Institutt for Informasjonsteknologi
Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo
Besøksadresse: Holbergs plass, Oslo

PROSJEKT NR. Group 0027
TILGJENGELIGHET Open

BACHELORPROSJEKT

Telefon: 22 45 32 00

HOVEDPROSJEKTETS TITTEL Web application development for an image accessibility workflow	DATO 24.05.2024 ANTALL SIDER / BILAG 102/2
PROSJEKTDELTAKERE 151, 186, 198, 255 Group 0027	INTERN VEILEDER Shrestha, Raju
OPPDRAGSGIVER OsloMet	KONTAKTPERSON Shrestha, Raju

SAMMENDRAG

In today's digital age, web applications play a crucial role in our daily interactions with technology. Accessible web images are an important yet often overlooked feature.

This thesis covers the development of ImageAble, a machine learning web application to help developers, writers, and editors describe web images easily, adhering to accessibility standards like NCAM (Freed & Rothberg, 2006).

ImageAble lets users upload images, generate AI descriptions, and evaluate them against accessibility guidelines. Developed using .NET and TensorFlow, it supports both guest and registered users. The app integrates a text evaluation model to ensure quality descriptions.

The web application supports custom models for description generation through API endpoints. This thesis includes an introduction, process documentation, product documentation, and a conclusion.

3 STIKKORD

Web Development

Machine Learning

1 Introduction

1.1 Summary

In today's digital age, web applications play a crucial role in our daily interactions with technology, from social media to e-commerce platforms. The design and functionality of these applications significantly impact user experience and accessibility. Accessible web images are an important yet often overlooked feature when developing web applications, writing articles, or blogs. The design and functionality of these web images significantly impact user experience and accessibility.

This thesis covers the planning and development of a machine learning image accessibility web application as a tool to help developers, writers, and editors describe web images with little effort. Our proposed solution, a web application named ImageAble, aims to make it easier for anyone to adhere to known accessibility standards, such as those set by NCAM (Freed & Rothberg, 2006).

The completed web app features the ability to upload one or multiple images, generate AI-made descriptions of the image contents, and evaluate the generated descriptions based on accessibility guidelines. Developed using .NET, ImageAble allows users to either use the application without an account or create an account to save generated image descriptions along with the uploaded images.

We have also developed our own image description generator model using TensorFlow, a flexible machine learning platform using the Python programming language, and incorporated it into the web app. Additionally, our supervisor, Raju Shreshta, provided a text evaluation model that evaluates the generated image descriptions in terms of select NCAM guidelines to ensure the quality of the description.

The web application is designed to support other custom models for description generation through the user interface by declaring an API endpoint, allowing for flexible use. We will discuss these and other features, as well as the development process and execution, in-depth in the following chapters.

This thesis is structured into four chapters, an introduction, the process documentation, the product documentation, and a conclusion with additional remarks.

1.2 Preface

Images in media are strongly influential in this era, which according to Harvard is used for conveying concepts and information and can reinforce information provided in text (Harvard University, u.d.). However, we need to address the challenges with image accessibility for developers and especially to the visually impaired community. It is either hard or easy to understand depending on the contents of the image. A well written description may enhance the understanding of the content of the image for the reader. Nevertheless, the description which is provided could be either human-authored or software-generated, which is found to be generic and often unreliable. (Shrestha, 2022)

Developers that are dependent on image accessibility might want a quick tool to generate a text for each image they use for development of for example, a web application or reports of concepts. Some might feel writing a satisfactory description may be hard and time consuming. Therefore, it may slow down the progress of development.

An image itself without a description or an image with non-reliable text can be quite challenging for an individual with visual impairments to understand what the contents of the image are.

We are a group consisting of four students pursuing two different study programs, Software engineering and Applied Computer Science. In the early meetings as a group, we discussed what we wanted to write our bachelor’s thesis on. Our project, “Web application development for an image accessibility workflow”, was proposed by Raju Shreshta, Associate Professor at OsloMet after some back and forth on what project we wanted to do.

His project proposes developing a tool for helping with image accessibility workflows within web development or other platforms that utilizes web images. Utilizing A.I machine learning models and .NET ASP CORE framework to conduct our application that can contribute to the research into the use of A.I and image accessibility for not only for developers and visually impaired individuals but also generally for everyone who seek better image accessibility.

There are many definitions of Artificial intelligence which is dependent on how it is approached but according to an IBM defines it as “technology that enables computers and machines to simulate human intelligence and problem-solving capabilities” (International Business Machines Corporation, n.d) It's common to be sceptical about the use of A.I tools or processes.

To resolve the challenges with inaccessibility in images and descriptions we conducted a web application using deep learning artificial intelligence models to generate a descriptive text for images and evaluate the text and give it a score on how well it is and how accurate the text is.

Working with the project has given us a great learning outcome. Throughout the process of the project we have learned multiple workflow methods, discussed agendas, addressing problems and overcoming the problems. Further, we enhanced our skills by facing challenges that were outside our competence, which yielded a lot of learning outcomes.

In this thesis we will go further into the depth of the project, where we are going through how we went through planning, process, methods, tools and finalizing the product. We will also go through the learning outcomes and conclusions about the project.

We would like to thank our Supervisor, Raju Shreshta, who proposed this project to us.

1.3 Table of Contents:

1 Introduction 2

1.1 Summary 2

1.2 Preface 3

1.3 Table of Contents: 5

2 Process documentation. 7

2.1 Preparation 7

2.1.1 Finding a project 7

2.1.2 Minimum Requirements 8

2.1.3 The possibility of expanding the project 11

2.1.4 Other Criteria and guidelines 12

2.2 Planning 13

2.2.1 Finalized plan. 13

2.3 Methods and workflow 15

2.4 Visual Design 18

2.4.1 Early phases 18

3 Product Documentation 32

3.1 Solution Introduction 32

3.2 Web application Development 32

3.2.1 ASP.NET core MVC 33

3.2.2 Web application architecture 34

3.3 Client-Side (front-end) architecture 35

3.3.1 Description of views 36

3.3.2 Views > Home 37

3.3.3 Views > Gallery 37

3.3.4 Views > Shared 38

3.3.5 User Interaction and Data Flow 38

3.4 Client-side (front end) Development 46
3.4.1 Home page development 48
3.4.2 Home page components & styling 48
3.4.3 Authentication pages styling 51
3.4.4 Gallery page Development 52
3.4.5 Gallery page – Search Component 54
3.4.6 About page development 55
3.4.7 Responsiveness grid system 55
3.4.8 Final Design adjustments 57
3.5 Server-side (back-end) Architecture 58
3.5.1 Introduction 58
3.5.2 Detailed description of components 58
3.6 Back-end (server-side) Development 64
3.6.1 Introduction 64
3.6.2 Data flow & Process 64
3.6.3 Database 69
3.6.4 API Gateway 71
3.6.5 Adding metadata to image 72
3.7 Security Considerations 73
3.7.1 Security Components: 75
3.8 Model development 76
3.8.1 Libraries and Technologies 77
3.8.2 ImageAble model architecture and preparations 77
3.8.3 Deploying model 87
3.8.4 Implementing Flask 87
3.8.5 Docker and Azure 88
3.8.6 Evaluation Model 89
3.9 Testing 90

3.9.2 Results on unit testing 96
3.9.3 Discussion: 97
4 Conclusion 98
4.1 Concluding summary 98
4.2 Personal learning and improvements 98
5 References 100

2 Process documentation.

2.1 Preparation

The first stage of preparation was finding the project. We made sure we found the right project for us to work on. The methods of finding the right project were quite straight forward since some of the projects was published on Canvas. Methods such as sending an e-mail to relevant business based on interest was also a method to find a project to work on. However, we chose the easier path where we chose the most relevant project based on our skills which was published in Canvas.

2.1.1 Finding a project

Our criteria for founding a project, such as it had to match our skills and within our competence, it should be relevant based on the subject we were taking. Finally, it should be a project where it could challenge our skills and give us an educational benefit.

While searching for a project we first found a project which consisted of making an app helping immigrants to learn the Norwegian language and helping them to find a job based on their earlier competence. This app required to be able to launch for both IOS and Android which meant we had to know programming languages such as Swift and Kotlin or React Core. Even though we did not have the optimal competence for the project we were still interested mostly in the learning outcome.

Furthermore, after discussions about whether we should take the project or not, which we decided to not take due to low competence and missing skills in app development, we decided to search further.

In about a week after we found the first project, we finally found this project which was very interesting and matched our criteria, proposed by Raju Shreshta, Associate Professor at OsloMet. We knew this could give us a great learning outcome because we wanted to learn more about A.I.

We were told the project was only made for a group of 3, but we discussed with the supervisor which almost was responsible for the project where we could extend the project for a group of 4. Therefore, we agreed with the supervisor to extend it and possibly make the «Image accessibility workflow" application also as a web browser extension.

The idea of the extension is to easily download it on chosen web browser and use it whenever a user needs it, instead of visiting our application every time they want to upload an image to generate a description.

The project had no requirements for frameworks and coding language. This was a huge advantage due to the wide variety of programming languages and frameworks, which can improve the project a lot. Without any written plan yet, a discussion came up within the group which consisted of the agenda about what tools we wanted to use to make the plan. As a group we quickly found out using a framework such as ASP.NET was the most optimal and most relevant way to solve the problem. Originally the project is a small and simple application, using a framework would be surplus and would be enough just only use HTML and CSS and JavaScript, but it would give us a huge advantage due to its scalability.

2.1.2 Minimum Requirements

Before the planning phase, a review of the minimum requirements is important to look at for an overview of the project. Having an overview of the requirements gives the plan a more structured basis and an idea of what the contents of the plan should be. Also, it is important to know what parts of the project to focus on.

We have defined non-functional requirements and functional requirements early in the process, which helped us prevent changes late in the development process. Since they are part of costly performance and design principles.

2.1.2.1 Functional requirements

Functional requirements are the basic requirements that a user specifically demands that the system should offer. This is requirements is needed to develop of a software or an application. However, these requirements vary depending on the type of application or software. These requirements are presented as data to be entered into the system along with the expected results. The user can directly see the implemented result of the requirements unlike non-functional requirements. (Geeks for geeks, 2024)

NR	**FUNCTIONAL REQUIREMENTS**	
	Register and log in: • User will be able to register and log in	*Implemented*
1	Image upload and handling: • User can upload images from their devices to the application • Uploaded images can be deleted and edited	*Implemented*
2	Generate and evaluate descriptions using the models	*Implemented*
3	User can manage account.	*Implemented*
5	User can log in and save images to gallery with description	*Implemented*
6	User can download image as a guest user	*Implemented*
7	User does not have to be logged in to edit the description	*Implemented*
8	User will be able to choose models to generate and evaluate descriptions	*Implemented*
9	User will be able to use own machine learning models by inserting API endpoint	*Implemented*

Table 2.1-1 – Functional requirements

2.1.2.2 Non-functional requirements

Non-functional requirements are just as important as functional requirements. They define how the functional requirements should work within the system and focus on how the system should be or behave or the characteristics of the application that it has and how it behaves through the tasks requested by the user. They help define and ensure a positive user experience by setting standards for usability. This was overwhelming for our web application where user satisfaction directly affects the success of the product through the interactions they make or ask for. Examples of issues the non-functional requirements deal with is as following:

- Security
- Reliability
- Portability
- Scalability
- Performance
- Flexibility

NR	**NON-FUNCTIONAL REQUIREMENTS**	
1	**Description**: The web application must handle images and generate description in an expected period of time. **Requirement:** Description should be generated and evaluated withing 30 seconds after the image is uploaded or user press “generate” submit button.	*Implemented*
2	**Description**: Secure web application user data should be handled in a proper and secure way. **Requirements:** Authentication must be implemented. Images must be secured with hash string or other form of security.	*Implemented*
3	**Description**: Web applications must be simple and easy to use.	
4	User friendly design. **Requirements**: Responsive design, WCAG recommendations.	*Implemented*
5	The web application should be supported by the major web browsers for example: Google Chrome, Firefox, Opera, Safari and Microsoft Edge. Requirements: Updated code syntax	*Implemented*
6	**Description**: The system must be available even though the user is not logged in. **Requirements:** Implement code logic to make the website accessible for guest users.	*Implemented*
7	**Description**: Application must be accessible for different needs, for example the visually impaired community and web/software developers. **Requirements**: Use of colour palette, font style and size, WCAG recommendations.	*Implemented*
8	**Description:** Web application should be well documented and easy to update. **Requirements:** XML documentation for each code line/block that needs an explanation or inspired by source. Folder structure for easy updates and changes.	*Implemented*

Table 2.1-2 – Non-functional requirements

2.1.3 The possibility of expanding the project

The decision to expand the project with other functionalities was not intended in the first place. The following functionalities were not prioritized; however, it could have been implemented if we had time to spare:

1. Allowing the user to modify the website, for example: change in font, font size, contrast/colour, dark/light-mode.
2. A web browser extension: allowing the user to use the tool on other websites he/she visits. The extension will gather images that are presented on the website. If the image(s) has an alt-text already coded, then output it. Otherwise, generate a description and evaluate it on selected images.

These functionalities were a way to extend the project and it could make the application more accessible and user friendly. Nevertheless, the idea of the functionality where we could accomplish to allow user to modify the font by for example by size, colour, or font family to improve readability. This would be implemented by font options on top corner either on right side or left side. The options would be “change colour”, “Change font size” buttons where the user could customize the experience.

A web browser extension would allow the user to use the tool without being dependent on visiting the web application itself. This means that when a web extension is installed a user can use it whenever they want while visiting other websites. The user has the option to install a web extension for their chosen standard browser such as Chrome or other browsers.

There are several reasons to use web extensions, but the main goal of making the application as an extension is to increase efficiency in workflow and increase accessibility. Users could have the benefit of this extension when users for example need a quick description of an image or if a user is not able to see the content of an image.

The interface of the extension would look a little like the application, however only with drop area to upload and a description will generate. The user can simply drag an image whether it's from a website or stored locally and drop it on the extension to generate.

2.1.4 Other Criteria and guidelines

There are two main guidelines we have used in our project for accessibility purposes. Both for our own website, and for the description generator. These will be explained more in detail later, but are as follows:

2.1.4.1 NCAM guidelines

Our application will use a machine learning model that evaluates descriptions. This model follows selected NCAM guidelines to evaluate reliable description for better image accessibility (Shrestha, 2022). Even though we do not have any components in our application that we are developing by ourselves that follow these guidelines. It is still important to address these guidelines which the model is provided by our supervisor which also took part in developing it. To get an enhanced comprehension about reliability of a satisfactory description we will need to understand the guidelines and the model works.

NCAM has guidelines for accessibility for most image types. According to an article written by Shrestha, 2022 Among 14 guidelines, ten of the guidelines was used to develop the model and a summarized guidelines is as follows:

1. *“The description should be succinct.”*
2. *“Colors should not be specified unless it is significant.”*
3. *“The description should be started with a high-level context and drilled down to details to enhance understanding.”*
4. *“The active verbs in the present tense should be used.”*
5. *“Spelling grammar, and punctuation should be correct.”*
6. *“Symbols should be written properly.”*
7. *“The description vocabulary should be added which adds meaning, for example: “map” instead of image.”*
8. *"The* material *should not be interpreted or analyzed; instead, the reader should be allowed to form their own opinions.”*
9. *“Physical appearance and actions should be explained rather than emotions and possible intention.”*
10. *The material should not be interpreted or analyzed; instead, the reader should be allowed for form their own opinions.”*

(Shrestha, 2022)

2.1.4.2 WCAG guidelines

With this project having such a big focus on accessibility, we wanted to integrate the Web Content Accessibility Guidelines, or WCAG (Henry, 2024). WCAG, as the name suggests, are guidelines for accessibility on web pages to make web content more accessible to disabled users. The guidelines incorporate standards for the visuals of a web page, such as colour contrast, font sizes, and object placements. We have tried to incorporate these guidelines throughout the development of the project.

2.2 Planning

After the preparation, the planning process was able to begin. The beginning of planning consists of brainstorming ideas and going through minimum requirements. Early meetings in the planning process we wrote down agendas before meetings with our supervisor. In the meetings there were discussions about what the contents of the plan should be. In the plan we implemented a list of requirements which was provided for us such as project requirements and minimum requirements for the application.

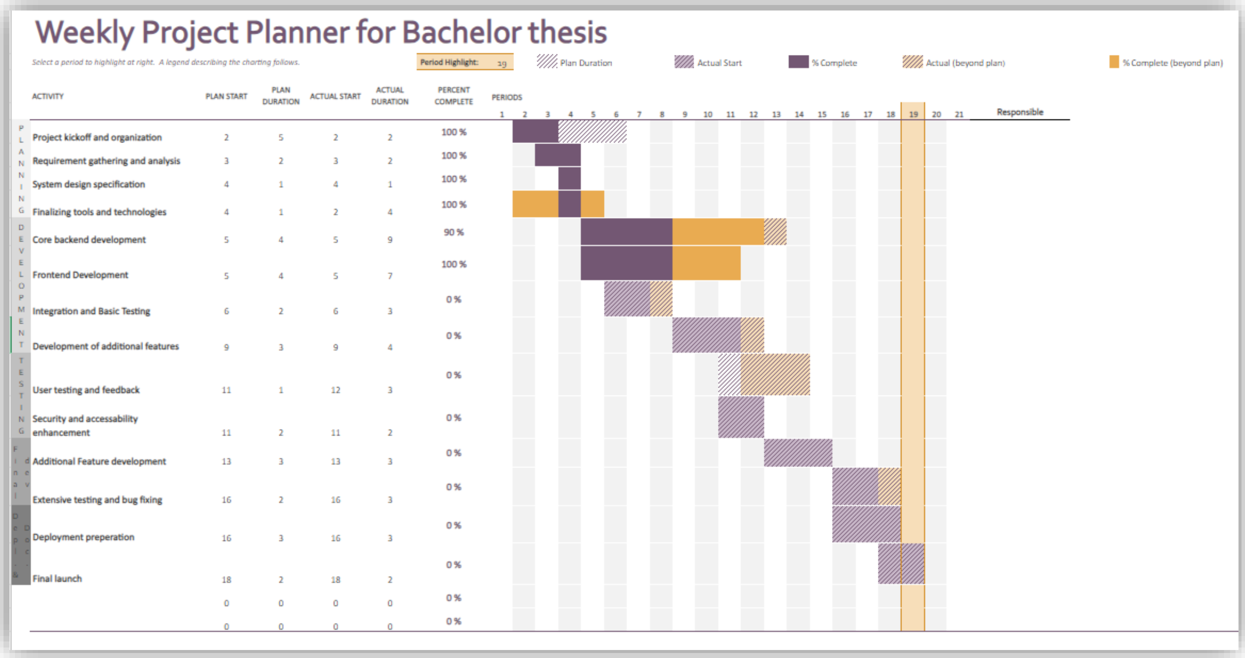
The process of producing a plan went quite quick and it was straight forward. The plan was created together in a meeting which was well thought out and as accurate as possible. Writing down and using Excel to make the plan.

2.2.1 Finalized plan.

Finalizing the plan took only a few days to make. Excel has planning templates which we could use and are uncomplicated. In the finalized plan we have divided the project into different activities to complete. These activities are a general description of huge parts of the project. Therefore, this template is not going into depth, instead we are using “Miro” to assign more detailed activities.

Figure 2.2-1 shows a template from excel and it includes “Activity”, “Plan start” Plan duration”, “Actual start”, “Actual duration”, “Percent complete”, “Periods”. In the “Periods” part shows number of weeks and a bar chart that changes based on the mentioned fields. Each bar chart has its own purpose to show “Plan duration”, “Completed” and “Actual beyond”.

The execution of the plan overall went slower and ineffectively than expected. Following the plan at the start went smoothly such as making designs and implementation front-end design until another assign hard tasks came through.



Weekly Project Planner for Bachelor thesis

Select a period to highlight at right. A legend describing the charting follows.

Period Highlight: 19

▨ Plan Duration ▨ Actual Start ■ % Complete ▨ Actual (beyond plan) ■ % Complete (beyond plan)

	ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE	Responsible
PLANNING	Project kickoff and organization	2	5	2	2	100 %	
	Requirement gathering and analysis	3	2	3	2	100 %	
	System design specification	4	1	4	1	100 %	
	Finalizing tools and technologies	4	1	2	4	100 %	
DEVELOPMENT	Core backend development	5	4	5	9	90 %	
	Frontend Development	5	4	5	7	100 %	
	Integration and Basic Testing	6	2	6	3	0 %	
	Development of additional features	9	3	9	4	0 %	
TESTING	User testing and feedback	11	1	12	3	0 %	
	Security and accessability enhancement	11	2	11	2	0 %	
Final dev	Additional Feature development	13	3	13	3	0 %	
	Extensive testing and bug fixing	16	2	16	3	0 %	
Depl. & Doc.	Deployment preperation	16	3	16	3	0 %	
	Final launch	18	2	18	2	0 %	
		0	0	0	0	0 %	
		0	0	0	0	0 %	

Figure 2.2-1 - Project planner.

The execution of the plan overall went slower and ineffectively than expected. Following the plan at the start from weeks two and six went smoothly such as making designs and implementation front-end design until another assign hard tasks came through between week six and eight.

Complicated tasks such as testing description models and evaluation models. The decision of trying to fine-tune own TensorFlow model was complicated and took a lot of time. Most of the time for fine-tuning a model was the research, training and testing. Other time consuming and complicated tasks was the back end of the application. We had a hard time implementing communication with front-end components and understand how to connect the application to the models.

To address the problem with slow and inefficient workflow, improving our methods was the main key to catch on work that were beyond deadline. There was a lot of improvements that could be done. The lack of meetings and poor communication led to slower workflow. The improvements of methods gave us a clear overview and greater communication which led to that we were able to keep up with the plan duration effectively.

2.3 Methods and workflow

Methods in the workflow to get progress major part of the project. Choosing a type of methods gave a huge impact on the workflow in the group. Several meetings were held to clarify which method and tools to use. In addition to our own meetings as group, we had meetings with the supervisor every other week where we could get advice on how to carry out methods.

2.3.1.1 Kanban work method

Finding a workflow method is to have an overview of the work that should be done. There are many different and unique methods to use, such as Agile and Waterfall methodology, Scrum, Kanban and vice versa (Eby, 2017) Each workflow method has its own   advantages and disadvantages. We will not go through all of the methods in this part, however a discussion between whether we should use Scrum or Kanban method escalated to an agreement of using Kanban framework which is agile management method and built on a philosophy of continuous improvement (Martins, 2024). The work items are represented visually on a Kanban board, which is a project management tool to visualize work, work-progress to maximize flow.

We had an agreement of not choosing Scrum, which also is an agile project management because of the irrelevancy which is often required to have assigned roles such as product owner, scrum master and a development team. The method is also using daily meetings and sprints which usually lasts two weeks, this made us unsure if we could keep up delivering small increments in such a short time span. Another disadvantage we discussed was the quality of the product using this method could be since it is a fast and agile method.

Comparing Scrum to Kanban, this method was more reliable and relevant for a group of 4 members. The visibility of the work gave us a more overview and we could see how tasks move throughout the process. Flexibility was a crucial part of the workflow; this is an advantage Kanban could give us. With Kanban we were able to reassess priorities and we were not dependent on a rigid project plan.

We were more familiar with how the Kanban worked due to working with multiple other projects in the different enrolled subjects. Work is displayed on a Kanban board with columns such as “In progress” and “Done”. As shown in figure 2.3-1, a Kanban board is made with the help of an online tool called “Miro”, a visual workspace for innovation. In the board it is split in multiple columns which consists of “Backlog, “In progress” and “Beyond Deadline”. The “backlog” column contains all the work we needed to do which each group member had the freedom to choose any task by changing the colour of the cards based on the colour code assigned for each group member. We chose the tasks based on what we were most comfortable with, which also was based on skills and knowledge. After choosing a task, each group member had to drag each card to “In progress”.

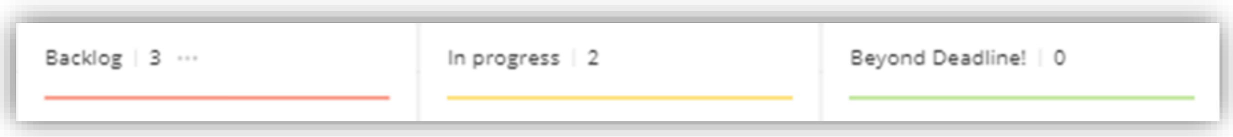
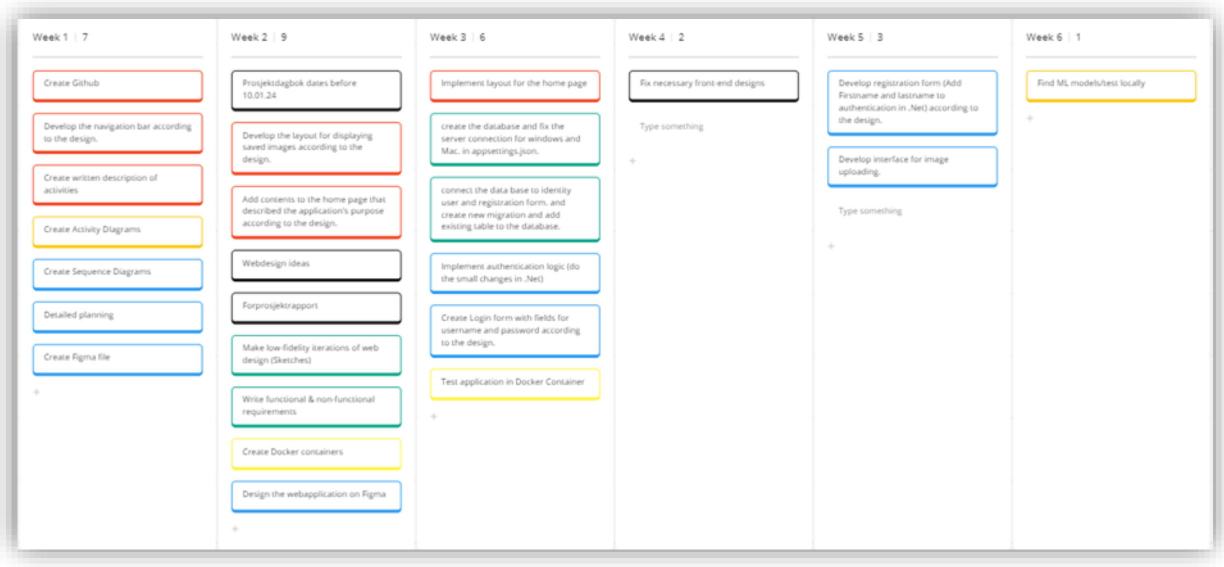


Figure 2.3-1 – Kanban board.

Further, we extended the board by adding in which weeks the tasks are done where each individual had to drag the assigned task to the week it was done as shown in the figure 2.3-2 below, which does not contain all of the weeks. However, it shows each card is dragged to the week when the task was done. The whole Kanban board consisted of 12 weeks in total.



Week 1 | 7
- Create Github
- Develop the navigation bar according to the design.
- Create written description of activities
- Create Activity Diagrams
- Create Sequence Diagrams
- Detailed planning
- Create Figma file

Week 2 | 9
- Prosjektdagbok dates before 10.01.24
- Develop the layout for displaying saved images according to the design.
- Add contents to the home page that described the application's purpose according to the design.
- Webdesign ideas
- Forprosjektrapport
- Make low-fidelity iterations of web design (Sketches)
- Write functional & non-functional requirements
- Create Docker containers
- Design the webapplication on Figma

Week 3 | 6
- Implement layout for the home page
- create the database and fix the server connection for windows and Mac. in appsettings.json.
- connect the data base to identity user and registration form. and create new migration and add existing table to the database.
- Implement authentication logic (do the small changes in .Net)
- Create Login form with fields for username and password according to the design.
- Test application in Docker Container

Week 4 | 2
- Fix necessary front-end designs

Type something

Week 5 | 3
- Develop registration form (Add Firstname and lastname to authentication in .Net) according to the design.
- Develop interface for image uploading.

Type something

Week 6 | 1
- Find ML models/test locally

Figure 2.3-2 – Full Kanban board.

2.3.1.2 Improvement methods

Kanban board is not the only method to improve our work. We observed that the work and following the plan went slower than expected. To address the problem of how inefficient our flow in development progress was, we had to gather methods to enhance our workflow to be able to reach our deadlines.

It was in our interest to mostly work from home and have meetings on Zoom or through other online communication applications, due to problems with laptops and it was more efficient to work with desktops at home. Even though we mostly worked from home, we were able to find relevant and most efficient methods to follow.

The improvement of workflow required us to evaluate what we were doing wrong or why we were executing work inefficiently. We believed that the inefficient workflow was caused by poor communication, multitasking, distractions and poor task management.

Solving this issue required implementing more meetings where we decided to have at least 1 to 3 meetings per week with agendas to increase better communication. Having a focus on conquering difficult tasks first was an effective way to power through easier tasks. Working on multiple tasks at the same time reduced productivity, therefore solved it by distributing tasks evenly so each group member could focus on one task at a time. Optimizing daily task and planning reduced poor task management. By optimizing daily tasks and daily plan to the fullest, we were able to manage and handle our tasks better and be more efficient.

2.3.1.3 Effectiveness of the methods

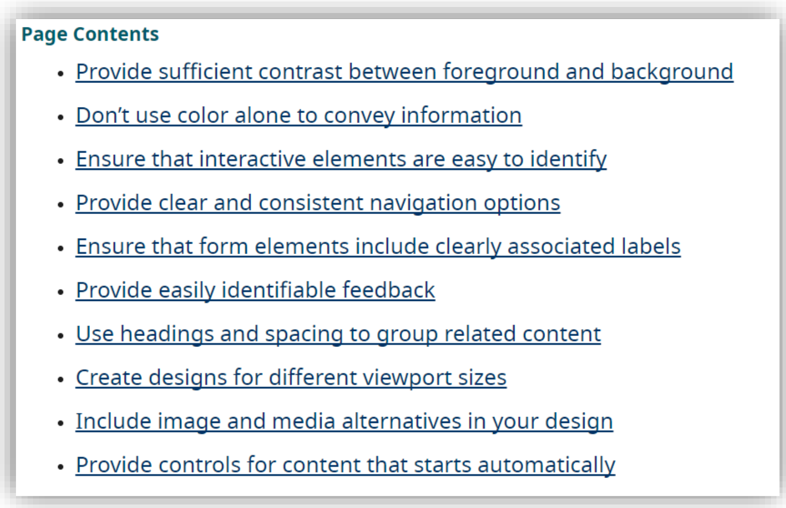
The effectiveness of the methods is mostly positive. The right choices were made when choosing methods. Being careful, taking smart decisions and getting help from supervisor built on the effectiveness of the methods. The Kanban framework improved workflow management and boosted productivity by visualizing work items. The approach with Kanban allowed us to change priorities and be flexible. Its simplicity does not require specific roles, and this led to better overview of the whole project and better communication between each group member.

Furthermore, our methods of improving workflow helped us evaluate our strengths and weaknesses. Changing methods and adapting methods gave us an immense learning curve and enhanced productivity.

2.4 Visual Design

For the design process we have heavily focused on meeting the guidelines from WCAG, as described in 2.1.4.2. Accessibility on our own web application is important to get right, especially in an app revolved around accessibility.

We have formed sketches and low- fidelity designs using both drawings and Figma.
WCAG Design & Develop guidelines introduces basic considerations that helps us with making a user interface. WCAG's official site contains detailed guidelines to follow as shown in Figure 3.4-1 (WCAG, n.d.)



Page Contents

- Provide sufficient contrast between foreground and background
- Don't use color alone to convey information
- Ensure that interactive elements are easy to identify
- Provide clear and consistent navigation options
- Ensure that form elements include clearly associated labels
- Provide easily identifiable feedback
- Use headings and spacing to group related content
- Create designs for different viewport sizes
- Include image and media alternatives in your design
- Provide controls for content that starts automatically

Figure 2.4-1 - WCAG's page content for overview of design guidelines. (WCAG, n.d.)

2.4.1 Early phases

In the development of our new web application, the goal was to create an intuitive and accessible platform that meets specific user needs. Therefore, design was an important part of the planning process. The initial sketches of the website's layout and functions were drawn on paper. To also emphasize how the sketches and meetings were important for the team, they served as a means of communication among us. The figures below show the first sketches we made.

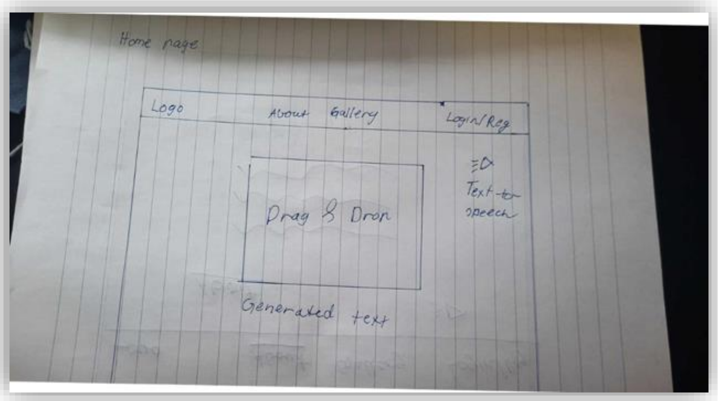


Figure 2.4-2 – Index page sketch.

