FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Meme generator
Student:	Bc. Ondřej Závodný
Supervisor:	Ing. Michal Valenta, Ph.D.
Study program:	Informatics
Branch / specialization:	Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2024/2025

Instructions

The goal of the thesis is to design, implement and test a tool for generating captioned memes from manageable templates.
1. Research existing meme generators.
2. Design and implement:
   a. an application for managing meme templates, including graphical and layout configuration (e.g. font settings and placement) in a graphical user interface;
   b. an application for generating memes via parametrization of existing templates.
3. Test both applications using functional tests and conduct usability testing.

Electronically approved by Ing. Michal Valenta, Ph.D. on 2 January 2024 in Prague.

Master's thesis

MEME GENERATOR

Bc. Ondřej Závodný

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Jaroslav Šmolík
January 11, 2024

Citation of this thesis: Závodný Ondřej. *Meme Generator*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

List of Figures

List of code listings

I would like to thank my fiancé for her support and also to Ing. Michal Valenta, Ph.D. for taking the time to help me with the thesis.

Declaration

In Prague on January 11, 2024 ....

Abstract

This masters thesis is focused on developing an application that enables management of meme templates and creating memes from them. It evaluates some existing meme generators and describes their functionality. The final application is built into a docker image and deployed to production. The application was tested using end-to-end tests and in usability testing.

Keywords meme, meme template, image editing, web application, frontend, nextjs, docker

Abstrakt

Tato magisterská práce se zabývá vývojem aplikace pro správu šablon memů a nasledným generováním memů z těchto šablon. Jsou evaluovány existující generátory memů a popsána jejich funkcionalita. Výsledná aplikace je sestavena do docker image a poté nasazena do produkce. Aplikace byla otestována end-to-end testy a byla součástí usability testingu.

Klíčová slova meme, memová šablona, editace obrázků, webová aplikace, frontend, nextjs, docker

Introduction

This thesis presents an in-depth exploration of one of the digital age's latest phenomenon: memes. The term "meme," originally coined by Richard Dawkins in 1976,[1], originally described as small units of culture that spread from person to person by copying or imitation, has evolved in the internet era to describe images with text meant for sharing and virality.

The initial chapter offers an overview of current meme generators and introduces the developed application, setting the stage for a comparative analysis. The subsequent chapter delves into the technical aspects of the application's implementation, including the evaluation of frontend libraries, and the use of Docker for building and deploying the application.

Central to this study is the development and analysis of a web application designed for creating and sharing memes. This application enables users to generate memes using both existing templates and user-created designs, focusing on the ease of sharing these creations.

Concluding the thesis, the final chapter is devoted to testing, emphasizing end-to-end, component and usability testing methodologies. This ensures the application's functionality and user-friendliness, thereby highlighting the effectiveness of the developed solution in the broader context of digital meme creation.

Chapter 1

Analysis

1.1 Existing solutions

There are many solutions online that allow users to create memes from templates.[2] These solutions are usually free to use, but they offer some extra features that are locked behind a subscription. The subscription usually unlocks the ability to remove the watermark from the generated memes, or the ability to use the application without ads. Two websites were chosen for a more detailed analysis, Imgflip and iLoveIMG. Imgflip is consistently ranked as one of the most popular meme generators on the internet[2][3], and iLoveIMG is one of the easiest sites to use when an inexperienced user wants to create a meme.[2] In the mobile application space, Mematic was chosen for a more detailed analysis, because it is often mentioned as one of the best meme generators on the market.[4][5][6]

1.1.1 Popularity

Popularity is difficult to find for websites. Android and iOS applications have a ranking system and a rough number of downloads listed in the respective stores, which can be used together to gain an insight into the popularity. Websites do not have such options, but search engines, like google, track search terms and make them available for analysis. Google offers the Google Trends, which shows how many times was each search term was searched in Google. Since this is not at all comparable to the any of the Android/iOS application metrics, websites and applications will not be compared between each other in this section.

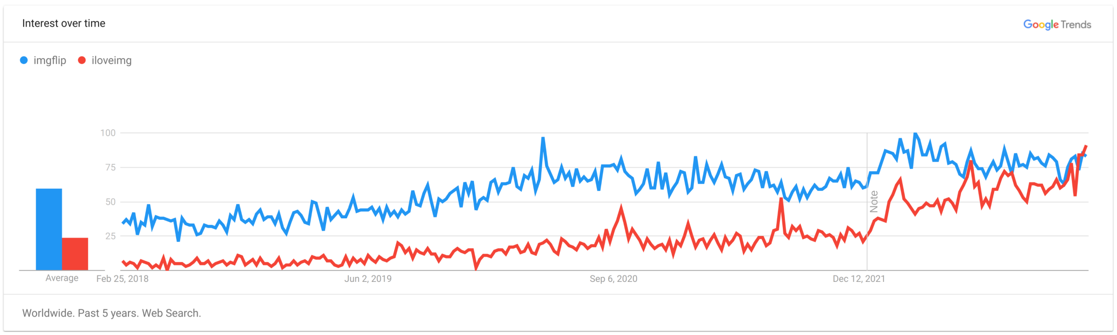


Figure 1.1 Google Trends for the search terms `imgflip` and `iloveimg` for the past 5 years. Taken on February 21. 2023.[7]

A graph from Google Trends is shown in the image 1.1, which shows that Imgflip was more searched than iLoveIMG for the past 5 years, except for January and February 2023, so it can be said that Imgflip is more popular than iLoveIMG.

Mematic has over 1 Million downloads on 48.9 Thousand reviews on Google Play[8] and 150.1 Thousand reviews on the App Store[9].

1.1.2 Imgflip Meme Generator

Imgflip Meme Generator [1] is a free online image maker that lets users add custom resizable text, images, and much more to templates. The generator can be used to customize established memes, such as those found in Imgflip's collection of Meme Templates [2]. It is also possible to upload custom templates or start from scratch with empty templates [10].

1.1.2.1 Creating a meme

The main Imgflip screen is shown in the image 1.2a. The image is split into four sections labeled with Roman numerals.

1.1.2.1.1 Section I

Section I. is the graphical editor of memes. There are 4 buttons at the top that can be used to manipulate the image. The first button rotates the whole image. The second button adds space, either to the top or to the button of the image. The third button allows the users to add some other images onto the background. The last button is used to draw on the image manually, being able to change the color and the thickness of the drawing tool. Below these buttons is the graphical editor. Here the user can move, rotate and resize text fields and images added to the meme.

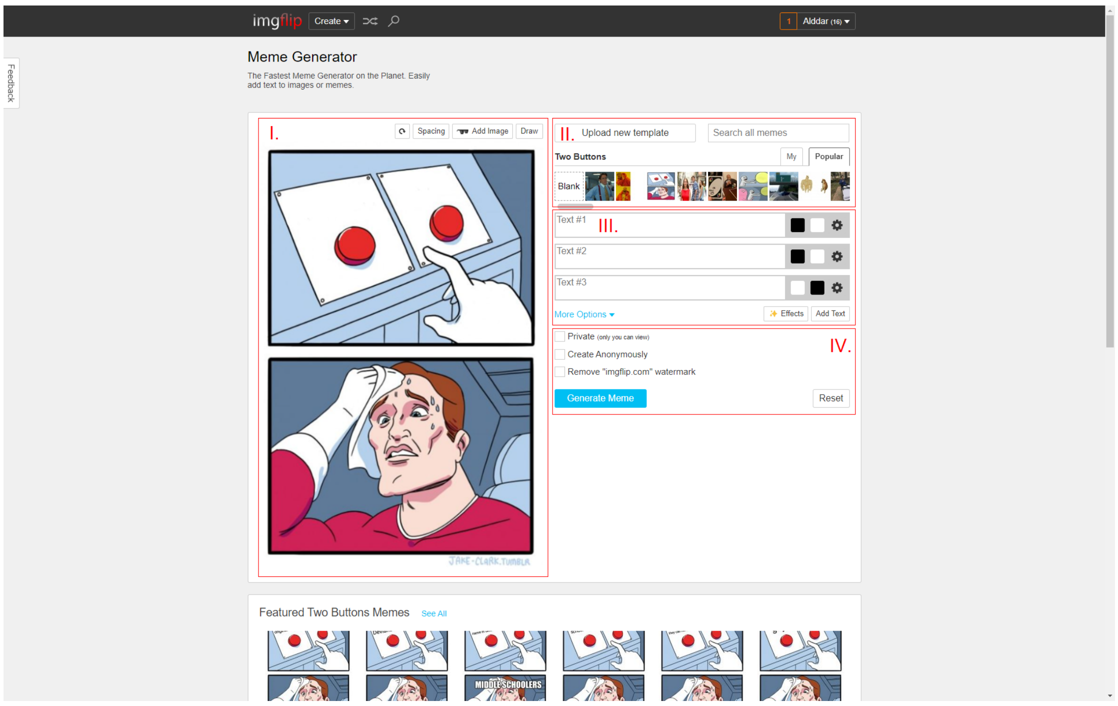
1.1.2.1.2 Section II

Section II. is where the user can upload a new template (see subsubsection 1.1.2.2), or select from one of the many templates made by other users. There are two tabs with templates, one has all the templates created by the current user and the other one has a selection of the most popular templates on the platform. There is also a search bar. If the user starts typing, a search dialog, which is shown in image 1.2b, will appear. It has pictures and names of all the relevant search results.

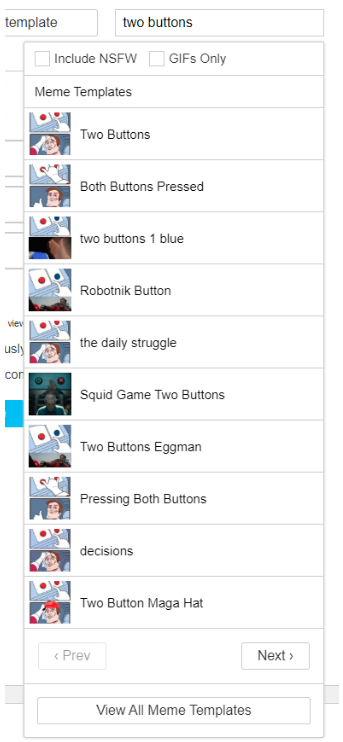
1.1.2.1.3 Section III

Section I. is where the user modifies other attributes of the image, mainly the texts. Each text field in section I. corresponds to a text box in the graphical image editor in section I. When any property in section III is changed, it is automatically updated in Section I. as well. The main text options are the actual text, its color and its outline color. The settings icon shows a dialog depicted in image 1.2f, which contains many more text options, like font, outline width, max font size and others. There isn't an option for font size, the text will always try to take up the maximum available space, but it is possible to specify a maximum font size. Below the text options is the `Add text` button and the `Effects` button. There are many effects to choose from, as shown in image 1.2d.

