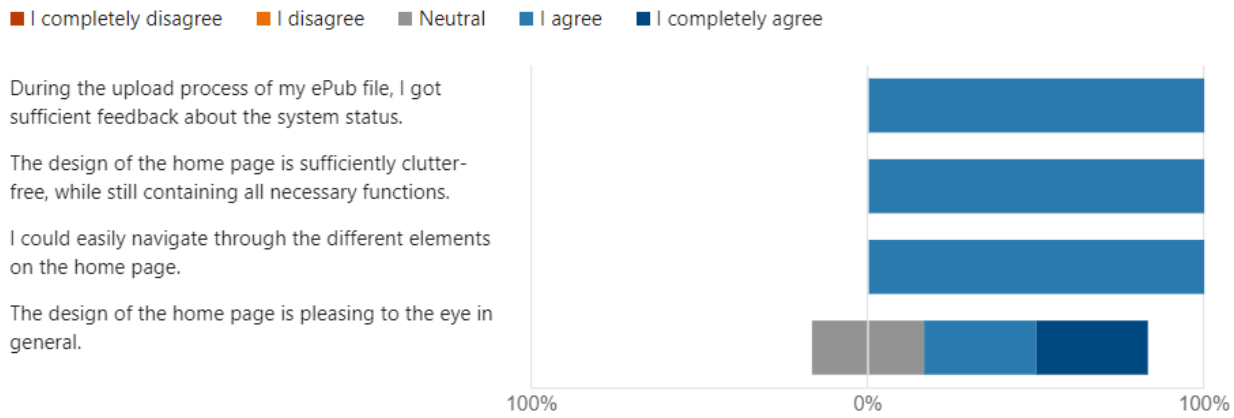


Figure 22: Bar chart displaying the results to the set of statements related to our Home View

Nevertheless, the results to the open question that followed this set of statements showed that several GUI issues were encountered with regards to uploading e-books on the Home Page. Based on these, we have made improvements accordingly - as elaborated on below.

Uploading of E-books

One of our evaluators experienced the following issue: "I downloaded three different EPUB files from Gutenberg.org, all three couldn't be validated by the system". To explain the latter, our system uses EPUBCheck to determine the validity of uploaded EPUB files [44], thereby rejecting the file as soon as errors are encountered. Based on this piece of feedback, we have realised that it suffices to merely display a warning to users, but still allow them to edit their e-books.

Furthermore, another evaluator pointed out that "uploading of e-books feels unintuitive", due to the fact that users had to press an "Upload" button before their e-book is processed. We agreed that this hinders efficiency of the platform, and thus have changed this behaviour to start processing the e-book as soon as users have selected their file. Removal of the "Upload" button is depicted in figures 23 and 24, showing the interface that our users had to evaluate and the improved version respectively.

[44] See Could Have requirement 1 in section 3.2.

Figure 23: Upload component on Home View interface presented to our evaluators

Figure 24: Upload component on Home View interface after improvements made based on evaluation results

7.2.2 The Editor View

Similar to the results related to the statements about the Home View, those related to the Editor View did not indicate severe issues [45] - as can be seen in figure 25. However, it can be observed that overall navigation throughout the Editor View was not experienced as user-friendly. Since this was confirmed by the results to our verbal evaluations, we have made several modifications to navigation buttons our Editor View - which are described below. On the other hand, from our verbal evaluations it also became evident that the visual progress bar on top of the editor interface is a user-friendly way of indicating the current system status.

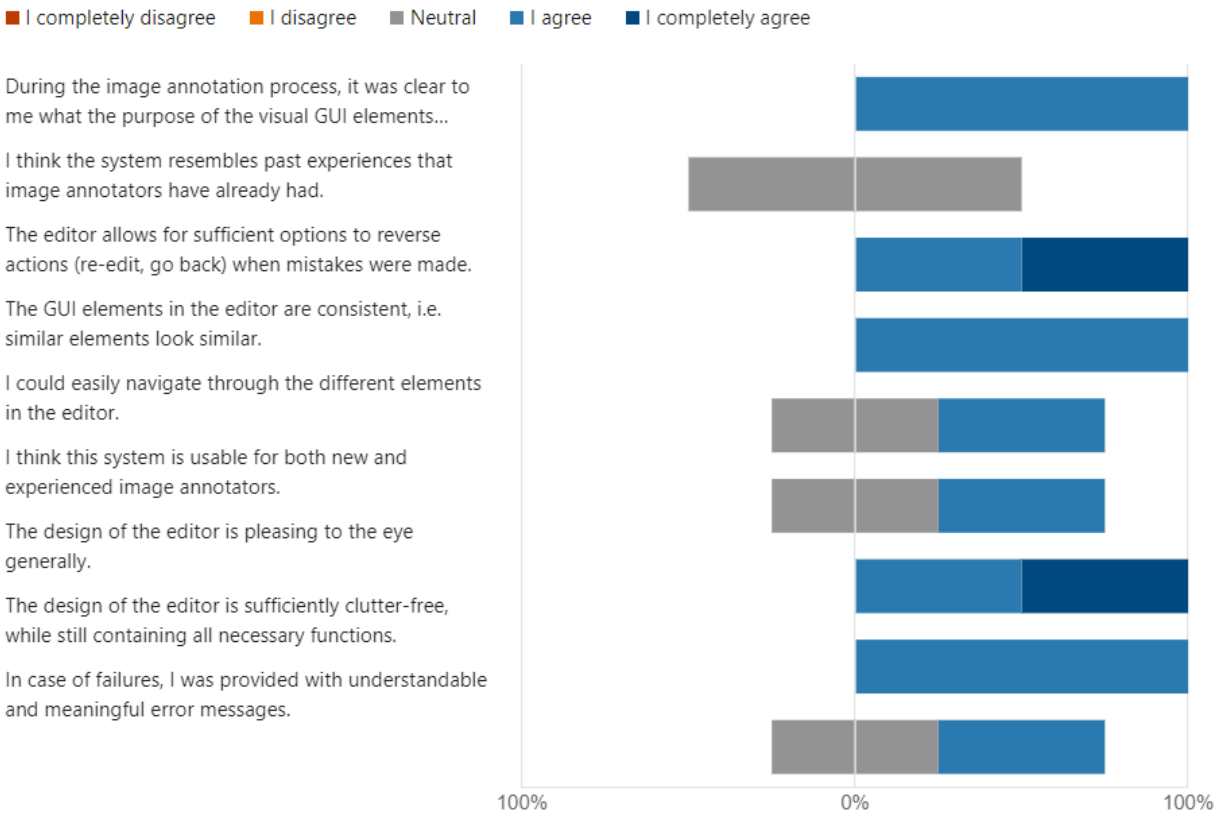


Figure 25: Bar chart displaying the results to the set of statements related to our Editor View

Navigation Buttons

An important usability issue was that the transition from annotating one image to another one was not straightforward. To illustrate this, throughout the process of image annotation users have the option to press a "Save" button in the edit panel *right of the e-book viewer* (see figures 26 and 27 below), automatically leading them to the next stage. When they arrive at the final stage however, there was no such button to continue to the next image. Instead, they had to press the "Next image" button *on top of the e-book viewer* - which is not intuitive (to new users). This prompted us to add a "Continue next image" button to the final step of the annotation process, in the edit panel on the right (see figure 27).