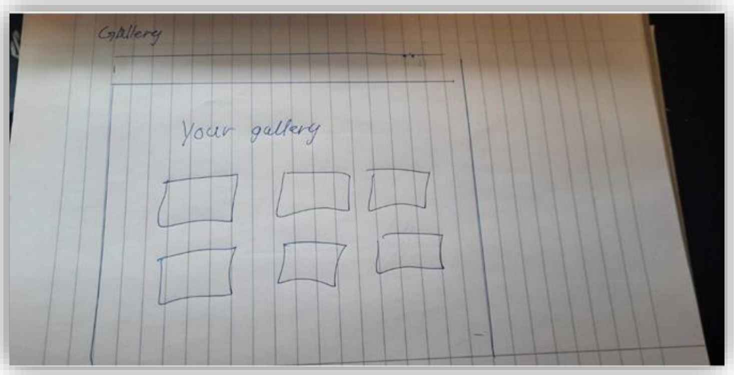


Figure 2.4-3 - Gallery page sketch.

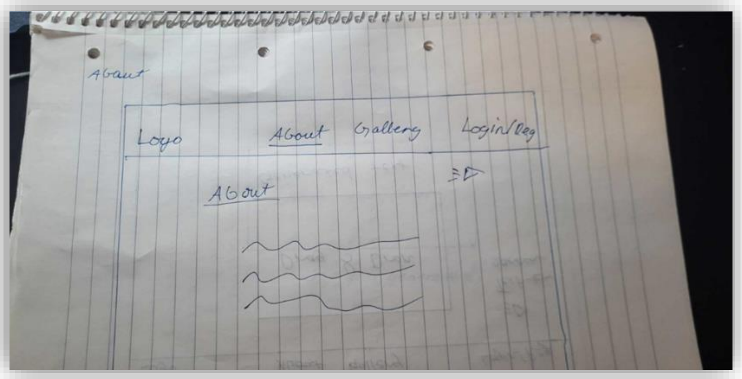


Figure 2.4-4 – About page sketch.

Early sketches were regularly shown to the entire team to get feedback. This ensured that all perspectives were considered and that the design continued to evolve in line with the project's goals. This provided valuable insights that led to several iterations before we moved on to more detailed digital prototypes.

2.4.1.1 Digital prototypes and iterations

After the paper sketches were completed and we all agreed on a common design based on our project's needs, we moved on to digital prototyping using Figma. By clearly utilizing the sketches, our team managed to evolve from the initial sketches to more detailed prototypes. By consulting with our supervisor and getting user feedback, which led to several internal reviews of the sketches and ideas, this resulted in multiple iterations in the design.

2.4.1.2 The first iterations

For each iteration, we discussed the key changes and choices that were made, including colour choices, layout, typography, and interactive elements. We conducted five design iterations. The figures below show the first two iterations of the prototype designs we created.

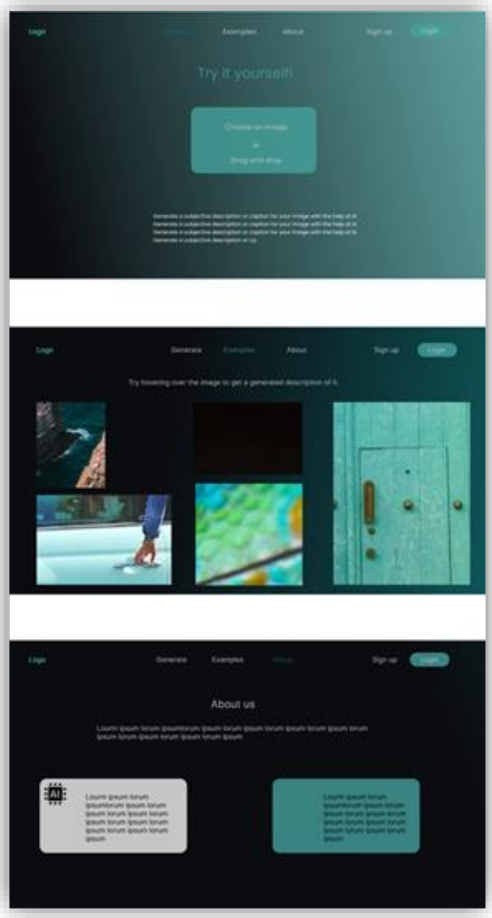


Figure 2.4-52.4-5 – First iteration.

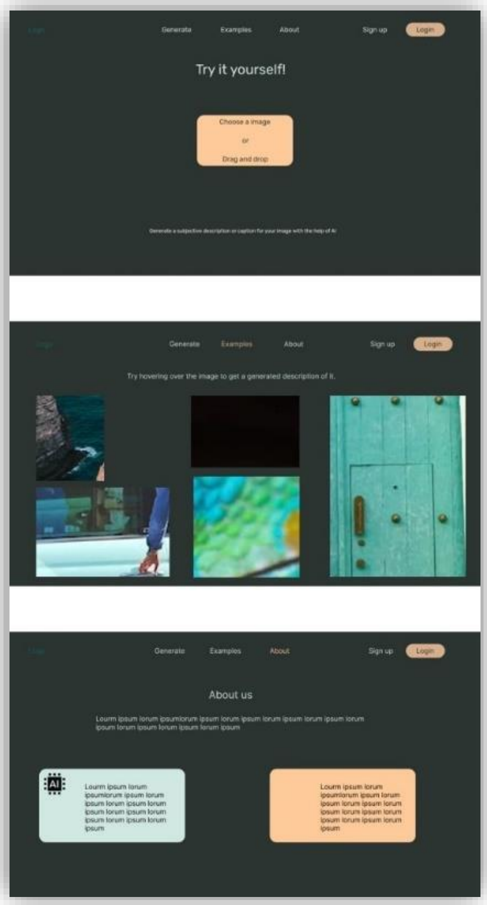
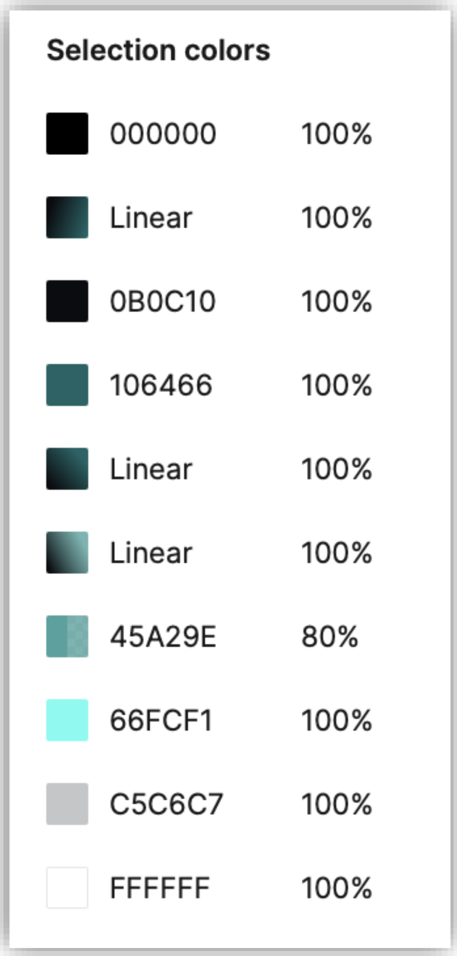


Figure 2.4-6- Second iteration.

Through meetings and discussions, we managed to complete five iterations, with the fifth being the final design we decided to continue with. We ensured that the design adhered to universal accessibility standards, taking into consideration people with visual impairments or colour blindness. Additionally, the four of us have diverse cultural backgrounds. This is reflected in a significant variation in colours, though the design itself remains largely consistent.



Selection colors

■	000000	100%
■	Linear	100%
■	0B0C10	100%
■	106466	100%
■	Linear	100%
■	Linear	100%
■	45A29E	80%
■	66FCF1	100%
■	C5C6C7	100%
□	FFFFFF	100%

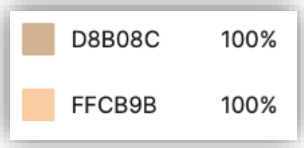
Figure 2.4-7 – First iteration colours.

This first iteration introduces a dark colour theme, featuring dark blue and turquoise tones that contribute to a calm and professional atmosphere. These colours were chosen to enhance the user's focus on the content, especially on interactive elements such as image uploads.

From the start of the design process, it was important to us to prioritize user friendliness and accessibility with an easy-to-understand user flow for our users.

The interactive elements, such as the upload button and the navigation bar, are prominently highlighted to engage the user immediately.

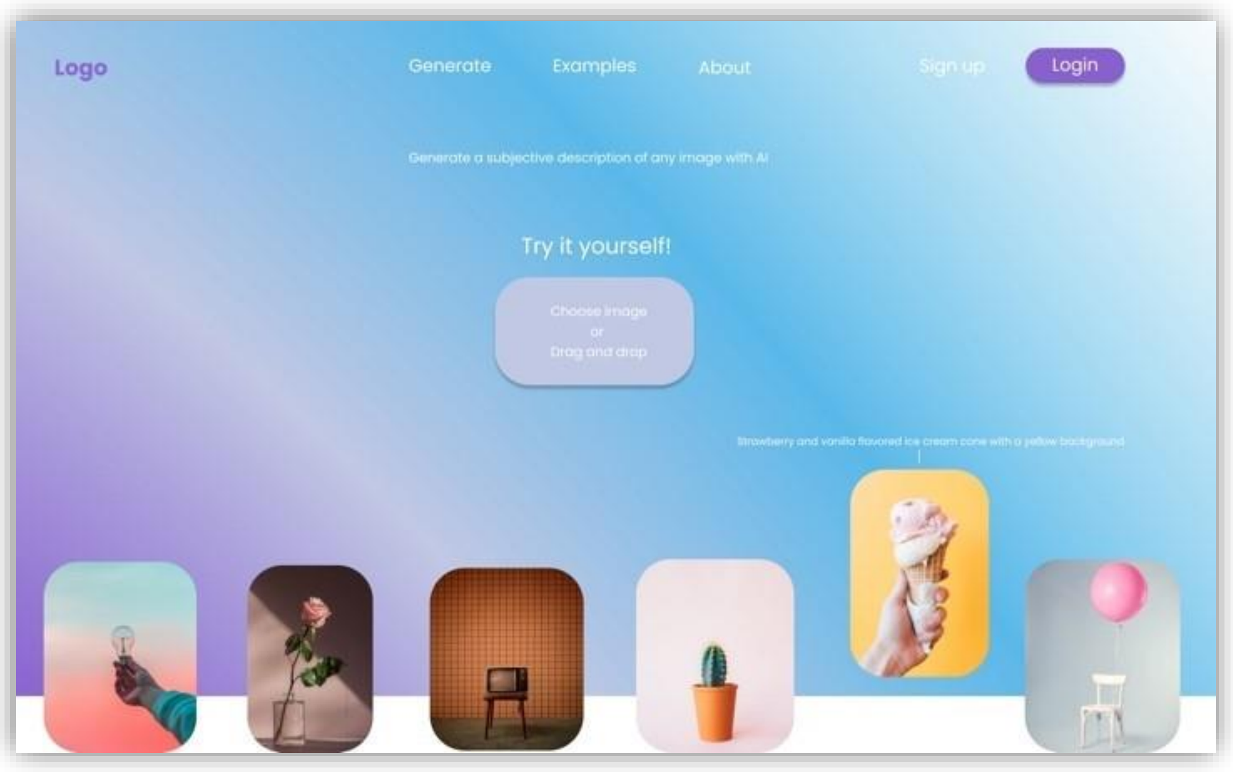
Second iteration: In the second iteration, we tested with a slight colour variation and introduced warmer colours such as soft teal (D8B08C) and orange (FFCB9B). These changes enhanced the app's visual appeal and helped differentiate between various user actions and functions.



■	D8B08C	100%
■	FFCB9B	100%

Figure 2.4-8 – Soft teal and Orange.

2.4.1.3 Third iteration

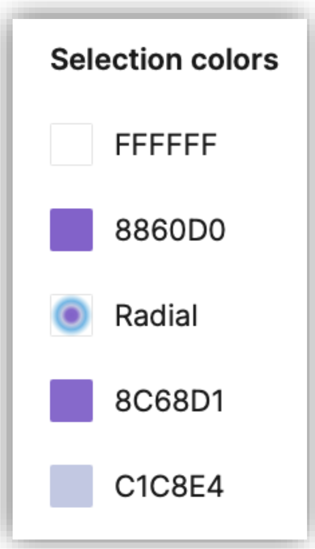


Generate a subjective description of any image with AI

Strawberry and vanilla flavored ice cream cone with a yellow background

Figure 2.4-9 – Third iteration index page.

This iteration brings a significant change in colors, adopting a more vibrant and livelier color scheme, which gives the website a more dynamic and friendly appearance. We moved to warmer tones, introducing bright colors such as pink and blue. We found that bright colors create a more energetic and engaging user experience and chose them to attract younger users.



Selection colors

⬜	FFFFFF
🟪	8860D0
◉	Radial
🟪	8C68D1
▨	C1C8E4

Figure 2.4-10

To enhance interactivity, we incorporated animations and transitions to make the user experience smoother. The softer corners on buttons and cards provide a more modern and accessible appearance.

By implementing better responses to user interactions, such as feedback text appearing on hover effects on images, we capture the attention of new users and provide a clear indication of what the site is about. Figure 2.4-10 shows the main colours of this iteration.

2.4.1.4 Finalized design.

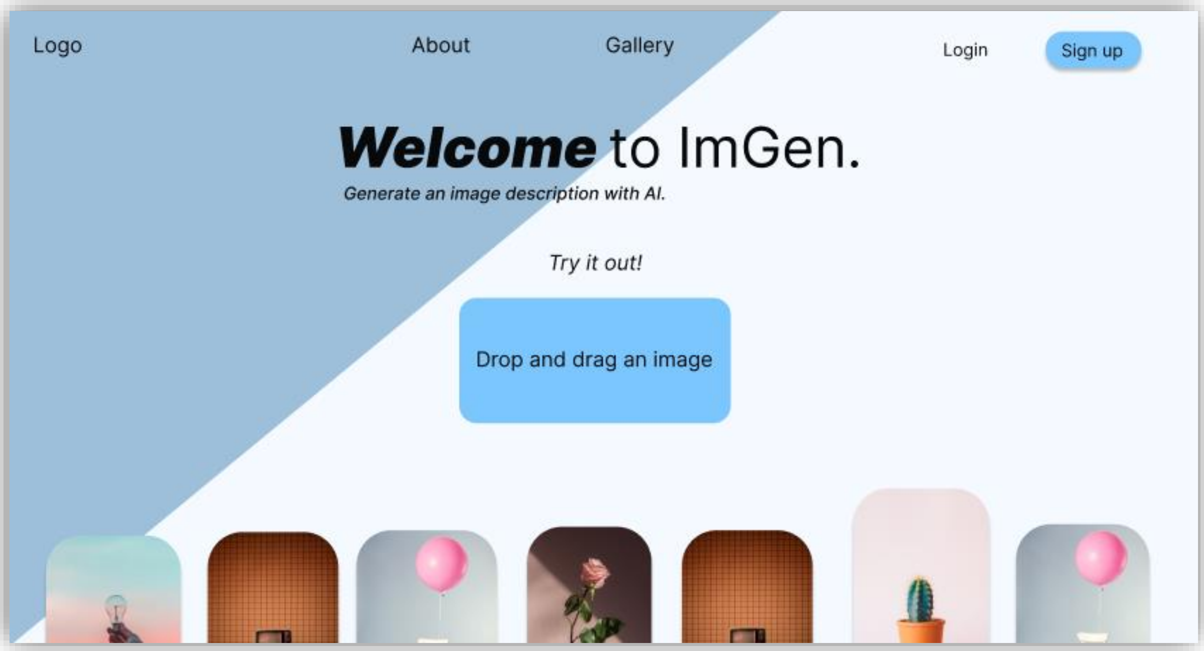
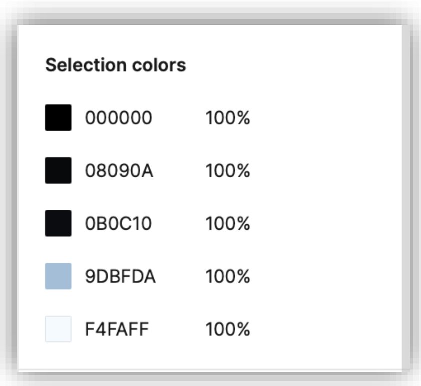


Figure 2.4-11 – Finalized design.

After three more iterations, we came up with a design that we agreed upon would work well for our web application. This would be the main inspiration for creating the final design in the development process.

It introduces a simpler colour palette, consisting of shades of blue, white, and black, which will end up being close to the final colour palette This design also aligns with the main WCAG guideline that influenced our choice under Principle 1, "Perceivable," (WCAG, n.d.)which states that text and text images should be readable and understandable.

Under this principle, guideline 1.4.8 "Visual Presentation" specifically advises on the use of font types that are easier to read for people with disabilities. It suggests avoiding or limiting the use of fonts with significant flaring, looping, or other decorative features that can distract or impede readability.



Selection colors

■	000000	100%
■	08090A	100%
■	0B0C10	100%
■	9DBFDA	100%
□	F4FAFF	100%

Figure 2.4-12 – Finalized sketch colours.

One of our main goals for the design was to ensure that anyone visiting ImageAble could find what they were looking for without any hassle, meaning easy access to all web app pages. We aimed to optimize user experience, promote ease of use, and ensure accessibility across various devices and user capabilities.

We decided to incorporate a top-level menu design. ImageAble employs a horizontal top-level navigation menu, including links to major sections such as Home, Gallery, About, and user account management options. In total, we created six navigation sites, each with its own function.

The navigation bar is planned to be responsive, ensuring that the website is equally navigable on desktops, tablets, and smartphones.

2.4.1.5 Responsive design

We aimed to make ImageAble more responsive and accessible for users with different devices. However, there were some exceptions due to time constraints and these planned illustrations do not accurately represent the final product.

We defined further suggestions for enhancement. For example, the current format of the data table looks good on desktops and larger screens like iPads but could be challenging to read on mobile devices. A more mobile-friendly approach, such as a card layout, could enhance usability.

Figure 2.4-13 explains how the application would look after these improvements, which we believe would result in a better user experience.

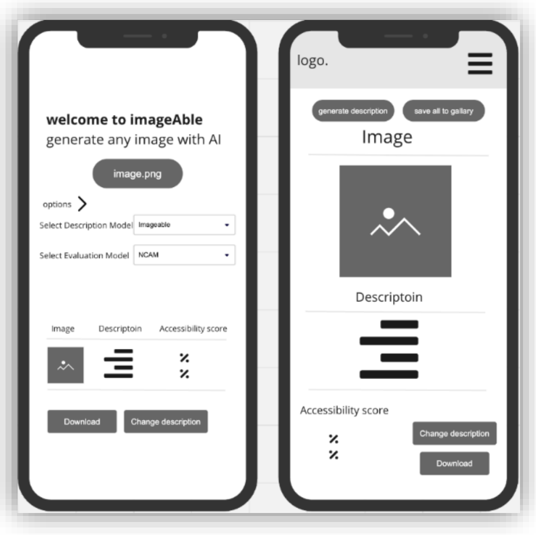


Figure 2.4-13

We identified additional areas for improvement, such as button adjustment for smaller screens. While functional, the buttons could better adapt to smaller screens by increasing the padding and adjusting the layout to ensure they are easily clickable on all devices.

2.4.1.6 Color consideration

When designing our website, we spent a lot of time figuring out the best colours to use. We tested many different colours. Colours don't just make the site look good; they also impact usability. We wanted to understand both the negative and positive aspects of different colour schemes, so we experimented with a variety of them to see what worked best.

We faced a significant challenge in deciding whether to use a white or dark background. Both have their advantages and disadvantages, especially depending on the type of website being created. This was a decision we needed to consider carefully.

As mentioned earlier, white backgrounds are clear and easy to read, making text easily readable without straining the eyes. This is crucial for our content-heavy site as it helps users read without discomfort. The high contrast ratios achieved with a light background enhance text readability, a principle supported by WCAG (WCAG, n.d.).

White backgrounds act as a blank canvas, allowing us to add any colour for buttons without worrying about clashing. As discussed in Moss51's analysis on web design, white is used effectively to create a sense of order and professionalism, key to modern web aesthetics (Moss51, n.d.).

However, if not complemented with other warm colours or graphical elements, a white background can make a site appear too sterile and boring, potentially diminishing user engagement if the space feels too empty.

We also considered using a dark theme to give our site a modern vibe. Dark themes can make elements like photos and graphics stand out, making them perfect for portfolios or art-related sites. Laurie Clarke explains in an article that dark themes can enhance the richness and contrast of images, making this a popular choice in the creative sectors (Clarke, 2019).

Ultimately, we settled on a white background with dark elements and text. This choice enhances readability by providing high contrast, ensuring that body text and headlines are easily readable, thus improving overall readability and user experience and is a decision that aligns with WCAG (W3C, 2023).

2.4.1.7 Final colour choice

Looking at our sketches, we frequently changed colors in each iteration. We wanted to try different versions and had many discussions about the color palette and its impact on the average user. As mentioned earlier, our goal was to make the design as user-friendly as possible without being harsh on the eyes. A blue themed seemed sensible to implement, as it is culturally considered a “calm” color _(Ślusarczyk, n.d.) . By reviewing our sketches, we realized that we almost always included a blue color palette in every iteration. We ultimately settled on a combination of white and blue-black, using a white and blue background.

These are the primary colors we ended up with, as shown in figure 2.4-14:

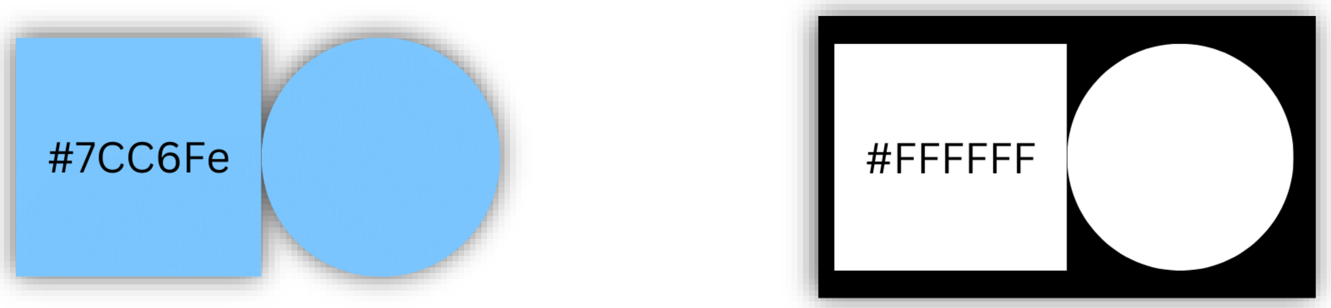


Figure 2.4-14 – Final primary colours.

These the secondary colours, as shown in figure 2.4-15, used for smaller elements such as buttons and text.

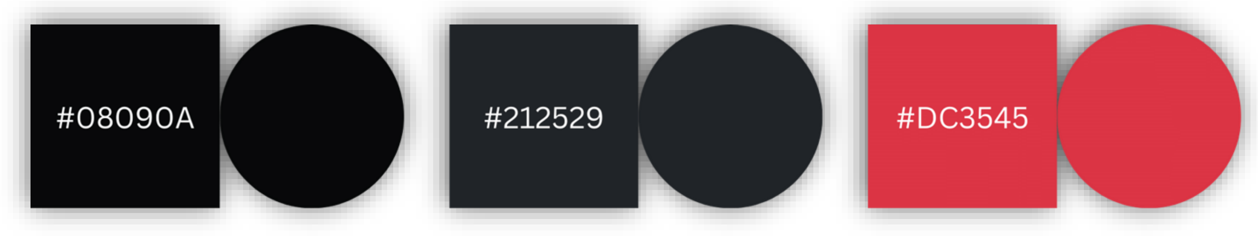


Figure 2.4-15 – Final secondary colours.

2.4.1.8 Font choice

For the text elements on our web application, we had to choose between to fonts sans serif and serif family. In the end we choose a sans serif font family for its clarity and readability. And the decision was made by several other key factors.

Sans serifs are easier to read on digital screens (Huang & Babich, 2023). They have cleaner and simpler lines which helps prevent the letters from blurring or appearing too crowded, especially on lower-resolution screens.

They also give a more modern and minimalistic look the website. This aligns with the aesthetic we wanted for ImageAble, aiming to create a user-friendly and visually appealing interface.

Figure 2.4-16 illustrates the differences between the two styles.

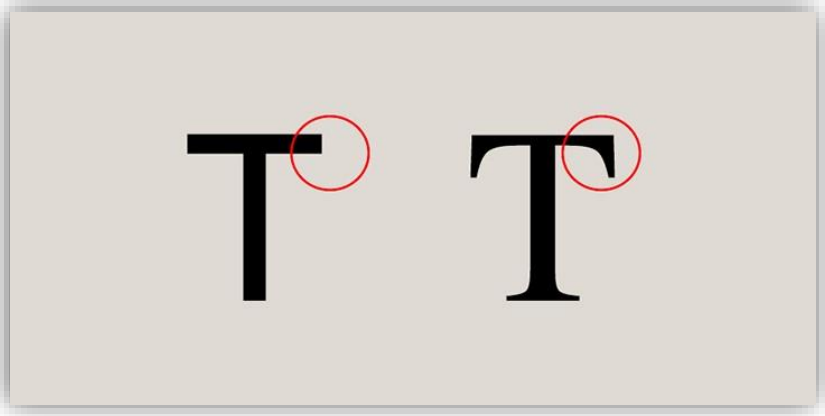


Figure 2.4-16 - Serif vs. sans serif. (Adobe, n.d.)

The font we ultimately went for is called “Inter”, a part of Google Fonts (Google, n.d.). We chose this font because of its readability and style.

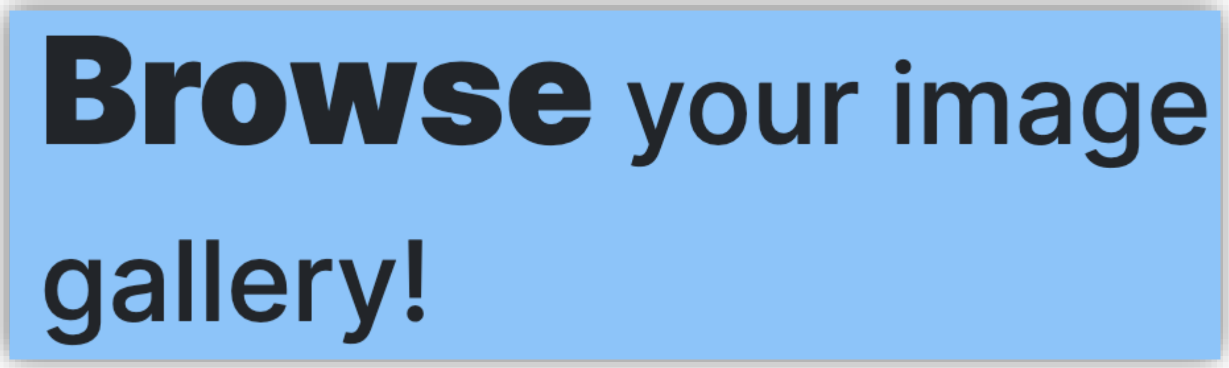


Figure 2.4-17 – Inter font in use.

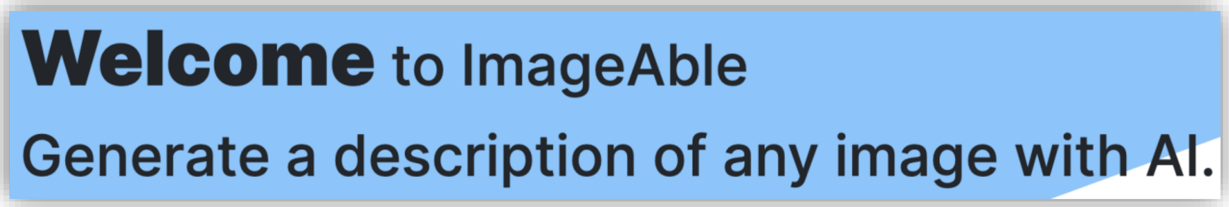


Figure 2.4-18 – Inter font on index page.

2.4.1.9 Buttons

When designing buttons for the ImageAble website, we prioritized a blend of aesthetics, functionality, and compliance with accessibility standards to ensure a cohesive and inclusive user experience. Colours has a significant impact on our perceptions and emotions if it used correctly. When designing the "Delete" button for ImageAble in the gallery, opted for red due its strong association with caution and alerts. This decision follows conventional UI design principles, which recommend red for actions that require extra attention to prevent unintended consequences. According to UX Planet (Babich, 2019) , red effectively draws attention and signals important actions within a user interface, aligning with accessibility standards outlined by WCAG. Figure 2.4-19 illustrate the gallery buttons.

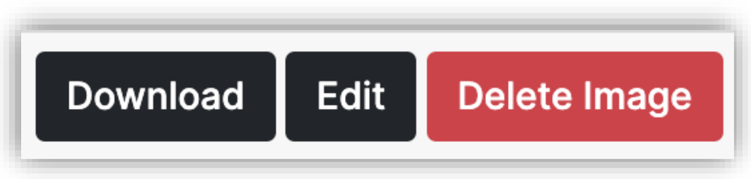
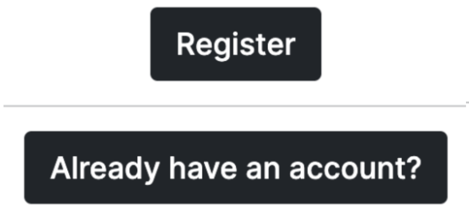
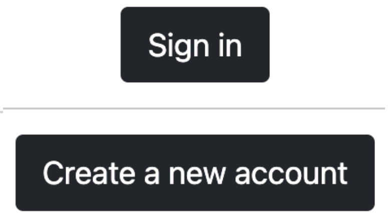


Figure 2.4-19 – Gallery image buttons.

Other buttons such as "Download" and "Edit" in the Gallery, and "Download" and "Edit Description" in main site and other in registering and login are styled consistently with overall design language of ImageAble, using our standard colour palette which enhances the visual harmony across the interface. These buttons are designed to be distinct yet complementary to the "Delete" button, ensuring a cohesive look while maintaining function clarity.





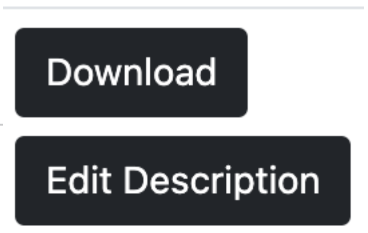


Figure 2.2.4-20 *Figure 2.4-21* *Figure 2.4-22*

We made sure to adding shadows-box beneath buttons like "chose image to upload" see figure 2.4-23 below. And other buttons, to help create sense of depth, making the buttons stand out against the background. This visual enhancement aids usability by making the interface elements feel more interactive and easier to recognize. Using shadows effectively can enhance the user's experience by providing a clear indication of clickable elements (Compassionate Design, 2024).

We choose a button design with rounded corners because they make the interface look more user-friendly and more inviting and make the webapp look more modern. Rounded corners are easier on the eyes than sharp edges and help draw attention to the centre of the buttons, making it clearer where to click. This design choice is in line with modern trends that prefer smoother, softer visual elements (Baig, 2023).

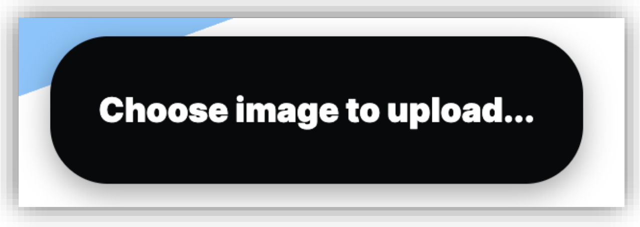


Figure 2.4-23 – Choose image button.

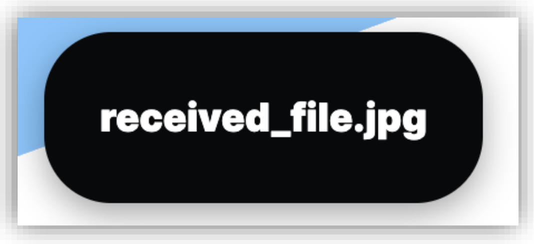


Figure 2.4-24 – Chosen image button.

Hover effects on buttons signal to users that the button is interactive when they place their cursor over it. This feedback is crucial for usability, especially for those less familiar with the interface. The rounded buttons, with their hover effects, offer a visual cue that they are ready for interaction, making the user experience smoother (Javatpoint, n.d.).

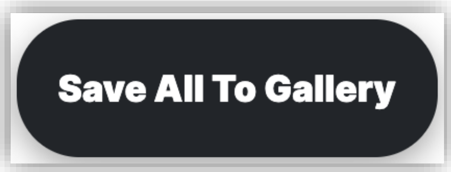


Figure 2.4-25 – Save to gallery.

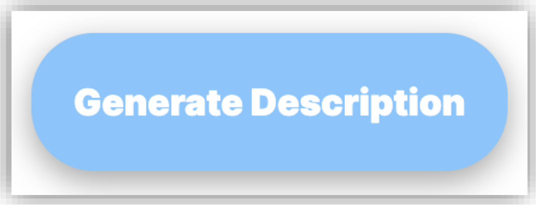


Figure 2.4-26 – Generate description.

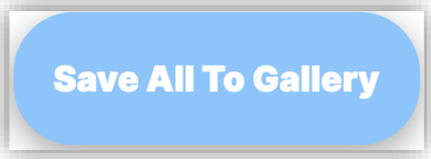


Figure 2.4-27 Save to gallery, hover.