[1] https://imgflip.com/memegenerator
[2] https://imgflip.com/memetemplates



(a) Main Imgflip meme generator screen.


template | two buttons

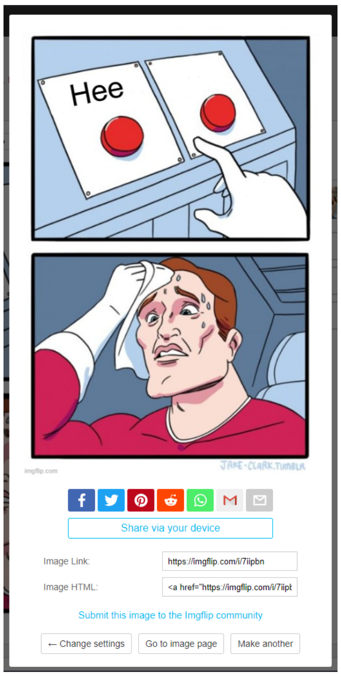
Include NSFW
GIFs Only

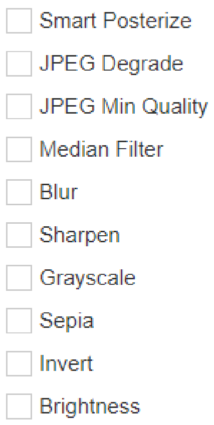
Meme Templates

- Two Buttons
- Both Buttons Pressed
- two buttons 1 blue
- Robotnik Button
- the daily struggle
- Squid Game Two Buttons
- Two Buttons Eggman
- Pressing Both Buttons
- decisions
- Two Button Maga Hat

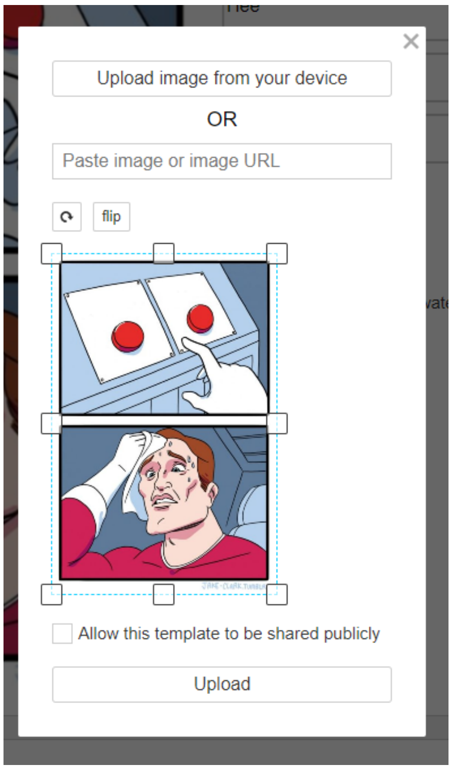
‹ Prev
Next ›

View All Meme Templates


Hee
Share via your device
Image Link: https://imgflip.com/i/7iipbn
Image HTML: <a href="https://imgflip.com/i/7iipt
Submit this image to the Imgflip community
← Change settings
Go to image page
Make another


- Smart Posterize
- JPEG Degrade
- JPEG Min Quality
- Median Filter
- Blur
- Sharpen
- Grayscale
- Sepia
- Invert
- Brightness

(b) Search dialog **(c)** Generated meme dialog. **(d)** Available effects



Upload image from your device

OR

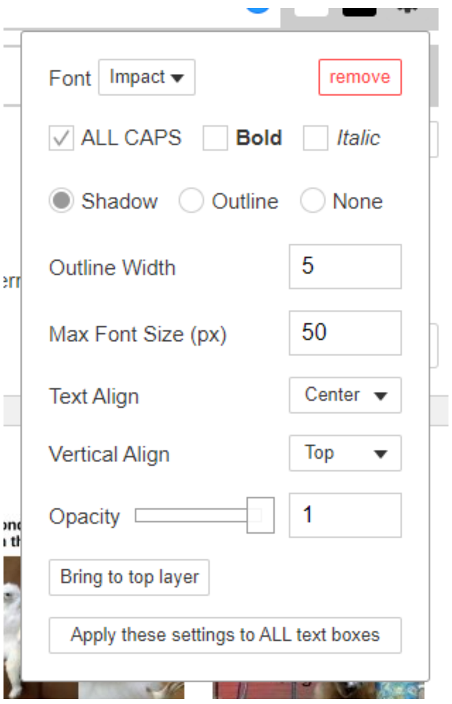
Paste image or image URL

flip

Allow this template to be shared publicly

Upload

(e) Upload dialog



Font: Impact ▾ — remove

[✓ ALL CAPS] ☐ **Bold** ☐ *Italic*

[● Shadow] ○ Outline ○ None

Outline Width: 5

Max Font Size (px): 50

Text Align: Center ▾

Vertical Align: Top ▾

Opacity: 1

Bring to top layer

Apply these settings to ALL text boxes

(f) Text Options

Figure 1.2 Screenshots of Imgflip

1.1.2.1.4 Section IV.

Section IV. is the final publishing of the meme. There is an option to make the meme private and also the option to create the meme anonymously. There is also an option to not have the `imgflip.com` watermark on the final generated meme, but that isn't available without getting a subscription (see subsubsection 1.1.2.4)

1.1.2.2 Uploading a template

When the user clicks the "Upload new template" button, an upload dialog opens. This dialog is depicted in image 1.2e. Here there are three possible ways to upload an image. The first one is to select it from the device. The second one is to paste an Uniform Resource Locator (URL) to the desired image. The third one is to paste an image from the clipboard.

After an image is uploaded, it can be flipped, rotated and cropped. There is also an option to make this template public, which means it will be accessible to other users of the platform.

1.1.2.3 Generating a meme

When the `generate meme` button is clicked, it shows the generated meme dialog shown in image 1.2c. The generated image is at the top of the dialog. There are many possible ways how to share the meme with people using several social networks and means of communication. The dialog has a generated link to the image and also an HTML code. There is also a `Submit this image to the Imgflip community` link, which would post it on the Imgflip social network.

1.1.2.4 Subscriptions

Imgflip is free to use, but it offers some extra features which are locked behind an "Imgflip Pro" subscription. The features that are available for the Meme Generator in the Pro version are the ability to remove the "imgflip.com" watermark from generated memes and the ad-free version of the page. There are also many other features for the other Imgflip products, like the Animated GIF Maker, which aren't a part of this thesis.

1.1.3 Iloveimg Meme Generator

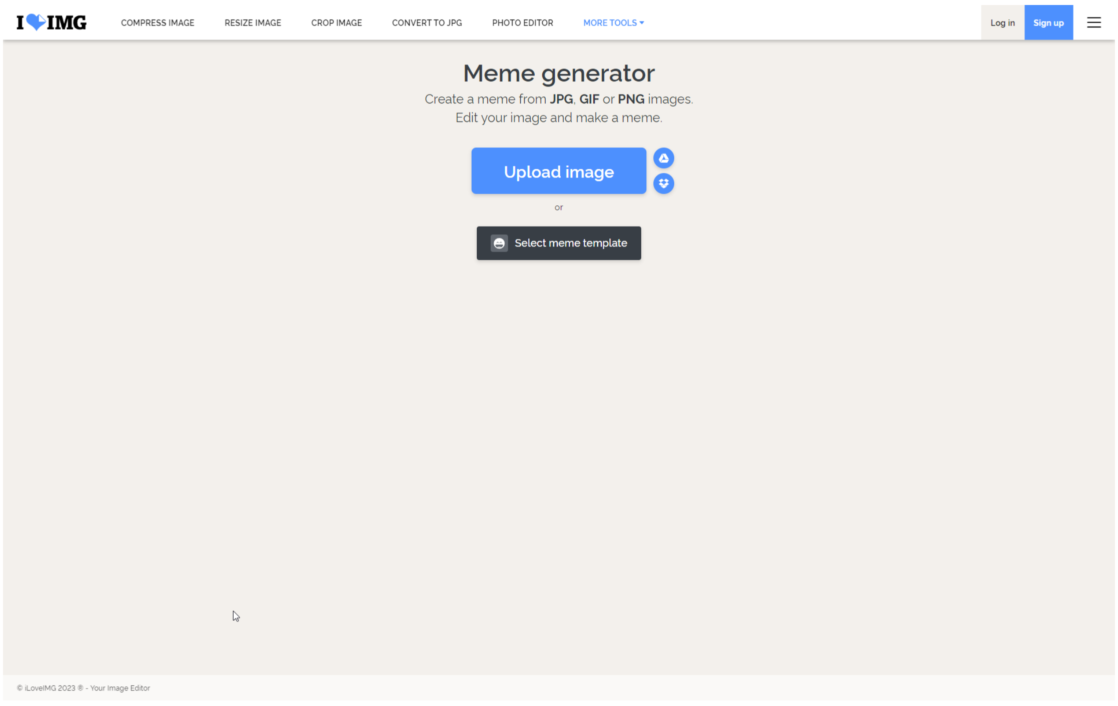
ILoveIMG Meme Generator [3] is a part of the iLoveIMG suite of free tools designed to make working with images easy [11].

1.1.3.1 Creating a meme

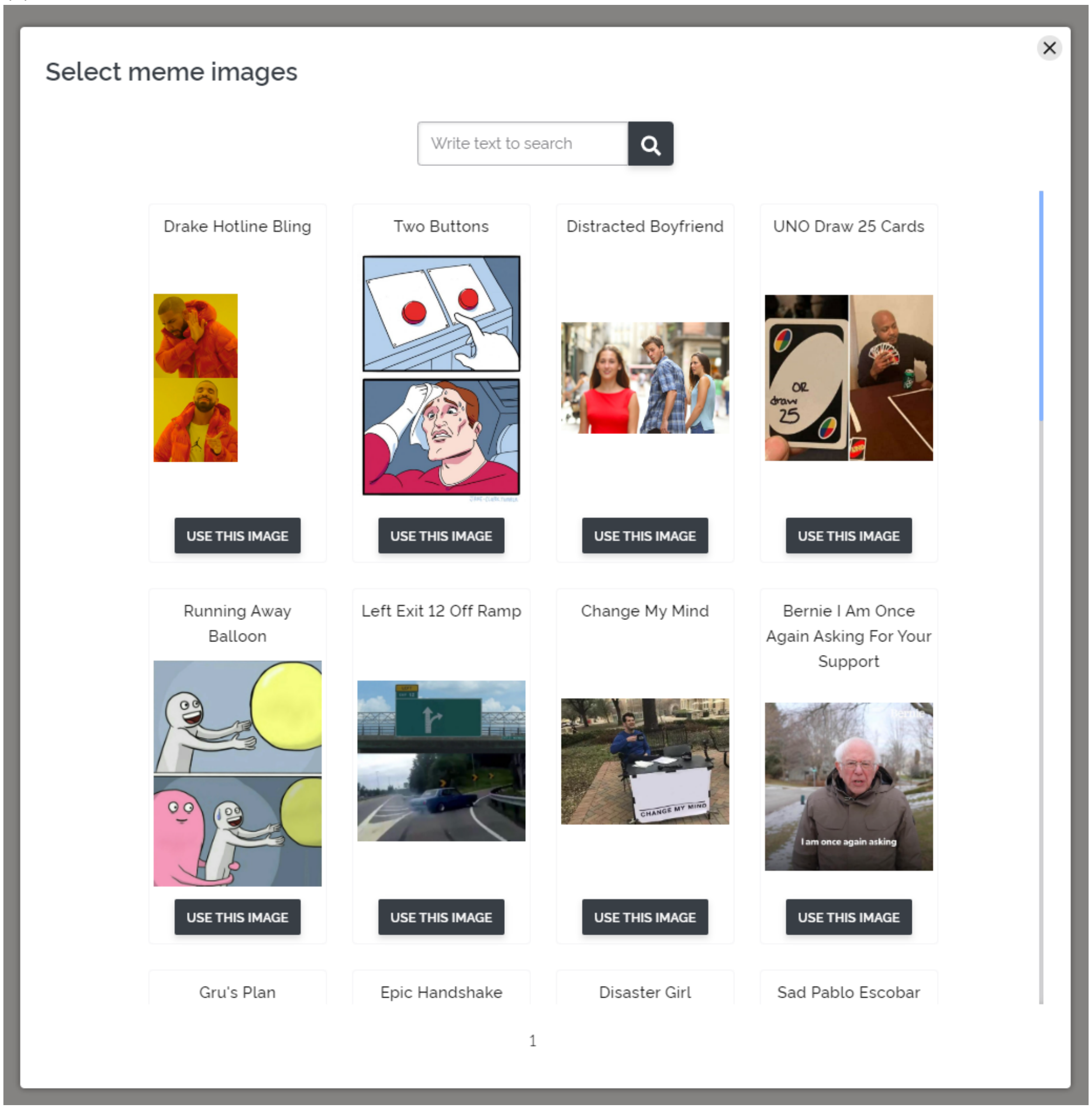
The intro screen to the generator, shown in image 1.3a, asks the user to upload a custom image, or to select from predefined meme templates. When the user clocks the `Select meme template` button, the dialog shown in image 1.3b appears. Here the user can select their desired template and use the search function to filter by name.