Furthermore, the number of buttons in the editor was described as a problem causing visual overload. This prompted us to implement a sidebar that contained infrequently used functionality such as a download button, which can be seen in figure 27 on the left. Moreover, the "Previous image" and "Next image" buttons were replaced with arrows - and the image number, making their functions more recognizable and the view less cluttered.

[45] These results are based on two evaluators only, as one of the evaluators was not able to upload his e-books

Finally, several evaluators mentioned that having images highlighted for ten seconds may not be sufficient to clearly indicate which image is under annotation. Hence, this has been changed to keep images highlighted until the next image visited.

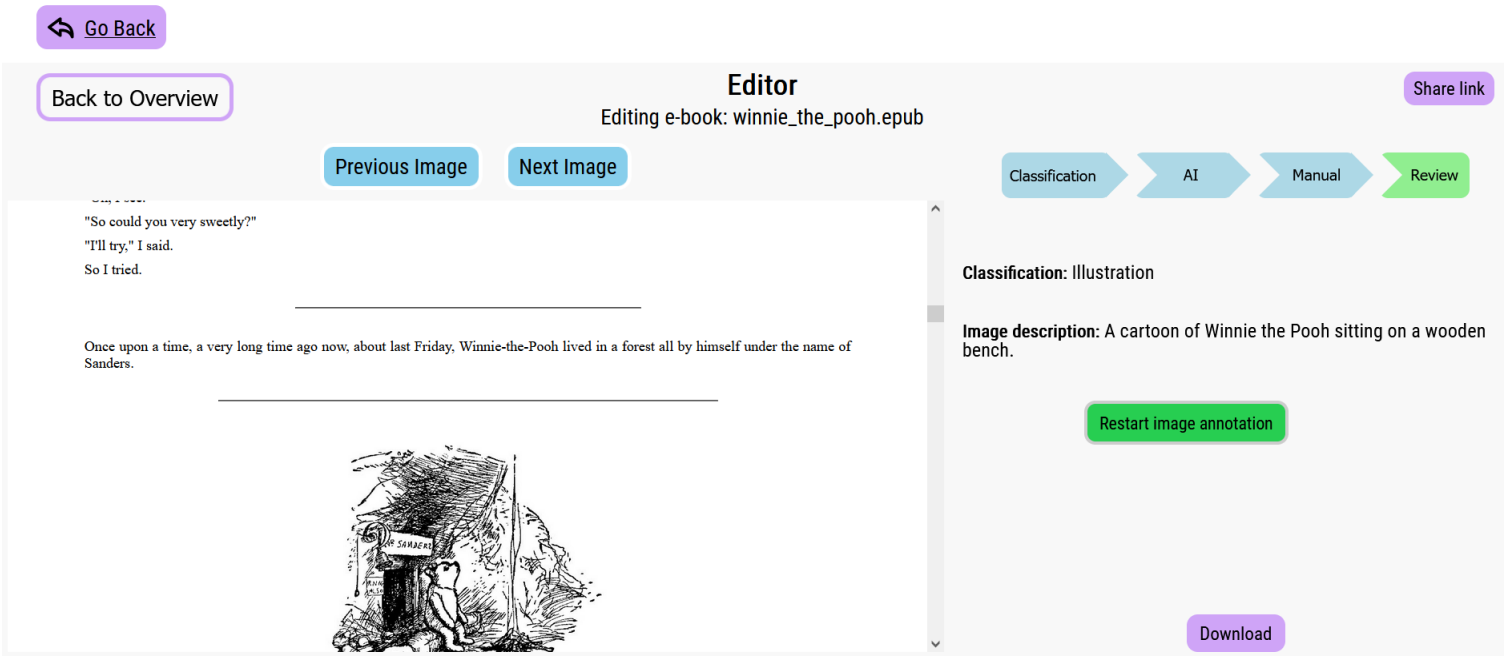