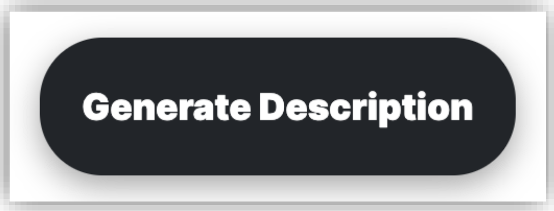
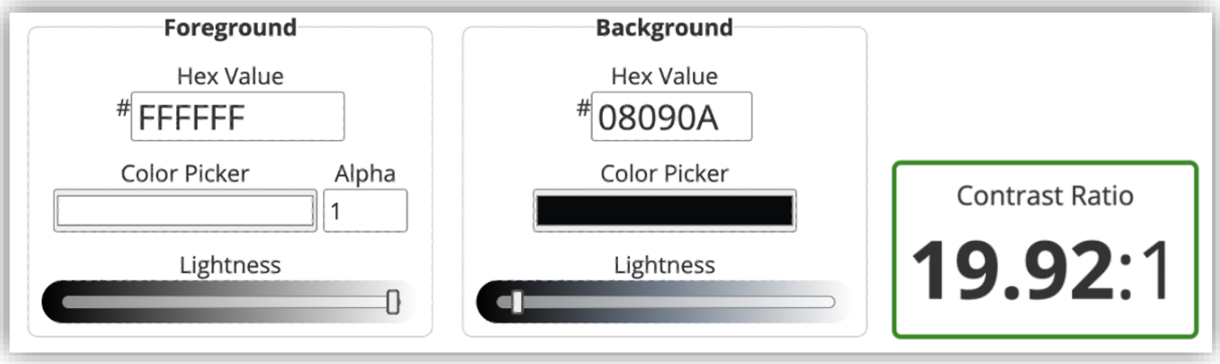


Figure 2.4-28 – Generate description, hover.

2.4.1.10 Design Testing

To verify that the contrast ratios of the elements on our website comply with the WCAG 2.1 guidelines, we used the contrast checker tool available on WebAim.org. This tool calculates the contrast values of the elements, allowing us to ensure they align with the recommendations set forth in WCAG 2.1.


Foreground

Hex Value

FFFFFF

Color Picker

Alpha

1

Lightness

Background

Hex Value

08090A

Color Picker

Lightness

Contrast Ratio

19.92:1

Figure 2.4-29 – Contrast checker, dark button. (WebAIM, n.d.)

The contrast ratio between button foregrounds and backgrounds, comply with accessibility standards. For instance, the button with a dark background has hex code (#08090A) and white text has a contrast ratio of 19.92:1, significantly above the minimum requirement, enhancing readability for all users.

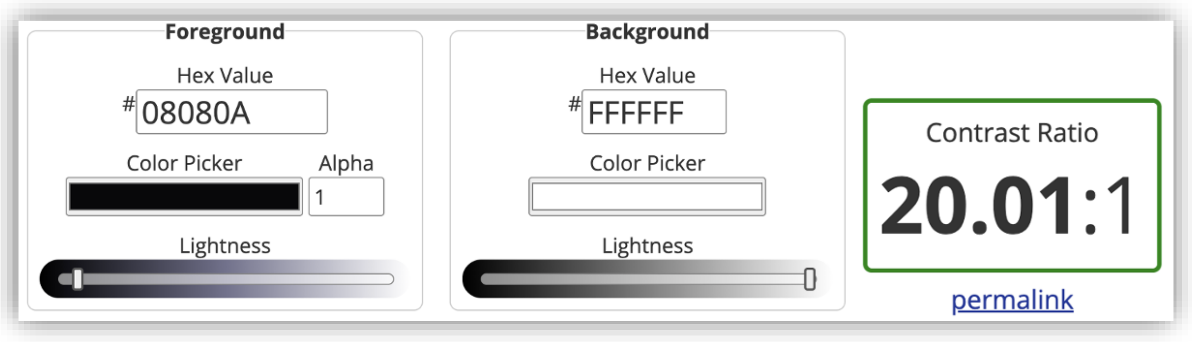


Figure 2.4-30 – Contrast checker, light button (WebAIM, n.d.).

Our black generated text for the images (#08080A) on a white background achieved a contrast ratio of 20.01:1, surpassing the required threshold for optimal readability and ensuring that all textual content is accessible and easy to read for users with visual impairments.

3 Product Documentation

3.1 Solution Introduction

In this chapter, our solution development of the application gave us a high quality of learning curve. Our solution consists of utilizing ASP.NET Core MVC to build an application for greater image accessibility. Creating an architecture for client-side and server-side by making diagrams gives us a visualization of how the solution will be. The process is demanding and in order to finalize a product, we have to go through the architecture and web application development for both front and back-end.

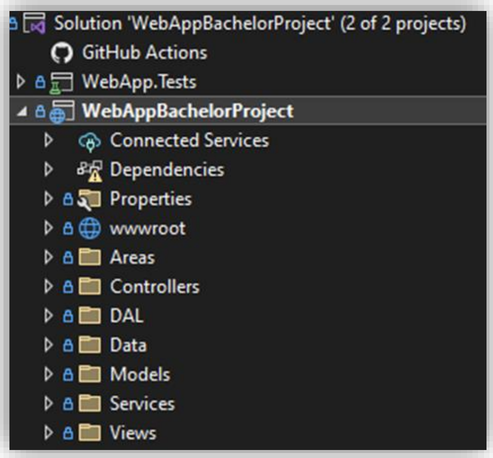
3.2 Web application Development

The development of web application includes how ASP.NET framework which offer a complete set of tools, functionalities and libraries which speed up development and creates a robust and secure web application. In the progress of development, we connect both front-end and back-end, connection with machine learning models and for image description generation and design implementations to create a fully function image accessibility application.

3.2.1 ASP.NET core MVC

In the application development Visual Studio for code editing and ASP.NET framework are the tools used to develop the application. Creating a ASP.NET core MVC project with individual accounts generated fully working application including razor components and EF core also known as Entity framework which is an open-source object-relational mapping(O/RM) (Microsoft, 2021). This enabled us to create, work with databases and write SQL queries.

In figures, which show a generated folder structure that includes folders such as, “Properties”, “wwwroot”, “Areas,” “Controllers”, “Data” , “Models”, “Services” and lastly “Views”. The “DAL” folder is data access layer which provides access between the stored data and the application.

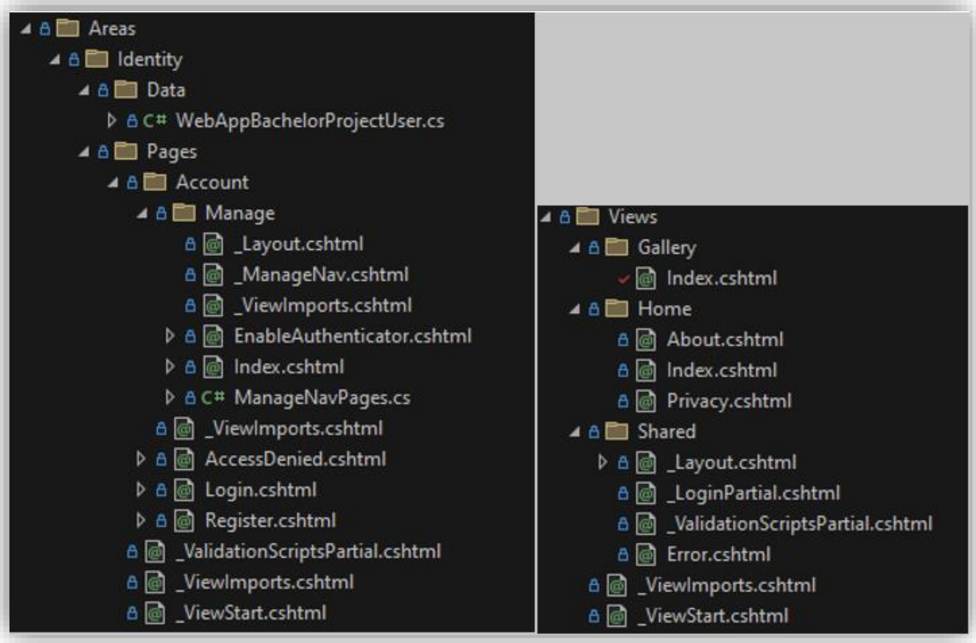


Solution 'WebAppBachelorProject' (2 of 2 projects)
GitHub Actions
WebApp.Tests
WebAppBachelorProject
Connected Services
Dependencies
Properties
wwwroot
Areas
Controllers
DAL
Data
Models
Services
Views

Figure 3.2-1 – Project overview.

The folder structure is a benefit for development where each folder has content of files or components which had its own purposes. The structure made the development effective and observant.

Each folder has relevant files and folders based on its given name. The “Connected services” is a folder consisting of SQL servers that are connected. This folder allowed us to have an overview of connected to databases. Folders such as “Areas” and “Views” consists of. cshtml files and .cs files which was the front-end part of the application, and the folders gave us an overview of each page where “Areas” consists of multiple other folders such as “Identity”, “Data”, “Pages”, “Account” and “Manage” as shown in figure 3.2-2.



```
Areas
  Identity
    Data
      WebAppBachelorProjectUser.cs
    Pages
      Account
        Manage
          _Layout.cshtml
          _ManageNav.cshtml
          _ViewImports.cshtml
          EnableAuthenticator.cshtml
          Index.cshtml
          ManageNavPages.cs
        _ViewImports.cshtml
        AccessDenied.cshtml
        Login.cshtml
        Register.cshtml
      _ValidationScriptsPartial.cshtml
      _ViewImports.cshtml
      _ViewStart.cshtml
```

```
Views
  Gallery
    Index.cshtml
  Home
    About.cshtml
    Index.cshtml
    Privacy.cshtml
  Shared
    _Layout.cshtml
    _LoginPartial.cshtml
    _ValidationScriptsPartial.cshtml
    Error.cshtml
  _ViewImports.cshtml
  _ViewStart.cshtml
```

Figure 3.2-2 – “Areas” and “view” folder structure.

However, the “View” folder is where we stored “Home”, “Gallery” and “Shared”– folders in which contained the most important pages in the applications. Besides those folders, “Controllers”, “Data” and “Model” allowed us to access the backend part of the application. These folders have controllers and models which represent the data and functions applied for each page on the application.

Furthermore, the framework has razor syntax for .NET based code for websites. This syntax consists of Razor markup, HTML and C#. It allows to set C# code into HTML in a readable way which also makes it easier to maintain and design websites.

The framework is flexible and supports a variety of technologies including MVC, Web forms, and ASP. The benefits of ASP.NET are that it has built-in security features such as authentication and authorization. It is also an event-driven model which means it is simpler to create interactive web applications.

3.2.2 Web application architecture

The architecture of the web application consists of client-side(front-end), server-side(back-end), database, API Gateway, AI model integration and Security Components. The client side of the application is the front-end process overall. To understand the architecture of the client-side and server side, diagrams are conducted to clear overview on what the application should include. The architecture of the client involves everything the user interacts with. The framework, diagrams and components are important to address in the process of development. In the server side, which is the backend of the application is the core of functionality. This handles databases, authentication and other operations. The database is essential for managing application data and storing it. Further, the API Gateway is the entry point for client requests, it is important to have route for the gateway so it can communicate with the services. The AI model integration is important for the application where it provides advanced functionalities such as description predictions and evaluation score prediction.

The security components such as authentication and authorization require implementation of authentication methods to ensure authorized users can access, for example the “Gallery” page to save images.

These two sides should communicate with each other in an appropriate way so that the functionalities in the application work properly and accordingly to the expectations of the user. Detailed architecture makes sure that the web application is functional, efficient, and scalable.

3.3 Client-Side (front-end) architecture

The client-side of our web application is built using Native JavaScript and ASP.NET CORE MVC Views. The combination gives us a robust, clean and maintainable structure which increases the developer experience. The client-side follows the Model-View-Controller (MVC) pattern provided by ASP.NET. With Views being the primary focus for the UI.

Views are important components in the front-end which is in charge of rendering the user interface. The role of views to generate HTML markup and handle logic to display for user interaction. ASP.NET, as mentioned, uses Razor syntax to embed server-based code into HTML that provides dynamic content.

The components of the front-end are elements that include buttons, forms and other interactive elements. While JavaScript handles the operations of the component such as uploading an image, validation of form and event handling.

The MVC pattern gives us a fundamental and maintainable overview of the concerns. The architecture is scalable where the services and components are separated which can be developed and tested.

3.3.1 Description of views

The Views folder is structured into several subfolders each having their own purpose in the application. In figure X show all the views subfolders such as “Home”, “Gallery”, “About” and “Shared”. The home folder consists of About.cshtml, Index.cshtml. These two .cshtml files are connected to “site.css” which is a universal CSS file which is integrated into all pages and generated by ASP.NET and we created another CSS files to style only for the home page and About page. These views are also connected to “HomeController.cs”.

The Gallery folder consists of only one “Index.cshtml” which has its own controller called “GalleryController.cs”. The controllers are inside another folder called “Controllers” as mentioned. However, each view is connected to a controller which has its purpose to manage the interaction between the application and the users.

Additionally, the “Shared” folder contains several views, and the following views is:

- Error.cshtml, this view is responsible to display errors that have occurred.
- _Layout.cshtml, this view is responsible for the front-end universal layout of the application.
- _Layout.cshtml.css, this is the css file for styling the layout.
- _LoginPartial.cshtml, this view displays a navigations bar for logged in user,
- _ValidationScriptsPartial.cshtml, this view displays validation and contains a validation script library.

In figure 3.3-1 we have made an overview of the folders.

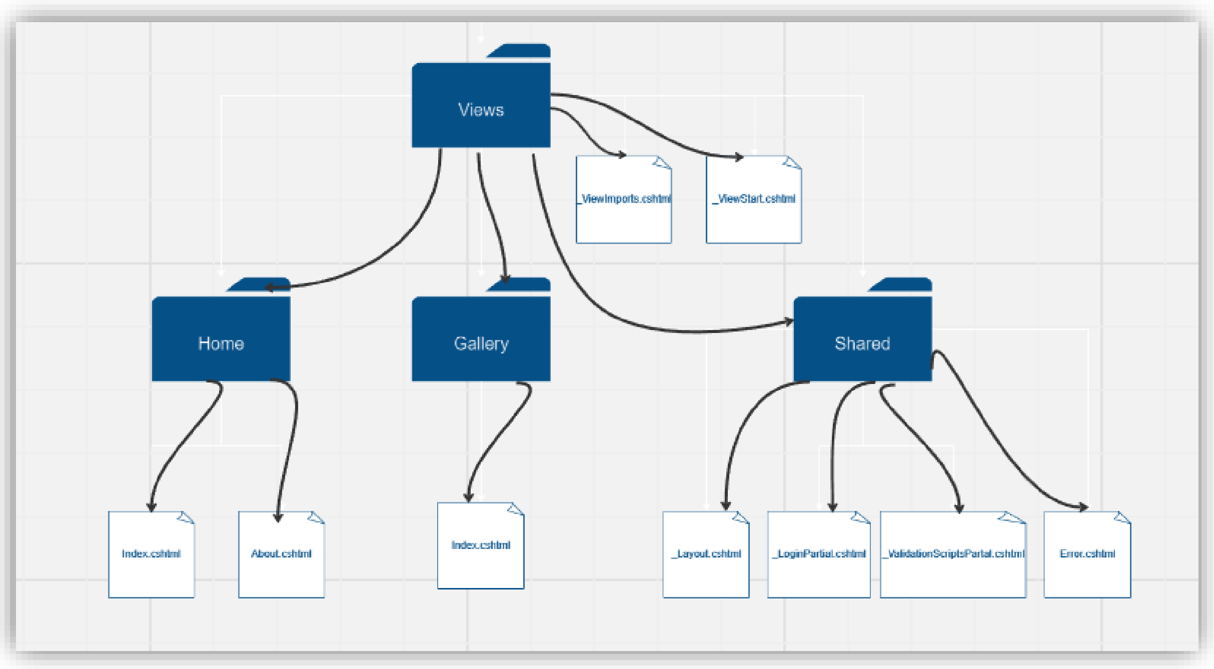


Figure 3.3-1 – "Views" folder overview.

3.3.2 Views > Home

Index.cshtml is the landing page (home page). It provides an overview and navigation options. This is also where the average user will spend most of their time. The Index.cshtml contains: Title of our application and a subtitle "Generate a description", Upload area for uploading images. Under the upload area there are example cards to show how examples of generated descriptions. About.cshtml contains information about the application and what we are trying to achieve. It has cards that are lined up horizontally and the information is split into two parts such as "What is AI?" and "How it works".

3.3.3 Views > Gallery

The gallery folder contains only Index.cshtml which is responsible for displaying the gallery of images a registered user has previously saved. It allows the user to view and manage their saved images. The main component of this view is the search bar where users are able to search by string, meaning that a user can search by description.

3.3.4 Views > Shared

The shared folder includes following views:

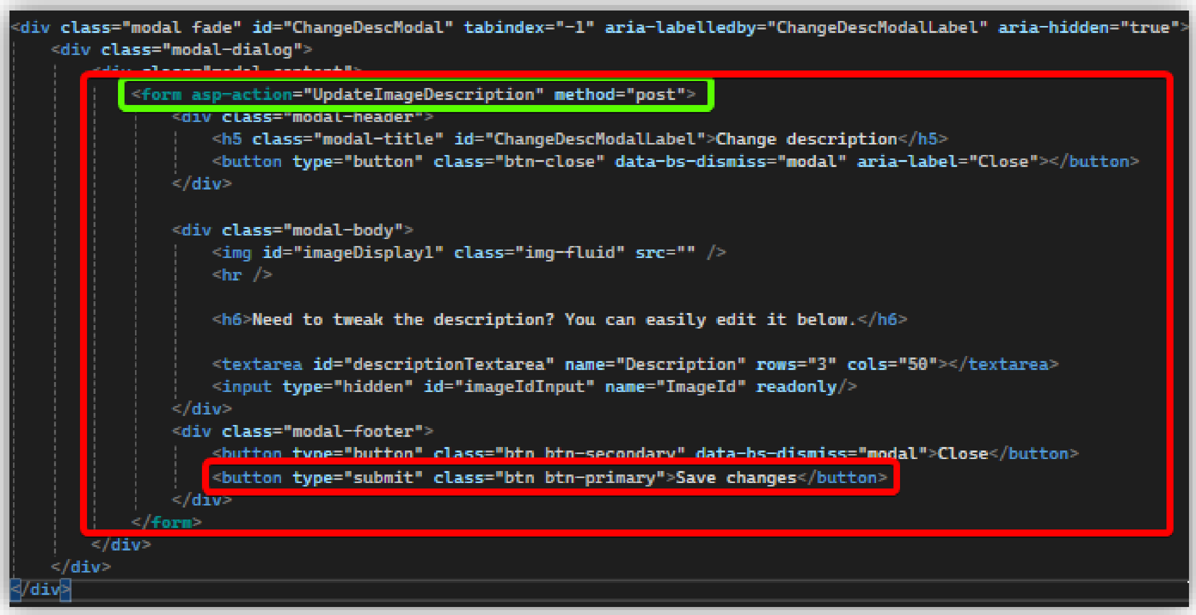
- _Layout.cshtml is the master layout view. This is where common structures for all the pages are coded. That includes the header, navigation bar, and links to different scripts.
- _LoginPartal.cshtml is where the links and navigation bar for users that are logged in. The navigation bar includes all the links for the different views. Additionally, the navigation bar also includes the username and log out button on the right side.
- _ValidationScriptsPartial.cshtml is the part where it displays validation and contains a validation scripts library. These scripts ensure the inputs are validated before they reach the server. It also helps prevent invalid data.
- Error.cshtml is the view that displays errors when errors occur. This view is rendered when an unhandled excpetion occurs which provides a user-friendly message.
- _ViewImports.cshtml and _ViewStart.cshtml are views that are used to manage namespaces and directives. These views are also responsible to apply settings within all views.

3.3.5 User Interaction and Data Flow

User interaction on the client-side is handled by JavaScript and ASP.NET Asp anchor helper. User interaction with JavaScript such as event handling that calls functions when a user clicks a button or form submissions. The application has various user interaction. Most of the user interaction will be on the landing page also known as “Home” page. In this page we have following user interactions:

- File Upload
- Collapsible Options
- Model selection.
- Edit description.
- Buttons

The applications also use ASP.NET Anchor tag helpers which enhances the standard HTML anchor by adding new attributes (Microsoft, 2022). The attributes has a prefixed name that is” asp-”. The asp- tags provide behaviours and attributes to the html anchor for example in the figure X which is a snippet of “Index.cshtml” in “Gallery” that contains a form that uses an asp anchor helper – “asp-action”. This asp anchor helper automatically triggers method located in the “GalleryController” since it is already in “Index.cshtml”. To send the data, the button tag is defined as a “submit” which will activate the input when the input is filled.



```
<div class="modal fade" id="ChangeDescModal" tabindex="-1" aria-labelledby="ChangeDescModalLabel" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <form asp-action="UpdateImageDescription" method="post">
                <div class="modal-header">
                    <h5 class="modal-title" id="ChangeDescModalLabel">Change description</h5>
                    <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
                </div>

                <div class="modal-body">
                    <img id="imageDisplay1" class="img-fluid" src="" />
                    <hr />

                    <h6>Need to tweak the description? You can easily edit it below.</h6>

                    <textarea id="descriptionTextarea" name="Description" rows="3" cols="50"></textarea>
                    <input type="hidden" id="imageIdInput" name="ImageId" readonly/>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>
                    <button type="submit" class="btn btn-primary">Save changes</button>
                </div>
            </form>
        </div>
    </div>
</div>
```

Figure 3.3-2 – ASP anchor helper example.

3.3.5.1 Uploading Images

The upload diagram in figure 3.3-3 is a summary of how the user can interact with upload images component. The interaction is clicking on the “Upload image” which triggers the file selection from user’s operative system. The user can then select multiple images or one image to upload. If the selected file is not an image format it will return an error message and return back to file selection. The user will have to choose a file that is an image then the name of the image will display at the application, however if the user has selected multiple images the number of files will display. The uploaded image(s) is also displayed in a table and a “Generate description” button will also appear.

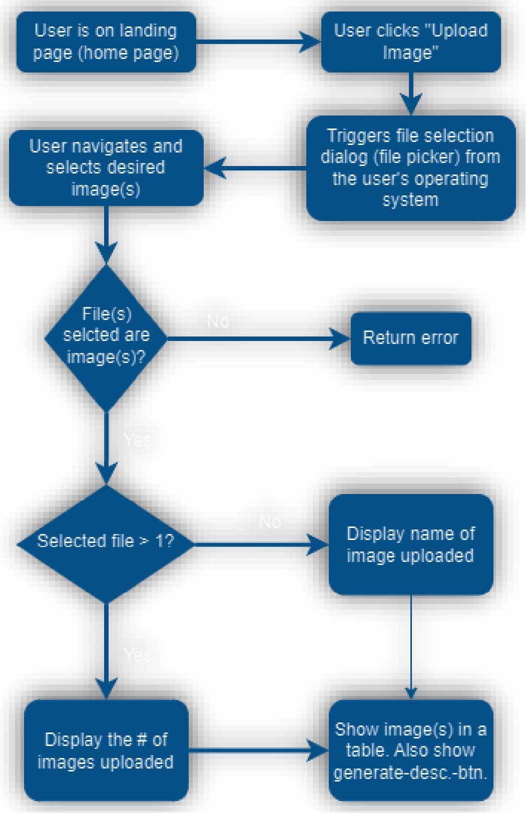
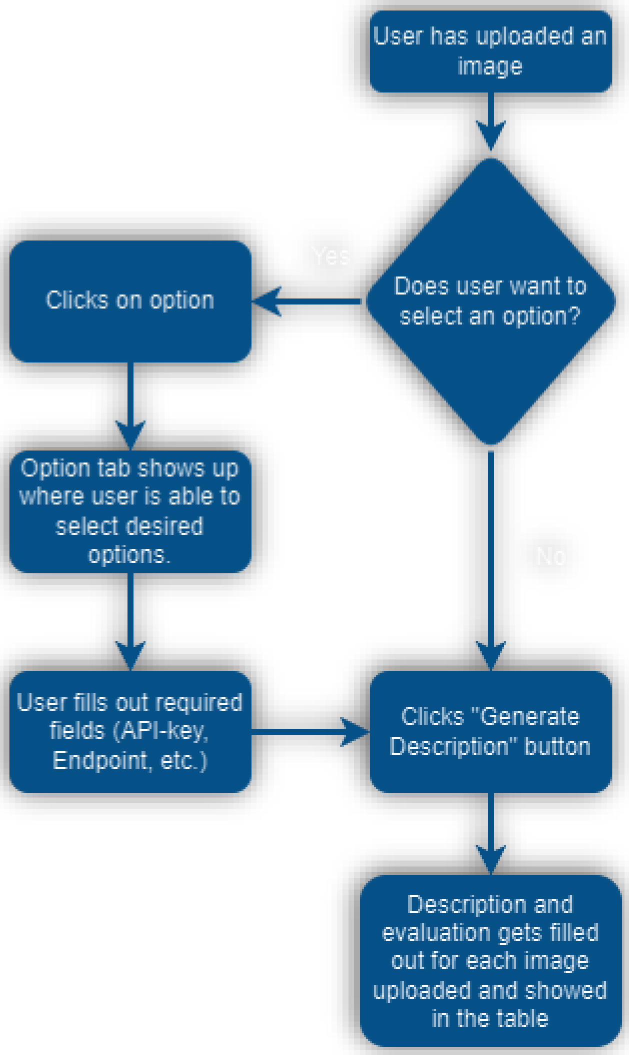

User is on landing page (home page)
User clicks "Upload Image"
Triggers file selection dialog (file picker) from the user's operating system
User navigates and selects desired image(s)
File(s) selcted are image(s)?
No
Return error
Yes
Selected file > 1?
No
Display name of image uploaded
Yes
Display the # of images uploaded
Show image(s) in a table. Also show generate-desc.-btn.

Figure 3.3-3 – User interaction – Upload images.

3.3.5.2 Generate Description and Evaluations for Images

The diagram in figure 3.3-4 demonstrates the next interaction when an image is uploaded. If a user wants to change the option, the user will have to click the “Option” button and then select the options such as using own model API, or other models already implemented. After selecting desired options, the user can click on the “Generate description” button. A description will be generated and displayed with evaluation score next to the description.



User has uploaded an image
Does user want to select an option?
Yes
Clicks on option
Option tab shows up where user is able to select desired options.
No
User fills out required fields (API-key, Endpoint, etc.)
Clicks "Generate Description" button
Description and evaluation gets filled out for each image uploaded and showed in the table

Figure 3.3-4 – User interaction – Generate description & evaluation.

3.3.5.3 Download image with metadata.

As shown in Figure 3.3-5, the diagram demonstrates user interaction where the user can download images. This interaction requires the user to wait until the evaluation and description generation is finished until it finally generates a “Download” button. This button gives the user opportunity to download and image with metadata. If a user clicks “Download” the web browser initiates the download, and the image is saved to user’s default download folder.

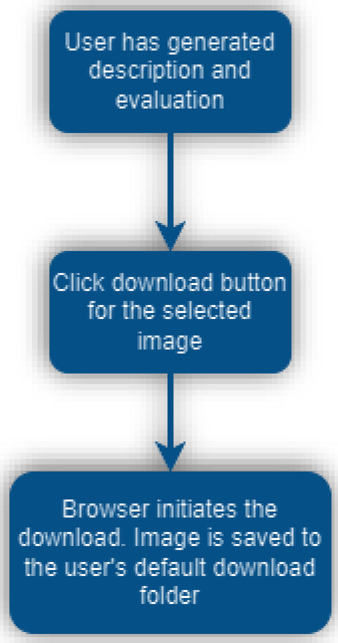

User has generated description and evaluation
Click download button for the selected image
Browser initiates the download. Image is saved to the user's default download folder

Figure 3.3-5 – User interaction – Download image w/ metadata.

3.3.5.4 Save Images to Gallery with Metadata

Saving images to the gallery part of our application is also a user interaction implemented into to the application. The diagram in figure 3.3-6 shows how an image can be saved to gallery. The user needs to upload and generate a description of the image and evaluation score for the description. Afterwards, the user can save images by clicking the “Save image”. Additionally, the user has to be logged in to be able to save it on gallery, if the user is not logged in the user is sent to log in page.

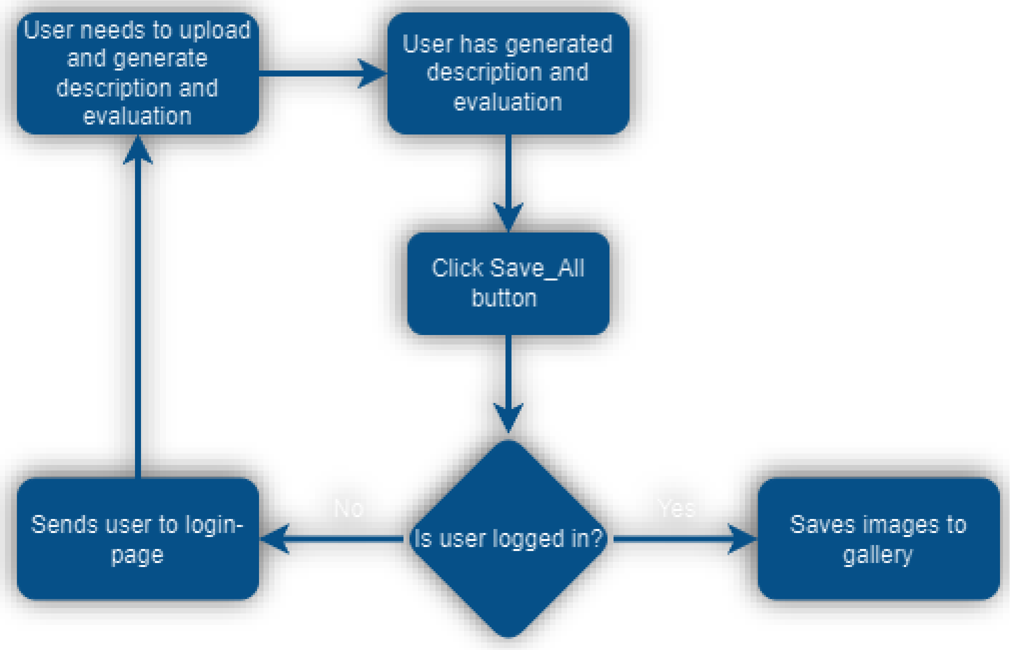

User needs to upload and generate description and evaluation
User has generated description and evaluation
Click Save_All button
Is user logged in?
No
Yes
Sends user to login-page
Saves images to gallery

Figure 3.3-6 – User interaction – Save all images to gallery w/ metadata.

3.3.5.5 Edit the Generated Description

Changing the description is an option the user can choose to do. The Figure 3.3-7 diagram below demonstrates how this interaction will be. When the user has uploaded and generated a description and an evaluation score the user will get the option to edit the description if they are not satisfied. By clicking the “Change description” a modal-view pops up with input field to change description. When the user has edited and satisfied, the user can choose between saving changes or cancelling the changes. If the save changes button is clicked the modal-view will close edited description will display with a new evaluation score.

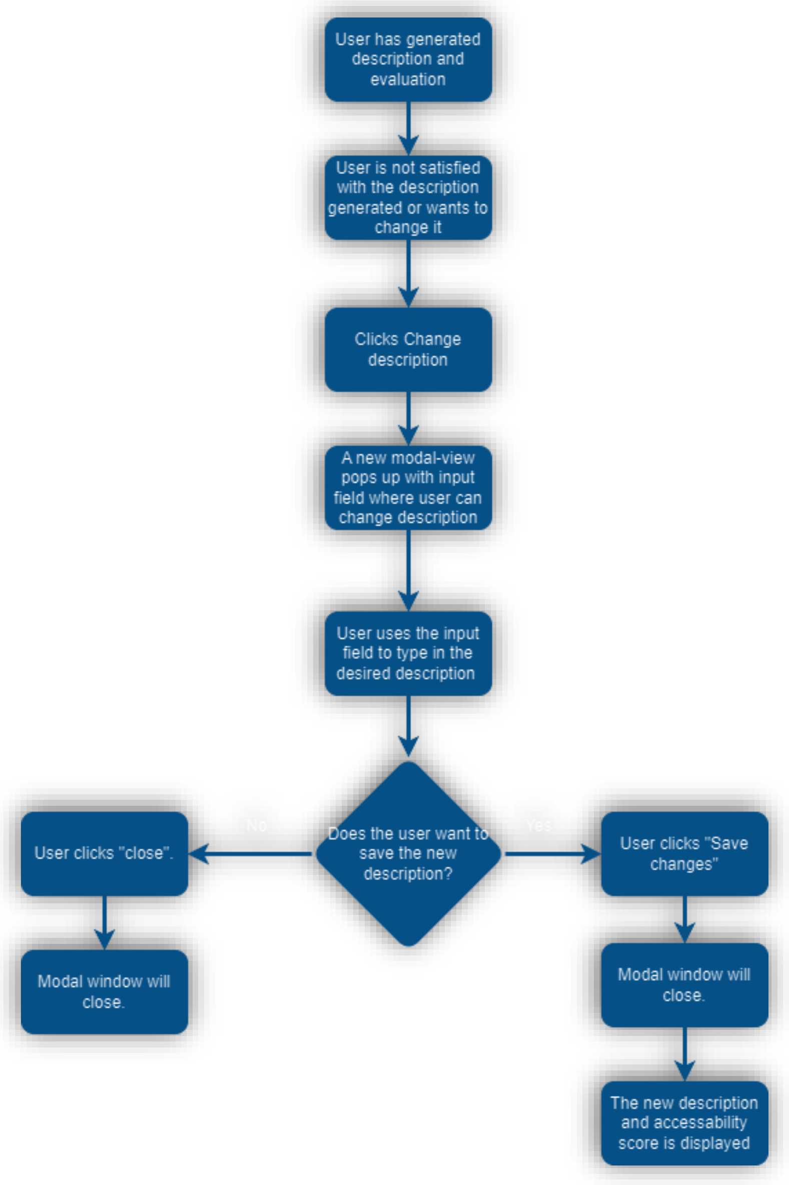

User has generated description and evaluation
User is not satisfied with the description generated or wants to change it
Clicks Change description
A new modal-view pops up with input field where user can change description
User uses the input field to type in the desired description
Does the user want to save the new description?
No
Yes
User clicks "close".
Modal window will close.
User clicks "Save changes"
Modal window will close.
The new description and accessability score is displayed

Figure 3.3-7 – User interaction – Change generated description.