When the background is selected, the screen shown in image 1.3c appears. Here, the user can use the graphical editor to edit the meme. Every template comes with two default texts, one on the bottom and one on the top. The user can switch between the texts being positioned inside the image or outside the image by using the buttons on the right. Unlike Imgflip, the template does not come with texts placed in strategic places in the template, the user has to add and position the texts themselves.

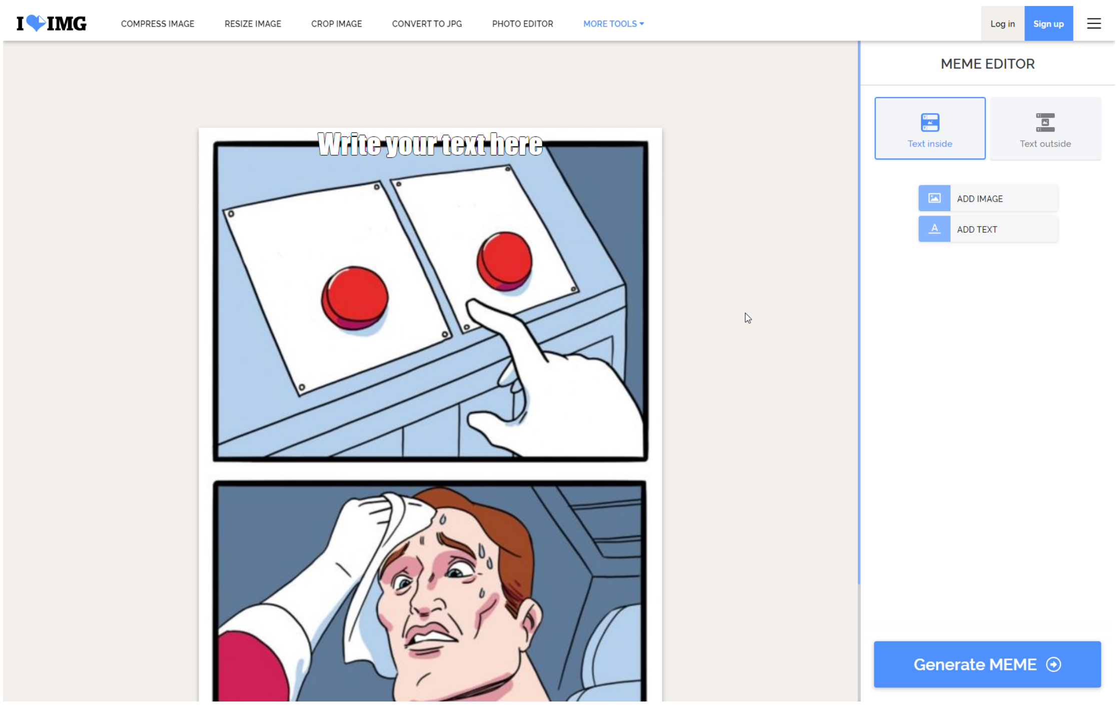
[3]https://www.iloveimg.com/meme-generator



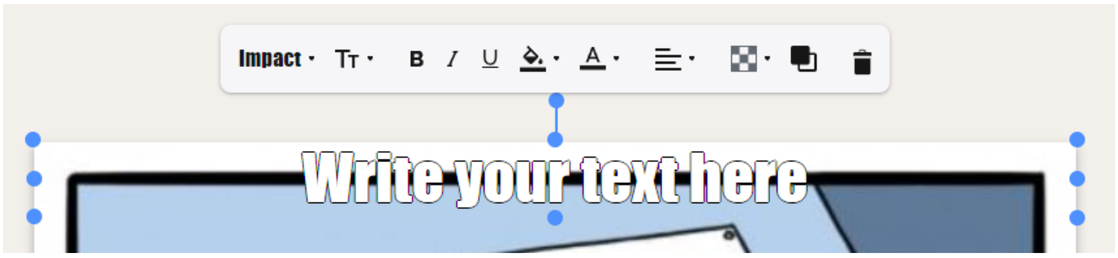
(a) Main iLoveIMG meme generator screen.



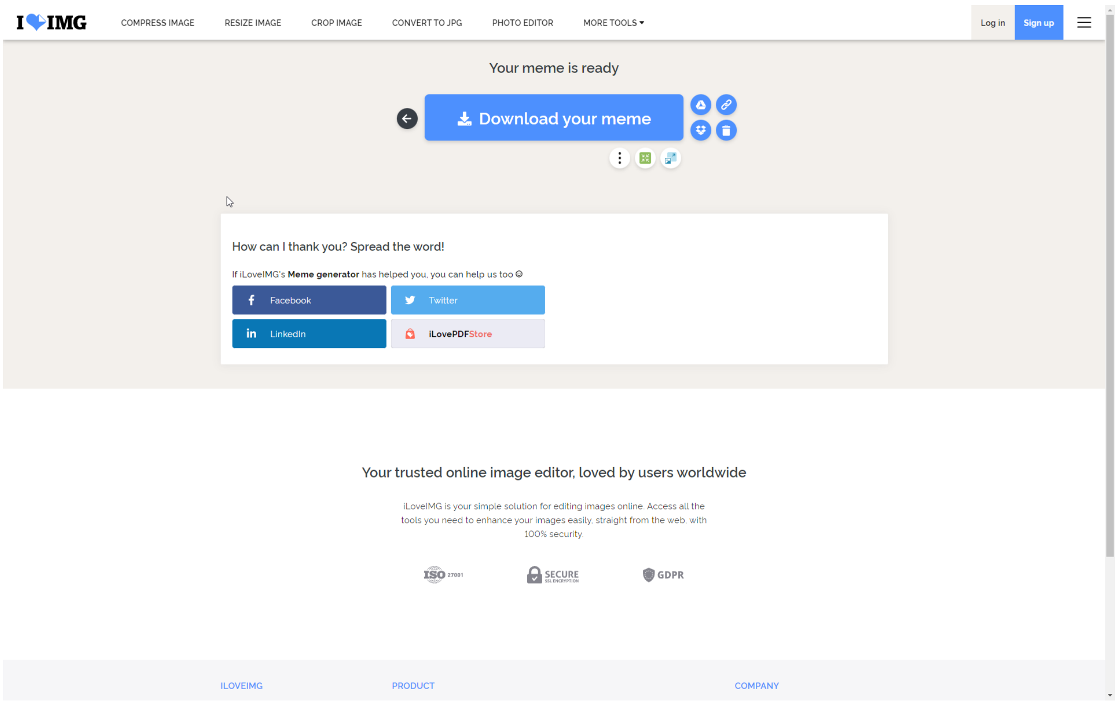
(b) Select meme dialog.



(c) Meme editor.



(d) Text editing tools



Your meme is ready

How can I thank you? Spread the word!

If iLoveIMG's **Meme generator** has helped you, you can help us too ☺

Your trusted online image editor, loved by users worldwide

iLoveIMG is your simple solution for editing images online. Access all the tools you need to enhance your images easily, straight from the web, with 100% security.

(e) Generated meme

Figure 1.3 Screenshots of iLoveIMG

Upon clicking on a text, a box with texts options appears on the top, shown on image 1.3d. This allows the user to modify other properties of the text, such as the font, text size, colors, and others. There are also buttons to duplicate the text and to delete the text.

1.1.3.2 Downloading a meme

When the user clicks the `Generate MEME` button, they are redirected to the page shown in image 1.3e. There is an option to download the image, or save it directly to supported cloud storage solutions, like Dropbox. There are also social media buttons to share the image on social networks. There are also buttons to further work with the image using other tools which are part of the iLoveIMG portfolio, such as cropping, compressing or resizing the image [11].

1.1.3.3 Subscriptions

ILoveIMG offers a subscription to unlock more features throughout all their applications, like an ad-free version of the site or customer support. The benefit of having the paid subscription for the meme generator is the ability to create images with files of up to four gigabytes, where the free version only supports files up to two hundred megabytes [12]

1.1.4 Mematic

Mematic is a free Android/iOS smartphone application for making memes, postcards, collages, and more. It offers controls adjusted for mobile phone usage.

1.1.4.1 Creating a meme

The initial screen of the application is shown in image 1.4a. It shows many options on how to start editing images, where some options are hidden behind the pro subscription (see subsubsection 1.1.4.3). The `Free Style` option is the most similar to the featured web applications.

When the `Free Style` option is selected, the template selection screen shows up as shown in image 1.4b. Here, the user can scroll through many available templates, or use the search bar to filter the template selection.

After the user selects a template, the editing screen, shown in image 1.4c, shows up. The template comes with texts positioned according to the meme template. The bottom row of actions allows the users to add texts, add images and modify the watermark. The watermark modification is not a part of the free version, but it is unlocked with the pro subscription.

When the user double-taps a text, the text editing options show up, as depicted on image 1.4d. The user can move, resize, rotate or remove the text in the graphical section. The bottom row of action also changes to offer other text editing options, like fonts, colors, or sizes.

1.1.4.2 Exporting a meme

When the user clicks the `Export` button, an export screen comes up, shown on image 1.4e. There is an option to remove the watermark, as well as to share or save the generated image. The `Share` button opens up the operating system sharing dialog.

1.1.4.3 Subscriptions

Mematic offers a paid subscription to unlock additional features. The subscription includes access to all the fonts, the possibility to customize the watermark, an ad-free version of the application and others.

1.2 Use Case Analysis

This section of the thesis provides detailed use cases of the application. A use case describes a sequence of interactions between a system and an external actor that results in the actor being able to achieve some outcome of value.[13]

1.2.1 Actors

New User is a user that has landed on the application but hasn't registered. Such a user has a very limited experience in the application.

Authenticated user is a user that has completed the registration process and thus has access to the whole application including all the features restricted for registered users.

1.2.2 Use Cases

Overview of all the use cases is shown in the figure 1.5. All the use cases are described in detail in the following table.

Authenticate

Actors	New User
Description	The New User expects to be able to authenticate into the application by providing valid credentials. There are two possible ways of authenticating, either by using an email address or by using an external identity provider.
Preconditions	The user has a valid email address, or they are registered with one of the valid external identity providers.
Normal Flow	1. Click the login button. 2. Fill out an email in the respective form field. 3. Click the `Sign in with Email` button. 4. Open the link that the user received on the email address filled out in step 2. 5. The user is authenticated in the newly opened session.
Alternate Flow	1. Click the login button. 2. Click on the identity provider that the user has an account at. 3. Go through the proccess of granting permission to the meme application from the external provider. 4. The user is redirected back to the meme application and is authenticated.
Postconditions	1. The user is authenticated. 2. User information is stored in the database.

List all templates

Actors	New User, Authenticated User
Description	The user expects to be able to list all the templates created by them and all the templates by other users which are publicly available.
Preconditions	There are any available templates created by the user or there are any publicly available templates.
Normal Flow	Visit the page that lists all templates.
Postconditions	User can browse all the templates available to them.