Figure 26: Editor View interface presented to our evaluators

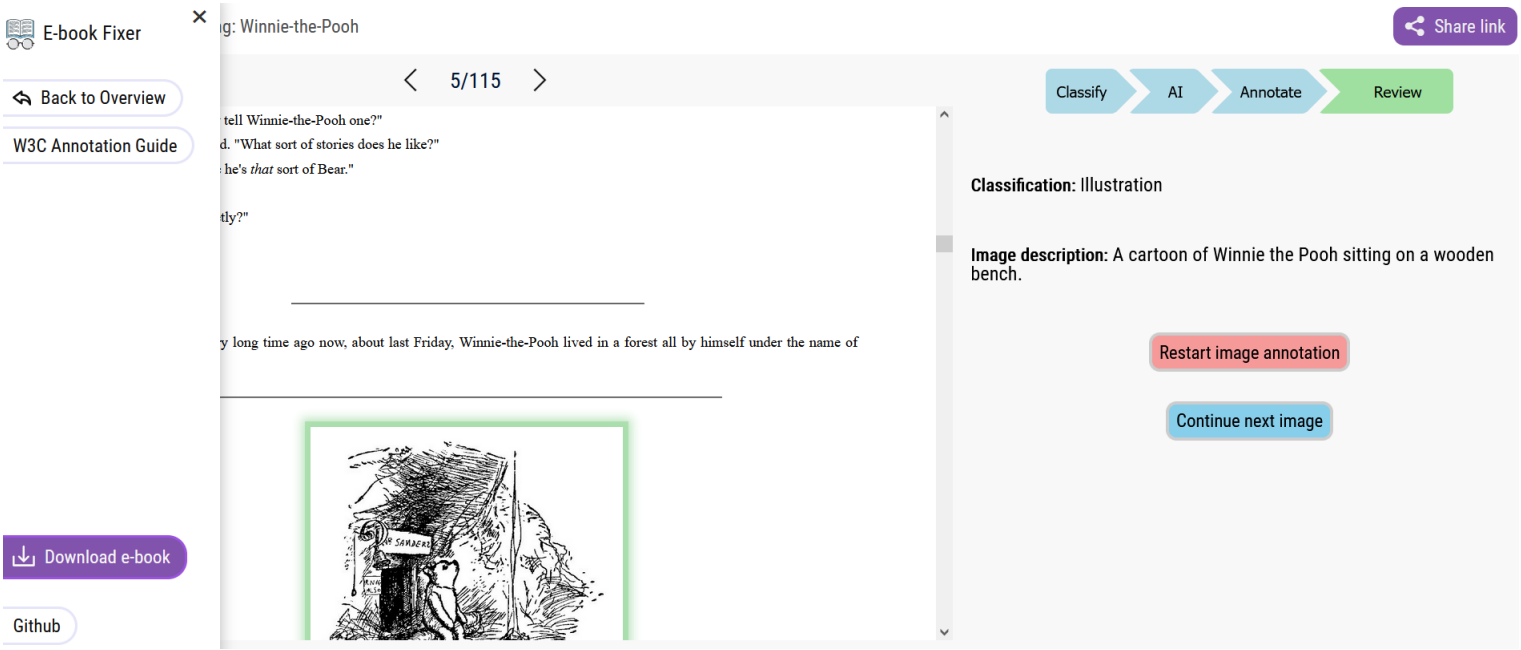


Figure 27: Editor View interface after improvements made based on evaluation results

Image Overview

During a verbal evaluation with one of our participants, it was pointed out that the image overview in our Editor Page can be improved by attaching labels to images ("Annotated", "Decorative", "Existing alt-text" etc.). We agreed that this can help users get a clear representation of the state of all images inside an e-book, allowing them to quickly estimate how much time may be required to remediate their e-book for instance. Figures 28 and 29 below display the improvements made.

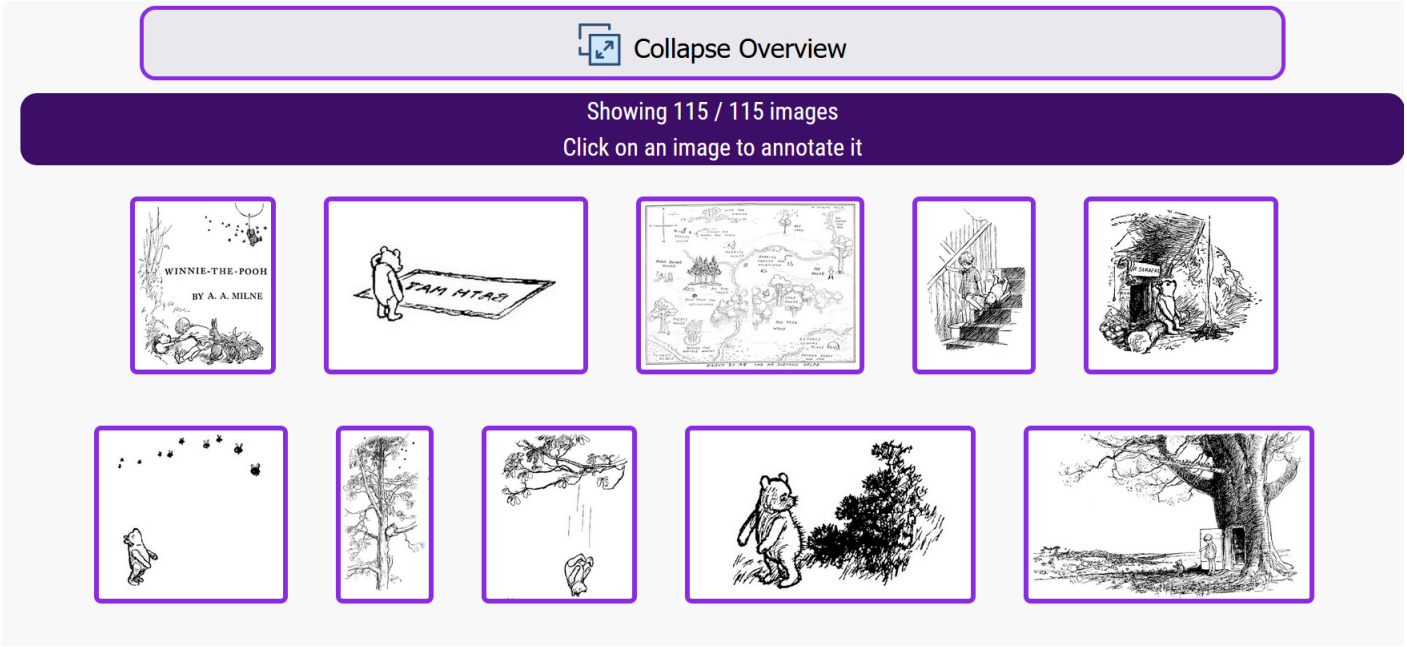


Figure 28: Image overview component in Editor View interface presented to our evaluators