3.3.5.6 Search Interaction

This interaction is about the search as shown in figure 3.3-8. The diagram shows when the user has access to the gallery, the user can search for images they have saved in the gallery page. Users have to write a search word and click on the "Search" button to activate the search function that filters by the search word. Furthermore, if a search filter finds a corresponding text it will display the image(s). If it does not find any image the user can click "Back to full list" or else, it will display no images.

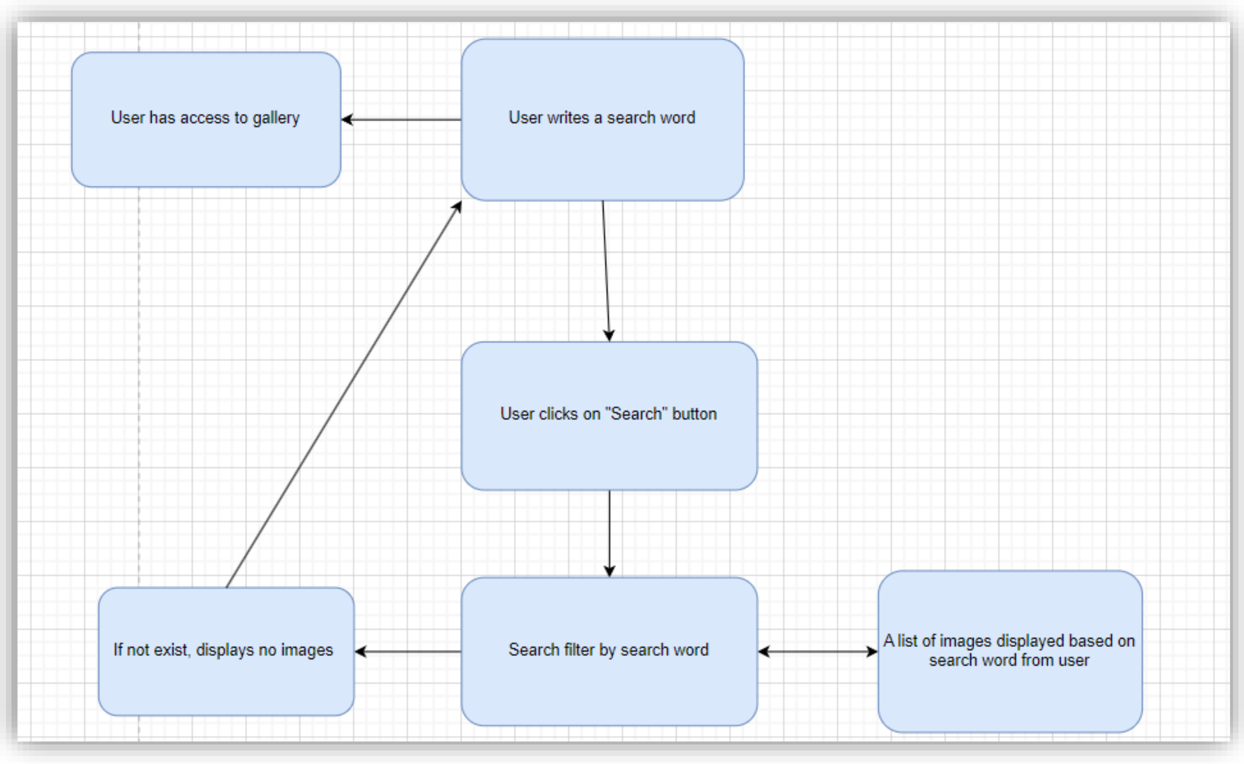

User has access to gallery
User writes a search word
User clicks on "Search" button
If not exist, displays no images
Search filter by search word
A list of images displayed based on search word from user

Figure 3.3-8 – User interaction – Search.

3.3.5.7 Authentication Interaction

This is the interaction flow chart as shown in figure 3.3-9 and it is about authentication. If a user has an account they can write in their log in credentials, if the credentials are wrong the user needs to try again. A "forgot password" link is accessible if the user has forgotten the password. A new password can be made, and the user will be sent back to start. However, if the user does not have an account the user can simply register and log in.

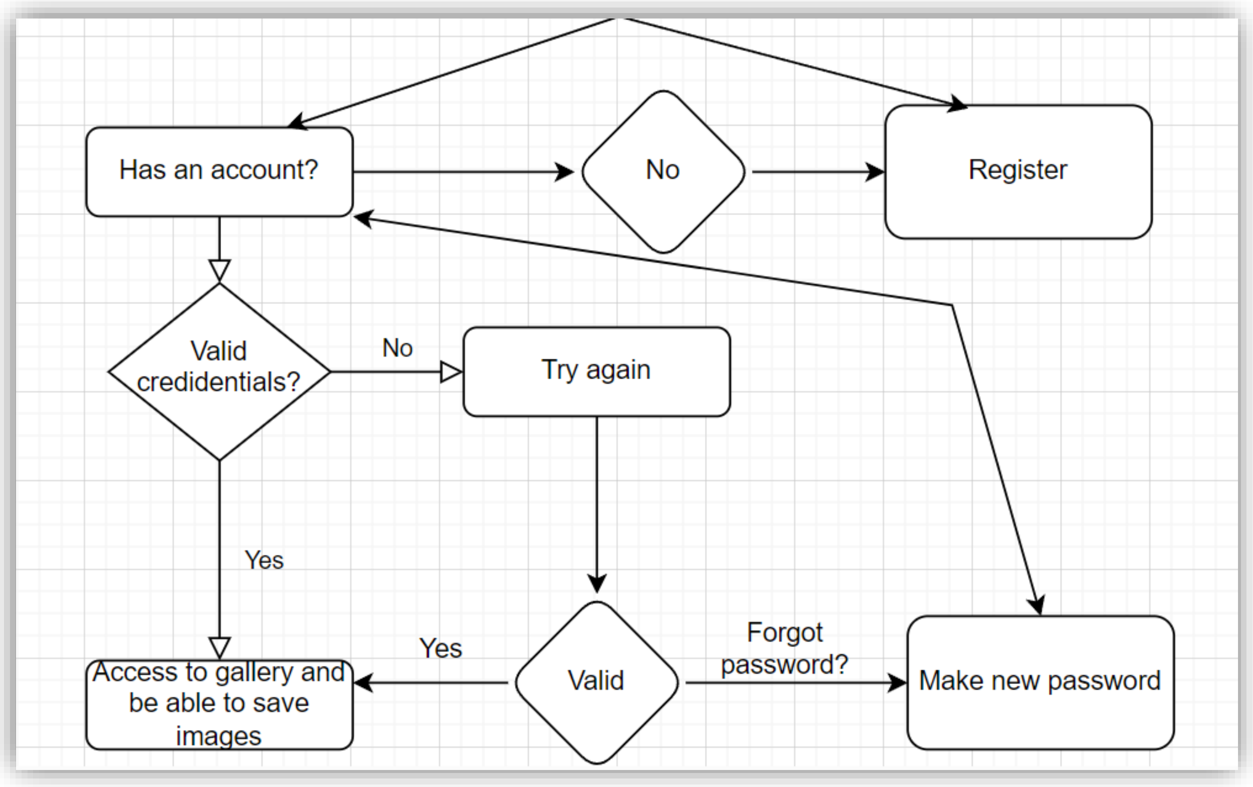
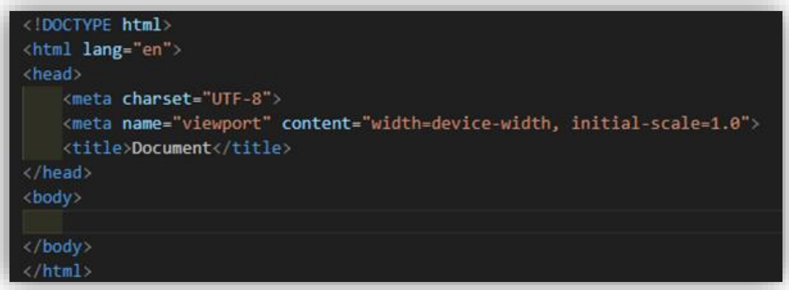


Figure 3.3-9 – User interaction – Authentication.

3.4 Client-side (front end) Development

Front-end represents the graphical user interface and is responsible for how the application looks. The front-end structure of the application is created with ASP.NET framework and includes HTML, CSS and JavaScript to create the user interface of the application. Therefore, we are able to code with HTML to create simple and basic structures of the website as shown in figure 3.4-1. This is a snippet of an HTML file with necessary tags that creates a basic structure of a website.

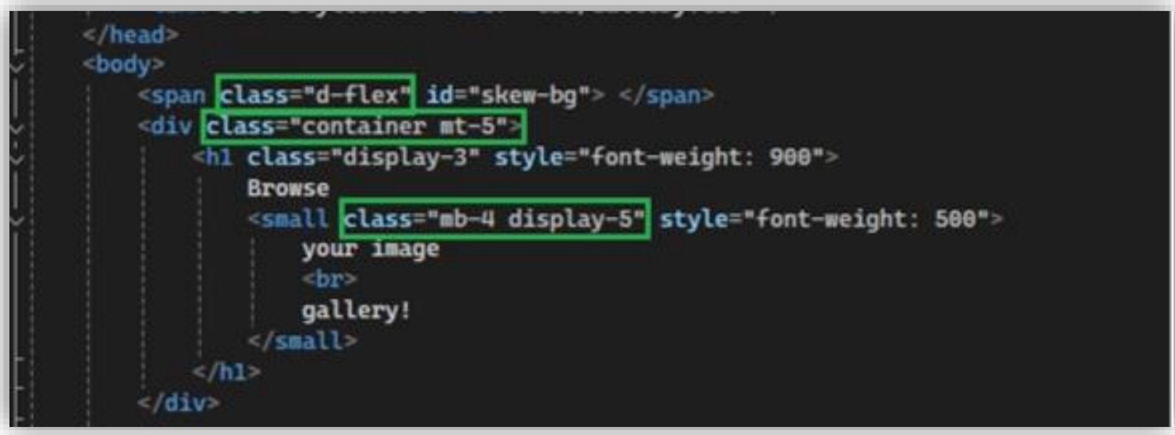


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

Figure 3.4-1 – HTML code snippet.

CSS is a way to style, shape, and enhance HTML blocks. The stylesheet is to give it a more polished and also allows interactive elements such as background color, shapes and graphic. JavaScript is also used in the front-end part where it gives elements such as buttons an interactive function.

Furthermore, since we used ASP.NET core framework which includes Bootstrap library, a powerful front-end toolkit, we could speed up the process and style the website using Bootstrap. The toolkit utilizes a prebuilt grid system, components and JavaScript plugins.



```html
</head>
<body>
    <span class="d-flex" id="skew-bg"> </span>
    <div class="container mt-5">
        <h1 class="display-3" style="font-weight: 900">
            Browse
            <small class="mb-4 display-5" style="font-weight: 500">
                your image
                <br>
                gallery!
            </small>
        </h1>
    </div>
```

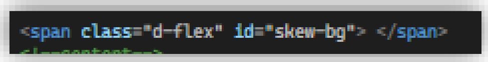
Figure 3.4-2 – Flex container.

In figure above, it shows a snippet from the gallery page where bootstrap classes are used to style the span-tag, div-tag, and small-tag. The bootstrap framework is used throughout the application to style it.

3.4.1 Home page development

In the first stages of development, we began to edit and create the contents of our “Home” page. This page will contain the most important components of the application which consists of uploading area, drop down menu to choose machine learning models, description and description evaluation scores.

The implementation according to the finalized sketch was about taking it slow, where we made each component one by one. Before making the components, we had to find the right font for the text and make a background to match the finalized sketch. We agreed to use a font from Google fonts which was called “Inter” this is a font that was readable for us and matches font for the website and an accurate font style with the sketch.

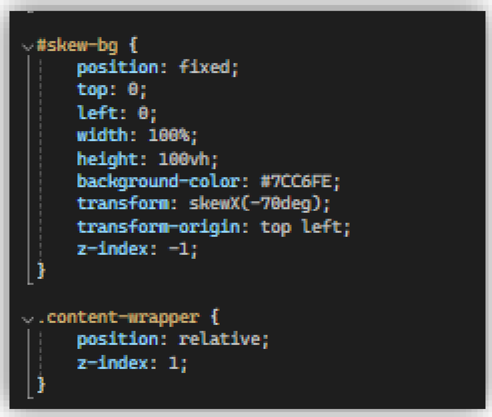


```
<span class="d-flex" id="skew-bg"> </span>
```

Figure 3.4-3 – Span-tag.

This is a snippet (Figure 3.4-3) of a span-tag that is the background, and it is applied for each page in the application. Span-tag is an inline container which is perfect for making a background.

In the figure below, CSS is used to style the span-tag and form a background with a skew with a light blue background colour taken from our colour palette.

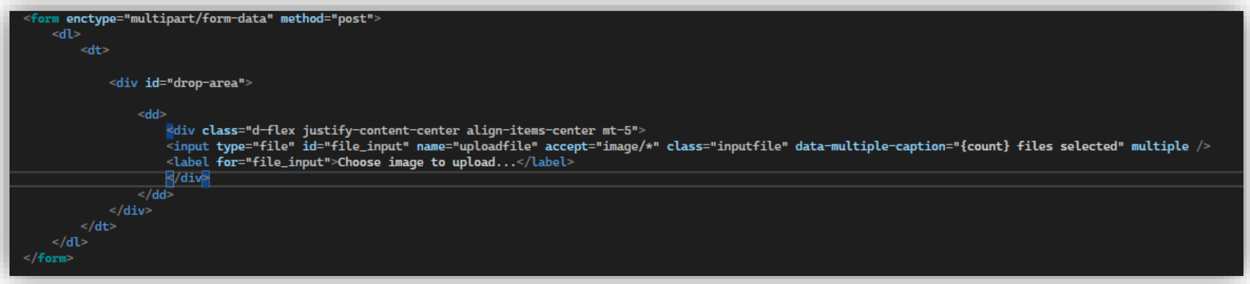


```
#skew-bg {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100vh;
    background-color: #7CC6FE;
    transform: skewX(-70deg);
    transform-origin: top left;
    z-index: -1;
}

.content-wrapper {
    position: relative;
    z-index: 1;
}
```

Figure 3.4-4 – CSS styling.

3.4.2 Home page components & styling

The first component we made was image upload area, this is the area where a user could upload their image to begin generating a description. The component needed to have a colour from our colour palette, and it should be as accurate and similar as possible to the finalized sketch. Therefore, the drop area has a black background and is in the centre of the application.



```html
<form enctype="multipart/form-data" method="post">
    <dl>
        <dt>

            <div id="drop-area">

                <dd>
                    <div class="d-flex justify-content-center align-items-center mt-5">
                    <input type="file" id="file_input" name="uploadfile" accept="image/*" class="inputfile" data-multiple-caption="{count} files selected" multiple />
                    <label for="file_input">Choose image to upload...</label>
                    </div>
                </dd>
            </div>
        </dt>
    </dl>
</form>
```

Figure 3.4-5

In this snippet, a drop area is made with ”div” ,”dl”,” dt“, and ”dd” tags with Bootstrap classes to make the component responsive. Inside of the last div-tags, input-tag and label-tag are used to create an upload area. Giving the input an id gives it a purpose later in JavaScript part to give the area a function. Besides, the styling of the input is styled with CSS to give the right colour and font-size. The snippet (Figure 3.4-6) shown below is the result of our upload area.

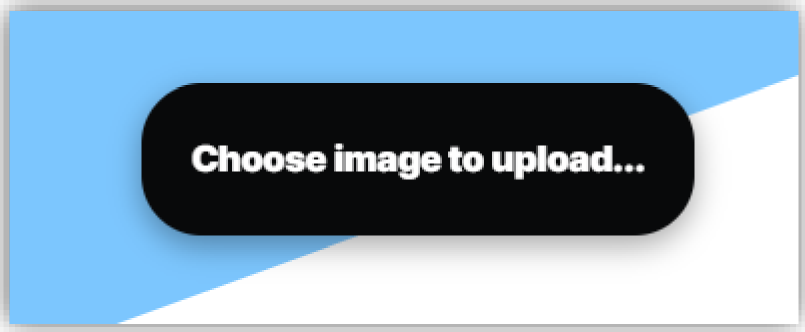
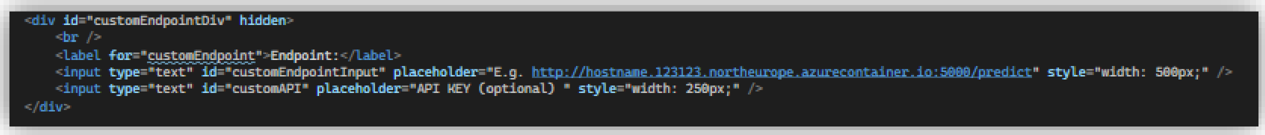


Figure 3.4-6 – Choose image button.

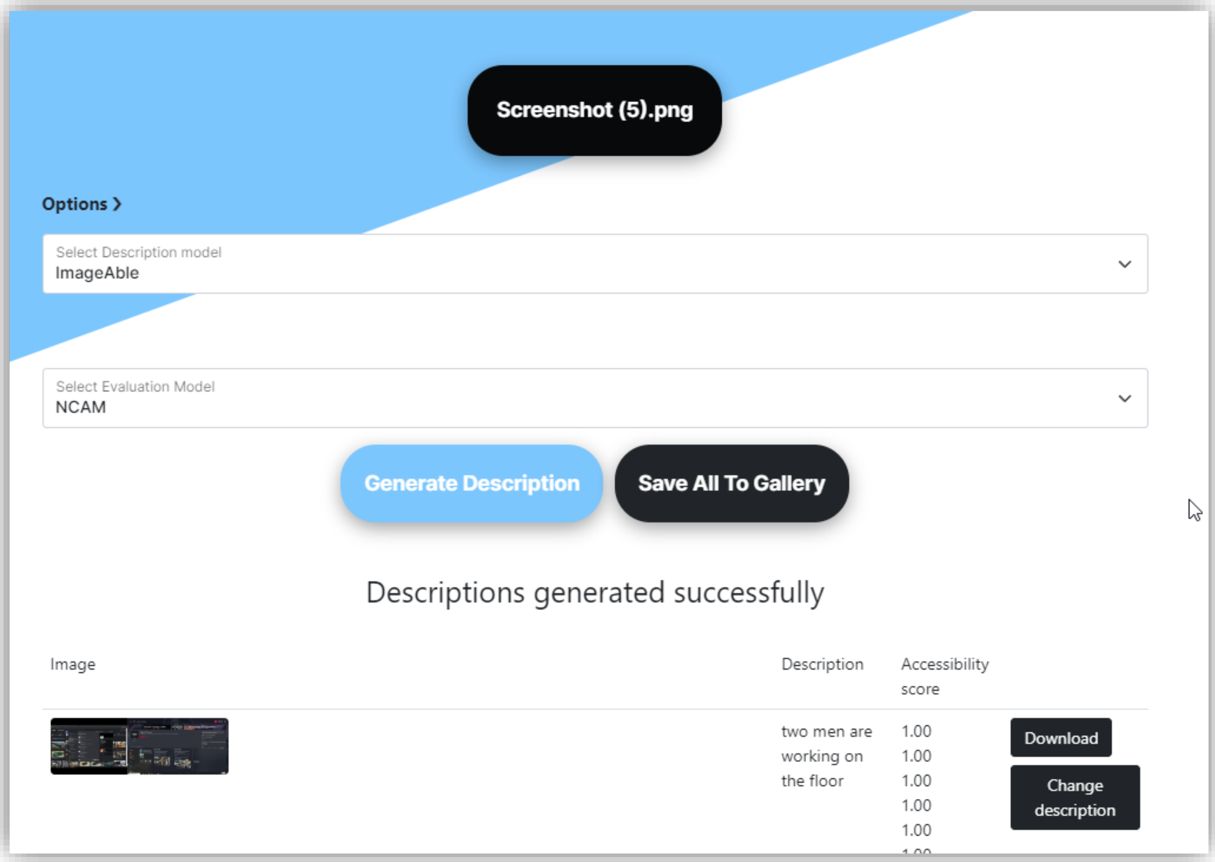
The second component is the description generation itself. This is the most demanding component for both front and back-end part. In figure 3.4-7 the implementation of a button to generate a description is straight forward using label and input tags that includes our model API which is styled with Bootstrap classes.



```
<div id="customEndpointDiv" hidden>
    <br />
    <label for="customEndpoint">Endpoint:</label>
    <input type="text" id="customEndpointInput" placeholder="E.g. http://hostname.123123.northeurope.azurecontainer.io:5000/predict" style="width: 500px;" />
    <input type="text" id="customAPI" placeholder="API KEY (optional) " style="width: 250px;" />
</div>
```

Figure 3.4-7 – Custom endpoint code snippet.

The model selection input and generation button are hidden until the user chooses the image. Which means we had to styled it in JavaScript. To keep the colour scheme, the button is a blue same blue colour as the background colour, but we made sure the button is on the white background to show contrast. Furthermore, the button has a hover function where it changes colour when hovered. The “Options” part is where a user can choose between models. This is implemented as an additional component which is not intended in the first place. Font style, weight and increased size of the inputs is to match the application, also make it more accessible.



Screenshot (5).png

Options ›

Select Description model: ImageAble

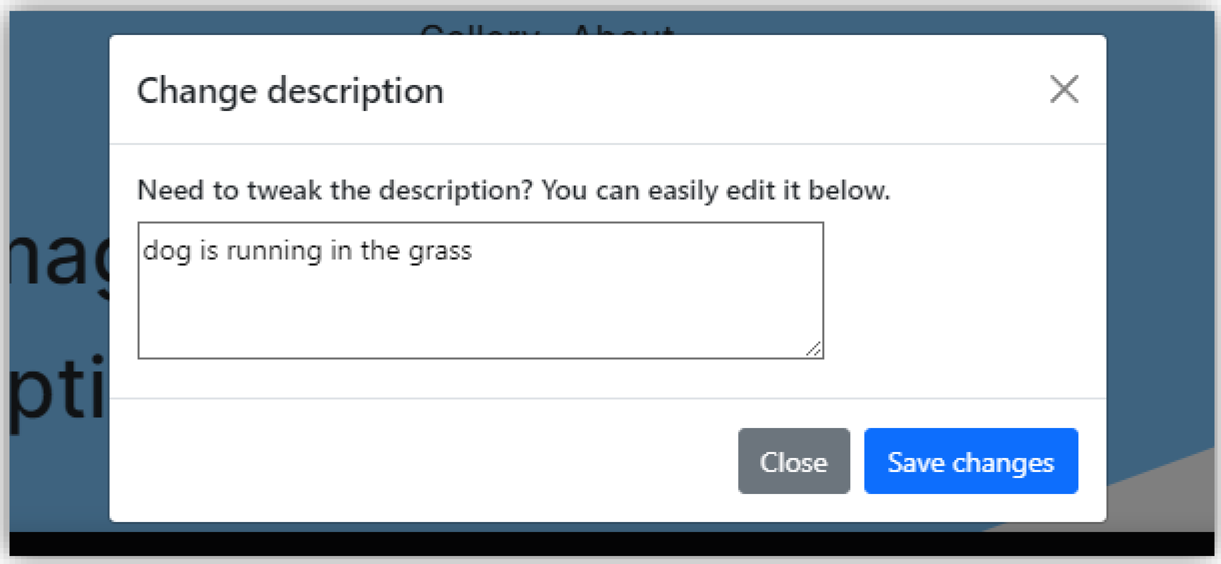
Select Evaluation Model: NCAM

Descriptions generated successfully

Image	Description	Accessibility score	
	two men are working on the floor	1.00 1.00 1.00 1.00 1.00	Download Change description

Figure 3.4-8 – “Generate description” page.

Further, since the “Download” and “Change description button is generated as shown in figure X. It is also style by JavaScript with bootstrap classes and a black colour chosen according to the colour scheme. The “Change description” will show a pop-up when clicked as shown in figure 3.4-9.


Change description

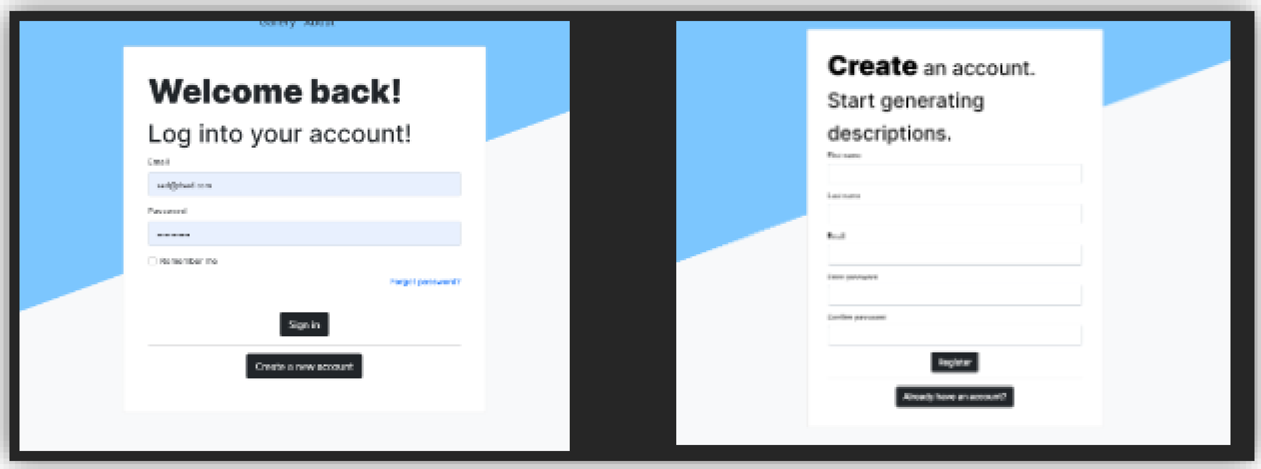
Need to tweak the description? You can easily edit it below.

dog is running in the grass

Figure 3.4-9 – Change description.

3.4.3 Authentication pages styling

As mentioned, ASP.NET core MVC with authentication generates a fully functional log-in, register and account management pages. These pages had already fully functional forms that we simply could use Bootstrap classes and style it. The snippet shown below is of each authentication page where Bootstrap is used.


Welcome back!

Log into your account!

Create an account.

Start generating descriptions.

Figure 3.4-10 – Authentication pages.

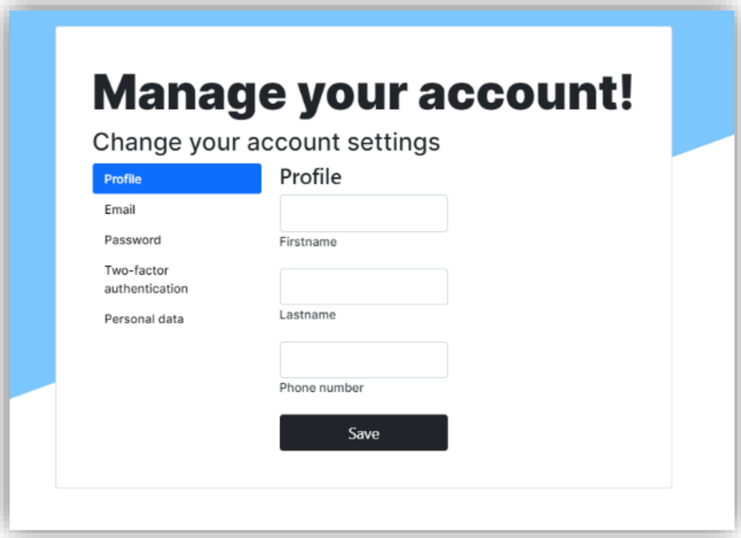
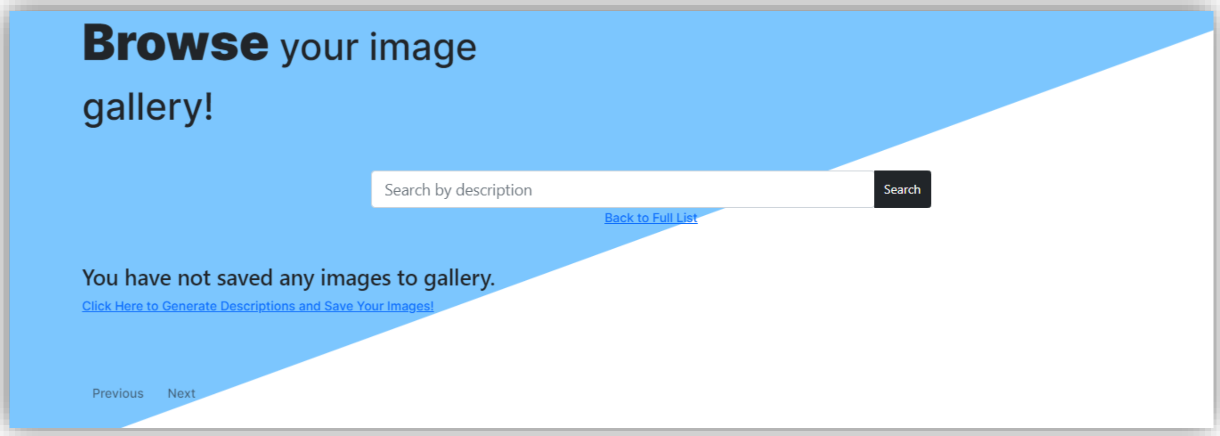


Figure 3.4-11 – Manage account.

To ensure to keep identical style in the application, the forms in the log-in, register and account management are styled inside a container with white background and border lines to separate the forms from the background. This page requires authentication to access and only shows when a user is logged in.

3.4.4 Gallery page Development

The gallery page contains images which each user has saved and a search bar that filters out by description and date. This page requires authentication to access and only shown when a user is logged in.


Browse your image gallery!

Search by description
Search
Back to Full List

You have not saved any images to gallery.
Click Here to Generate Descriptions and Save Your Images!

Previous Next

Figure 3.4-12 Gallery page.

Even though, “Gallery” page does not contain much and only a search bar an if –statement and foreach loop is implemented in the code as shown in figure below. This is to check if a user has any saved images in gallery, if not a text will appear to tell the user to upload an image and generate a description to save in gallery.

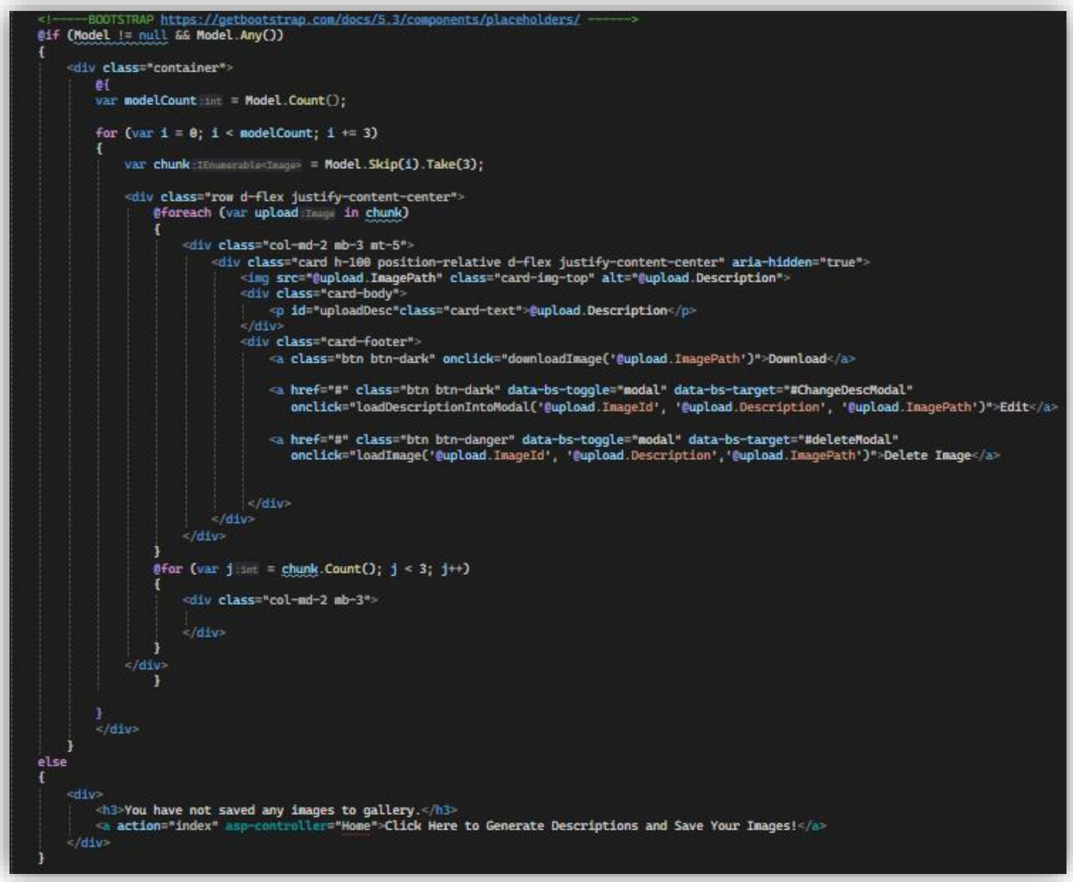

```
<!-----BOOTSTRAP https://getbootstrap.com/docs/5.3/components/placeholders/ ------->
@if (Model != null && Model.Any())
{
    <div class="container">
        @{
        var modelCount :int = Model.Count();

        for (var i = 0; i < modelCount; i += 3)
        {
            var chunk :IEnumerable<Image> = Model.Skip(i).Take(3);

            <div class="row d-flex justify-content-center">
                @foreach (var upload :Image in chunk)
                {
                    <div class="col-md-2 mb-3 mt-5">
                        <div class="card h-100 position-relative d-flex justify-content-center" aria-hidden="true">
                            <img src="@upload.ImagePath" class="card-img-top" alt="@upload.Description">
                            <div class="card-body">
                                <p id="uploadDesc"class="card-text">@upload.Description</p>
                            </div>
                            <div class="card-footer">
                                <a class="btn btn-dark" onclick="downloadImage('@upload.ImagePath')">Download</a>

                                <a href="#" class="btn btn-dark" data-bs-toggle="modal" data-bs-target="#ChangeDescModal"
                                   onclick="loadDescriptionIntoModal('@upload.ImageId', '@upload.Description', '@upload.ImagePath')">Edit</a>

                                <a href="#" class="btn btn-danger" data-bs-toggle="modal" data-bs-target="#deleteModal"
                                   onclick="loadImage('@upload.ImageId', '@upload.Description','@upload.ImagePath')">Delete Image</a>


                            </div>
                        </div>
                    </div>
                }
                @for (var j :int = chunk.Count(); j < 3; j++)
                {
                    <div class="col-md-2 mb-3">

                    </div>
                }
            </div>
                }

        }
        </div>
    }
else
{
    <div>
        <h3>You have not saved any images to gallery.</h3>
        <a action="index" asp-controller="Home">Click Here to Generate Descriptions and Save Your Images!</a>
    </div>
}
```

Figure 3.4-13 – Gallery page code snippet.