Use a template

Actors	New User, Authenticated User
Description	The user expects to be able to use an available template to create a meme.
Preconditions	There is at least one available template.
Normal Flow	Click on a template on the create page.
Alternate Flow	Click on a template on the list templates page.

Postconditions	User was redirected to the create page and the template is loaded into the editor.

Import an image	
Actors	New User, Authenticated User
Description	The user expects to be able to import an image either from their local device or the internet, or alternatively, utilize the integrated text-to-image AI functionality for image generation.
Preconditions	The user has a valid image file on their device, or they have a valid URL to an image, or they have a valid text to generate an image from.
Normal Flow	**1.** Click the `Add image` button. **2a.** Click the upload button and select the image file. **2b.** Paste the URL to the image and click the `Use` button. **2c.** Paste the text to generate an image from and click the `Generate` button. **3.** Click the `Next` button. **4.** Use the cropping tool to select the part of the image to use. **5.** Click the `Next` button. **6.** Click the `Remove Background` button. **7a.** Click the `Use original image` button. **7b.** Click the `Use image without background` button.
Postconditions	The image is imported and can be used in the application.

Modify a text in a template	
Actors	New User, Authenticated User
Description	The user expects the ability to change the contents and the properties of a text
Preconditions	**1.** The user is editing a template. **2.** The template has a text element in it.
Normal Flow	**1.** Click and drag the text's corners to resize it. **2.** Click and drag the control point above the text to rotate it. **3.** Click and drag the text to move it. **4.** Use the input field to change the content. **5.** Click on the colors in the text properties to change the outline and fill colors of the text. **6.** Click on the settings icon to open the text properties dialog. **7.** Change the text's size in the properties dialog.
Postconditions	The text is modified.

Modify an image in a template	
Actors	New User, Authenticated User

Description	The user expects the ability to change how the image is displayed in the template.
Preconditions	1. The user is editing a template. 2. The template has a text element in it.
Normal Flow	1. Click and drag the image's corners to resize it. 2. Click and drag the control point above the image to rotate it. 3. Click and drag the image to move it.
Postconditions	The image is modified.

Exporte a meme

Actors	New User, Authenticated User
Description	The user expects to be able to export a meme and download it.
Preconditions	The user has a template loaded into the editor.
Normal Flow	Click on the `Export Meme` button.
Postconditions	The meme is exported and a dialog with the image is shown.

Save a template

Actors	Authenticated User
Description	The user expects to be able to save an edited template.
Preconditions	The user has a template loaded into the editor.
Normal Flow	1. Click on the `Save Template` button. 2. Edit the template's name. 3. Set the template's visibility. 4. Click on the `Save` button.
Postconditions	The template is saved and can be used in the future.

Share a meme

Actors	Authenticated User
Description	The user expects to be able to get a sharable URL to the meme.
Preconditions	The user has exported a meme and the dialog with the image is shown.
Normal Flow	1. Click on the `Copy` button.
Postconditions	The URL to the meme is copied to the clipboard.

List exported memes

Actors	Authenticated User
Description	The user expects to be able to list all the memes exported by them.

Preconditions	The user has exported at least one meme and has not deleted it.
Normal Flow	Visit the page that lists users memes.
Postconditions	User can view all the memes that they have exported.
Postconditions	The template is saved and can be used in the future.

Remove a meme	
Actors	Authenticated User
Description	The user expects to be able to remove a meme.
Preconditions	The user has exported at least one meme and has not deleted it. The user is on the page that lists all the memes exported by them.
Normal Flow	1. Click on the `Delete Meme` button.
Postconditions	The meme is deleted.

Remove a template	
Actors	Authenticated User
Description	The user expects to be able to remove a template.
Preconditions	The user has saved at least one template and has not deleted it. The user is on the page that lists all the templates they have saved.
Normal Flow	1. Click on the `Delete Template` button.
Postconditions	The template is deleted.

1.3 Wireframes

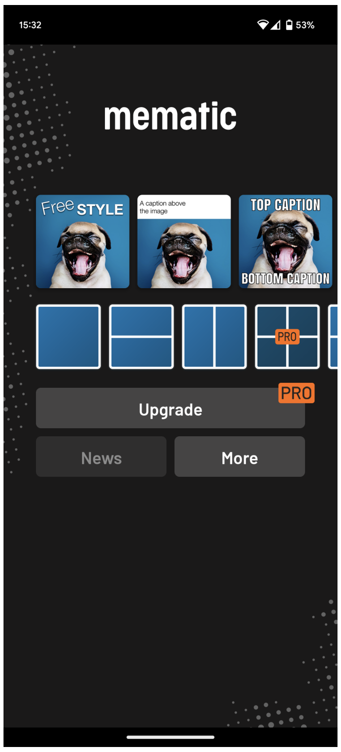
A wireframe is the basic blueprint that illustrates the core form and function found on a single screen of your web page or application [14]. The wireframes do not exactly resemble the finished product, but they provide an insight into the basic functionalities and structure of the application. All of the individual pages of the application are featured in the figure 1.6 and the import image dialog is shown in the figure 1.7.

1.3.1 Create Page

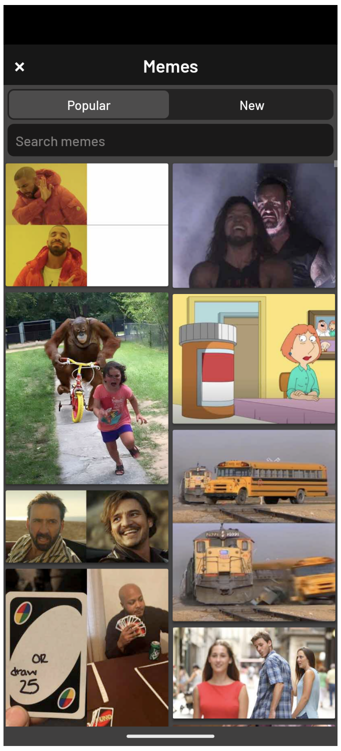
The `Create Page` is the main page of the application where the user lands at. The wireframes show it in two versions, one where a user has logged in to the application (image 1.6b) and one where the user is anonymous (image 1.6a).

The main part of the page is mostly the same for both types of users. The top part of the page is used to get a base template, which can either be selected from the list of available templates or imported from an image.

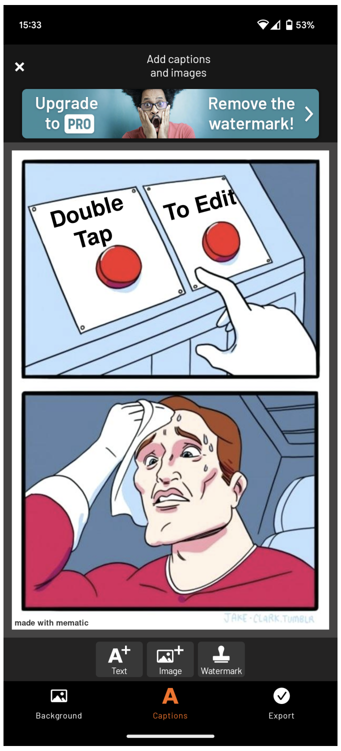
To the left is the graphical section. It features a canvas where the user can see the template, the texts and the added images. The user can move the texts and images around, rotate them



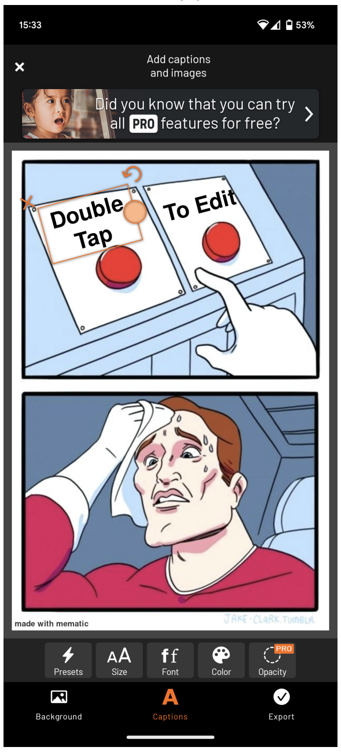
(a) Initial screen of the Mematic application



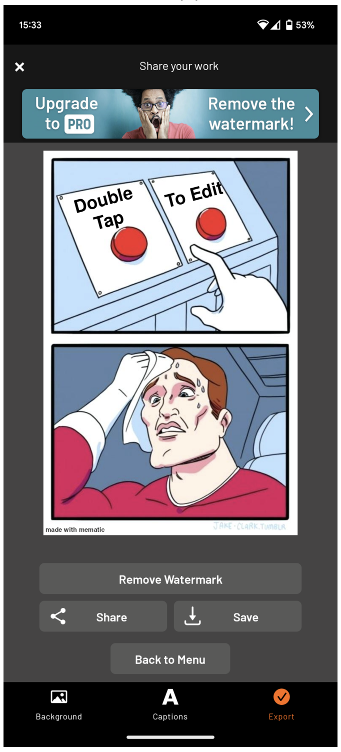
(b) The template selection screen.



(c) The meme edit screen.



(d) The text editing controls.



(e) Exporting a meme.

Figure 1.4 Screenshots of Mematic

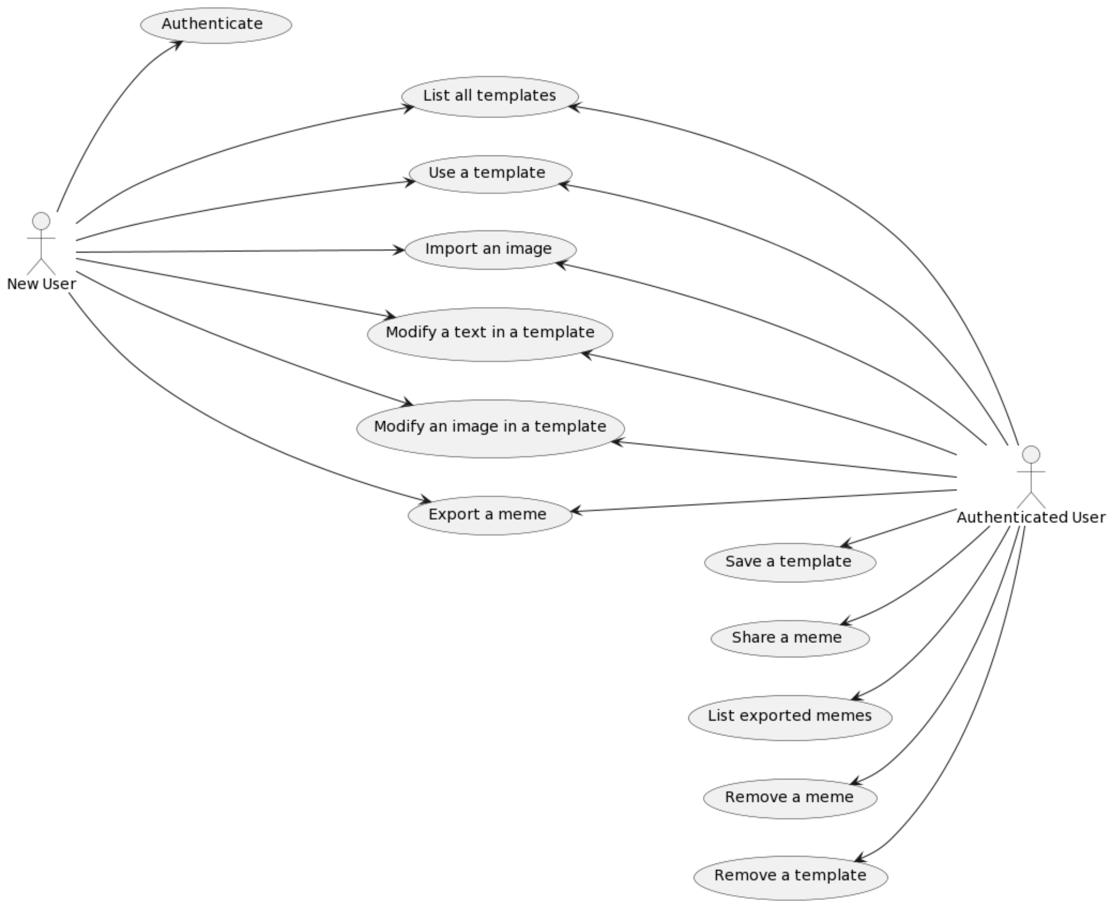


Figure 1.5 Use Case Diagram.

and resize them. The user can also add new texts and images to the canvas, where the images would be imported using the import image dialog (see subsection 1.3.6).