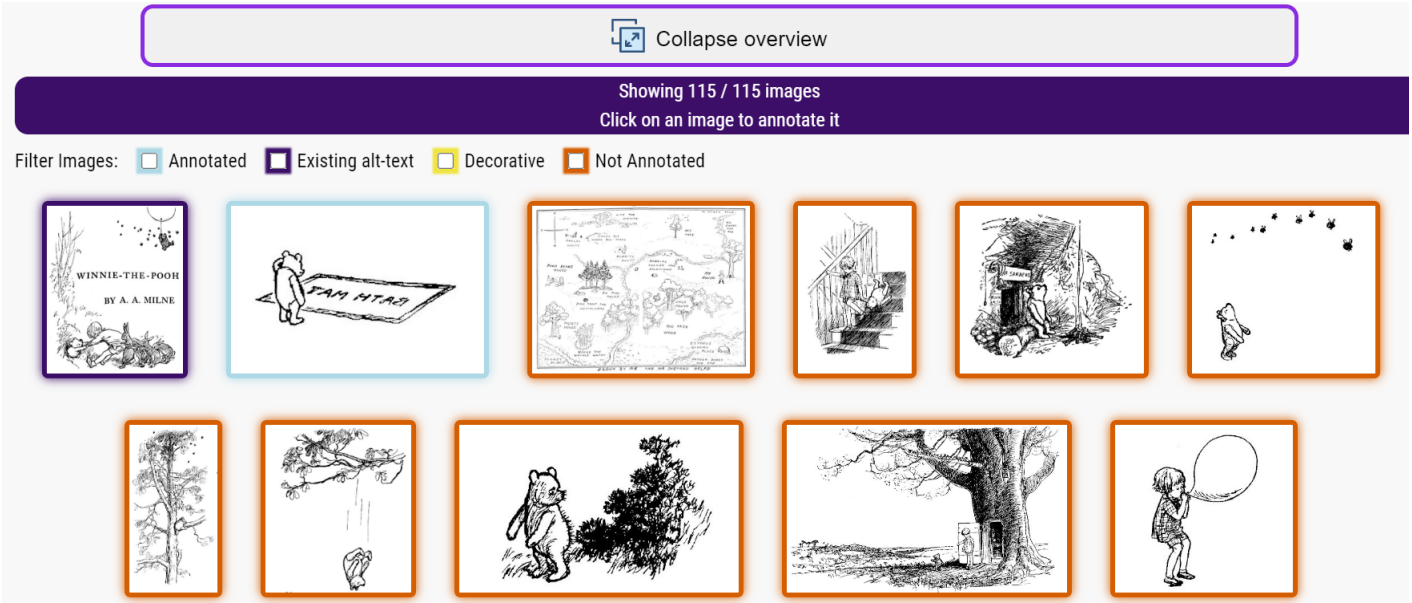


Figure 29: Image overview component in Editor View interface after improvements made based on evaluation results

8 Product Discussion and Future Recommendations

This chapter aims to explain how we look back on our final product, as well as what system extensions we envision for the future. Section 8.1 lays out the most important points that came out of our reflection on the image annotation platform we have implemented. Afterwards, section 8.2 motivates various future recommendations that can help in making e-books even more accessible in the future.

8.1 Reflection on Final Product

Below we present the results to the reflection on the final product (and process towards it) we had with our client Ted van der Togt, as well as the reflection we had as a developers team.

Reflection by Client

After the implementation of our demonstrator was completed after our seventh sprint (week 4.8), a final meeting was conducted with our client Ted van der Togt to reflect on the resulting image annotation platform. Overall, he was pleasantly satisfied with our final product, including all features we managed to implement within the time frame of ten weeks. In his opinion the requirements laid out at the beginning of the project have been satisfied. Moreover, he experienced the entire working process with us as positive. In particular, he was content about the communication between us in the form of weekly meetings. During these meetings, we were able to update on our progress and have transparent discussions about certain system requirements - thereby possibly modifying these.

All in all, our client Ted is hopeful for the future and confident in continuing the project further so that additional functionality can be incorporated and it can progress to a pilot project. Finally, he is curious about any additional ideas that we may have with regards to making e-books even more accessible.

Reflection by Development Team

We as a team are convinced that we have been able to fulfil the project goals as presented in section 2.3. Specifically, we all agree we have been able to build a user-friendly system that can help in making the process of image annotation more efficient than existing solutions have been able to do so far. In the end, we have completed all of the Must Have and Should Have requirements as specified in 3.2.1. As for our Could Haves, we have managed to implement five out of the total twelve.

So far, we have implemented a data processing pipeline that validates EPUBs (Could Have 1) and adds HTML language tags to EPUBs (Should Have 4) as additional accessibility feature. In the future, this pipeline can be conveniently extended with further improvements to make e-books accessible. Examples are coverting EPUB2s to EPUB3s (Could Have 2), making EPUBs automatically accessible (Could Have 3) or checking annotated EPUBs against accessibility specifications (Could Have 4). Whereas we started implementing these three Could Haves, this did not succeed for reasons that will be elaborated on in section 8.2.

Furthermore, applying heuristics for estimating the purpose of an image (Could Have 9) required research that was not feasible within our time frame. Instead, our system relies on manual image classification. In fact, this may be even more accurate than what current image classification technologies such as Google's AutoML have been able to produce. The downside is that expensive time is required for manual classification however. Finally, Could Haves 10, 11 and 12 are related to user management, which we considered outside the scope of our project.

All in all, we are satisfied with our final image annotation platform and the development process. Although we did not complete all Could Have requirements, these were not necessary in the scope of our project. They can greatly improve user experience however. Hence, we have built our platform in such a way that additional features can easily be integrated. Our choices for Django and React turned out to be convenient in building coherent pieces of code to keep the system maintainable. Moreover, communication between the Django server and React client is effortless. As for quality control, we managed to achieve 98% coverage on the backend software, whereas we initially aimed for 80%. Additionally, the prototype evaluation as discussed in chapter 7, as well as discussions with our coach and client have proven to us that E-BOOK FIXER is indeed helpful for facilitating the process of image description.

8.2 Future System Recommendations

This section presents several future recommendations for improving the image annotation platform E-BOOK FIXER. We are highly convinced that these system extensions can eventually lead to higher accessibility of e-books. Moreover, we believe these recommendations are feasible to implement by potential new developers that may build upon this open-source system. A subset of these recommendations are based on suggestions made by our client Ted van der Togt and TU coach Dr. Christoph Lofi.

Collaborative Image Annotation

In the future, we think the platform should support collaboration between users, which will greatly speed up the process of classifying and annotating images in a single e-book. Furthermore, it will reduce the amount of data that we need to store on the server and its database. Currently, the system already allows users to share links to their e-book or image. To elaborate, below is shown a list of specific potential extensions to enhance user collaboration:

- **Shared e-books**
  When a user uploads an e-book that already exists in the system [46], they could get the option to continue working on the existing book to reduce the amount of work, as it might already contain annotations created by other users.
- **Display History of User Annotations**
  By giving the user access to annotations created earlier by himself or other users, the process of manual image annotation can be facilitated, This feature can be implemented by attaching a user ID and timestamp to each annotation, allowing the system can keep track of when which annotation was made by which user.

Extended Image Classification

Our system allows users to classify images manually into a certain category defining its purpose. However, we currently only distinguish between decorative and non-decorative images in the sense that decorative images should not be annotated. In the future, this image classification could be used more efficiently to facilitate the process of image annotation. To this end, we have two suggestions:

- **Automatic Image Classification**
  The system could incorporate AI systems that can classify the purpose of an image given its representation and/or its surrounding context. An example of an existing image classifier is Google's AutoML, which was also used in a pilot project by Fondazione LIA as mentioned in 2.4.2. This is going to eliminate the cumbersome job of having the user manually classify each image separately.
- **Service by Image Classification**
  Depending on the classification given to an image, the process of image annotation might include different steps. As illustration, different AI image captioning systems are generally suitable for different types of images. Furthermore, the platform could provide users with templates and/or guidelines to follow, for annotating a specific type of image. This allows for consistency, thereby also guiding new users using the system.