The @if-statement and @foreach is possible due to razor syntax which allows a combination of C# and HTML within the same file. This code loops through the “Model” which represents data passed from the controller and then creates rows and columns, if there are no images saved a text will show up to tell the user there are no images saved.

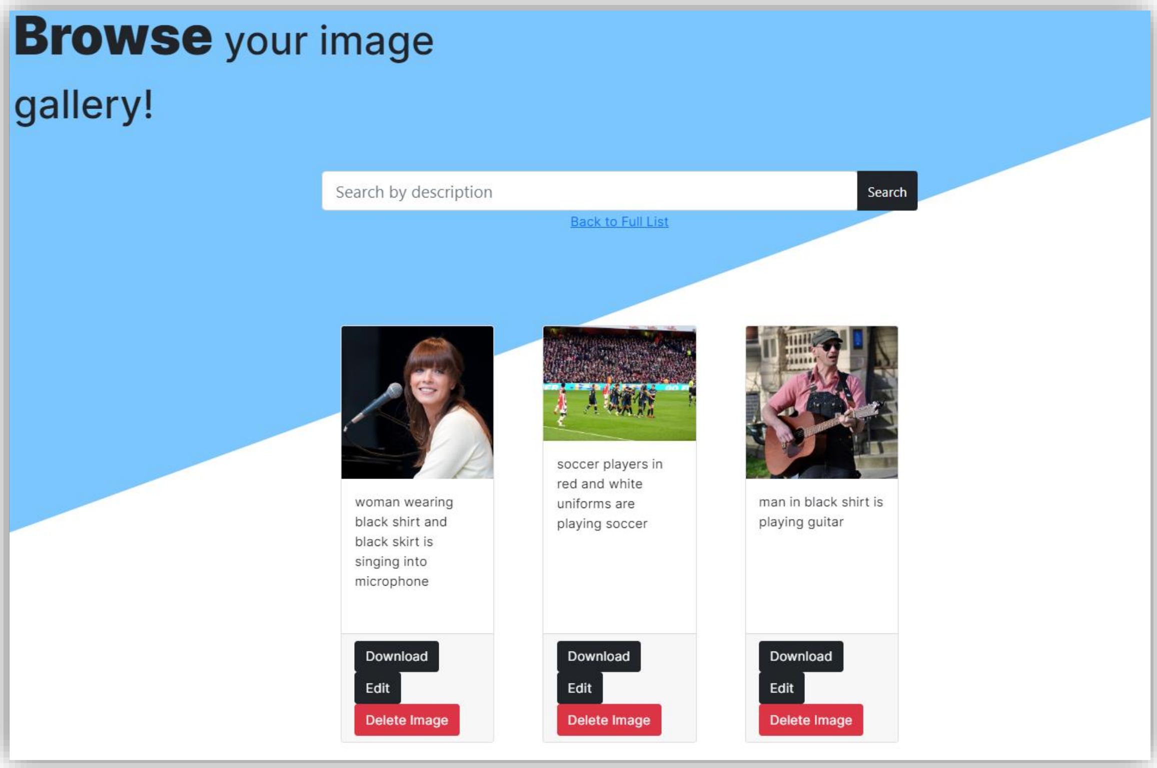
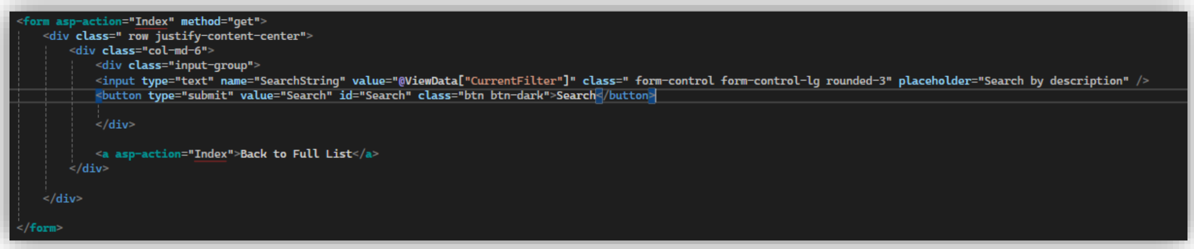

Browse your image gallery!
Search by description
Search
Back to Full List
woman wearing black shirt and black skirt is singing into microphone
soccer players in red and white uniforms are playing soccer
man in black shirt is playing guitar
Download
Edit
Delete Image

Figure 3.4-13-1 – Image gallery view.

3.4.5 Gallery page – Search Component

The search component as shown in figure X for the gallery is implemented as an input with div-tags wrapped around Bootstrap classes to keep responsiveness and it is also styled with these classes.



```
<form asp-action="Index" method="get">
    <div class=" row justify-content-center">
        <div class="col-md-6">
            <div class="input-group">
            <input type="text" name="SearchString" value="@ViewData["CurrentFilter"]" class=" form-control form-control-lg rounded-3" placeholder="Search by description" />
            <button type="submit" value="Search" id="Search" class="btn btn-dark">Search</button>

            </div>

            <a asp-action="Index">Back to Full List</a>
        </div>

    </div>

</form>
```

Figure 3.4-14 – Search function code snippet.

The final look of the search bar is shown in 3.4-15. It has a large input and a button on a side. The search bar will be accessible for the user and when the user wants to go back to the full list of the gallery they can click the button below the search bar.

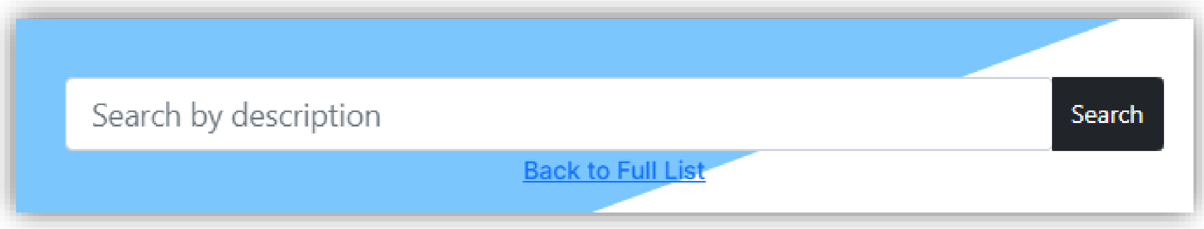
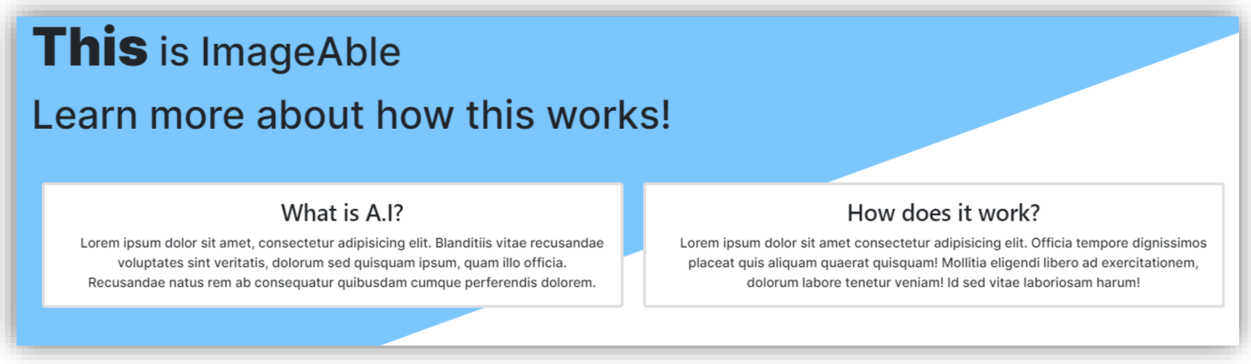


Figure 3.4-15 – Search bar.

3.4.6 About page development

According to the finalized sketch the implementation of about page is straight forward where we created two cards horizontally and each card has the information and user needs to use the web application and some information about the models. These cards are made with div containers and styled with Bootstrap to give it responsiveness. Furthermore, some extra Bootstrap classes such as “bg-white” and “shadow” to give the cards styling white background and shadow to give the cards 3D effect to make it look like cards and separate it from the background. This will make sure that the information provided is readable and user-friendly. In figure 3.4-16 is the result of implementation of the “About” page.



This is ImageAble

Learn more about how this works!

What is A.I?

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Blanditiis vitae recusandae voluptates sint veritatis, dolorum sed quisquam ipsum, quam illo officia. Recusandae natus rem ab consequatur quibusdam cumque perferendis dolorem.

How does it work?

Lorem ipsum dolor sit amet consectetur adipisicing elit. Officia tempore dignissimos placeat quis aliquam quaerat quisquam! Mollitia eligendi libero ad exercitationem, dolorum labore tenetur veniam! Id sed vitae laboriosam harum!

Figure 3.4-16 – “About” page.

3.4.7 Responsiveness grid system

Responsive web design enables automatic adaptation to different screen sizes. Implementing responsiveness on the application required small and uncomplicated practice of using Bootstrap library we were able to add Bootstrap predefined classes that includes a responsive gid system. We have used a predefined class “d-flex” which gives a display flex grid system to the “span”- tag, in this case is the background of the web application.

In figure 3.4-17as shown below shows a grid system used in the application which in this case the figures are only screenshots of the home page. However, implementation of grid system is applied to the whole application.

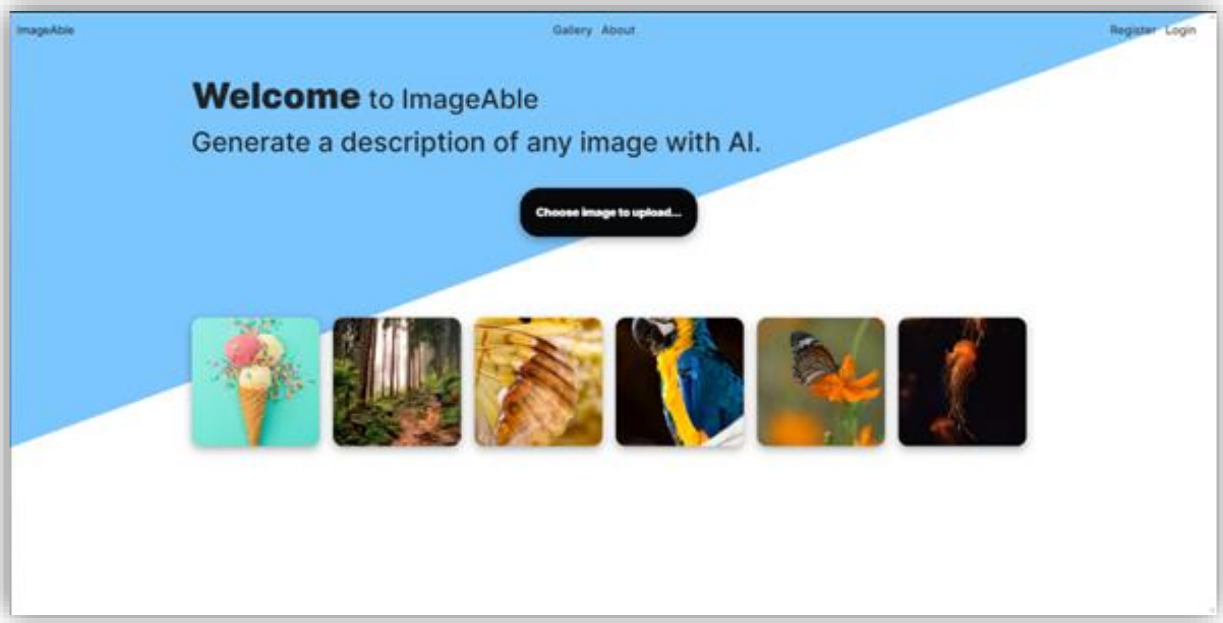

ImageAble
Gallery About
Register Login
Welcome to ImageAble
Generate a description of any image with AI.
Choose image to upload...

Figure 3.4-17 – Main page in 1920x1080p resolution.

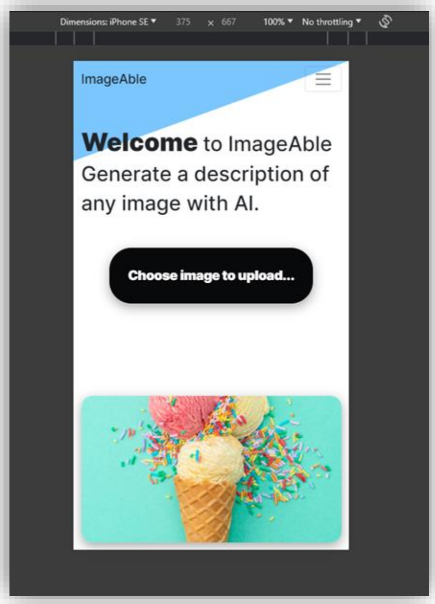

Dimensions: iPhone SE 375 x 667 100% No throttling
ImageAble
Welcome to ImageAble
Generate a description of any image with AI.
Choose image to upload...

Figure 3.4-18 – Responsiveness on a smartphone.

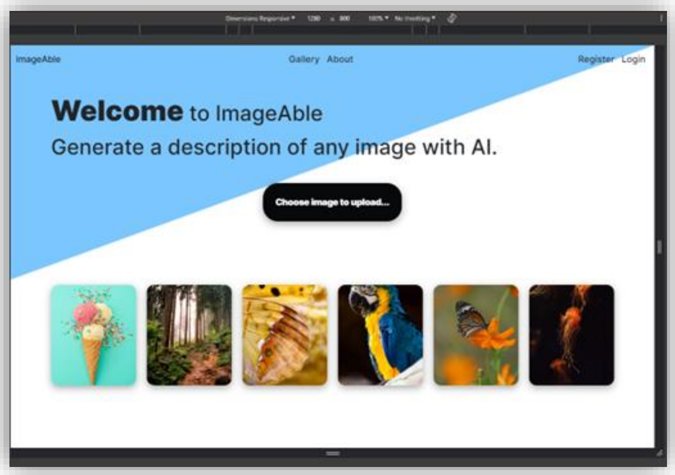
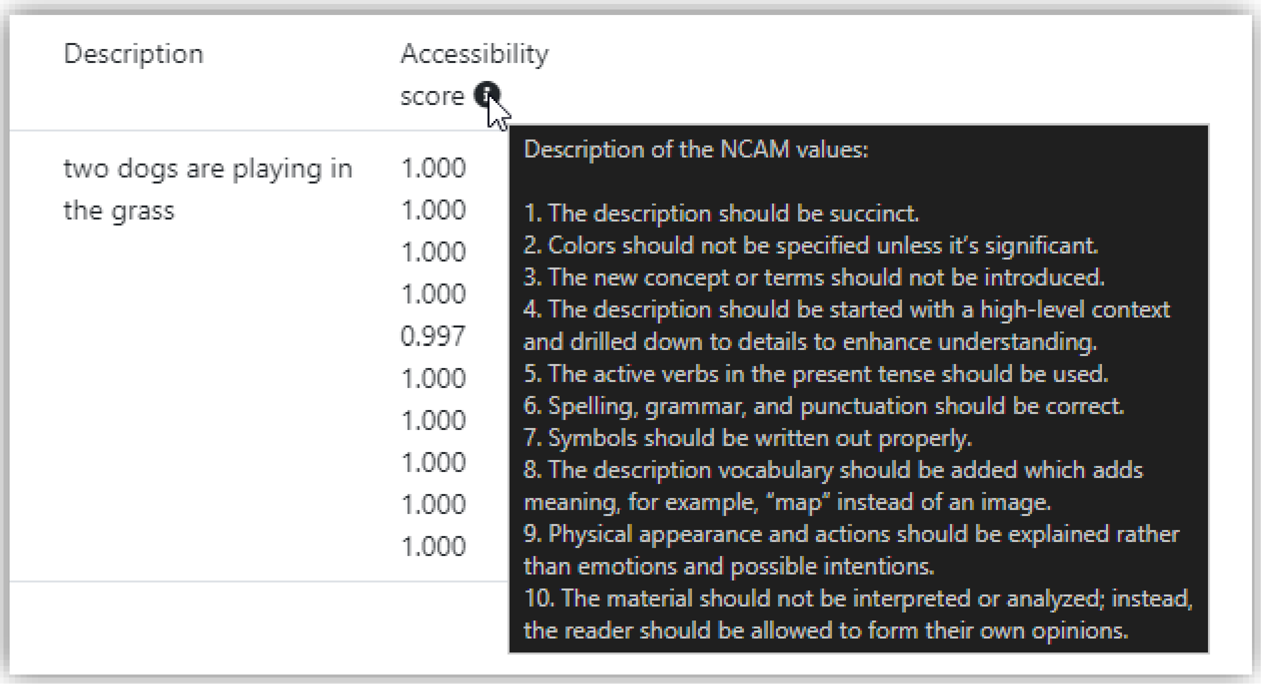

ImageAble
Gallery About
Register Login
Welcome to ImageAble
Generate a description of any image with AI.
Choose image to upload...

Figure 3.4-19 - Responsiveness in common laptop screen size.

3.4.8 Final Design adjustments

The final front-end adjustment is based on the feedback from our supervisor. This is adjustments such as adding an information icon for the NCAM guidelines which the evaluation model is based on. This is shown in figure 4.4-20 where a pop-up shows information.



Description	Accessibility score
two dogs are playing in the grass	1.000 1.000 1.000 1.000 0.997 1.000 1.000 1.000 1.000 1.000

Description of the NCAM values:

1. The description should be succinct.
2. Colors should not be specified unless it's significant.
3. The new concept or terms should not be introduced.
4. The description should be started with a high-level context and drilled down to details to enhance understanding.
5. The active verbs in the present tense should be used.
6. Spelling, grammar, and punctuation should be correct.
7. Symbols should be written out properly.
8. The description vocabulary should be added which adds meaning, for example, "map" instead of an image.
9. Physical appearance and actions should be explained rather than emotions and possible intentions.
10. The material should not be interpreted or analyzed; instead, the reader should be allowed to form their own opinions.

Figure 3.4-20 – Evaluation view.

Adjustments such as making the application more responsive by resizing buttons cards. Rewriting CSS code and improving the site overall.

3.5 Server-side (back-end) Architecture

3.5.1 Introduction

The server side is organized into different components where each have their own aspects of the application’s functionality. Components that typically include are controllers, services, models, data access layer, utilities and routing. The architecture should be clean which has several reasons. It allows the concerns to be separated for easy maintainability that enables us to modify specific components without the entire system to be affected. A clean architecture provides scalability which allows for adding new features or changes without affecting existing code.

3.5.2 Detailed description of components

3.5.2.1 Controllers

Controllers are fundamental components in the MVC architecture. The purpose of the controller is to serve as a middleman between the UI and business logic. The controller is among other things responsible for responding to the requests/calls from client. For example, AJAX-requests and other similar requests using tag-helpers will should always trigger the Controller as the next step.

The controller will then act as a coordinator, determining the appropriate action to take and possibly delegating tasks to other more specific layers such as the Service or Repositories. Separating the responsibilities enhances the maintainability and keeps responsibility more separate.

In our application we have the following three controllers: “GalleryController*”*, “HomeController*“and* “ImageController*”*. Each have their own responsibilities.

- The GalleryController handles calls and functions related to the Gallery View, for example handling the search functionality on the gallery page.
- The HomeController is responsible for rendering the Home page and About page, this is because both are connected to the View “Home”. This controller does not handle any other

functionality. We do not handle any other functions in this controller, leaving the rest for the ImageController.

- Rest of our functionality happens in the ImageController which is by far our largest controller. This controller handles the requests for image description, evaluation, saving the images and more. Basically, centralizing the image-related functions in a single controller.

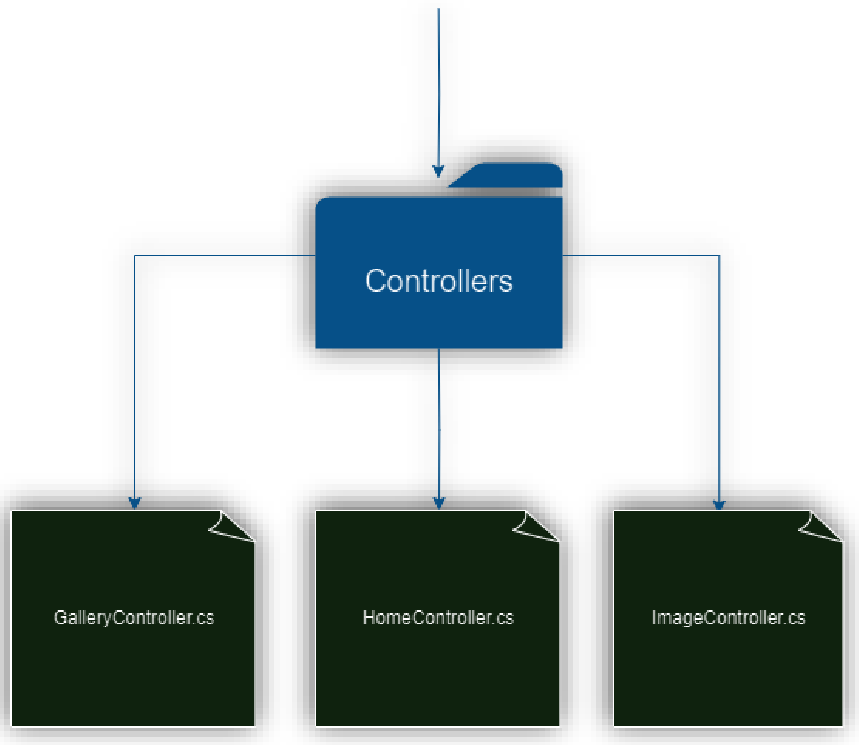


Figure 3.5-1 – Controllers.

3.5.2.2 Service

- The service layer handles the business logic and operations that the application performs. The purpose of this layer is to provide a clear separation between the controller and business logic. In a larger application which might have a decent number of controllers the more important the service layer becomes. This is because it promotes modularity and reduces duplication within the application.

In our application, the service folder contains the following three services: “ExternalApiService.cs”, “ImageService.cs” and “ImageProcessingService.cs”. Each of these services has its own interface.

- ExternalApiService is responsible for interacting with the external API’s. For example, the custom endpoints of ChatGPT.
- ImageService is responsible for handling core functionalities related to the image such as saving the image to a folder and downloading the image.
- ImageProcessingService handles the process forwarding the requests to ExternalApiService for description/evaluation generating.

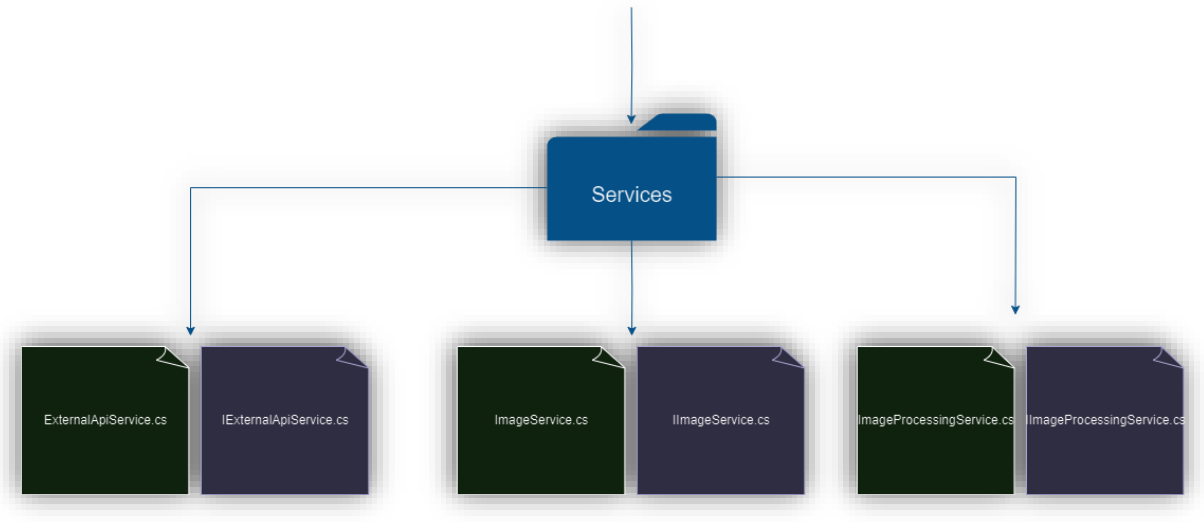


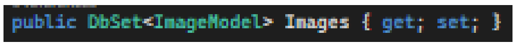
Figure 3.5-2 – Server-side architecture: Services.

3.5.2.3 Data Access Layer (DAL)

The Data Access Layer (DAL) is a critical component in the architecture. The component manages the interactions with the database. The purpose with DAL is to handle data-related operations away from the business logic, providing a more organized structure.

Our DAL is split up into three subfolders: “Context”, “Migrations” and “Repositories”. Inside these folders we have “ApplicationDbContext.cs”, “Migrations” and “ImageRepository/IImageRepository” respectively.

- ApplicationDbContext file represents the entity framework for our application. It brings the database and the application together. This file lets us to do queries and save records. Much of this file is automatically generated with .NET, however we are responsible to add the new tables/models we want the DB to generate. In our case we have requested to add the model "Images" to the database. More on the models later.



```
public DbSet<ImageModel> Images { get; set; }
```

- The migrations folder contains the files generated by the Entity framework. Each migration is coded with the database scheme. For example, if we would like to add a new model/table to the database – we apply migrations, and the migrations contain the code for the new model.
- Repositories' purpose is to provide access to the data logic. We handle all operations that queries with the database here. For example, creating, reading, updating, deleting (CRUD) and more.

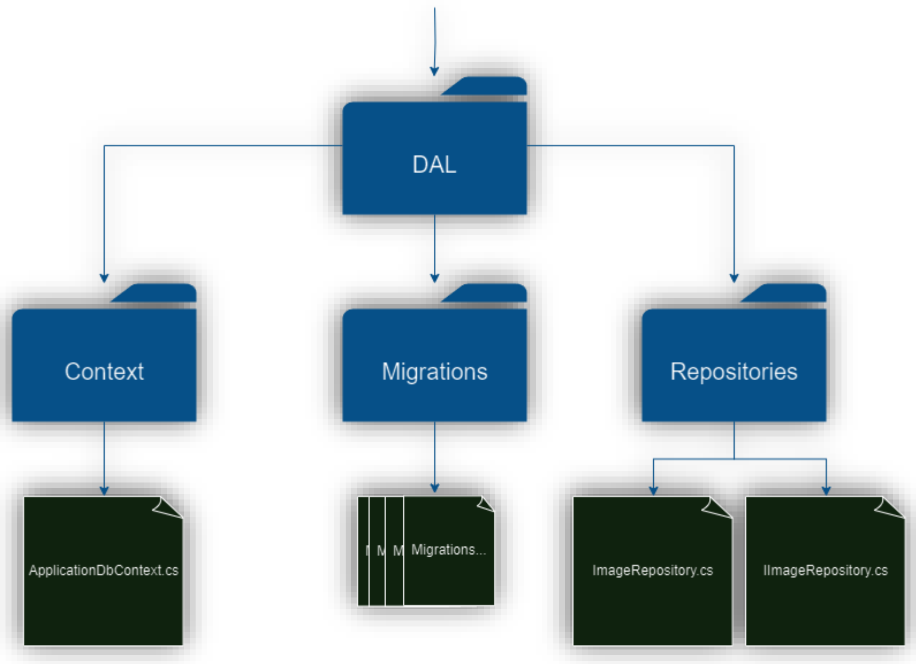


Figure 3.5-3 - Server-side architecture: DAL

3.5.2.4 Models

Models represent the data and the business rules of the application. We have tried to organize the different models into several models based on their purpose. “HelperModels”, “Models” and “RequestModels”. Which contains the following files, “PaginatedList”, “Image”, “ErrorViewModel”, “EvaluationRequest” and “ImageUploadRequest” respectively.

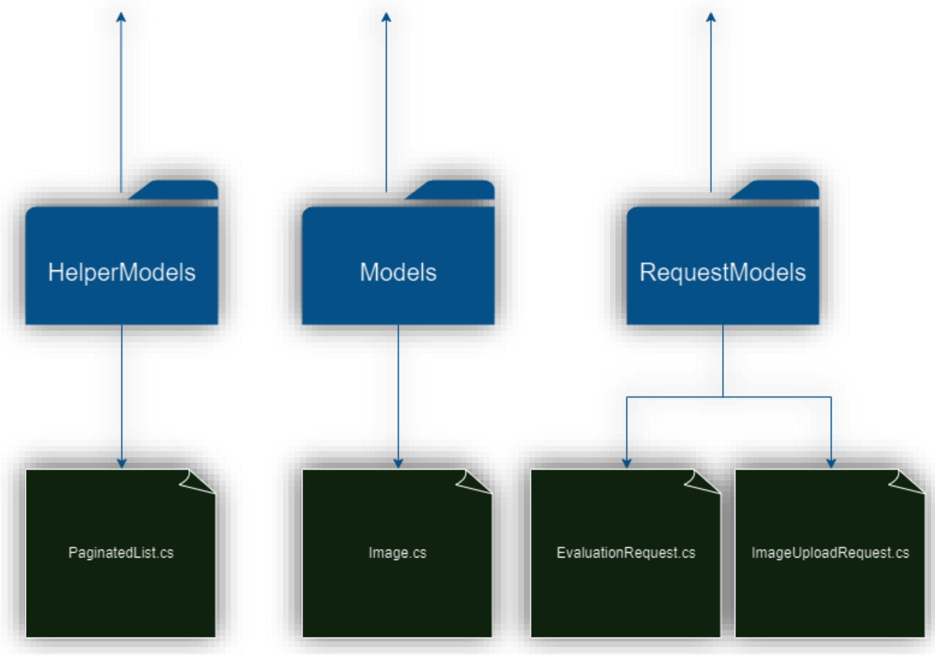
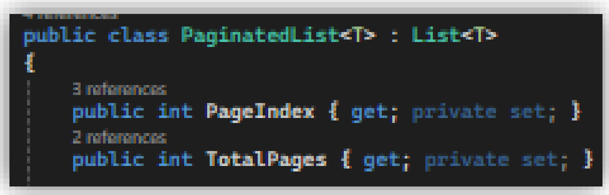


Figure 3.5-4 - Server-side architecture: Models.

- **PaginatedList:** This is a helper model used to display the paginated images. For now, only used to showcase the images in the gallery page.



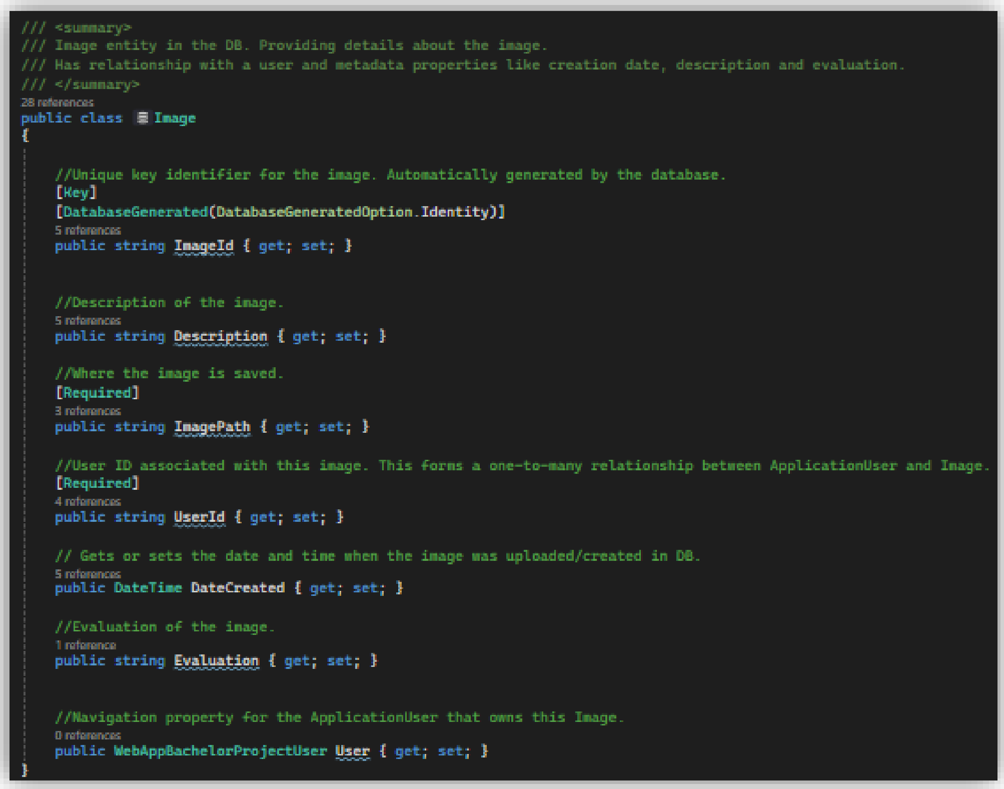
```
public class PaginatedList<T> : List<T>
{
    3 references
    public int PageIndex { get; private set; }
    2 references
    public int TotalPages { get; private set; }
```

Figure 3.5-5 - Server-side architecture: PaginatedList

- **Image:** This is the primary model that is mapped to the DB. It represents the images saved in the DB. It contains the following properties.
    - ImageId (PK): The ID for the image. This gets automatically generated by the DB.