To the right is the properties section of the page. It displays a list of all the texts and images that are currently on the canvas. The user can use this section to modify the content of each individual text and it's the outline and fill colors. The order in which the texts and images are displayed in the properties section is the same as the order in which they are displayed in the graphical section. Texts and images can be moved up and down in the list using the arrows on the left side of each item. The user can also remove texts and images from the canvas using the `Remove` button on the right side of each item.

The bottom of the right section is dedicated to saving the template and exporting a meme image. Only logged-in users can save the template, but both types of users can export a meme image.

1.3.2 All Templates Page

The `All Templates` page shows all the available templates that the user can modify and use them to generate memes. If the user is logged in, the page will show all the user's templates and all the public templates from other users. If the user is not logged in, the page only shows public templates.

The `Load More` button is only visible when there are more templates to be loaded. Clicking it will load more templates.

Clicking any of the displayed templates redirects the user to the `Create page` and loads the clicked template into the editor.

1.3.3 My Templates Page

The `My Templates Page` is only available to logged-in users. Here, the users can view all the templates they created and saved. There are two options for each template, opening it in the `Create Page`, or deleting it.

1.3.4 My Memes Page

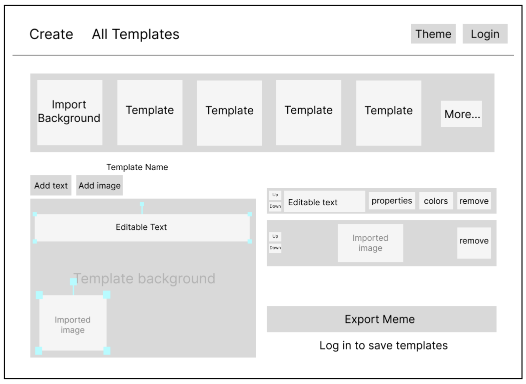
The `My Memes Page` has a similar design to the `My Templates Page`, but instead of a list of templates, it shows a list of memes. There is an option to open the meme's details, which would show a larger version of the image and the link to the meme's image.

1.3.5 Profile Page

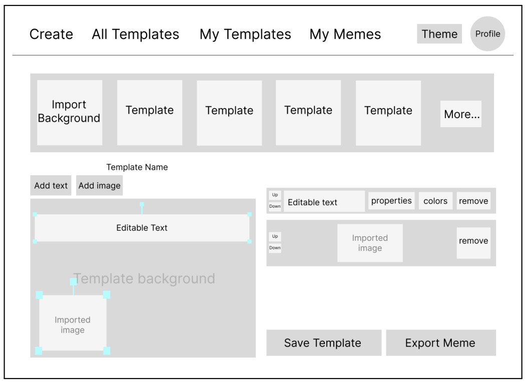
`Profile Page` shows the user's statistics and information. There is a section with the user's information, like their name, email, and profile image. There is also a section with the user's statistics, like the number of templates and memes they created. The user is able to change their profile image by clicking on icon in the top left corner the image and selecting a new one. They can also change their username using the form below the statistics.

1.3.6 Import Image Dialog

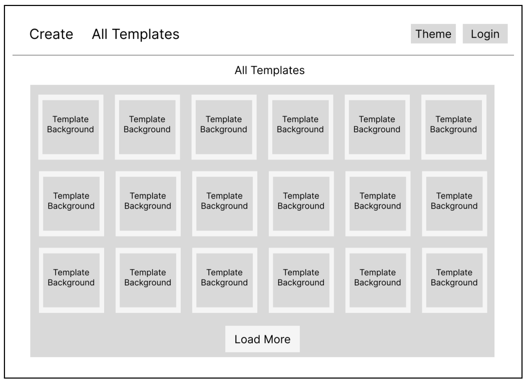
The `Import Image Dialog` is used to import an image to the application, which is used as a template background or is a part of the template. The dialog has three steps.



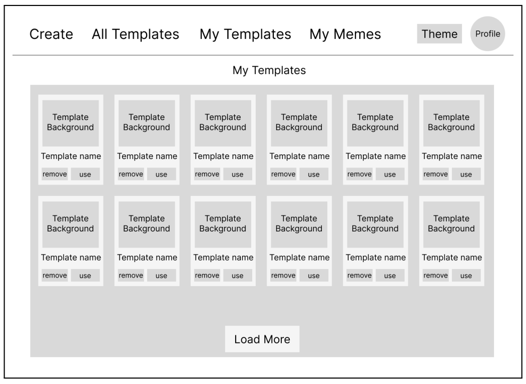
(a) Create page, anonymous user.



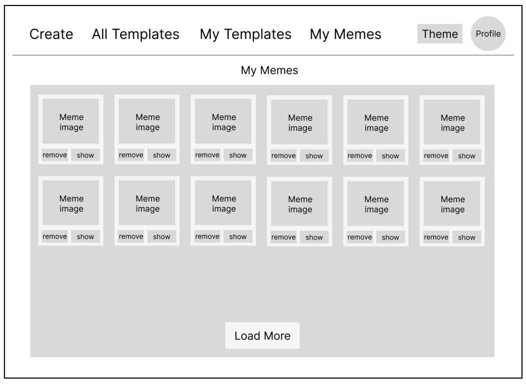
(b) Create page, logged-in user.



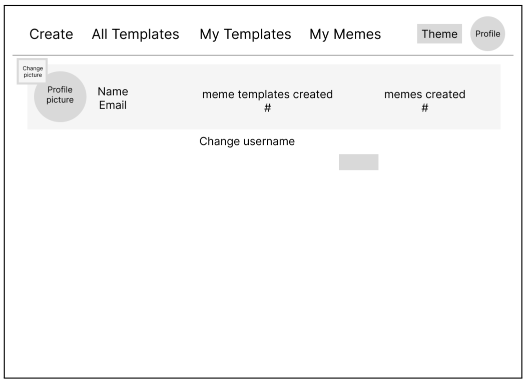
(c) All templates page.



(d) My templates page.



(e) My memes page.



(f) Profile page.

Figure 1.6 Wireframes of the application.

Step 1 is used to select the image to import. It is shown on image 1.7a. The user can either select an image from their device, paste an URL to an image or paste a text to generate an image from.

Step 2 is used to crop the image. It is shown on image 1.7b. The user moves the cropping rectangle to select the part of the image to use.

Step 3 is used to remove the background from the image. If the user doesn't want to remove the background, they can skip this step using the `Confirm` buttons, which is shown on image 1.7c.

When the `Remove background` button is clicked, the application will remove the background from the image and show both the original image and the result, as shown on image 1.7d. The user can then either use the image without the background or use the original image.

1.4 Database Schema

The database schema is shown in the figure 1.8. The schema is designed to be as simple as possible, while still being able to support all the features of the application. The schema is designed to be easily extensible, so that new features can be added without having to change the schema.

The `Image` model acts as a foundational entity, encapsulating the core attributes of an image file within the system. The database itself doesn't store the image data directly, but rather a reference to the image file stored in the image storage. This allows for efficient storage and retrieval of images, as well as the ability to scale the image storage independently of the database.

The `TemplateText` and `TemplateImage` models define the components of a meme template, allowing dynamic placement and styling of text and images. They include attributes for positioning (`x`, `y`), dimension (`width`, `height`), styling (`fontSize`, `color`, `strokeColor`), and other visual properties (`scale`, `rotation`, `zIndex`). These models are crucial for creating templates that can be customized and reused, with `TemplateText` managing the textual elements and `TemplateImage` managing the imagery within templates.

The `Template` model serves as an aggregator, linking multiple `TemplateText` and `TemplateImage` objects to form a complete meme template. It includes a unique identifier, a name for the template, a visibility flag (`isPublic`), and relationships to the `Image` and `User` models, indicating the creator and the background image of the template.

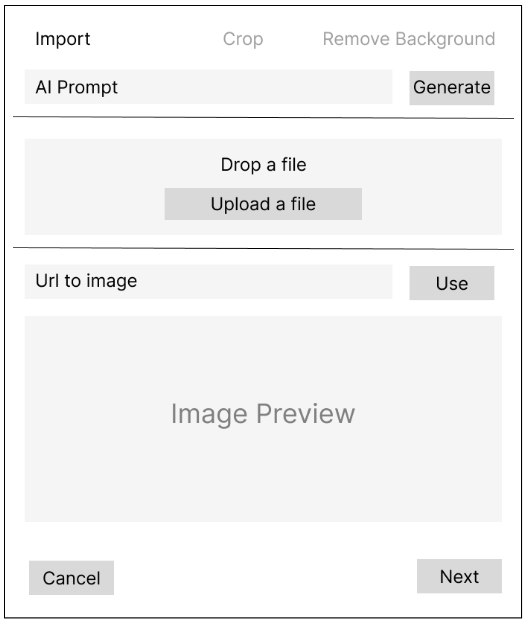
The `Meme` model represents the final product, a personalized meme created by a user. It is linked to the User and Image models, connecting a meme to its creator and the image it utilizes.

The `Account` model is designed to manage user accounts, including authentication and authorization information. It supports integration with external authentication providers and maintains access and refresh tokens for user sessions. The unique constraint on the combination of `provider` and `providerAccountId` ensures that each external account is uniquely linked to a user account within the system.

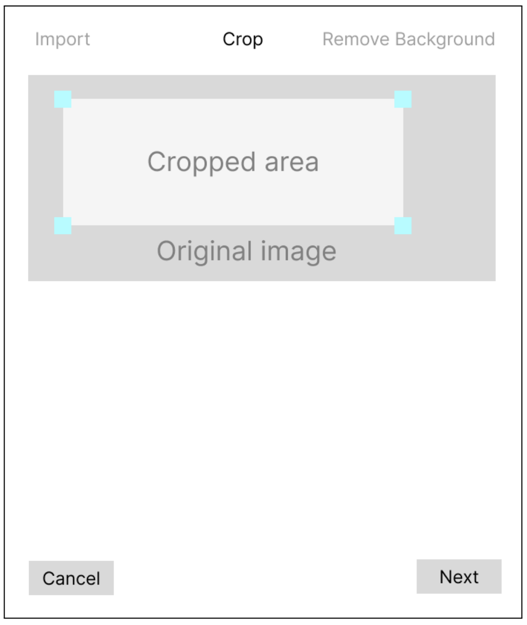
The `Session` model handles user sessions, with a unique session token and an expiration date. This model is essential for tracking active user sessions and managing session expiration.

The `User` model represents individuals interacting with the system. It includes personal information such as name and email, with the email field being unique to prevent duplicate accounts. The User model has one-to-many relationships with `Account`, `Session`, `Template`, and `Meme`, indicating the various entities a user can have ownership of or association with.