Extended Data Processing Pipeline

Besides improving image annotations, there are more steps that can be performed on the e-book file uploaded by the user - to allow for better accessibility of the e-book. Currently, our system already has a data processing pipeline that validates EPUBs upon upload (against EPUB specifications). The following steps could be added to this pipeline in the future [47]:

- **Convert any uploaded EPUB2 file to EPUB3**
  As mentioned earlier in section 2.2, the newer EPUB3 format serves as a better foundation for accessibility with an improved structure among other features. To this end, we have already attempted to integrate the existing Sigil plugin EPUB3-itizer [48] as a command-line tool, which was not trivial as it does not come with packages that could be easily integrated into our Python server. Our suggestion for future developers would be to find another existing tool to do this job, or to write their own code to convert EPUB2s to EPUB3s.

[46] Duplicate e-books can be identified by comparing ids of e-books, found in the *package.opf* file of the corresponding EPUB files.
[47] These are Could Have requirements 2, 3 and 4 as specified in section 3.2.
[48] `https://github.com/kevinhendricks/ePub3-itizer`

- **Making EPUB File Accessible in Automated Fashion**
  Making an EPUB file "accessible" involves various steps that essentially come down to making structural changes to the file. Note that annotation of images is a step that involves human intervention, which is the core of our system. As for the steps that can be performed automatically, we would recommend trying to use the existing Sigil plugin Access-Aide [49], or a similar tool. Access-Aide is compatible with EPUB3 files only however, meaning the previous recommendation is required.

- **Check Accessibility of Resulting e-book**
  After the user is done annotating images in the uploaded e-book, the resulting accessibility of that e-book could be evaluated against EPUB Accessibility Specifications [50] using an automated, open-source tool such as Ace by DAISY [51]. Again, this tool is compatible with EPUB3 files only.

Additional e-book Accessibility Functionality

- **Improved Extraction of Textual Context**
  The unique aspect about our system is that it uses the text analysis tool Yake to generate keywords, by analysing the surrounding text of an image. However, extracting this "textual context" accurately requires more research than we had time for. Hence, our current system relies on a naive approach of selecting the four paragraphs preceding and following the image under annotation. In the future, this extraction can be improved by incorporating heuristics that analyse the relatedness across words. Word2Vec is an existing model that considers concepts including synonym detection, concept categorization and analogy. Moreover, this tool can be used in combination with Python's GenSim library, allowing for simple integration into our current system.

- **Usage of HTML longdesc attribute**
  Our system currently stores all human image annotations in the "alt attribute" of that image, within the corresponding HTML file. However, HTML also allows for the optional "longdesc attribute", which can hold a URL to an external resource containing more detailed information. This could be used by the system (in combination with the "alt attribute"), to store detailed descriptions of infographics for instance.

- **Support for Non-commercial e-books**
  We believe it is important that this system can help in making plenty of non-commercial e-books accessible in the future, so that everyone can enjoy learning. Since this involves an enormous number of books, efficiency plays a major role. Thus, we think the system may benefit from a dedicated workflow for free open-access educational e-books. In an even further future, the system could use automatic translation of image annotations (with a human-in-the-loop for quality assurance) to bring educational books to even more people.

Tolerance for Legal Implications

As the system may grow into a real pilot, there may be users that try to run the system on copyrighted e-books. Sharing a link to access that e-book in our system and letting others access it could count as illegal distribution of copyrighted material. Furthermore, users may abuse AI systems for doing image description on huge sets of their own images which would cost money for anyone hosting the product. We believe it is crucial that these legal matters are dealt with by a future implementation system (whether it is for-profit or not), despite the fact that these legal issues are not directly related to accessibility of e-books.

- **Closed Alpha Release**
  A closed alpha release would be highly advantageous to anyone making a fully-fledged product from our system, as that way only trusted individuals have access to the tools. When moving to an open beta release, it would be wise to make AI usage a paid feature, or implement guards against high usage of the third-party AI tools it provides.

- **Partnership with Publishers**
  Another solution is partnering or releasing the product with a major publisher (or online library) of EPUBs. This would grant safe and easy access to a large collection of EPUBs, thereby making the upload functionality not necessary anymore as users can just pick a book to annotate from the collection.

That said, it was our mission to help make e-books more accessible for anyone, regardless of their wealth or connections to companies. Our dream scenario is keeping this interactive image annotation platform non-commercial, allowing it to be used by volunteering image annotators.

[49] `https://github.com/kevinhendricks/Access-Aide` or command-line version `https://github.com/JingMatrix/sigil-cli`
[50] https://idpf.org/EPUB/a11y/
[51] `https://daisy.org/activities/software/ace/`

Conclusion

The aim of this report was to document the development of a demonstrator for an interactive image annotation platform ("E-BOOK FIXER"), which combines automated and manual image descriptions to make the process of image annotation efficient. The relevance of our system can be understood from the fact that creating meaningful image descriptions - to make e-books accessible to visually impaired users - is a challenging and time-consuming process for humans.

An important requirement was allowing users to classify their image before annotation starts. By having this feature, unnecessary time and resources are saved, since images labelled as "decoration" do not need to be described. Moreover, our system can display to the user which images inside an e-book are decorative, as well as which have been annotated already. This feature helps users well when it comes to estimating time needed to annotate images inside their e-book.

Following image classification, it was required that our system can generate automatic suggestions for *conceptual* image descriptions - by analysing the image. To this end, we have incorporated two state-of-the-art image recognition systems. Google Cloud Vision API - which generates labels - and Microsoft Azure Computer Vision API - which can detect both labels and descriptions. As these descriptions come close to human annotations, our system can paste this description to the manual editing field for convenience.

Nevertheless, we have observed that none of theses AI suggestions are sufficiently descriptive to replace human annotations yet. A major reason for this is that humans can analyse the surrounding text fragments of an image. Since the latter is a laborious job however, we identified another related requirement: support for automated text analysis systems to generate suggestions for *non-visual* image descriptions. To fulfil this, have integrated the keyword extractor Yake. It turned out that selecting the text fragment belonging to an image is not straightforward. Thus, our system simply considers the four paragraphs that precede and follow an image as "textual context". Despite this naive solution, it became clear that Yake still provides the user with non-trivial keyword suggestions. Hence, we can conclude that incorporating this text analysis tool is a useful addition on top of image recognition engines. Moreover, our system allows users to click on a keyword suggestion, after which this word/phrase is highlighted in the text. This adds to the accessibility of our image annotation platform, as users can quickly verify where that keyword appeared in the text - after which they may perform further analysis of the text.