- Description: Required is the description of the image.
- ImagePath: Where the image is saved (required field).
- UserId: The user who added this image (required field).
- DateCreated: Date and time when the image was uploaded.
- Evaluation: Evaluation/Accessability score of the image.

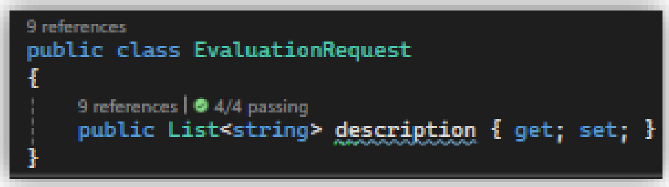
It also has a one-to-many relationship with the user. This is so we users can later retrieve the images they uploaded.



```
/// <summary>
/// Image entity in the DB. Providing details about the image.
/// Has relationship with a user and metadata properties like creation date, description and evaluation.
/// </summary>
28 references
public class Image
{

    //Unique key identifier for the image. Automatically generated by the database.
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    5 references
    public string ImageId { get; set; }


    //Description of the image.
    5 references
    public string Description { get; set; }

    //Where the image is saved.
    [Required]
    3 references
    public string ImagePath { get; set; }

    //User ID associated with this image. This forms a one-to-many relationship between ApplicationUser and Image.
    [Required]
    4 references
    public string UserId { get; set; }

    // Gets or sets the date and time when the image was uploaded/created in DB.
    5 references
    public DateTime DateCreated { get; set; }

    //Evaluation of the image.
    1 reference
    public string Evaluation { get; set; }


    //Navigation property for the ApplicationUser that owns this Image.
    0 references
    public WebAppBachelorProjectUser User { get; set; }
}
```

Figure 3.5-6 - Server-side architecture: Image Model.

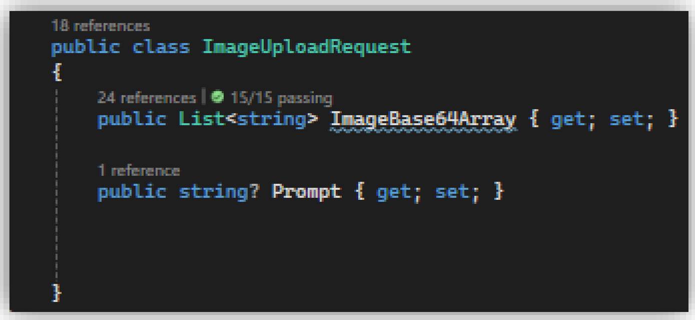
- **EvaluationRequest:** This is a request model which is used when retrieving a list of descriptions to generate an evaluation for. It contains a string list of descriptions.



```
9 references
public class EvaluationRequest
{
    9 references | 4/4 passing
    public List<string> description { get; set; }
}
```

Figure 3.5-7 - Server-side architecture: EvaluationRequest

- **ImageUploadRequest:** This is also a request model which is used with the ChatGPT option when attempting to generate a description for an image. It contains the ImageBase64 array and an optional prompt.



```
18 references
public class ImageUploadRequest
{
    24 references | 15/15 passing
    public List<string> ImageBase64Array { get; set; }

    1 reference
    public string? Prompt { get; set; }



}
```

Figure 3.5-8 - Server-side architecture: ImageUploadRequest.

3.6 Back-end (server-side) Development

3.6.1 Introduction

A development of strong, secure, and scalable server-side is very important for our application. The following sections discuss in detail the different building blocks and practices followed in developing and maintaining the backend architecture. A well-structured and efficient back end is a must for dealing with client requests and data management and integrating features like API requests to AI models. This section will delve deeper into the data flow, sequence diagrams, database, API development, security implementations, business logic, integration of external services and lastly logging.

...this will in total give a thorough overview of the backend development.

3.6.2 Data flow & Process

The diagram below illustrated the journey of the flow from the client through the whole backend and back. The flow is crucial for understanding how the different components in our application interact with each other. It is also important to keep in mind that not all sequences/flows necessarily go through all the components.

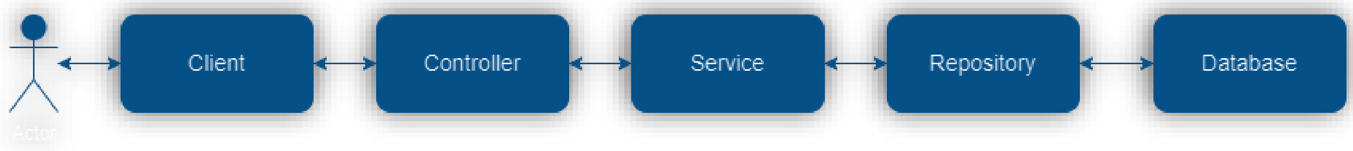


Figure 3.6-1 – Server-side Data Flow: Flow Journey

3.6.2.1 Sequence diagrams

We will go through the most important functions/methods of our application with sequence diagrams. The visualization will help to get a better understanding of how the flow and process will work.

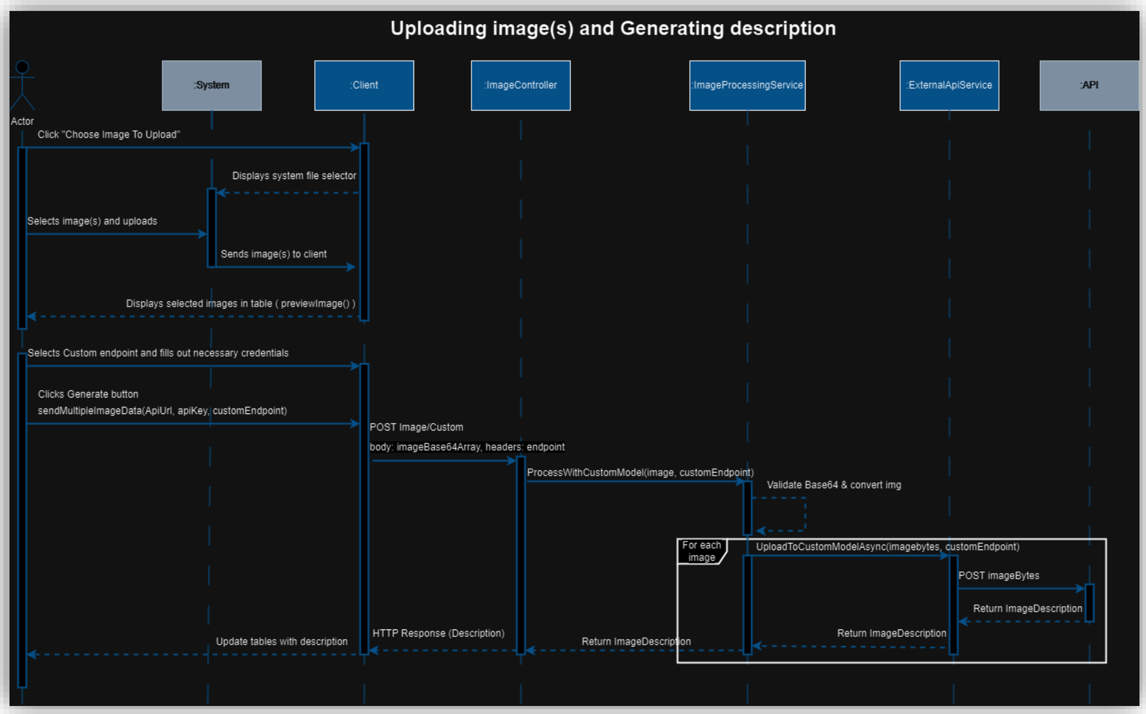

Uploading image(s) and Generating description
Actor
:System
:Client
:ImageController
:ImageProcessingService
:ExternalApiService
:API
Click "Choose Image To Upload"
Displays system file selector
Selects image(s) and uploads
Sends image(s) to client
Displays selected images in table (previewImage())
Selects Custom endpoint and fills out necessary credentials
Clicks Generate button
sendMultipleImageData(ApiUrl, apiKey, customEndpoint)
POST Image/Custom
body: imageBase64Array, headers: endpoint
ProcessWithCustomModel(image, customEndpoint)
Validate Base64 & convert img
For each image
UploadToCustomModelAsync(imagebytes, customEndpoint)
POST imageBytes
Return ImageDescription
Return ImageDescription
Return ImageDescription
HTTP Response (Description)
Update tables with description

Figure 3.6-2 - Server-side Data Flow: Sequence Diagram #1.

The sequence diagram above illustrates the process where the user uploads an image and generates a description using the Custom API endpoint. It involves multiple components having to work together to reach the external API, including the user interaction, client-side, controller, service logic.

Simplified explanation:

1. User interacts with the upload button and uploads an image.
2. Client responds with presenting the image.
3. User interacts with options, adds the necessary inputs and clicks Generate Description button.
4. Client gathers the input and sends the request through AJAX POST request to ImageController.
5. The ImageController validates the input fields and sends the request forward to the ImageProcessingService for further processing.
6. For each image uploaded, the ImageProceesingService sends a request to ExternalApiService and waits for the return.
7. The externalApiService sends a POST request to the endpoint and sends the result back to the ImageProccesingService.
8. Once all the images have been processed through the externalApiService, we send the response in an array back all the way back to the client.
9. The client presents the description in the table for user to see.

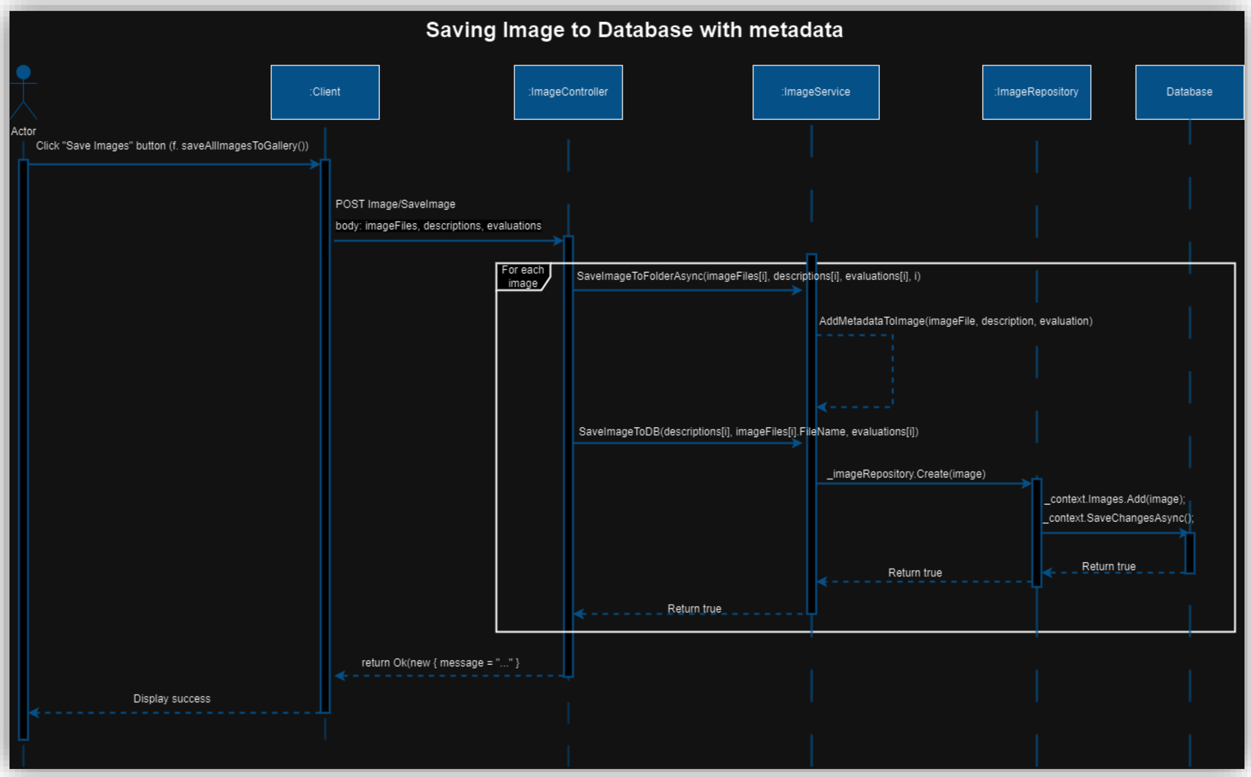

Saving Image to Database with metadata
:Client
:ImageController
:ImageService
:ImageRepository
Database
Actor
Click "Save Images" button (f. saveAllImagesToGallery())
POST Image/SaveImage
body: imageFiles, descriptions, evaluations
For each image
SaveImageToFolderAsync(imageFiles[i], descriptions[i], evaluations[i], i)
AddMetadataToImage(imageFile, description, evaluation)
SaveImageToDB(descriptions[i], imageFiles[i].FileName, evaluations[i])
_imageRepository.Create(image)
_context.Images.Add(image);
_context.SaveChangesAsync();
Return true
Return true
Return true
return Ok(new { message = "..." }
Display success

Figure 3.6-3 - Server-side Data Flow: Sequence Diagram #2.

Simplified explanation:

1. User clicks ”Save images” button in the UI.
2. The client reads the input and sends an AJAX POST request to the ImageController.
3. For each Image:

    3.1 the controller sends the image together with description and evaluation to the ImageService.

    3.2. When the data is received in the ImageService it adds the metadata to the image saves the image to the folder.

    3.3. The ImageController then sends another request to the imageService to save the image to the DB.

    3.4. The imageService calls upon the ImageRepository which handles the Database requests and created the image with the input given.

    3.5 ImageRepository sends queries to the DB.
4. When all images have been saved, we return an OK status back to the client.
5. The client will at last display a success message for the user to see.

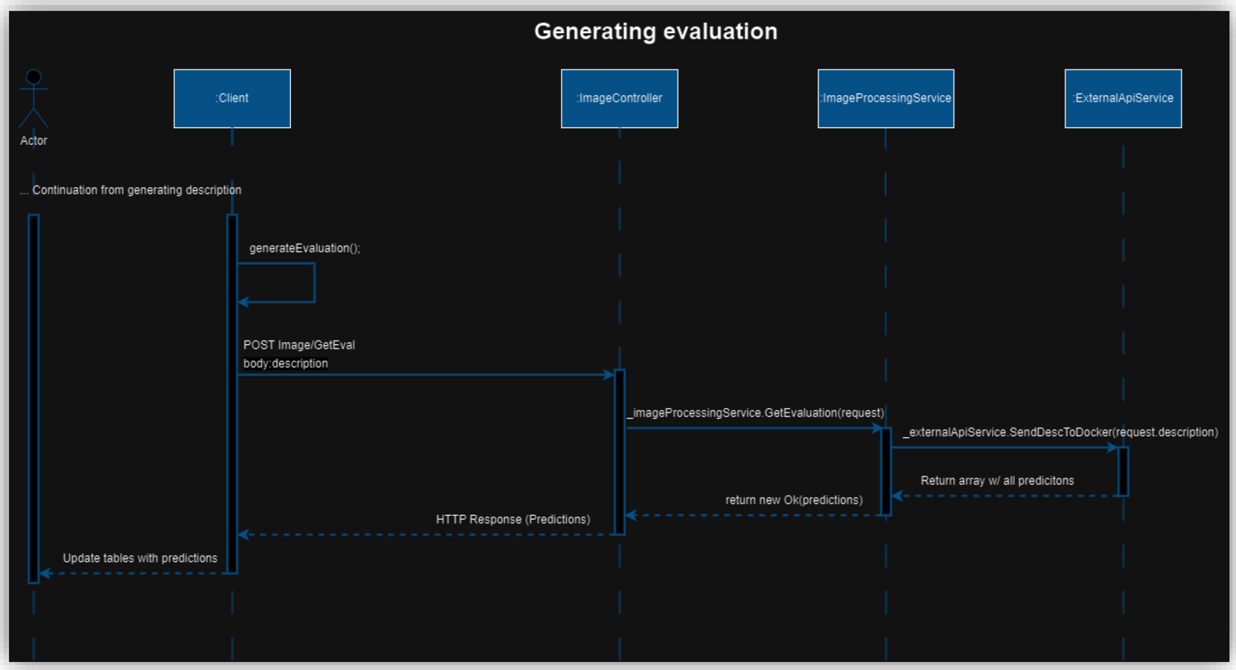

Generating evaluation
Actor
:Client
:ImageController
:ImageProcessingService
:ExternalApiService
... Continuation from generating description
generateEvaluation();
POST Image/GetEval
body:description
_imageProcessingService.GetEvaluation(request)
_externalApiService.SendDescToDocker(request.description)
Return array w/ all predicitons
return new Ok(predictions)
HTTP Response (Predictions)
Update tables with predictions

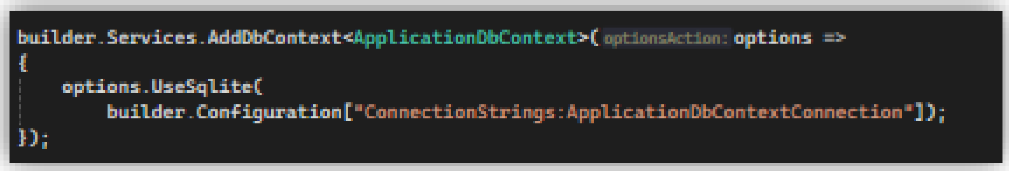
Figure 3.6-4 - Server-side Data Flow: Sequence Diagram #3.

Simplified explanation:

1. This is a follow up after generating the description. In other words, this process activates automatically. No need for user interactions for this process to start.
2. The client will run the generateEvaluation for the evaluation to start right after finishing generating the description.
3. The client will send AJAX POST calls to the ImageController with the description as input.
4. The ImageController validates and sends the request for further business logic to ImageProcessingService.
5. The ImageProcessingService sends the request to the ExternalApiService.
6. The ExternalApiService sends the request to the API and returns the array with all predictions through all the components and update the table in the UI for the user to see.

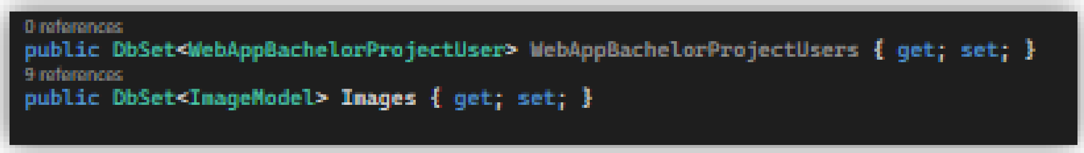
3.6.3 Database

For saving account information and images connected to them, we decided to use SQLite as our database. SQLite is a lightweight file-based database that is included in ASP.NET Core. This type of database is easy to set up because it is serverless and has a zero-configuration database engine. This is also embedded for applications such as ours and has the benefit of supporting most SQL92 standards that includes complex queries, triggers and indexing.



```
builder.Services.AddDbContext<ApplicationDbContext>( optionsAction: options =>
{
    options.UseSqlite(
        builder.Configuration["ConnectionStrings:ApplicationDbContextConnection"]);
});
```

Figure 3.6-5 – Defining database.



```
0 references
public DbSet<WebAppBachelorProjectUser> WebAppBachelorProjectUsers { get; set; }
9 references
public DbSet<ImageModel> Images { get; set; }
```

Figure 3.6-6 – Calling database.

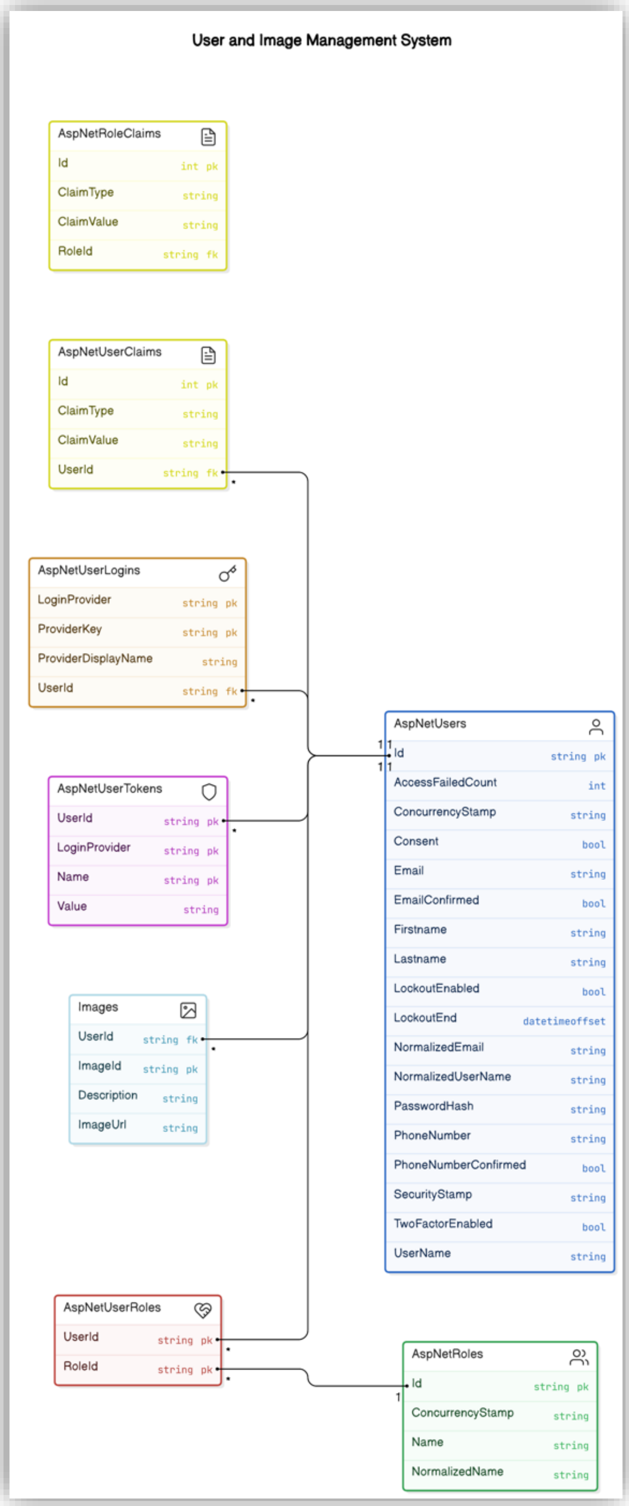


Figure 3.6-7 – Database overview.

The entity-diagram is related to web application user and image management. Based on the diagram the data is stored in related database tables such as AspNetUser, AspNetUserClaims, AspNetUserLogins, AspNetUserTokens.

AspNetUser table stores user information such as username, email, password hash, etc. AspNetUserClaims stores claims or attributes associated with each user. AspNetUserLogins table tracks the information about the login accounts. AspNetUserTokens table stores tokens related data for each user.

The relationship between these entities shows the connection between them, for example where “Images” has one-to-one relationship with “AspNetUsers”, this means that each user can have one or more images. While relationships between AspUsers have one-to many relationships with AspNetUserClaims, which means that a user can be associated with multiple AspNetUserClaims entities.

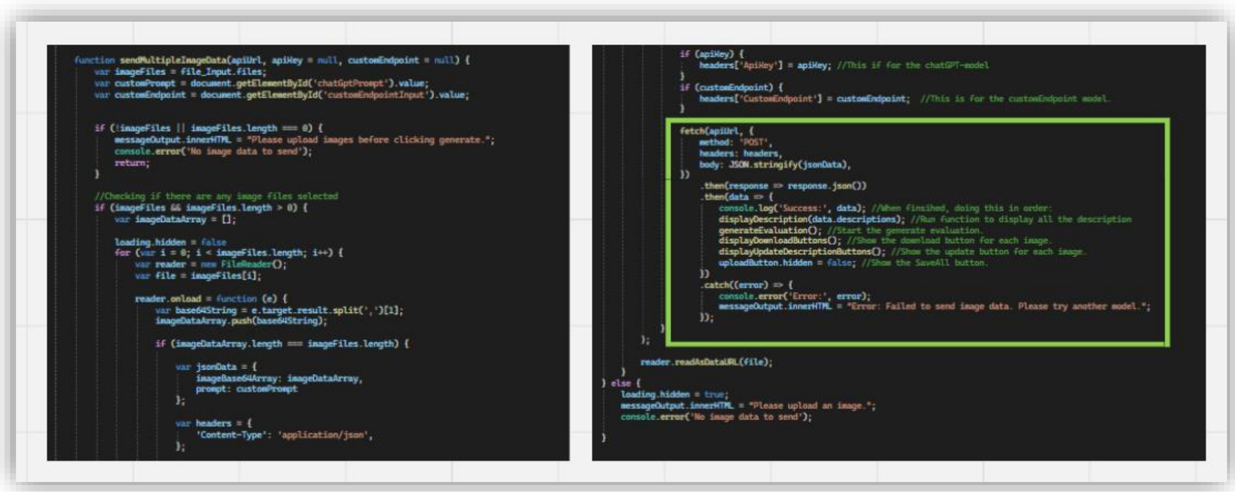
In addition, this schema is designed to be code-first approach which means that application entity relationships automatically change depending on the application's code logic. Therefore, defining data annotation, for example properties such as “int”, “string” and “bool” allows us to add metadata to our code that includes properties, classes and methods.

3.6.4 API Gateway

In part API- Gateway is a crucial part in our solution. AJAX is a benfit for us , when applied for instance forms which enables forms dynamatically without having to reload the web page. Therefore, increases the applications performance.

AJAX is short for Asynchronous JavaScript and XML. AJAX refers to a multiple of technologies such as XHTML, CSS, DOM, XMLHttpRequest, XML, HTML and XSLT to develop web applications (IBM, 2021). These technologies enable the web application user to interact with the web page without constant web page reloading. A method of initiating client to server communication is used to provide a way to enable partial page updates.

The implementation of API gateway where we used AJAX fetch in client server is shown in figure 3.6-8.



```javascript
function sendMultipleImageData(apiUrl, apiKey = null, customEndpoint = null) {
    var imageFiles = file_Input.files;
    var customPrompt = document.getElementById('chatGptPrompt').value;
    var customEndpoint = document.getElementById('customEndpointInput').value;


    if (!imageFiles || imageFiles.length === 0) {
        messageOutput.innerHTML = "Please upload images before clicking generate.";
        console.error('No image data to send');
        return;
    }

    //Checking if there are any image files selected
    if (imageFiles && imageFiles.length > 0) {
        var imageDataArray = [];

        loading.hidden = false
        for (var i = 0; i < imageFiles.length; i++) {
            var reader = new FileReader();
            var file = imageFiles[i];

            reader.onload = function (e) {
                var base64String = e.target.result.split(',')[1];
                imageDataArray.push(base64String);

                if (imageDataArray.length === imageFiles.length) {

                    var jsonData = {
                        imageBase64Array: imageDataArray,
                        prompt: customPrompt
                    };

                    var headers = {
                        'Content-Type': 'application/json',
                    };
```

```javascript
                    if (apiKey) {
                        headers['ApiKey'] = apiKey; //This if for the chatGPT-model
                    }
                    if (customEndpoint) {
                        headers['CustomEndpoint'] = customEndpoint;  //This is for the customEndpoint model.
                    }

                    fetch(apiUrl, {
                        method: 'POST',
                        headers: headers,
                        body: JSON.stringify(jsonData),
                    })
                        .then(response => response.json())
                        .then(data => {
                            console.log('Success:', data); //When finished, doing this in order:
                            displayDescription(data.descriptions); //Run function to display all the description
                            generateEvaluation(); //Start the generate evaluation.
                            displayDownloadButtons(); //Show the download button for each image.
                            displayUpdateDescriptionButtons(); //Show the update button for each image.
                            uploadButton.hidden = false; //Show the SaveAll button.
                        })
                        .catch((error) => {
                            console.error('Error:', error);
                            messageOutput.innerHTML = "Error: Failed to send image data. Please try another model.";
                        });
                }
            };

            reader.readAsDataURL(file);
        }
    } else {
        loading.hidden = true;
        messageOutput.innerHTML = "Please upload an image.";
        console.error('No image data to send');

    }
```

Figure 3.6-8 – AJAX fetch.

In this figure which shows a JavaScript code used for the home page, we have implemented multiple AJAX fetch calls in following methods:

- 'SendMultipleImageData'
- 'generateEvaluation'
- 'SaveAllImagesToGalley'
- 'downloadImage'

However, to give an example of the use of AJAX “fetch” call the figure only shows “SendMultipleImageData” The method sends image data to an API endpoint. To break it down, it validates input by checking any image files that have been selected, then processes the image data using a loop. Furthermore, it creates a “FileReader” object that reads the file asynchronously, then converts the image to base64-encoded string. The method then sends data over to server, by constructing a JSON object that contains an array of base64 image data. Lastly, it sends a “POST” request to the apiUrl endpoint with JSON data by using “fetch” API.

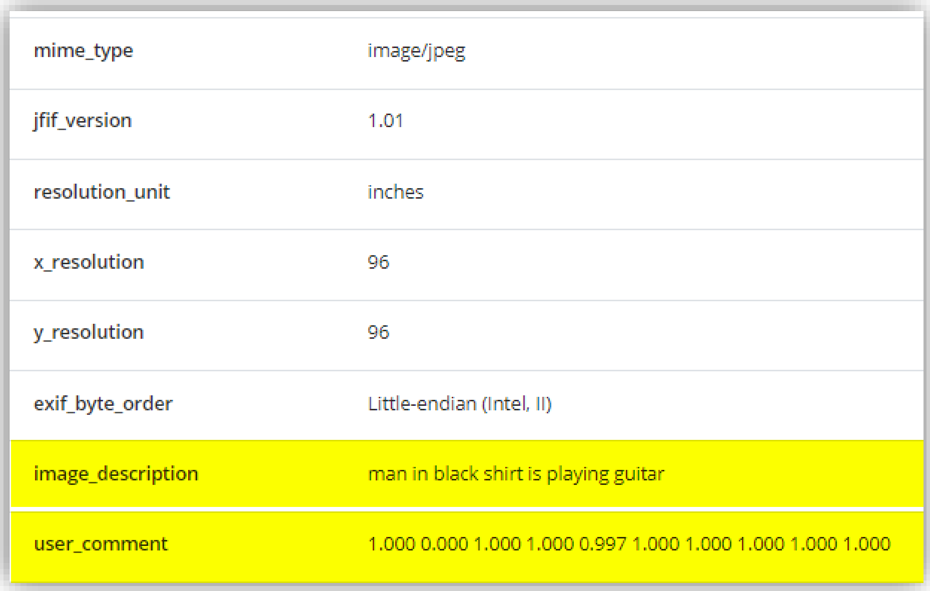
3.6.5 Adding metadata to image

In our application we enhance the images we save to gallery or download by adding descriptive metadata using the “AddMetadataToImage” function in the ImageService. The process involves embedding the information directly into the image file’s EXIF. The reasoning behind this idea is that we believe it lets developers easily use this as an ALT-text when adding an image to a website. It might also be useful for searching and organizing images based on the content in the ImageDescription tag.

We had a variety of EXIF tags to choose from. We went through the list of property tags from Microsoft and luckily found the property tag image_description which worked out perfectly. However, for the evaluation we were not that lucky. We could not find something that corresponds with what the evaluation is for, so we added it under the property tag “user_comments”.