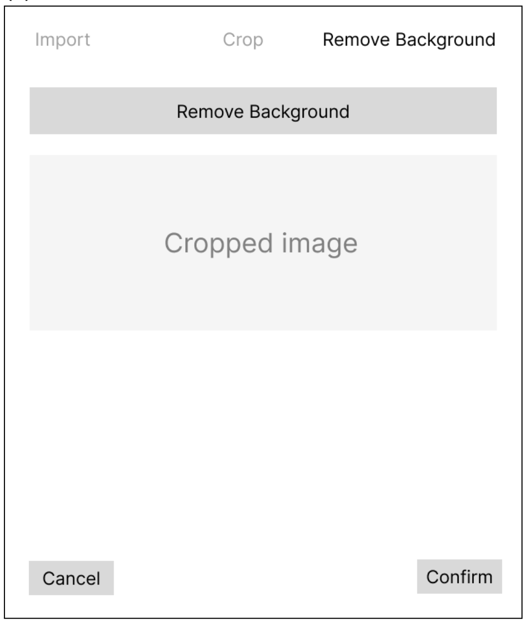
Finally, the `VerificationToken` model is employed for account verification process when using email authentication. It includes a unique token and an expiration date, ensuring the token is used within a certain timeframe.



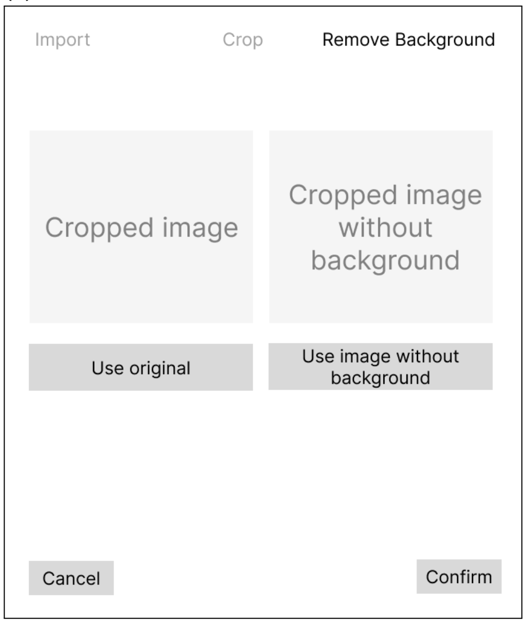
(a) Step 2, import.



(b) Step 2, crop.



(c) Step 3, remove background, before `Remove Background` button is clicked.



(d) Step 3, remove background, after `Remove Background` button is clicked.

Figure 1.7 Wireframes of the import image dialog.

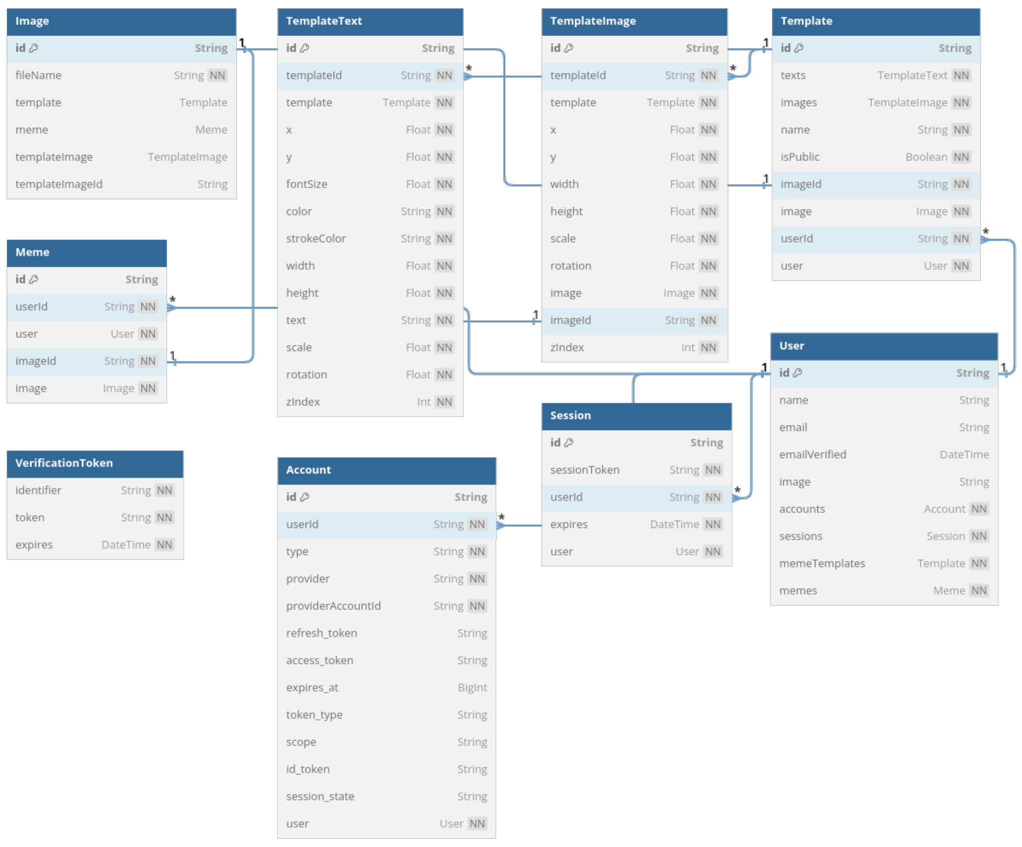


Figure 1.8 Database schema.

Each model is designed with a focus on modularity and extensibility, allowing for scalability as the system grows in complexity and user base. The relationships between models, particularly the one-to-many and many-to-one relationships, are carefully crafted to reflect the real-world interactions between the different entities. For instance, a User can create many Templates and Memes, but each Template or Meme is associated with exactly one User.

1.5 High-level architecture

The architecture of the application is modular, consisting of five distinct but intercommunicating components.

1.5.1 Pages

The `Pages` module is primarily responsible for delivering the statically generated and server side generated frontend to the user's browser. This segment of the application encompasses all the static elements, as well as the necessary scripting to enable dynamic capabilities. The frontend interacts with the API to facilitate access to an array of resources such as the database, image repository, and authentication mechanisms.

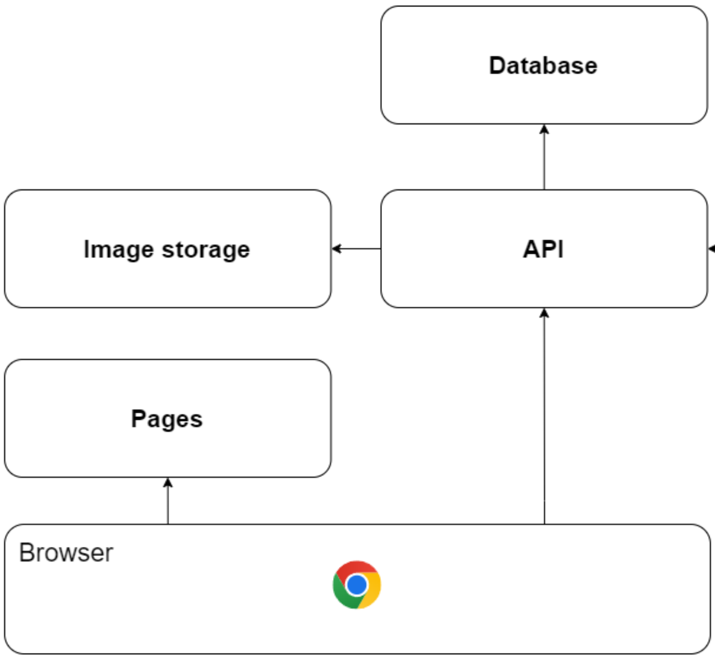


Figure 1.9 High-level architecture overview.

1.5.2 API

The API mediates all interactions between the client-side browser and the other constituent components. It offers a suite of procedures that the frontend can invoke, thereby enabling the retrieval and submission of information as per user requests.

1.5.3 Database

The `Database` is a cornerstone component that holds all data related to the application, barring images. Its design ensures efficient storage, retrieval, and management of data, thereby supporting the application's operational integrity.

1.5.4 Image Storage

Dedicated to image management, the `Image Storage` component is tasked with housing all visual assets utilized by the application. This includes a variety of images such as meme content and images relating to templates, essential for the application's visual functionalities.

1.5.5 Authentication

The Authentication module is critical for maintaining the application's security. It interfaces with various authentication providers and manages email-based authentication processes. Additionally, it leverages the database for storing essential information related to user identities, accounts, and session data, thereby ensuring a secure and personalized user experience.

Chapter 2

Development

This chapter describes the developmental framework of the application, providing a comprehensive overview of the various libraries implemented and the rationale behind their selection. It explains the synergistic integration of the application's components, detailing how each part contributes to the cohesive functioning of the whole system.

2.1 Technologies

This section has provided an overview of the framework selection process and the integration of various complementary technologies for the development of the application.

2.1.1 Framework

The initial step in determining the appropriate framework[15] for the application involved an analysis of the project requirements. This analysis encompassed understanding the application's functionality, and user experience goals and is described in detail in the previous chapter. Other requirements included my expertise, documentation, community support, and maturity of the framework.

The following frameworks were considered based on their popularity, measured by Stack Overflow Survey[16], and relevance to the application's domain:

1. Next.js
2. Nuxt.js
3. Remix

Based on all factors mentioned above and internet research[17], mainly my extensive experience, Next.js was chosen as the framework for the application. Next.js is a React framework that provides a variety of features that are essential for the application, such as server-side rendering, static site generation, and routing. It also provides a variety of other features that are not essential for the application, but are useful, such as image optimization, internationalization, and many others.[18]

2.1.1.1 Authentication

`NextAuth` was selected as the authentication solution for its seamless integration with Next.js, the primary framework used for the application. It offers a straightforward and secure approach to handling user authentication and session management.[19] The simplicity of setting up providers for OAuth, Email, and other authentication mechanisms with `NextAuth` significantly accelerated the development process, while ensuring robust security and compliance with industry standards.

2.1.1.2 State Management

For state management within the application, `Zustand` was chosen for its minimalistic and straightforward API. Its non-boilerplate approach and ease of integration with React components made it an ideal choice for managing the application's state.[20] Zustand's simplicity and efficiency in handling state across components helped in maintaining a clean and manageable codebase, improving both development speed and application performance.

2.1.1.3 API Library

`TRPC` was implemented to enable end-to-end type-safe API calls, enhancing the robustness and reliability of the application. It provided a simplified way of creating APIs without the need to define types and validation schemas multiple times, thereby reducing redundancy and potential for errors. This integration of `tRPC` brought a significant improvement in development efficiency and contributed to a more maintainable and error-resistant codebase.

2.1.1.4 Database Management

`Prisma` was incorporated as the database management tool, owing to its powerful and type-safe ORM capabilities. It offered an intuitive and straightforward way to interact with the database, ensuring a strong alignment with the application's data modeling and business logic requirements.[21] Prisma's emphasis on type safety and its ease of use in defining and evolving the database schema played a pivotal role in enhancing the application's data integrity and overall backend efficiency.

2.1.1.5 Database Backend

SQLite was chosen as the database backend for its distinctive advantages that perfectly align with the needs of the project. A primary benefit of SQLite is its embedded nature within the application, eliminating the need for managing a separate database service. This integration simplifies the overall setup, which is crucial for developer velocity.

Starting up a new database with SQLite is remarkably uncomplicated compared to other databases like Postgres. There's no intricate setup process; it's essentially just a file. This simplicity extends to production migrations and seed operations, which can be executed with minimal effort. The application heavily utilizes these features, with the database being created and seeded during the build process. This approach ensures that the application is ready to use immediately after deployment, with no additional setup required.