The suggestions generated by Yake cannot be used as final image description however, as they are mere keywords. Thus, it was also required for our system to allow for manual image annotations. Integrating this into our system, the quality of image descriptions is preserved, while at the same time saving time and labour. It was important that this editor is intuitive to use for different future users. Therefore, the system has an e-book viewer that allows users to inspect all pages. Quick navigation between images only is also possible. Moreover, the addition of a progress bar inside the editor is a user-friendly way to indicate which step of the image annotation process the user is in, as concluded from the results to our user evaluation conducted on a prototype of our system. Another feature that we have built into our system to ease the process of image annotation is letting users download their updated e-book file. However, our system comes with an even more convenient alternative to continue progress: by providing users with a URL to their e-book (or a specific image). Most importantly however, this URL can be shared with other users to collaboratively annotate images, thereby increasing efficiency.

As the system will become open-source, we have various recommendations for future extensions that can make image annotation even more efficient [52]. Firstly, the system could make more clever use of image classifications - by having separate workflows for certain image types. This may include using a specific set of AI systems that perform particularly well for that image type. Providing users with guidelines for describing a particular type is a possibility as well. A second recommendation would be to improve on selecting textual context of an image, by using heuristics that can analyse the relatedness across words. Finally, collaborative image annotation could be extended by incorporating a user management system. This way, annotators can improve upon image descriptions made by others and changes can be traced back to specific users. We are highly convinced that the above recommendations shall help in increasing efficiency of image annotation, thereby contributing to remediating inaccessible e-books for visually impaired users.

[52] Section 8.2 presented a full list of recommendations already, but here we point out the most crucial ones that are directly related to image annotation.

References

[1] Blindness and vision impairment. (2021, October 14). WHO.
Retrieved May 5, 2022, from *https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment.*

[2] Act, E. A. (2019) "European Directive to Improve the Accessibility of Mainstream Ebooks", pp. 1-3

[3] Wang, L et al., 2020, "Novel Object captioning surpasses human performance on benchmarks"

[4] A. Petrovai, A. D. Costea and S. Nedevschi, "Semi-automatic image annotation of street scenes," 2017 IEEE Intelligent Vehicles Symposium (IV), 2017, pp. 448-455, doi: 10.1109/IVS.2017.7995759.

[5] Hodosh, M., Young, P. and Hockenmaier, J. (2015) 'Framing image description as a ranking task: Data, models and evaluation metrics', IJCAI International Joint Conference on Artificial Intelligence, 2015-Janua, pp. 4188–4192.

[6] Petrie, H. et al. (1999) 'Describing images on the Web : a survey of current practice and prospects for the future Centre for Human Computer Interaction Design City University London Northampton Square 2 The importance of describing images on the Web'.

[7] Campus, G., (2019) Digital Single Market and Cloud Services: the legal implications of the Cloud on copyright laws and CSPs' liability [online] Kluwer Copyright Blog. Available at: http://copyrightblog.kluweriplaw.com/2019/03/14/digital-single-market-and-cloud-services-the-legal-implications-of-the-cloud-on-copyright-laws-and-csps-liability [Accessed 11 May 2022].

[8] E. Enge (2019), "Image Recognition Accuracy Study", Perficient.com. [Online]. Available: https://www.perficient.com/insights/research-hub/image-recognition-accuracy-study [Accessed: 22- Apr- 2022].

[9] Schwaber, K. (1997). SCRUM Development Process. In: Sutherland, J., Casanave, C., Miller, J., Patel, P., Hollowell, G. (eds) Business Object Design and Implementation. Springer, London. https://doi.org/10.1007/978-1-4471-0947-1_11

[10] Nielsen, J., Landauer, T.K. (1993). A mathematical model of the finding of usability problems. Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems.

[11] Nielsen, J. (1994, April). Enhancing the explanatory power of usability heuristics. In Proceedings of the SIGCHI conference on Human Factors in Computing Systems (pp. 152-158).

Appendix A: Frontend Manual Testing Plan

Below is displayed our manual test plan used to verify the behaviour of the GUI of our web-based image annotation platform E-BOOK FIXER - as referred to earlier in section 5.1. Since software quality assurance for our client system (frontend) mainly relies on manual testing, it is crucial that we provide sufficient tools to allow for these tests. Hence, this plan is made to help people outside of our development team to test our user interface manually.

E-BOOK FIXER - Frontend Manual Testing Plan

This test plan describes how our web application can be tested manually, with the main focus on our frontend GUI interface.
The following will be covered:

- Navigating through the **Home page** and uploading e-books
- Navigating through the **Editor page** and annotating images
- General usability of website

Functional Testing Checklist - Home page

- ☐ Does the interface give status updates while your uploaded e-book is being processed?
- ☐ Does uploading a file type other than .epub give a proper warning?
- ☐ Does uploading an invalid EPUB file give a prompt for downloading the error report?
- ☐ Will uploading a valid EPUB file automatically redirect you to the editor?
- ☐ Does pressing the “Menu” button show a sidebar containing buttons such as “W3C Annotation Guide”?

Functional Testing Checklist - Editor page

- ☐ Is the correct e-book title rendered on top of the editor?
- ☐ Are there issues with expanding and collapsing the overview of images?
- ☐ Does the overview display all images in your e-book?
- ☐ Does the overview indicate the status of each image using colours? (‘Decorative’, ‘Existing alt-text’, ‘Annotated’, ‘Not Annotated’)
- ☐ Does the next/previous image button work as intended?
- ☐ Is the progress bar (four arrows) on top highlighting the correct step you are currently in?
- ☐ Does the selected image get highlighted?
- ☐ In the “Classify” step, is the dropdown menu working as intended?
- ☐ When selecting the classification ‘Decoration’, do you get a message saying that this image should not be annotated?
- ☐ After classifying an image, are you redirected to the next “AI” step?
- ☐ After classifying an image, are all arrows in the progress bar clickable, and leading you to the correct stage?
- ☐ In the “AI” step, does pressing the information icon display textual information about each AI choice in the dropdown menu?
- ☐ After having selected an AI engine, are you automatically redirected to the next “Annotate” step?