In the implementation of the metadata manipulation, we used Sixlabors.Imagesharp. It is a versatile image processing library allowing us to manipulate metadata and save images. (Microsoft, 2022)

Figure 3.6-9 shows an example of an image where we manipulated this exact metadata.



mime_type	image/jpeg
jfif_version	1.01
resolution_unit	inches
x_resolution	96
y_resolution	96
exif_byte_order	Little-endian (Intel, II)
image_description	man in black shirt is playing guitar
user_comment	1.000 0.000 1.000 1.000 0.997 1.000 1.000 1.000 1.000 1.000

Figure 3.6-9

3.7 Security Considerations

The security considerations are to protect the user data. There are several security issues and common issues such as:

- User authentication
- Server authentication
- Authorization
- Encryption

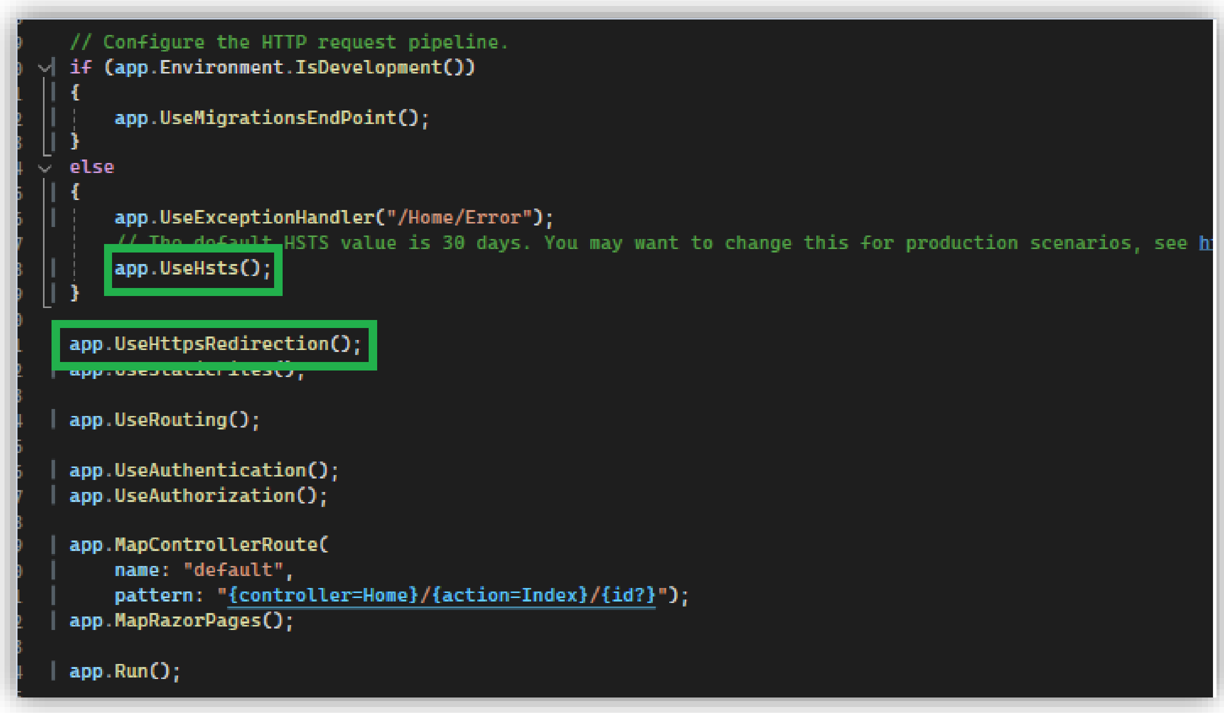
The client-side security involves data-transmission security, form validation, CSRF Protection, CSP (Content Security Policy), Secure Cookies, Data Intregity Checks and additional measures.

ASP.NET has built–in support for authentication and authorization. The ASP.NET Identity has password hashing algorithms such as PBKDF2, Argon2 and bcrypt are used to store each individual's password securely. ASP.NET also has Account Lockout, meaning that it has lockout policies to prevent attacks by locking out users after multiple failed login attempts. In addition, it also has Multi-Factor Authentication that provides another layer of security.

In our application we were able to enable two-factor identification (2FA). This type of authentication uses two methods of authentication, for example where users log in with a password and then receive a verification code on their phone. The users are able to set up 2FA to login to their account.

The application already includes HTTPS for all communication between the client and server. HTTPS encrypts data in transit, which prevents MITM short for man-in-the-middle attacks. ASP.NET Core has middleware to enforce HTTPS with the use of “UseHttpsRedirection” to redirect HTTP request to HTTPS. The framework built in support for HTTP Strict Transpor Security (HSTS) which make sure the browser only connects to the web application through HTTPS.

Figure 3.7-1 shows the built-in support for HTTPS and HSTS.



```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseMigrationsEndPoint();
}
else
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see h
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
app.MapRazorPages();

app.Run();
```

Figure 3.7-1 – HTTPS and HSTS support.

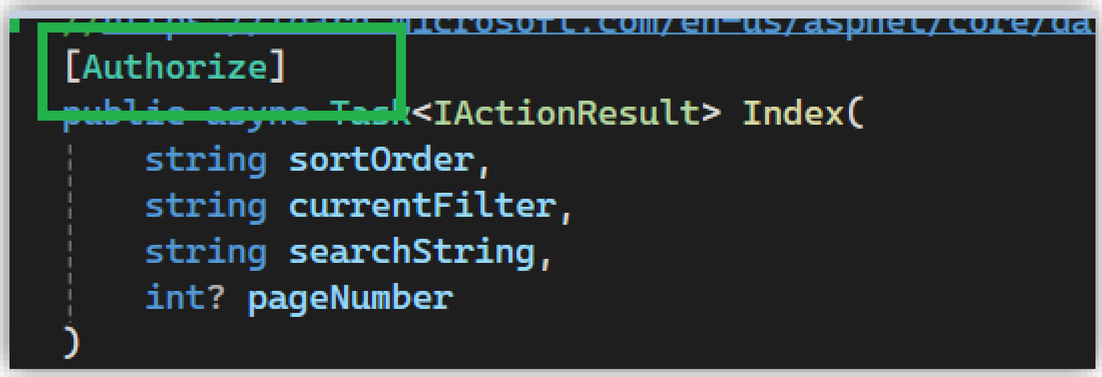
3.7.1 Security Components:

There are regulations that are important to take a look at to ensure the protection a user needs against cyber-attacks. General Data Protection Regulation (GDPR) is a regulation in the EU that is focused on data protection and privacy for individuals within the EU. It has guidelines for the collection, processing and storage of personal data. (Wolford, n.d.)

To ensure that the application has security features or components such as authorization services, for example when a user wants to see a gallery and save an image, authorization is required. Authorization measures are important since it helps protect sensitive information and prevents unauthorized users from modifying or viewing data (Unacademy, n.d.). Our application does not contain Role-based Access Control (RBAC). However, we give access to gallery and for saving images to gallery for users who has registered a user profile in the application. It also gives access to other functions such as downloading, editing and deletion of images that are saved in the gallery.

To implement this function, we used the “[Authorize]” an ASP-NET attribute as shown in figure X. This attribute simply checks if the user is authenticated, otherwise it returns a HTTP error status that says, “Unauthorized user”. In our case we implemented so if an unauthorized user tries to go to gallery the user gets sent to log-in page.

In addition, as mentioned in security considerations 2FA are also implemented in the user management page where user can set up 2FA.



```
[Authorize]
public async Task<IActionResult> Index(
    string sortOrder,
    string currentFilter,
    string searchString,
    int? pageNumber
)
```

Figure 3.7-2 – Authorize code snippet.

3.8 Model development

For this project, we decided to develop our own image description/captioning model using the Tensorflow 2 library with Python 3. Image captioning is a challenging task that requires the model to understand the visual content of an image and generate a coherent and contextually relevant text description. Our approach combines a convolutional neural network (CNN) for image feature extraction with recurrent neural networks (RNNs) for sequence generation.

The architecture of our model consists of three main components: The image encoder, the sequence processor, and the decoder. Even though creating a model of our own was not a requirement for the project, we chose to create our own to have more control over our web application. and because most third-party image captioning/description models require a paid subscription. This gives us full control over the data it has been trained on and makes it possible to tune and fit our purpose. It has also been a very interesting and challenging task which has given us an opportunity to broaden our skillset and learn a dig deeper into a very relevant subject in today's age.

The finished model will also have to be accessed through an API. We chose to use the Flask 3.0 Python library for this, running inside a dedicated Docker container, which will be discussed in detail later.

This process has been time-consuming despite our introductory course to machine learning, requiring us to delve a lot deeper into the subject. Our model is not perfect due to at-home hardware limitations, but it recognizes basic elements within images, serving more as a proof of concept. We will not be going very deep into explaining the intricate details that make up the model development as we still have a lot to learn within the subject, but we will be explaining the basic process of how we created our own image description model.

3.8.1 Libraries and Technologies

The most notable libraries and technologies that are essential for our model creation are as follows:

The most notable libraries and technologies that are essential for our model creation are as follows:

- **Python:**
  - pickle - Saving and loading Python objects for persistence.
  - numpy - Handling of arrays for scientific computing in Python.
  - Tensorflow.keras. * - A vast deep learning library for building and training models. In this context it is used for fetching the pre-trained VGG16 model, image preprocessing, layering, and loading models.
  - nltk (Natural Language Toolkit) - Calculating BLEU score.
  - Flask - Routing HTTP requests.
- **Docker**
- **Azure**

3.8.2 ImageAble model architecture and preparations

The model architecture and training methods we used have been heavily inspired by sources such as Hackers Realm (Hacker's realm, 2022) and aswintechguy's GitHub (aswintechguy, n.d.).

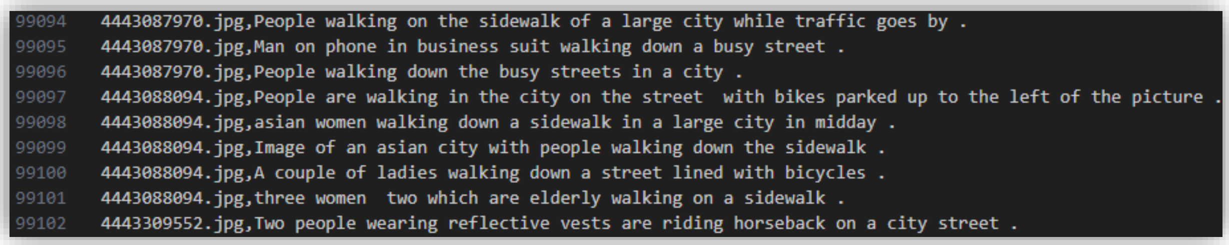
This has been implemented using Jupyter Notebook script, which allows us to facilitate iterative development and testing, which is standard practice in machine learning.

Our model creation involves several steps: extracting image features, preprocessing text data (captions), setting up our model structure, and then training the model with an image-caption dataset.

We also need to limit the amount of data the model can be trained on at a time, since a regular approach might result in crashes because of the limited computational resources we have.

3.8.2.1 Data preparation

Creating a machine learning model requires a lot of data to efficiently extract and understand distinct features from images. This requires a large dataset, which is easily available on the web. The dataset we have decided to use for training our ImageAble image description model is the Flickr30K dataset (H$anke$ara, n.d.). This is a public domain dataset which consists of about 30 000 images and five caption variations for each respective image. In other words, around 150 000 text sentences in a single .cvs file as shown in figure 4.8-1.



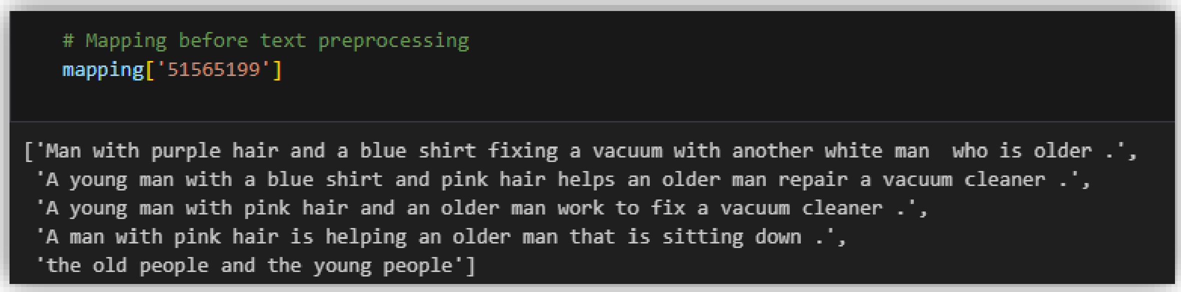
```
4443087970.jpg,People walking on the sidewalk of a large city while traffic goes by .
4443087970.jpg,Man on phone in business suit walking down a busy street .
4443087970.jpg,People walking down the busy streets in a city .
4443088094.jpg,People are walking in the city on the street  with bikes parked up to the left of the picture .
4443088094.jpg,asian women walking down a sidewalk in a large city in midday .
4443088094.jpg,Image of an asian city with people walking down the sidewalk .
4443088094.jpg,A couple of ladies walking down a street lined with bicycles .
4443088094.jpg,three women  two which are elderly walking on a sidewalk .
4443309552.jpg,Two people wearing reflective vests are riding horseback on a city street .
```

Figure 3.8-1 – Flickr30k captions.

The captions file, which includes image IDs and captions, is loaded into our notebook. Each line in the captions file consists of an image ID and a caption, with a new image ID every 5 lines. We create a dictionary to map each image.

e ID to its respective captions. For each line in the captions file, we split the image ID and the caption into tokens using line.split(‘,’), skipping the line if it’s empty and also removing the file extension of each image ID. Now we can extract the image ID with its respective captions and store them in the mapping dictionary using mapping.append. This allows us to store the mappings efficiently.

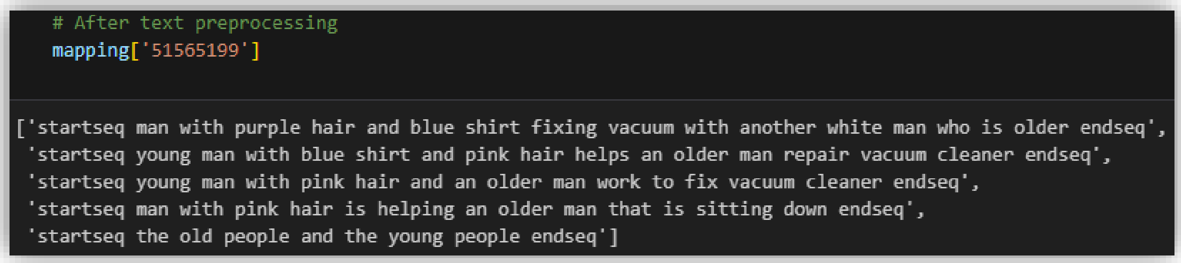
When calling the mapping of a specific image ID, we will get what’s shown in figure 4.8-2.



```
# Mapping before text preprocessing
mapping['51565199']

['Man with purple hair and a blue shirt fixing a vacuum with another white man  who is older .',
 'A young man with a blue shirt and pink hair helps an older man repair a vacuum cleaner .',
 'A young man with pink hair and an older man work to fix a vacuum cleaner .',
 'A man with pink hair is helping an older man that is sitting down .',
 'the old people and the young people']
```

Figure 3.8-2 – Mapping.

The next step is to preprocess the text data. We need a way to declare where each caption starts and ends, so the model has a consistent way of reading the data. We iterate over each mapped caption to convert all letters to lowercase, delete special characters, digits, and additional spaces, and add a start and end tag to each caption, “startseq” and “endseq”. After this step, we end up with a finalized captions mapping like in figure 4.8-3.



```
# After text preprocessing
mapping['51565199']

['startseq man with purple hair and blue shirt fixing vacuum with another white man who is older endseq',
 'startseq young man with blue shirt and pink hair helps an older man repair vacuum cleaner endseq',
 'startseq young man with pink hair and an older man work to fix vacuum cleaner endseq',
 'startseq man with pink hair is helping an older man that is sitting down endseq',
 'startseq the old people and the young people endseq']
```

Figure 3.8-3 – Pre-processed caption data.

The maximum length of the captions in the dataset is also important to note, since we don’t want our model to be able to generate unnecessarily long image descriptions. The max length of a caption in the dataset is fetched from the caption list and stored in a variable.

The captions also need to be tokenized. This step is important because neural networks don’t understand raw text, but rather numerical data. By using a tokenizer, we can store the captions in a format the model understands. A built-in Keras tokenizer is initialized, and the captions are processed and assigned an integer index to each unique word. It is then saved in another Pickle file which is used for both training and running the model later.

When the tokenizer has been stored, we can check how unique words are in the vocabulary our model will be trained on, which is more than 18 000 in our case.

3.8.2.2 Feature extraction

The method we are using to build our model requires the ability to extract distinct features from images. When training the model, we need a way for the model to learn different features of images like people, animals, objects, and actions. This is where a model for feature extraction is useful.

For image feature extraction, we have decided to use a modified version of the Keras-included pre-trained VGG16 model. The VGG16 model is a convolutional neural network (CNN) originally made for image classification (Simonyan & Zisserman, 2015). It has been pre-trained on the ImageNet dataset (Image-Net, n.d.) which helps leverage learned feature maps without starting from scratch. It contains 16 convolutional layers, which has helped the model to learn a large hierarchy of features. In other words, the VGG16 model works as our image encoder.

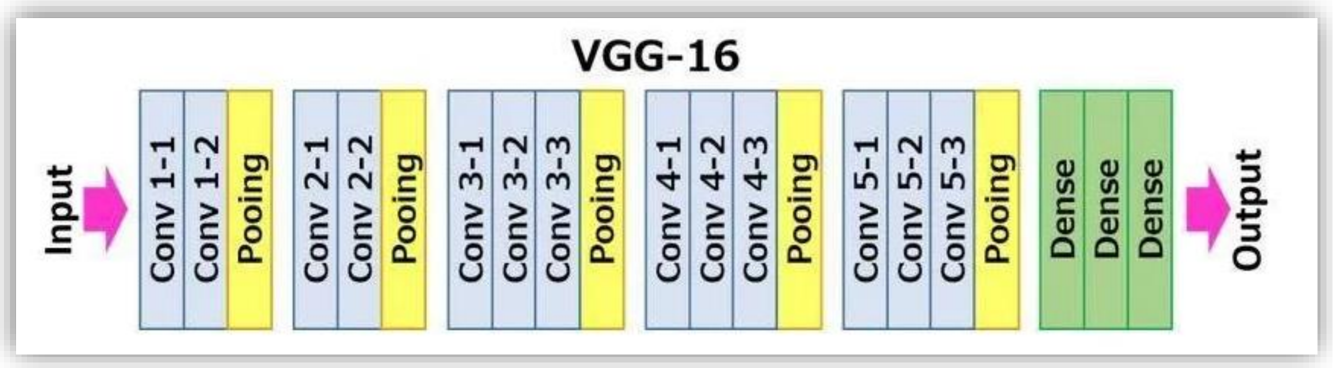
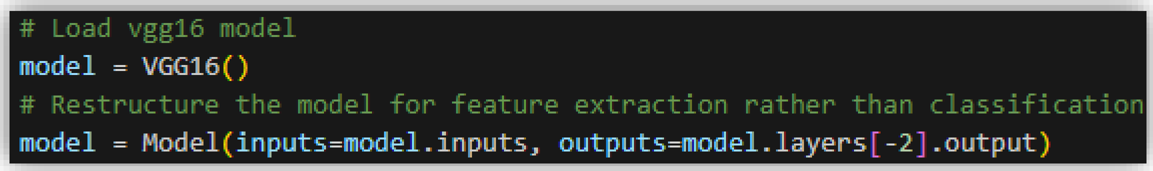


Figure 3.8-4 – VGG16 model structure (Great learning, 2021)

Because VGG16 is pre-trained on a comprehensive image dataset, it is able to recognize a variety of features in images, which eliminates our need to manually do feature extraction for training our model. Its simplicity also makes it easy to use without knowing the in-depths about how it functions, which is an advantage for us since this is our first time creating any sort of machine learning model that uses image feature recognition.

When loading the VGG16 model, it can be restructured to better fit our purpose, as shown in figure 4.8-5.



```
# Load vgg16 model
model = VGG16()
# Restructure the model for feature extraction rather than classification
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
```

Figure 3.8-5 – VGG16 restructuring.

By changing the model.layers output to discard two layers, we can use the model for feature extraction instead of classification. Next, we have to extract the image features from our dataset.

This is done by declaring an array for features and the images directory. Then we iterate for each image in the dataset. Each image data is converted to a numpy array and reshaped to fit the VGG16 model before going through preprocessing using Tensorflow's own VGG16 model preprocessor. This step is needed for the VGG16 model to be able to extract features from images of any size.

The features are then extracted using model.predict with the image as an argument and stored in a Pickle file. This allows us to later load the extracted image features on demand when training and running the finished model.

3.8.2.3 Train test split

Preparing the model for training requires setting up a train test split. This divides the image set into two subsets, a training set and a test set. The train set is used to train the model, while the test set is used to evaluate the performance of the trained model. This helps us to assess how well the model generalizes to unseen data and is important because of two reasons: It prevents overfitting the model, or in other words making sure the model doesn't just memorize the input data, but adjusts accordingly to new, unseen data. It also helps with providing an estimate for the performance of the model.

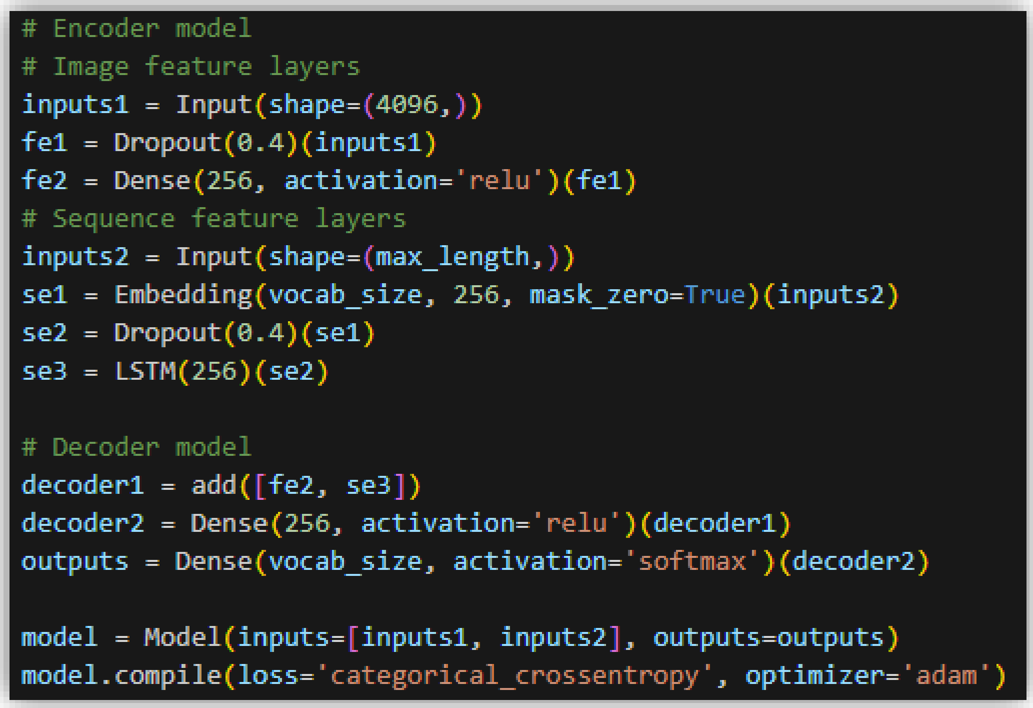
The data is split so that 90% of it is used for training, and 10% is used for testing.

3.8.2.4 Model creation

We have already covered the VGG16 model as our image encoder. What remains is the sequence processor and the decoder.

The sequence processor is an embedding layer followed by an LSTM (Long Short-term Memory) network. The embedding layer converts the input (captions) of our previously saved mapping into dense vectors of a fixed size, which are then passed through to the LSTM to capture the temporal dependencies in the sequences. The LSTM output represents the processed sequence features.

The decoder then combines the LSTM output (sequence features) and image features using an addition operation, followed by a dense layer to generate the final output.



```
# Encoder model
# Image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# Sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

# Decoder model
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Figure 3.8-6 – Model creation.

To efficiently handle the large dataset and prevent memory overflow, we have implemented a data_generator function, which yields batches of input data and corresponding target outputs to the model during training. The function takes in a list of Image IDs, the mapping we created earlier, and the features and tokenizer and creates batches from them. The batch size can also be tuned depending on the hardware capabilities the model is trained on.

3.8.2.5 Training

The model is set to train for 100 epochs (one round of training using all the data), but with early stopping implemented to prevent overfitting the model. Early stopping monitors the calculated loss and terminates training if there are no improvements made in 10 consecutive epochs. Again, due to hardware limitations, we only let the model run for 16 epochs, or about 12 hours, which was nowhere close to triggering early stopping. We used a batch size of 32 for the data_generator, as this seemed to give us the best combination of performance and speed during training. We can monitor the loss value for each epoch. Loss is a metric to estimate how accurate the model is, where a lower score is better. (Google, 2022)

After manually stopping the training, we can save the trained model weights locally to a .h5 file. The model is now ready to be tested and used.

3.8.2.6 Testing

With an ending loss value of 3.07, we can already tell that our model is not perfect. No model is, but running the training for longer would have yielded a lower loss.

There are different methods of testing how the model works in practice. For our purposes, we have decided to test it using BLEU (Bilingual evaluation understudy) (Wikipedia, n.d.), which is an algorithm for evaluating a machine learning model’s text output. It compares a model's output text (candidate) to one or more references, in our case the Flickr30k dataset captions. The output gives a score of 0-1, where a higher score is better.

We have used the nltk (Natural Language Toolkit) Python library to calculate a BLEU score for our model. We created two lists which will contain our predicted descriptions, and the actual descriptions from our dataset. We then iterate over the previously declared test data from the train test split and predict descriptions for each image in the test dataset. The predicted descriptions and actual descriptions are put in their respective lists. Now we can feed the BLEU algorithm our predicted descriptions and the actual descriptions and compare them.

We tested the BLEU score using two different wights for the function and called them BLEU-1 and BLEU-2, which is one standard method (NLTK, n.d.).

BLEU-1 is used with the weights “(1.0, 0, 0, 0)”, meaning it only considers unigram precision, meaning how many single words in the predicted descriptions are in the actual descriptions.

BLEU-2 is used with the weights “(0.5, 0.5, 0, 0)”, which means it considers both unigram and bigram precision, how many both single and two-word sequences in the predicted descriptions are in the actual descriptions.

We can adjust these weights further for N-gram words we would like to check, but for our purposes, only checking uni and bigram precision is sufficient.

Doing this, we end up with these scores shown in table 4.8-1

BLEU-1	0.558210
BLEU-2	0.307648

Table 3.8-1 – BLEU scores.

From these results we can see that our model isn't very precise, but it does consistently get words correct.

For the BLEU-1 score, we can interpret that on average our model predicts about 55.8% of single words correctly compared to the actual descriptions, which is not too bad, but not very impressive either. This suggests that our model predicts the most prevalent features of an image correctly, but not necessarily in the right context or order.

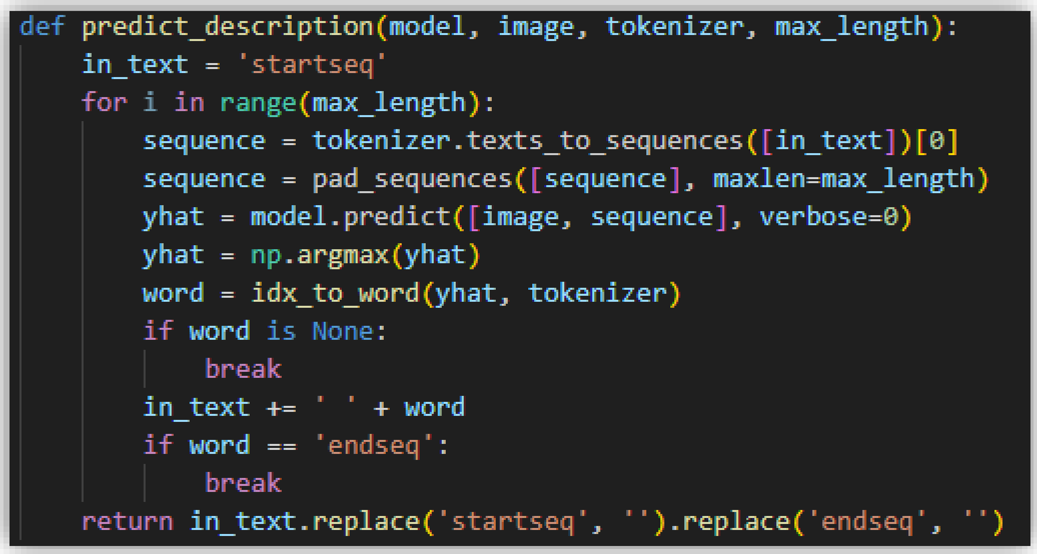
For BLEU-2, we get a lower score, which is expected since bigram precision is harder to achieve than unigram precision. It has an accuracy of about 33.8% which means it gets two consecutive words right about this much of the time. This shows that while our model gets individual words right more than half of the time, it is less effective at getting pairs of consecutive words correct.

It is also important to note that BLEU and other metrics for evaluating natural language are not perfect either, but it gives us an idea of how accurate our model is compared to the dataset's actual image descriptions.

3.8.2.7 Model results

To use the model for predicting descriptions we have made a simple separate Python script. First, we preprocess the image using the previously discussed method of converting the image to a numpy array, which can then be passed into the model prediction. This preprocessing step also allows us to use any image, no matter the size or file extension it has, making it very accessible to use.

By creating a function "predict_description", that takes in the model, tokenizer, mag length, image, and image features as arguments, we can call TensorFlow's model.predict and return its created string, which is the generated description. The image is Since we have defined the model to start the description with "startseq" and "endseq", we can simply replace all instances of these words with blanks, which leaves us with just the predicted image description.



```
def predict_description(model, image, tokenizer, max_length):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([image, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = idx_to_word(yhat, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq':
            break
    return in_text.replace('startseq', '').replace('endseq', '')
```

Figure 3.8-7 – Predict function.

Here are some example outputs using images from the Flickr30k dataset and one of its actual descriptions:

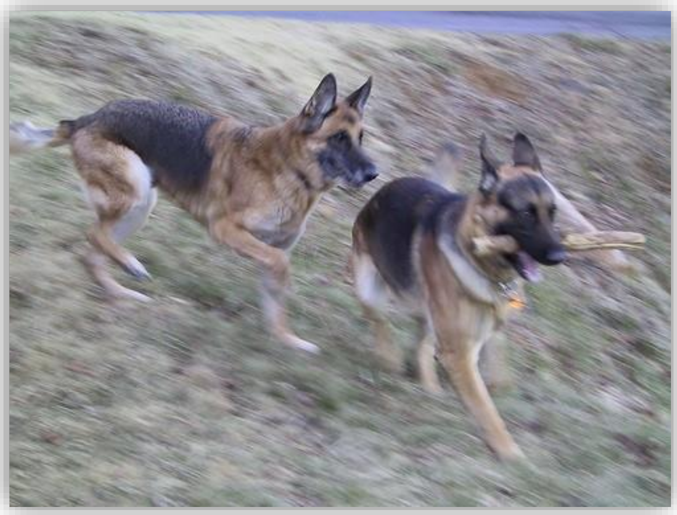


Figure 3.8-8 – Flickr30k, image 78984436.

Predicted: "two dogs are running on the beach "

Actual: "A German Shepherd dog following another German Shepherd who has a stick in its mouth. "

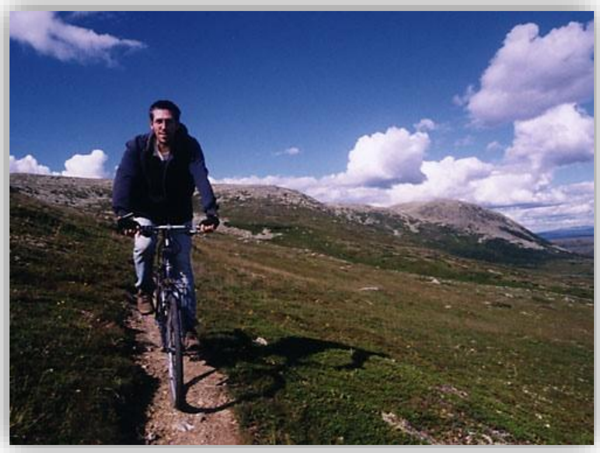


Figure 3.8-9 – Flickr30k, image 95728660.

Predicted: “man in blue shirt is riding mountain bike “

Actual: “A guy is riding a bike up the side of a hill.”

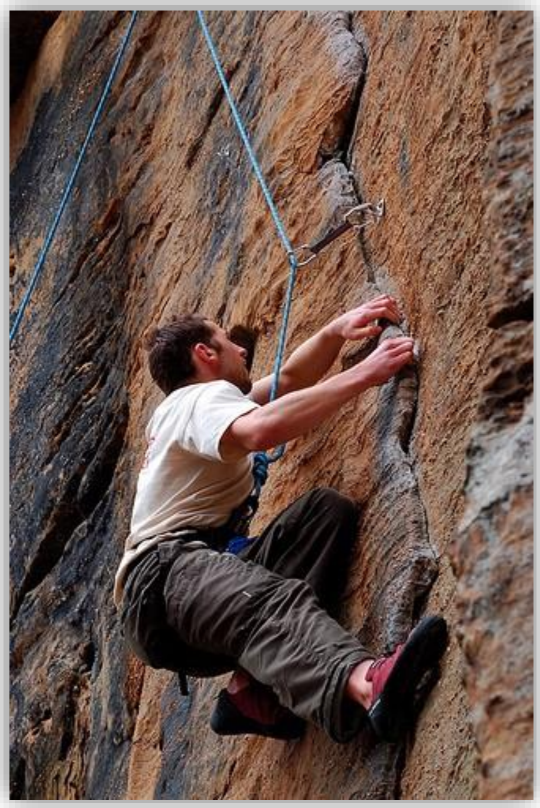


Figure 3.8-10 – Flickr30k, image 111537217.

Precited: “man climbing rock wall “

Actual: “A man climbing a rock wall”

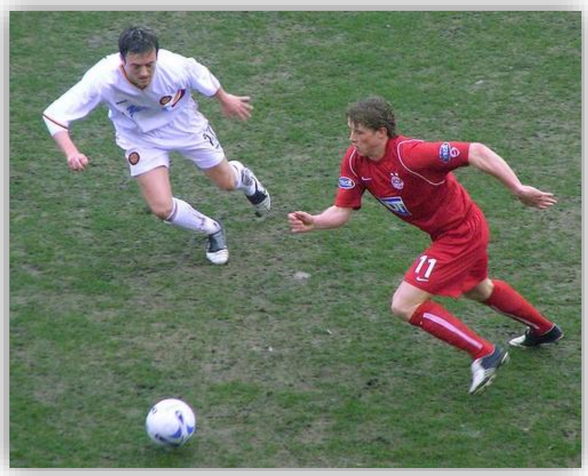


Figure 3.8-11 – Flickr30k, image 11864790.

Predicted: “two men are playing soccer in a field “

Actual:

“Two football players chase after the ball. “

The first thing to take note of here is that the predicted descriptions are a lot less detailed than the actual descriptions. Our model does, however, capture the basic features and elements of images.

3.8.3 Deploying model

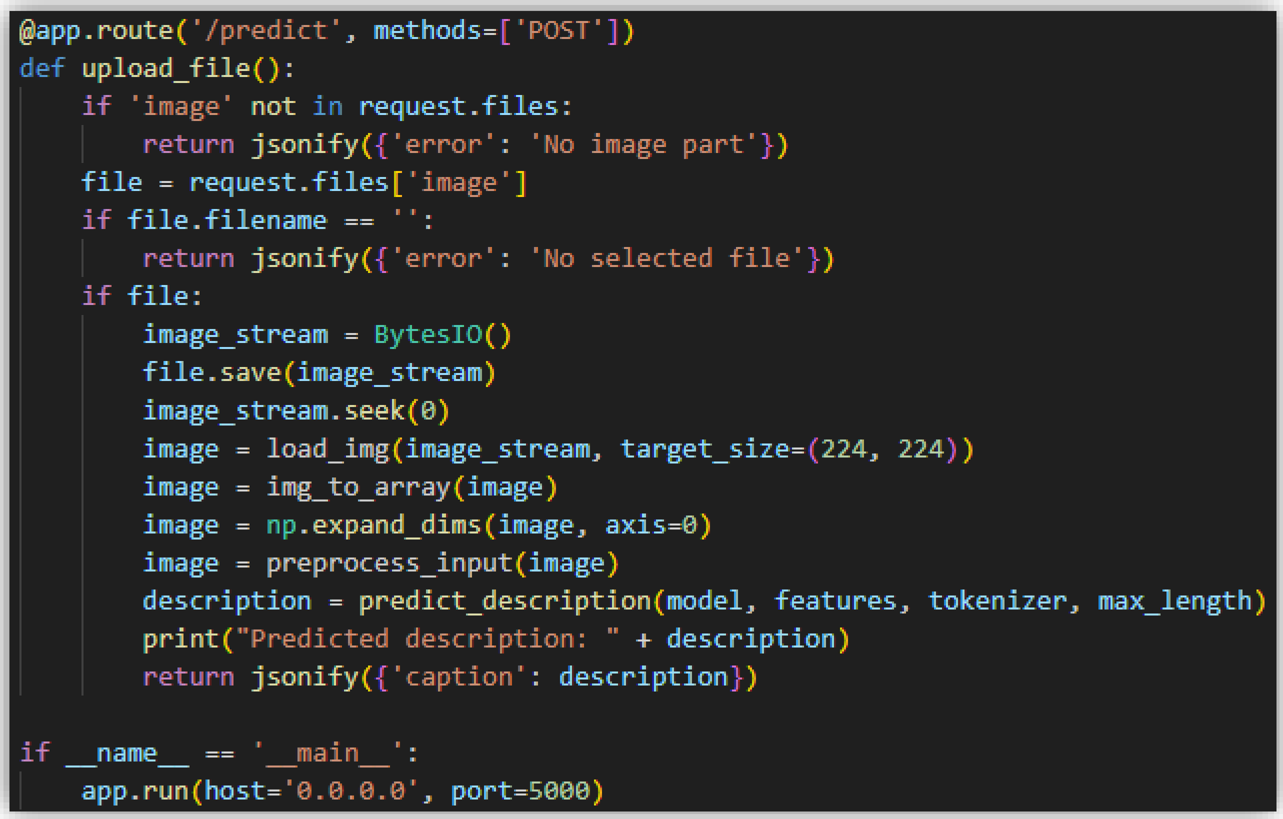
The next step is to consistently be able to run the created model. Right now, we only have the model weights, features, and tokenizer, with a script to run it locally. To be able to connect it to our web application, we need a way for the model and back end of the web application to communicate.

3.8.4 Implementing Flask

To integrate the model into our web application, we have chosen to use Flask for serving the model. Flask is a lightweight WSGI (Web Server Gateway Interface) (WSGI, n.d.) web application framework in Python. It enables rapid development and easy integration with machine learning models.

While we first tried implementing TensorFlow/Serving (TF/Serving) (TensorFlow, n.d.), it became clear that it was difficult to adjust our model to be able to be saved in a TF/Serving format and that it gave us little flexibility when it came to minor adjustments like removing the start and end tags from the generated descriptions on the server-side. Flask was chosen instead because of its simplicity and flexibility, as it is pretty straight-forward to set up and allows us to edit the generated description and return custom error messages. It provides the necessary tools to route web requests and handle sessions.

Flask is implemented within our script for predicting descriptions and is structured to handle HTTP requests to send and receive json packages. The core of the functionality is handled by Flask routes. The “/predict” route handles image uploads, processes the image, and returns the generated caption.