Furthermore, this ease of use significantly benefits the testing phase of development. Complex database setups often lead to time-consuming evaluations of how to mock databases at the ORM level to avoid the challenges of running and connecting to a database during testing. SQLite, being just a file, simplifies this process substantially. Each test can have its own isolated database environment with minimal setup, allowing for more efficient and accurate testing scenarios. This level of ease and efficiency in setting up isolated test environments is harder to achieve with other database systems.[22]

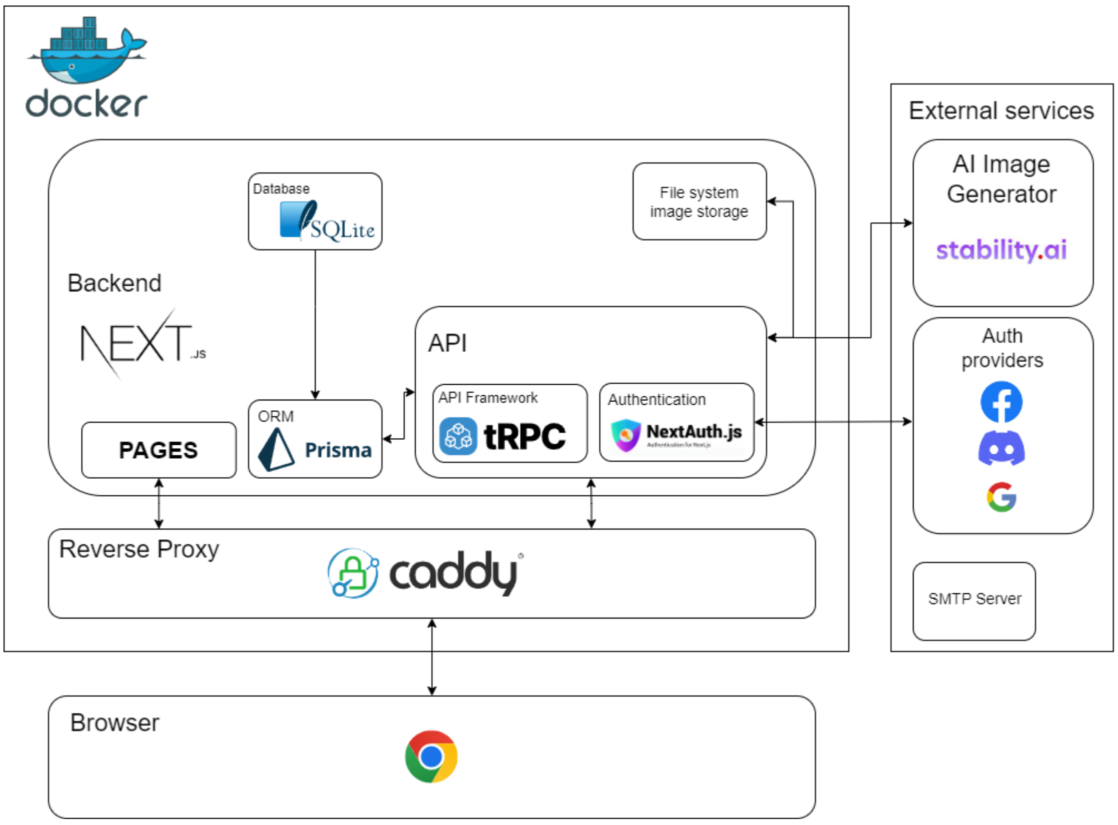


Figure 2.1 Application diagram.

2.2 Application design

Figure 2.1 presents the high-level architecture of the application, showcasing the interaction between its components as detailed in the preceding section. The system utilizes Docker for containerization, streamlining deployment, which is further elaborated in Section 2.9.

At its core, the application consists of statically generated pages and an API for client-server interaction. Statically generated pages enhance performance by reducing server load, while the API ensures efficient data transfer and processing. Prisma, serving as the Object-relational mapping (ORM), abstracts database interactions, simplifying database operations.[23]

Authentication is streamlined via three external providers, optimizing the sign-in and registration workflows. There is also an email option, which utilizes an external Simple Mail Transfer Protocol (SMTP) server.

The application offers its users the ability to generate images using text-to-image Artificial Intelligence (AI) models. This functionality is facilitated by an external API that is integrated into the application.

2.3 Configuration

The application is configurable through the use of environment variables. The variables can either be set as environment variables on the host system, or they can be filled into the .env file in the directory from which the application is launched. The environment variables are validated on startup, and the application will not start if any of the required variables are missing. The environment variable schema is configured in the `src/env.mjs` file.

This section describes all the configuration options and the environment variables that are used to configure them.

2.3.1 Data Storage

The database configuration within the application is specified by the `DATABASE_URL` environment variable, which is a standard practice for defining connection strings. In the context of this application's architecture, the flexibility to switch between different database providers is restricted by the limitations of the Prisma framework.[24] As such, SQLite is the exclusive database provider utilized. The format of the URL for connecting to the SQLite database is explicitly structured as `file:file-location`, which indicates the file path where the SQLite database is located on the filesystem. By using this approach, the application benefits from SQLite's simplicity and ease of use, while also allowing for the database to be easily referenced and managed within the application's codebase.

The configuration for image storage within the application is governed by the `IMAGE_UPLOAD_DIR` environment variable. This variable is responsible for specifying the directory path where all images are to be stored. It is crucial to ensure that the designated directory is already established on the server's filesystem and that the necessary permissions are set so that the application has the requisite write access. The proper configuration of this environment variable is essential for the seamless operation of the image upload and storage features of the application.

2.3.2 Authentication

`NextAuth`, a critical component for handling authentication in the application, necessitates the configuration of essential environment variables as outlined in its documentation[25]. The `NEXTAUTH_URL` variable is pivotal, required to be set to the canonical URL of the deployed website, ensuring that OAuth callbacks are correctly resolved. Furthermore, for the sake of security, the NEXTAUTH_SECRET is used as an encryption key for session tokens and for hashing email verification tokens, enhancing the integrity and confidentiality of the authentication process.

In an effort to facilitate a flexible and dynamic authentication system, the default NextAuth configuration has been augmented. This customization permits the activation of different authentication providers through environment variable configurations, adhering to the principle of infrastructure as code. Presently, the system supports up to three distinct OAuth providers and also an email sign in capability.

The built-in OAuth providers supported by NextAuth are extensive[26], and this application harnesses the capability to integrate three popular services: Facebook, Google, and Discord. Each service requires the setting of two specific environment variables: [PROVIDER]_CLIENT_ID and [PROVIDER]_CLIENT_SECRET. These variables function as the application's credentials for interacting with the respective OAuth service, and their values are crucial for the initiation and secure maintenance of OAuth flows. The process of setting up each provider is detailed in the NextAuth documentation[26]. The application will enable any of these providers if both of their respective environment variables are set.

The configuration of the email provider is determined by two essential environment variables. The EMAIL_SERVER variable is designated to define the SMTP server details that NextAuth will utilize to send out the authentication emails. The expected format for the EMAIL_SERVER variable aligns with the specifications provided in the Nodemailer library's documentation[27], ensuring compatibility and ease of setup.

Additionally, the EMAIL_FROM variable defines the sender's email address for all outgoing messages. This address represents the application in the users' inboxes and should be chosen to reflect the application's identity and to avoid being filtered as spam. Email sign in is enabled if both of these variables are set.

2.3.3 AI Image Generation

The application utilizes an external API for generating images from text. The API is provided by the stability.ai platform.[1] The stability.ai platform is a paid service, and it requires a registered account to be able to take advantege of its API. The STABILITY_AI_API_KEY environment variable is used to specify the API key, which is used to authenticate the application with the stability.ai platform.

2.4 API

This section dives deeper into the implementation of all the application API endpoints and procedures. Most of the API functionalities are implemented using the tRPC library, except for the endpoints which send images between the frontend and backend. TRPC only supports sending payloads which are serializable to JavaScript Object Notation (JSON)[28] which is a limitation that makes tRPC not a suitable tool for this purpose. Images could be serailzed using base64 encoding, but that is generally considered to be a bad practice because of the performance and size limitations[29].

2.4.1 Image upload

This endpoint handles POST operations to the /api/image-upload route, processes the file upload using multer, and responds with the path of the uploaded image. The API endpoint configures Next.js to handle multipart/form-data requests, which are essential for file uploads, by disabling the default JSON body parsing. This is a crucial aspect because image uploads involve binary

[1] https://platform.stability.ai

data that doesn't fit into the standard JSON format. By customizing the `bodyParser` configuration, the API ensures that the raw data from multipart forms can be processed correctly, catering specifically to the needs of file upload operations.

A notable feature of this API is the implementation of a unique filename generation mechanism for each uploaded image. This is achieved using the `uuid` library to generate an Universally Unique Identifier (UUID) for each file, ensuring that each uploaded image has a distinct filename, thereby eliminating conflicts and overwrites in the server's storage. Additionally, the API leverages the `mime-types` library to accurately determine and append the appropriate file extension based on the MIME type of the uploaded file. This approach contributes to better organization and retrieval of uploaded files.

The API employs `multer`, a popular middleware for handling `multipart/form-data` HTTP requests, to manage the incoming file data. The configuration of `multer` is tailored to store files in a designated directory defined by the configuration environment variables, with filenames determined by the aforementioned unique naming strategy. The final part of the API is an express route set up using `next-connect` - a minimalist framework for routing and middleware.

2.4.2 Generate AI Image

This endpoint handles `POST` operations to the `/api/generate-ai-image` route and facilitates the interaction with an external AI image generation service, specifically Stability AI's platform. The function is structured to respond to HTTP requests by generating images based on text prompts provided by the user, showcasing integration with an advanced AI-powered service in a web application.

The handler function first checks for the presence of a 'prompt' in the query parameters of the request. If the prompt is missing, it responds with a 400 status code and an error message, ensuring that the necessary input for generating an image is provided. This is a crucial step to validate the incoming request and maintain the integrity of the API's functionality.

Once a valid prompt is received, the handler makes a `POST` request to the Stability AI's API endpoint. It constructs the request with appropriate headers, including the necessary API key obtained from the environment variables. The body of the request includes the text prompt, along with specifications for the generated image such as its height, width, number of samples, and steps for the generation process. These parameters are essential for tailoring the image generation to the specific requirements of the application.

The response from the Stability AI API is expected to contain the generated artifacts in the form of a base64-encoded string representing the generated image. The handler function checks whether it received the artifacts and forwards them to the client. However, if the call to the API didn't succeed, a 500 status code with an appropriate error message is returned.

2.4.3 Remove background

This endpoint handles `POST` operations to the `/api/remove-background` route and facilitates the interaction. It employs the same techniques for image upload and manipulation as the `image-upload` endpoint described in Section 2.4.1 with some key differences. The `multer` configuration for this endpoint does not specify an upload directory, but stores the received images into memory instead.

The handler utilizes the `sharp` and `rembg` libraries to process the uploaded image and remove its background. First, the handler has to check whether an image file has been uploaded. If no file is found, a 400 status response is sent with a relevant message. Otherwise, the code processes the uploaded image using `sharp` and `rembg`, removing the background. If successful, the processed image data is returned as a response with a 200 status. In the case of errors during image processing, a 500 status is returned with error details.

2.4.4 tRPC

TRPC has been instrumental in optimizing interactions between the frontend and the database. By facilitating direct, procedure-based communication between the client and server, it eliminates the complexities and overheads associated with traditional REST or GraphQL APIs. This streamlined approach, enhanced by TypeScript's strong typing, ensures consistent data structures across both ends, significantly reducing runtime errors and improving data integrity, especially in complex database operations. The integration of TRPC thus not only accelerates development cycles but also fortifies the application's reliability.[30] The section is structured into router subsections, each of which describes the procedures implemented within the router.