- [ ] When selecting Google Vision, are there issues with displaying the labels for that image as word cloud in a text box labelled as “Generated labels”?
- [ ] When selecting Microsoft Azure, are there issues with displaying the labels for that image as a word cloud in a text box labelled as “Generated description”?
- [ ] When selecting Microsoft Azure, are there issues with displaying the generated description in its text box?
- [ ] When selecting Microsoft Azure, does pressing the “Copy over” button underneath the generated description indeed paste that description into the user box?
- [ ] For images with existing ALT-text, is that annotation visible in the editor?
- [ ] After having saved a manual annotation, and pressing the “Review” arrow, do you see your annotation and classification chosen earlier?
- [ ] Does pressing on a generated keyword in the box labelled as “textual context keyword” highlight that word/phrase inside the e-book?
- [ ] When revisiting an image classified and annotated earlier, do you get an overview of the correct classification and annotation saved earlier?
- [ ] Does pressing “Download e-book” under the “Menu” download your EPUB file on your device, with all annotations added?

General Usability Checklist:

- [ ] Do all UI elements and content (text, images) render fully on the Home page and the Editor page?
- [ ] Can the user navigate the UI?
    - [ ] Are all links, the sidebar, and sub-menus accessible and clickable? Are there any broken links?
    - [ ] Is there a sidebar on every screen/page?
    - [ ] Are there any issues with the text content of buttons, fields, tooltips, messages, navigation items, or menus?
    - [ ] Are there any spelling errors?
- [ ] Are there any visible layout issues?
    - [ ] Is the order of menu and submenu items correct?
    - [ ] Do all widths, margins, and paddings match the specifications?
    - [ ] Is the content of UI elements clearly visible (e.g. not truncated due to the width of the element)?
    - [ ] Do fonts match the specifications?
- [ ] Are there any major issues with the structure of the page?
- [ ] Is the page intuitive to navigate through / to use?
- [ ] Are all GUI elements consistent?
- [ ] Is the page too cluttered?
- [ ] Does the system provide sufficient status indication?

Appendix B: General Task Division

Having clear and well-distributed division of tasks is a vital part of planning our project, as mentioned earlier in section 5.3. The image annotation platform to be built is made up of various components: a data processing pipeline, API integration for generating automated suggestions for image descriptions and a web user interface among others. Table 2 below displays a general work distribution we have created that guides us during the development phase of our project.

Table 2: Distribution of high-level technical tasks in our project

	Aratrika	Denis	Filip	Nadine	Vlad
Backend Django system	X	X	X	X	
Frontend React system	X			X	X
GUI Designs				X	X
System Architecture Designs	X	X	X	X	X
Data Storage System Architecture	X	X	X	X	X
Processing of e-books		X	X		
Integration of AI for Automated Image Annotations	X		X		X
Handle Human Image Annotations	X	X		X	
Integrate Docker	X	X			
Integrate Version Control System (GitHub)		X			
Organize Prototype Evaluation on GUI				X	X

Appendix C: User Evaluation Instructions

Below is displayed the form containing instructions to our prototype evaluation, as referred to in section 7.1. This form was sent out to all participants, thereby striving to instruct and inform them sufficiently.

Instructions Prototype Evaluation – Image Annotation Platform

Fixing e-books for visually impaired users!

Thank you for your participation!

This project is a collaboration between the Delft University of Technology (TU Delft) and the National Library of the Netherlands (KB). It is part of our 2nd year course Software Project (CSE2000), for the bachelor programme Computer Science and Engineering at TU Delft.

Using the results to this *prototype* evaluation, we hope to detect usability issues in the GUI of our image annotation platform, which is currently still in development.

It will take approximately 15 to 30 minutes.

Information about the system

This interactive image annotation platform "E-book Fixer" is created to assist experts/volunteers in the process of describing images manually, as is it known that creating meaningful image descriptions is a time-consuming process - thereby introducing issues of cost, scalability, timeliness, and quality.

In this system however, users can get suggestions for image labels / descriptions generated automatically by external AI captioning systems (Google Cloud Vision API e.g.). Afterwards, the textual context around the image is analyzed in order to improve these automated descriptions even more. (*Note that this has not been implemented yet)* Finally, users can create manual image annotations which get saved in the e-book.

Instructions evaluation

Open the online evaluation form via: **https://forms.office.com/r/QWNWAmG85u**

The survey will start with several introductory, demographic questions about you as evaluator.

Afterwards, the hands-on part starts.

- Navigate to the web application via **https://www.denistsvetkov.com/**. You will automatically be lead to the home page.
    - Browse through the home page, try uploading an .epub file to be annotated. (Downloaded from https://www.gutenberg.org/ for example, which has over 60.000 public domain e-books)
    - Fill in **Part 1: Home page** in the survey form.
- After having uploaded your e-book, you will automatically lead to the editor page, in which the image annotation happens.
    - Try navigating through your e-book and annotating images, by generating AI suggestions for example.
    - Fill in **Part 2: Editor page** in the survey form.

We would appreciate receiving your results by Fri 10 June 😊

Informed consent

I consent voluntarily to be a participant in this study and understand that:

The results that come out of the evaluation will be collected to analyze usability issues in our system. Moreover, these results will be documented in the report that we are writing for our course Software Project CSE2000.	☐
I can refuse to answer questions and I can withdraw from the study at any time.	☐