```python
@app.route('/predict', methods=['POST'])
def upload_file():
    if 'image' not in request.files:
        return jsonify({'error': 'No image part'})
    file = request.files['image']
    if file.filename == '':
        return jsonify({'error': 'No selected file'})
    if file:
        image_stream = BytesIO()
        file.save(image_stream)
        image_stream.seek(0)
        image = load_img(image_stream, target_size=(224, 224))
        image = img_to_array(image)
        image = np.expand_dims(image, axis=0)
        image = preprocess_input(image)
        description = predict_description(model, features, tokenizer, max_length)
        print("Predicted description: " + description)
        return jsonify({'caption': description})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Figure 3.8-12 – Flask routing.

Flask will listen for incoming POST requests on port 5000 and check the contents of the POST request. The script has been configured to only accept JSON packages with an image included. If there are no image objects or attached image file in the package, the script will return an error. If the JSON contains an image object and an image, it will proceed with image preprocessing and call the "predict_description" function using the preprocessed image. The description will then be returned to the client.

This setup defines our API endpoint for our model. The next step is to deploy the model and the serving script so that it can be run on any device.

3.8.5 Docker and Azure

For deploying the model, we have decided to use Docker, which is a free tool for testing, developing, and deploying applications. Using Docker, we can containerize our model using an image (Docker, n.d.), which is a custom, standalone and executable package that includes the minimum requirements to run a piece of software. This can include a compact operating system with libraries, runtime, configurations, and environment variables. A Docker Image can be containerized, which allows it to run on any system that supports Docker, no matter what the hardware or software the system is running on.

For our purposes, we needed a compact image that includes the necessary basics needed to run our model. For this, we chose the official Python 3.8 slim Docker Image. This image includes a basic Python 3.8 installation running on Linux. To include the other libraries needed to run our model, we have created a Dockerfile to customize the installation process. With the Dockerfile, we can add the dependencies and libraries needed to the model. This includes most of the libraries from 3.4.2. A Dockerfile is easy and simple to set up. By creating an extensionless file named “dockerfile” we have instructed Docker to build an image that uses our declared parent image, install package dependencies, and copy our local model and files to the container when running the image.

3.8.6 Evaluation Model

A requirement for this project was to use a machine learning evaluation model for evaluating the image descriptions our own model, or any other model, generates. Our supervisor, Raju Shrestha, has provided us with the models from his article on “A transformer-based deep learning model for evaluation of accessibility of images descriptions” (Shrestha, 2022), specifically developed for evaluating machine generated text in terms with the NCAM accessibility guidelines (Diagram Center, n.d.)

This includes 10 models, each trained to evaluate one of 10 guidelines listed in the article, ref. 2.1.4.1.

In order to run these 10 models, we have taken the same approach as with the ImageAble model. We have integrated the model into a Python Flask app that takes in a text string as an input and returns the 10 different evaluations created by the evaluation model. Using the same method for predicting descriptions in the model app, we iterate for each of the 10 models and save return all 10 outputs.

3.9 Testing

We test because human errors occur all the time. Finding these errors increases the overall user experience. The role of testing is to reduce the risk of errors occurring under production. The goal of testing is to find the errors, increase the trust of the application and reduce the chance of errors happening in the future. We often separate between two different types of testing: static and dynamic testing. Each has their own purpose/goal. With static testing we mean to do tests without actually running the code/application. For example, checking if which each have their own purpose/goal. Unit-testing, Acceptance testing, and maintenance testing. Unit testing’s purpose is to find as many errors as early as possible. The acceptance testing ensures that the system works as intended. Maintenance testing is to check if any new errors have been introduced after changes in the code.

3.9.1.1 Test planning.

Two of us had prior experience with software testing from the ADTS2310 – Testing of software class. If all of us did not understand the importance of testing, we got informed at the planning stage. We got informed of the 7 principles of testing at later stages of planning – especially point two and three, “Exhaustive testing is not possible” and “Early testing saves time and money” (boxuk, n.d.) Knowing that postponing testing for last would be a bad idea. Beginning the testing as early as possible would let us find errors easier than doing it in the later stages. So, we also planned to implement a routine of agile testing where we did the testing when a new method was implemented or as soon as possible. We also thought it would be good to split the testing and programming to different team members, meaning if someone programmed a method then someone else would do the testing for that method. This allows us to have more “set of eyes” on the code. The person writing the code might be blind to his own code resulting in less effective tests.

Static testing:

- Test-cases: Creating test case forms which would help us get an idea of all the different inputs that can happen in a method and with that try to cover all decisions.

Dynamical testing:

- Unit testing: This is a more thorough test where we exclusively look into the function in detail. This allows us to refine the code and improve code quality (Geeks for geeks, 2024). We can use the test cases to see if we cover all of the decision areas in the actual code with unit testing. We planned to use an integrated tool in Visual Studio called, NUnit to do these tests.

- System testing: Here we test the whole application as one unit. This involves doing tests on functions to see if we get the expected result and stress testing to see how much the application can handle (Yasar & Black, 2023). Since we have prior experience with Selenium, which can cover these tests, we saw it reasonable to use it for the system testing.
- Acceptance testing: This type of testing involves presenting the application and demonstrating its features to ensure they meet the requirements and expectations. It allows the developers to get feedback at an early stage. (Testsigma, n.d.)

3.9.1.2 The test processes.

3.9.1.3 Acceptance testing w/ supervisor.

We can call parts of our bi-weekly meetings with the supervisor as an informal acceptance testing. From early stages of our project, we presented our web application in these meetings. We ran the application and tried out the new functions we had been working on. This gave us valuable feedback although it was not a formal acceptance testing. These meetings were collaborative focusing on the requirements and how we should or had implemented them.

3.9.1.4 Testcases

To get a better overview of the necessary tests on our most important methods/functions we decided to create testcases. The table will try to cover all possible ways the method/function could be triggered. It helps us to think about what decisions we should consider in the coding and also helps us later with what to test in the unit test.

#1 - Generating description from custom endpoint

Testcase #	Input	Expected Result
1	No input data	Bad request (invalid input)
2	No images provided	Bad request (invalid input)
3	Empty image list	Bad request (invalid input)
4	Valid images, no endpoint	Bad request (invalid input)
5	Valid images, valid endpoint, API fails	Bad request (API error)
6	Valid images, valid endpoint, API success	Returns descriptions (OK)

#2 - Saving image to gallery page

Testcase #	Input	Expected Result
1	No input data	Bad request (invalid input)
2	No images provided	Bad request (invalid input)
3	No description provided	Bad request (invalid input)
4	No evaluation provided	Bad request (invalid input)
5	Different count of image files, description and evaluation	Bad request (invalid input)
6	User not logged in.	Forbidden (401)
7	Valid image files, descriptions, and evaluations, but an error occurs during processing	Server error (500)
8	Valid image files, descriptions, and evaluations, processing succeeds	Returns OK (200)

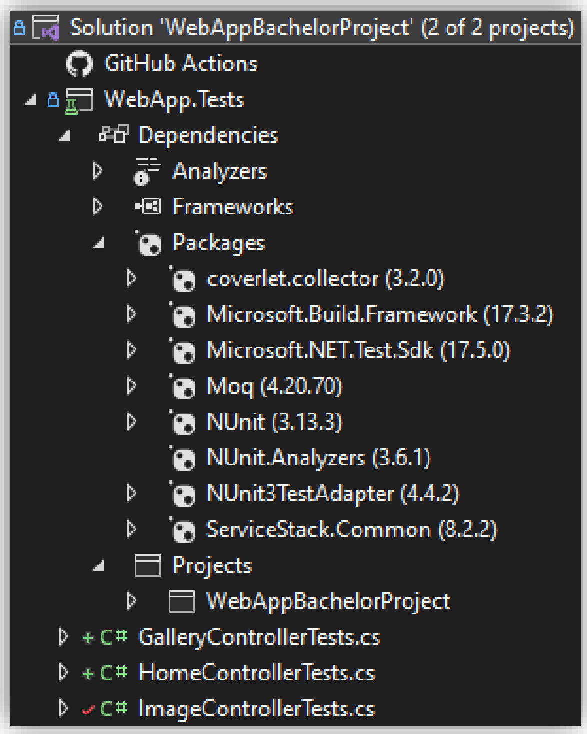
#3 - DOWNLOADING IMAGE WITH METADATA

Testcase #	Input	Expected Result
1	No input data	Bad request (invalid input)
2	No images provided	Bad request (invalid input)
3	No description provided	Bad request (invalid input)
4	No evaluation provided	Bad request (invalid input)
6	Valid image file, description, and evaluation, but a general error occurs during processing	Server error (500)
7	Valid image file, description, and evaluation, processing succeeds	Returns image containing metadata

3.9.2 The unit tests.

3.9.2.1 Structure for unit testing.

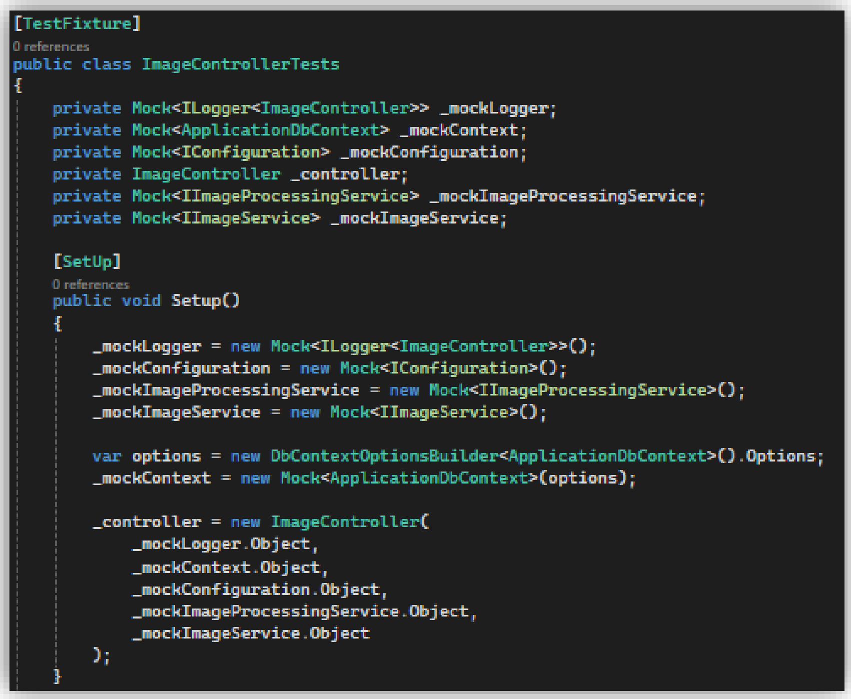
First, we created a new project under the solution where also the web application is located and called it *WebApp.tests*. We had to link the test project to the web application, we did this by simply adding a reference to it. We saw it reasonable to split the unit tests of different controllers to each C# file with their classes. For example, ImageControllerTests.cs.



```
Solution 'WebAppBachelorProject' (2 of 2 projects)
    GitHub Actions
  WebApp.Tests
    Dependencies
      Analyzers
      Frameworks
      Packages
        coverlet.collector (3.2.0)
        Microsoft.Build.Framework (17.3.2)
        Microsoft.NET.Test.Sdk (17.5.0)
        Moq (4.20.70)
        NUnit (3.13.3)
        NUnit.Analyzers (3.6.1)
        NUnit3TestAdapter (4.4.2)
        ServiceStack.Common (8.2.2)
      Projects
        WebAppBachelorProject
    GalleryControllerTests.cs
    HomeControllerTests.cs
    ImageControllerTests.cs
```

Figure 3.9-1

Before writing the actual tests, we need to decorate the class we want to test with [TestFixture]. Then, we set up the test with mocks. These mocks imitate the controller class, allowing us to call it in the tests. Below is an example of how we set up the ImageControllerTest. It replicates the construction of the ImageController.



```
[TestFixture]
0 references
public class ImageControllerTests
{
    private Mock<ILogger<ImageController>> _mockLogger;
    private Mock<ApplicationDbContext> _mockContext;
    private Mock<IConfiguration> _mockConfiguration;
    private ImageController _controller;
    private Mock<IImageProcessingService> _mockImageProcessingService;
    private Mock<IImageService> _mockImageService;

    [SetUp]
    0 references
    public void Setup()
    {
        _mockLogger = new Mock<ILogger<ImageController>>();
        _mockConfiguration = new Mock<IConfiguration>();
        _mockImageProcessingService = new Mock<IImageProcessingService>();
        _mockImageService = new Mock<IImageService>();

        var options = new DbContextOptionsBuilder<ApplicationDbContext>().Options;
        _mockContext = new Mock<ApplicationDbContext>(options);

        _controller = new ImageController(
            _mockLogger.Object,
            _mockContext.Object,
            _mockConfiguration.Object,
            _mockImageProcessingService.Object,
            _mockImageService.Object
        );
    }
```

Figure 3.9-2 – Setup for ImageControllerTests

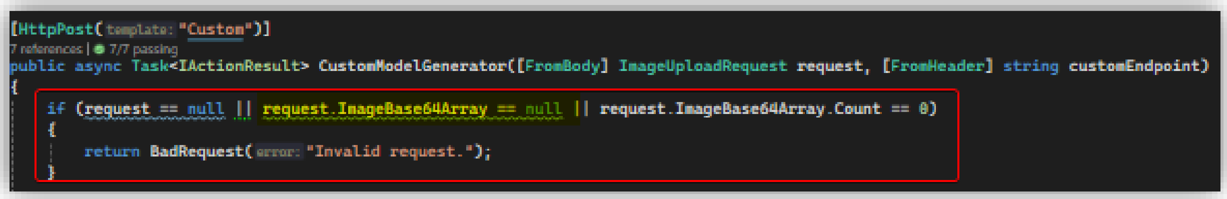
FIGURE X – SETUP FOR IMAGECONTROLLERTESTS

After the setup is done, we can begin writing the tests for the different functions. Throughout the whole unit testing we used the AAA concept. It is a three-step process where we first arrange the test, for example creating the object or initializing the parameters. The next step is the act which executes the test and runs the actual function with the way we ‘arranged’ it in the first step. At last, we have the Assert which checks and verifies the result with expected results. (Dash, 2023)

In the example below we can see the AAA concept in a simple test. We want to test the customModelGenerator to return bad request if ImageBase64Array is null. Firstly, we arrange that ImageBase64Array is set to null. In the act we call the customModelGenerator with a random endpoint, the important part here is that it is not null, because that it not what we are testing in this test. Finally, we assert – verifying if it will return a bad request.

3.9.2.2 Example of a unit test

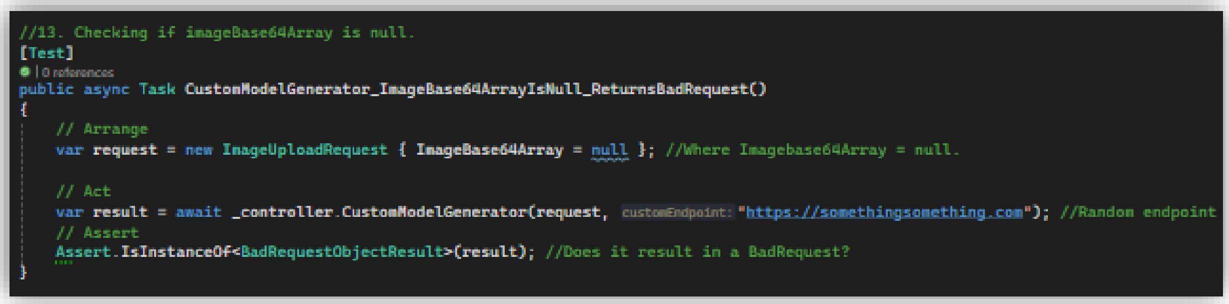
In the example below, we demonstrate the AAA (Arrange, Act, Assert) concept in a simple test. Our goal is to test that the CustomModelGenerator returns a bad request if ImageBase64Array is null. The test shown will try to test whether the if-condition in the ImageController.CustomModelGenerator function succeeds as expected. This test is also an included in the testcase for CustomModelGenerator.



```
[HttpPost(template: "Custom")]
7 references | 7/7 passing
public async Task<IActionResult> CustomModelGenerator([FromBody] ImageUploadRequest request, [FromHeader] string customEndpoint)
{
    if (request == null || request.ImageBase64Array == null || request.ImageBase64Array.Count == 0)
    {
        return BadRequest(error: "Invalid request.");
    }
```

Figure 3.9-3

1. First, we decorate the class with a [Test].
2. Then, we arrange by setting ImageBase64Array to null.
3. In the act, we call the customModelGenerator with a random endpoint. The endpoint isn't crucial for this test since it's not the focus of our verification.
4. Finally, we assert by checking if it returns a bad request.



```
//13. Checking if imageBase64Array is null.
[Test]
| 0 references
public async Task CustomModelGenerator_ImageBase64ArrayIsNull_ReturnsBadRequest()
{
    // Arrange
    var request = new ImageUploadRequest { ImageBase64Array = null }; //Where Imagebase64Array = null.

    // Act
    var result = await _controller.CustomModelGenerator(request, customEndpoint: "https://somethingsomething.com"); //Random endpoint
    // Assert
    Assert.IsInstanceOf<BadRequestObjectResult>(result); //Does it result in a BadRequest?
}
```

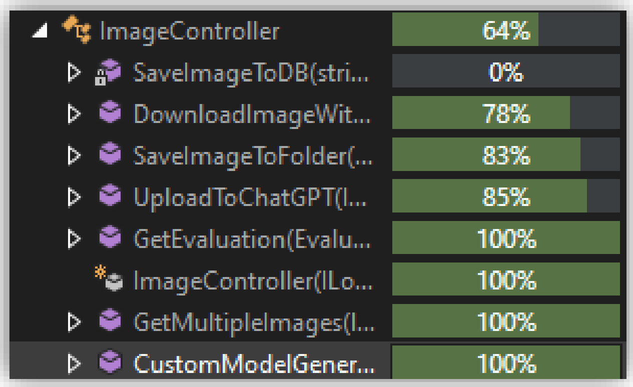
Figure 3.9-4

3.9.3 Results on unit testing

3.9.3.1 Code Coverage

After setting up our unit tests gathered a thorough evaluation of the test for each controller in our application. We used .NET-coverage utility extension to calculate this for us. The results are shown below:

Controller	Total Tests Conducted	Code Coverage percentage
ImageController	35	64%
GalleryController	3	19%
HomeController	0	0%



```
◢ ImageController           64%
  ▷ SaveImageToDB(stri...     0%
  ▷ DownloadImageWit...      78%
  ▷ SaveImageToFolder(...    83%
  ▷ UploadToChatGPT(I...     85%
  ▷ GetEvaluation(Evalu...  100%
    ImageController(ILo...  100%
  ▷ GetMultipleImages(I...  100%
  ▷ CustomModelGener...     100%
```

Figure X – Code coverage percentage for the image controller.

3.9.3.2 Reflection

Our unit testing results shows that even though we created a lot of tests, the overall code coverage is lower than desired. It was unexpectedly low. This indicates that there still are areas in the controller that were not tested thoroughly. There are several reasons for this.

1. Some parts of the code contained code that was difficult to test. For example, functions that invoked multiple services and repositories. We lacked knowledge of how to test this efficiently.
2. Lot of late changes in the code made many of our tests to not work so we had to redo a lot of the unit testing.
3. Not all of us had experience with testing so there was limited of resources and time.

3.9.3.3 Future considerations

It is too late to make these changes in our project, but we recognize that we could have improved the unit testing and code coverage. We should have:

1. Increase our knowledge to conduct tests on more complex logic.
2. Allocated more time to the unit testing.

3.9.4 Discussion:

When looking back at the testing, we clearly see that we did not finish what we had planned. We should have done more planning, to minimize having to rewrite much of the code and retest. Creating more static tests, especially for each crucial function would benefit us. It would make the testing more efficient, instead, it seems like we relied heavily on trial and error. Having to redo much of the code had a big impact on what we managed to do.

We also see that we are in need of a system test to be surer of the application we built. We missed out on stress testing which could have given us valuable insights on high traffic, potential bottlenecks and more.

4 Conclusion

4.1 Concluding summary

In this project, we aimed to develop a web application for image accessibility. We have successfully created a working web app, ImageAble, that fulfils the requirements for the given project. All functional and non-functional requirements have been met. The web app consists of these main parts: The website itself, a database that saves user accounts and images, a robust back end which allows for connecting to other machine learning models via an API, a successfully connected image description generator model and an image description evaluation model. The project has met its objectives by providing a functional tool that adheres to accessibility standards. It is, however, not perfect, and we see a lot of ways this application could be improved even further.

We did not have enough time to create a web extension that would link up to the web application, which we had hoped to do as it would expand the possible use cases for the application. Some parts of the application could be improved, especially the description generator model. The model fails to recognise detailed features within images and could have handled errors better. It has a hard time recognising features of images with no living entities like animals or humans. We realise this is because we did not have the resources to properly train the model but are happy that we managed to create a working model, even if it is not perfect.

Most of the time spent was the process of implementing functionalities, architecture and design into the web application itself. The front-end design part of the application was the most uncomplicated implementation we had to do. However, typically the back-end part is the most demanding part of the application. Connecting the front-end and back-end for it to communicate to ensure the user interaction from the front-end side works as it should, we had to adjust back-end code to meet a user-friendly interaction expectation.

We are very happy with how the project turned out overall.

4.2 Personal learning and improvements

Throughout this project, our team gained significant insights into web accessibility, web development, and machine learning. This has been a very challenging process, but seeing the final result has been fulfilling for us. The group has had different courses throughout our degree which has made it challenging to help each other in some cases, especially when it came to application testing and developing in .NET as only half our group has had introductory courses in these subjects.

None of us had any idea how to make an image description model when we took upon this project, but after weeks of research and development, we managed to create one ourselves. Even though we still don’t have a complete understanding of how everything within it works, we have drastically improved our understanding on the subject.

We have also improved our knowledge of APIs and connecting different parts of a project. Making sure the web application and the models connect correctly has been an interesting and educational experience, as is for developing a feature to use custom API endpoints.

Working as a team for such a big project has also had its challenges. Planning this far ahead is something we have never done before and has been a fantastic learning experience. We do however admit that it could have gone better. Demotivation from being stuck in certain parts of the project made us fall behind schedule, but we managed to catch up once we overcame those obstacles. If there’s something, we would have done differently it was to work evenly throughout the semester.

In conclusion, this project has been an invaluable experience that has not only expanded our technical skills but also taught us important lessons in teamwork and project management. The challenges we faced and the solutions we developed have prepared us well for future projects in our careers. We are proud of what we have achieved and look forward to applying the knowledge and skills gained from this project to future projects and professional opportunities. We are excited about the potential impact our work can have on making technology more inclusive and accessible for all users.

5 References

aswintechguy. (n.d.). *GitHub*. Retrieved from GitHub web site: https://github.com/aswintechguy/Deep-Learning-Projects/tree/main/Image%20Caption%20Generator%20-%20Flickr%20Dataset

Babich, N. (2019, January 17). *UXplanet*. Retrieved from UXplanet web site: https://uxplanet.org/using-red-and-green-in-ui-design-66b39e13de91

Baig, I. U. (2023, August 13). *Medium*. Retrieved from Medium web site: https://medium.com/@IrfanUlahBaig/comparing-the-aesthetics-of-rounded-and-rectangular-buttons-6294cae40061

boxuk. (n, d). *The seven principles of testing*. Retrieved from boxuk.com: https://www.boxuk.com/insight/the-seven-principles-of-testing/

boxuk. (n.d.). *The seven principles of testing*. Retrieved from boxuk: https://www.boxuk.com/insight/the-seven-principles-of-testing/

Compassionate Design. (2024, February 29). *LinkedIN*. Retrieved from LinkedIN web site: https://www.linkedin.com/pulse/leveraging-shadows-uxui-design-tale-enhanced-user-v5ihc/

Dash, D. (2023, November 16). *C-sharp Corner*. Retrieved from C-Sharp corner website: https://www.c-sharpcorner.com/article/introduction-to-nunit-testing-framework)

Diagram Center. (n.d.). *Diagram Center*. Retrieved from Diagram Center web site: http://diagramcenter.org/table-of-contents-2.html

Docker. (n.d.). *Docker*. Retrieved from Docker Docs: https://docs.docker.com/guides/docker-concepts/the-basics/what-is-an-image/),

Eby, K. (2017, February). *What's the Difference? Agile vs Scrum vs Waterfall vs Kanban*. Retrieved from smartsheet.com: https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban

Freed, G., & Rothberg, M. (2006, April). *GBH Educational Foundation*. Retrieved from WGBH: https://www.wgbh.org/foundation/services/ncam/accessible-digital-media-guidelines

Geeks for geeks. (2024, May 23). *Geeks for geeks*. Retrieved from Geeks for geeks web site: https://www.geeksforgeeks.org/unit-testing-software-testing/

Geeks for geeks. (2024, May 22). *Geeks for Geeks*. Retrieved from Geeks for Geeks website: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/

Google. (2022, July 18). *Google Developers*. Retrieved from Google developer documentation: https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

Google. (n.d.). *Google Fonts*. Retrieved from Google Fonts web site: https://fonts.google.com/specimen/Inter

Great learning. (2021, September 23). *Medium*. Retrieved from Medium web page: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918.

H$anke$ara. (n.d.). *Kaggle*. Retrieved from kaggle web site: https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset

Hacker's realm. (2022, April 27). *Hacker's realm*. Retrieved from Hacker's realm website: https://www.hackersrealm.net/post/image-caption-generator-using-python

Harvard University. (n.d.). *Digital Accessibility*. Retrieved from Harvard.edu: https://accessibility.huit.harvard.edu/use-images-and-media-enhance-understanding#:~:text=Images%20and%20media%20are%20powerful,reinforcing%20information%20provided%20in%20text.

Henry, S. L. (2024, March 7). *World Wide Web Consortium*. Retrieved from W3: https://www.w3.org/WAI/standards-guidelines/wcag/).

IBM. (2021, March 4). *IBM*. Retrieved from IBM web site: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=page-asynchronous-javascript-xml-ajax-overview#:~:text=Asynchronous%20JavaScript%20and%20XML%20(Ajax,user%20makes%20an%20input%20change

Image-Net. (n.d.). *Image-Net*. Retrieved from Image-Net web site: https://www.image-net.org/download.php

International Business Machines Corporation. (n.d). *IBM*. Retrieved from IBM website: https://www.ibm.com/topics/automation

Javatpoint. (n.d.). *Javatpoint*. Retrieved from Javatpoint web site: https://www.javatpoint.com/css-button-hover-effects

Martins, J. (2024, January 19th). *What is Kanban? Free Kanban template, with examples*. Retrieved from Asana.com: https://asana.com/resources/what-is-kanban

Microsoft. (2021, May 25). *Microsoft Learn*. Retrieved from Microsoft Learn web site: https://learn.microsoft.com/en-us/ef/core/

Microsoft. (2022, February 23). *Microsoft Learn*. Retrieved from Microsoft Learn web site: https://learn.microsoft.com/en-gb/windows/win32/gdiplus/-gdiplus-constant-property-item-descriptions

Microsoft. (2022, June 3). *Microsoft Learn*. Retrieved from Microsoft Learn web site: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/anchor-tag-helper?view=aspnetcore-8.0

NLTK. (n.d.). *NLTK*. Retrieved from NLTK Documentation: https://www.nltk.org/_modules/nltk/translate/bleu_score.html

Shrestha, R. (2022, June 21). *ACM digital library*. Retrieved from A transformer-based deep learning model: https://dl.acm.org/doi/pdf/10.1145/3529836.3529856

Simonyan, K., & Zisserman, A. (2015). *Arxiv*. Retrieved from Arxiv web site: https://arxiv.org/pdf/1409.1556

TensorFlow. (n.d.). *TensorFlow*. Retrieved from TensorFlow documentation: https://www.tensorflow.org/tfx/guide/serving

Testsigma. (n.d.). *Test Sigma*. Retrieved from Test Sigma Web Site: https://testsigma.com/guides/acceptance-testing/

Unacademy. (n.d.). *Unacademy*. Retrieved from Unacademy web site: https://unacademy.com/content/bank-exam/study-material/computer-knowledge/authorization-an-important-measure-in-computer-security/#:~:text=Authorization%20is%20an%20important%20measure,resources%2C%20and%20other%20system%20objects

WCAG. (n.d.). *Designing for Web Accessibility*. Retrieved from w3.org: https://www.w3.org/WAI/tips/designing/

WebAIM. (n.d.). *WebAIM*. Retrieved from WebAIM web site: https://webaim.org/resources/contrastchecker/

Wikipedia. (n.d.). *Wikipedia*. Retrieved from Wikipedia BLEU article: https://en.wikipedia.org/wiki/BLEU

Wolford, B. (n.d.). *GDPR EU*. Retrieved from GDPR EU web site: https://gdpr.eu/what-is-gdpr/

WSGI. (n.d.). *WSGI*. Retrieved from WSGI documentation: https://wsgi.readthedocs.io/en/latest/what.html

Yasar, K., & Black, R. (2023, March). *Tech Target*. Retrieved from Tech target web page: https://www.techtarget.com/searchsoftwarequality/definition/system-testing