Name of participant: _____

Signature: _____

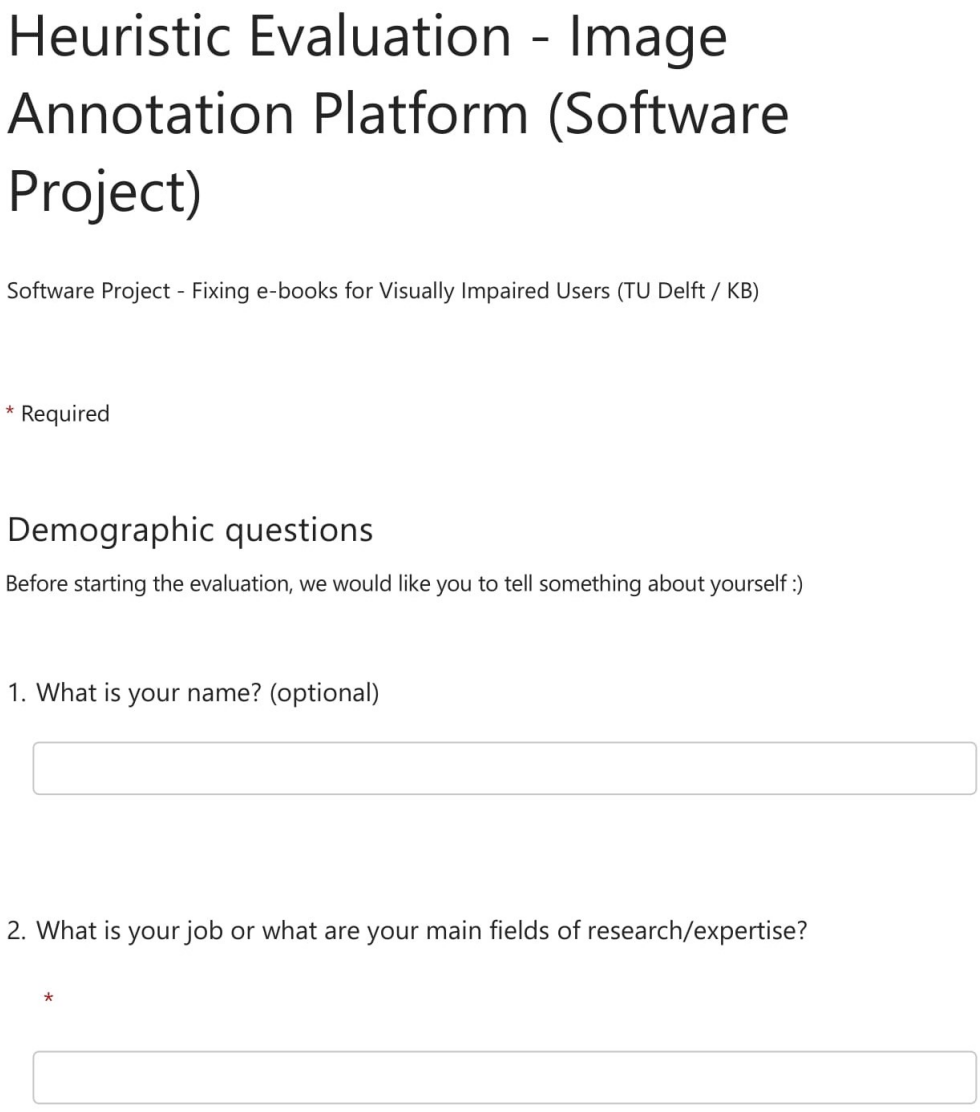
Date: _____

Appendix D: User Evaluation Survey Form

Below is displayed the online survey (created using Microsoft Office Forms) that was sent out to all five participants of our user evaluation, as mentioned in 7.1. Three out of them completed the survey eventually, whereas with the other four we have conducted a verbal evaluation. The final results and conclusions drawn are elaborated on in section 7.2.

The survey starts with demographic questions about the user himself, which are followed by a "hands-on" part. For both the Home Page and Editor Page, the evaluators are asked to interact with the system and give answers to a set of statements related to that interface. Each of these statements is linked to a predetermined heuristic (or usability principle) as defined by Jacob Nielsen in 1993 [11]. Hence, this evaluation can be referred to as *heuristic* evaluation. These heuristics were not displayed to the users, but are indicated on the pages below (in green) as illustration.

After this set of statements, users had the chance to report on usability issues in an open text field. Thereby, we asked them to do so in a predefined format. This way, we try to get a specific description from our evaluators. Besides, by having this predefined format we can easily identify problems that are repeated by different evaluators but stated in different words.



Heuristic Evaluation - Image Annotation Platform (Software Project)

Software Project - Fixing e-books for Visually Impaired Users (TU Delft / KB)

* Required

Demographic questions

Before starting the evaluation, we would like you to tell something about yourself :)

1. What is your name? (optional)

2. What is your job or what are your main fields of research/expertise?

*

Part 1: Home page

The following statements will be about the GUI of our home page, which allows users to upload their e-books. To guide (new) users, information about the platform is provided.

4. Please select the most applicable option.

		I completely disagree	I disagree	Neutral	I agree	I completely agree
Visibility of system status	During the upload process of my ePub file, I got sufficient feedback about the system status.	◯	◯	◯	◯	◯
An aesthetic and minimalist design	The design of the home page is sufficiently clutter-free, while still containing all necessary functions.	◯	◯	◯	◯	◯
Recognition rather than recall	I could easily navigate through the different elements on the home page.	◯	◯	◯	◯	◯
An aesthetic and minimalist design	The design of the home page is pleasing to the eye in general.	◯	◯	◯	◯	◯

5. If there are one or more problems you encountered on the home page, please report on these below in the following format:

   1. Problem description: a brief description of the problem

   2. Likely/actual difficulties: the anticipated difficulties that the user will encounter as a consequence of the problem

   3. Specific contexts: the specific context in which the problem may occur

   4. Assumed causes: description of the cause(s) of the problem

Part 2: Editor page

The following statements will be about the editor of our system, which allows users to navigate through the uploaded e-book and annotate selected images. By generating (different types of)

6. Please select the most applicable option.

		I completely disagree	I disagree	Neutral	I agree	I completely agree
Visibility of system status	During the image annotation process, it was clear to me what the purpose of the visual GUI elements (buttons, forms) are.	◯	◯	◯	◯	◯
Match between system and the real world	I think the system resembles past experiences that image annotators have already had.	◯	◯	◯	◯	◯
User control and freedom	The editor allows for sufficient options to reverse actions (re-edit, go back) when mistakes were made.	◯	◯	◯	◯	◯
Consistency and standard	The GUI elements in the editor are consistent, i.e. similar elements look similar.	◯ completely disagree	◯ I disagree	◯ Neutral	◯ I agree	◯ completely agree

Recognition rather than recall	I could easily navigate through the different elements in the editor.	◯	◯	◯	◯	◯
Flexibility and efficiency of use	I think this system is usable for both new and experienced image annotators.	◯	◯	◯	◯	◯
An aesthetic and minimalist design	The design of the editor is pleasing to the eye generally.	◯	◯	◯	◯	◯
An aesthetic and minimalist design	The design of the editor is sufficiently clutter-free, while still containing all necessary functions.	◯	◯	◯	◯	◯
Help users recognize, diagnose and recover from errors	In case of failures, I was provided with understandable and meaningful error messages.	◯	◯	◯	◯	◯

7. If there are one or more problems you encountered on the home page, please report on these below in the following format:

   1. Problem description: a brief description of the problem

   2. Likely/actual difficulties: the anticipated difficulties that the user will encounter as a consequence of the problem

   3. Specific contexts: the specific context in which the problem may occur

   4. Assumed causes: description of the cause(s) of the problem