2.4.4.1 Profile Router

The profile router encapsulates all the procedures related to user profile management. It contains two primary mutations: `updateUserName` and `updateProfileImage`. Both mutations are secured with `protectedProcedure`, indicating that they require user authentication. Each mutation accepts a single string input; username and profile image URL. The core functionality involves updating the user's name or profile image in the database using `Prisma`, with the current user's ID obtained from the session context supplied by `NextAuth`.

2.4.5 Template Router

The template router handles all the interactions related to meme templates. Its procedures are secured with `protectedProcedure`, ensuring that only authenticated users can access them

	createTemplate
Description	This procedure enables authenticated users to create new templates.
Parameters	Input object with properties: 'name' (string), 'isPublic' (boolean), 'fileName' (string), 'texts' (array of objects validated by the database schema), 'images' (array of objects validated by the database schema).
Operations	Authenticates the user, creates a new image and template in the database, and adds associated texts and images. Each text and image is validated and processed individually, ensuring data integrity and consistency.
Return value	Returns an object including the newly created template details, along with the texts and images associated with it.

	updateTemplate
Description	This procedure is designed for updating existing templates.
Parameters	Input object with properties: 'template' (object with template details, excluding sensitive fields like 'imageId' and 'userId'), 'texts' (array of objects validated by the database), 'images' (array of objects validated by the database schema).
Operations	Performs ownership verification of the template, deletes existing texts and images related to the template, then updates the template with the new data.

Return value	The response includes an updated template object, featuring the new sets of texts and images, indicative of a complete and secure update operation.

deleteTemplate	
Description	This procedure allows users to delete their templates.
Parameters	Input object with a single field 'id' (string), specifying the template to be deleted.
Operations	Checks if the user is authorized to delete the specified template. Upon successful verification, proceeds to delete the template from the database, ensuring data consistency and security.
Return value	Doesn't return any value, but the procedure throws an error if something unexpected happens or if the user is not authorized to delete the template.

getTemplate	
Description	The procedure is a public procedure for retrieving specific templates.
Parameters	A single input 'id' (string) to identify the template to be fetched.
Operations	Includes conditional logic to ensure that private templates are accessible only by their owners. Retrieves a specific template from the database, along with its associated images and texts.
Return value	The detailed representation of the template, including associated images and texts.

getInfiniteTemplates	
Description	This procedure is designed to fetch templates in a paginated manner.
Parameters	Input object with properties: 'fetch' (enum: fromUser, public, all), 'limit' (number, min: 1, max: 100), 'cursor' (string, nullish), 'searchString' (string, optional).
Operations	Conducts a paginated query based on the input parameters, retrieving templates from the database. The operation involves filtering templates, providing a mechanism for fetching data in a manageable and scalable way.
Return value	An object containing the fetched items and a 'nextCursor' for pagination, enabling the frontend to fetch the next page of items.

2.4.5.1 Meme Router

createMeme	
Description	This procedure enables authenticated users to create new memes.

Parameters	Input object with property: ’fileName’ (string).
Operations	Authenticates the user, creates an image entry in the database using the provided file name, then creates a meme associated with the image and the current user.
Return value	Returns the newly created meme object, including the meme details and the path to the associated image.

deleteMeme	
Description	The procedure allows users to delete their memes.
Parameters	Input object with property: ’id’ (string) specifying the meme to be deleted.
Operations	Verifies the ownership of the meme, then proceeds to delete both the meme and its associated image from the database.
Return value	Doesn’t return any value, but the procedure throws an error if something unexpected happens or if the user is not authorized to delete the template.

getInfiniteMemes	
Description	This procedure is designed for fetching memes using pagination to limit the number of memes returned in each query.
Parameters	Input object with properties: ’limit’ (number, min: 1, max: 100), ’cursor’ (string, nullish)
Operations	Retrieves memes created by the current user in a paginated manner, based on the provided limit and cursor for pagination.
Return value	An object containing the fetched memes and a ’nextCursor’ for pagination.

2.5 State Management

State management is a critical aspect of building robust and maintainable React applications. It allows developers to handle data and application state in a predictable and efficient manner. State management solutions, like Zustand, simplify the process of managing state in React applications.[31] This section explains in details how Zustand is used and how its features are leveraged to efficiently manage application data.

The module contains a single store with all the application data and functions for manipulating the data. This store is shared across the while application and every page/component has access to its data and function The individual pieces that make up the store are outlined in this section. Each piece has a TypeScript type definition which illustrates the structure of the data and the functions.

2.5.1 Popups

```
export type PopupType = {
    id: number
    type: 'info' | 'warning' | 'success' | 'error'
    message: string
}

type PopupState = {
    popupCounter: number
    popups: PopupType[]
    addPopup: (popup: Omit<PopupType, 'id'>) => void
    removePopup: (id: number) => void
}
```

This part of the state is used to display pop-up notifications. On the screen.

The `PopupType` interface defines the structure of pop-up notifications and ensures consistency in how pop-up notifications are defined and rendered.

The `PopupState` type defines the state for the store. It includes a `popupCounter` to keep track of the unique IDs for pop-ups, an array to store the pop-up notifications, an `addPopup` function to add new pop-ups to the state, and a `removePopup` function to remove pop-ups by their unique IDs.

The `addPopup` function takes a popup object as a parameter, omits the id field (which is automatically generated), and pushes the new pop-up into the popups array. Additionally, it sets a timeout to remove the pop-up after a specified duration defined by a constant `POPUP_TIMEOUT`, ensuring that pop-ups disappear automatically. The unique ID assigned to each pop-up is essential to identify and remove them later.

The `removePopup` function is responsible for removing pop-ups by their unique ID.

2.5.2 Dialogs

```
type DialogType = 'addImage' | 'save' | 'exportMeme'

type DialogState = {
    dialogs: {
        [key in DialogType]: boolean
    }
    setDialog: (dialog: DialogType, open: boolean) => void
}
```

This section of the state allows for easy control of different dialogs' open or closed states.

The `DialogState` type contains a `dialogs` object, where each key corresponds to a `DialogType`, and the associated value indicates whether the respective dialog is open or closed. This structure ensures that the state of each dialog can be tracked individually.

The `setDialog` function is designed to update the state of a specific dialog, effectively toggling the visibility of the specified dialog. It takes two parameters: `dialog` to specify which dialog to update and `open` to indicate whether to open or close the dialog.

2.5.3 Import Image

```
export type ImageImportStep = 'import' | 'crop' | 'remove-bg'

type ImageImportState = {
    isImportingBackground: boolean
    importCurrentStep: ImageImportStep
    importedImage: File | null
    importNextStep: (image: File) => void
    openImport: (isImportingBackground?: boolean) => void
}
```

This part of the state is designed to facilitate the step-wise import and manipulation of images, which is used when importing a background of a template or an image as a part of a template. It defines the `ImageImportState` type, which encompasses essential attributes and functions. Firstly, it includes `isImportingBackground` to destinguish between an image being imported as a background or as a part of a template. The `importCurrentStep` attribute is used to keep track of the current import step defined by `ImageImportStep`, and `importedImage` to hold the imported image file.

The `importNextStep` function allows seamless transitions between import steps. Depending on the current step, it updates the state accordingly. For example, if the current step is `import`, it advances to `crop`, and if it's `crop`, it proceeds to `remove-bg`. When the last step is advanced from, the image is either set as the background or it is added to the template's images.

The `openImport` function initializes the image import process. It accepts an optional parameter `isImportingBackground` to determine the type of import, but can default to the value in the store. It resets the state and opens the import dialog, preparing the application for the user to begin importing and manipulating images.

2.5.4 Template Texts

```
type FTemplateText = Partial<TemplateText> &
    Omit<TemplateText, 'id' | 'templateId'>

type TextState = {
    texts: {
        [key: string]: FTemplateText
    }
    addText: () => void
    deleteText: (key: string) => void
    updateText: (key: string, text: FTemplateText) => void
}
```

This chunk of the state manages text elements within a meme template, enabling dynamic addition, modification, and deletion of text elements. It defines the `TemplateTextState` type, which includes attributes and functions for handling text elements within a meme template.

The `texts` property stores text elements where each key represents a unique identifier for each element, and the associated value is an object containing various properties of the text which are defined by the database schema. It initially starts as an empty object, but text elements are added dynamically.

The `addText` function is responsible for adding a new text element to the texts object. It generates a unique identifier using the `uuid` library and initializes the properties of the text element with default values, such as position at (0,0), default colors, width and height, an empty text string, and other styling properties. The zIndex property is incremented to ensure that the next element appears on top of existing ones.

The `updateText` function allows for updating the properties of a specific text element identified by its `id`. It receives the `id` and the new text properties and updates the state by replacing the existing text element properties with the new ones.

`deleteText` function removes a text element from the state based on its `id`. It uses the delete operator provided by javascript to delete the corresponding text element from the texts object, effectively removing it from the template.

2.5.5 Template Images

```
 type FTemplateImage = Omit<TemplateImage, 'id' | 'templateId' | 'imageId'> &
    Partial<Pick<TemplateImage, 'id' | 'templateId' | 'imageId'>> &
    (
        | { imageFile: File; imageUrl?: never }
        | { imageUrl: string; imageFile?: never }
    )

type ImageState = {
    images: {
        [key: string]: FTemplateImage
    }
    addImage: () => void
    updateImage: (key: string, image: FTemplateImage) => void
    deleteImage: (key: string) => void
}
```

This part of the state enables dynamic addition, modification, and deletion of image elements in templates. It also integrates with the `openImport` function to simplify the process of adding images by triggering the image import dialog, which handles all to logic realted to importing images. The underlying images are either represented by binaray data in the form of a `File` object or by a link to an image, depending on whether they have been uploded to the server or not.

The `addImage` function is responsible for adding a new image element to the images object. When called, it invokes the `openImport` function, which initiates the image import process.

The `updateImage` function allows for updating the properties of a specific image element identified by its `id`. It receives the `id` and the new image properties, and it updates the state by replacing the existing image element properties with the new ones. This function is essential for customizing the images within the template, like editing its size, rotation and other properties.

The `deleteImage` function removes an image element from the state based on its `id`. It uses the javascript delete operator to delete the corresponding image element from the images object, effectively removing it from the template.

2.5.6 Template

```
type TemplateState = {
    template: Pick<Template, 'isPublic' | 'name'> & Partial<Template>
    currentZIndex: number
    switchZIndexes: (
        obj1: { type: 'text' | 'image'; id: string },
        obj2: { type: 'text' | 'image'; id: string }
    ) => void
    updateTemplate: (updates: Partial<Template>) => void
    saveTemplate: (overwrite: boolean) => void
    useTemplate: (id: string) => Promise<void>
}
```

This part of the state is responsible for managing the currently edited template. It defines the `TemplateState` type, which includes the `template` object, which stores the template details, and the `currentZIndex` attribute, which is used to keep track of the current highest z-index value.

The `switchZIndexes` function is used to switch the z-indexes for text and image elements in the template. It takes two elements as parameters, each containing the type of the element and its `id`. It finds the objects in the template and switches their z-indexes, ensuring that they appear in the correct order when rendered.

The `updateTemplate` function allows for updating the template information by merging the provided updates into the existing template state, ensuring that the template data remains up-to-date. When updating the template, the images that were previously uploaded are not uploaded, the link to the image is reused.

The `useTemplate` retrieves a template from an API, initializes the application's state with the template data, and sets the background, texts, and images. It also handles z-index calculations to ensure proper layering of elements.

The `saveTemplate` function is responsible for saving meme templates. It checks whether the template is being updated or created and makes the appropriate API calls to save the template and associated elements (images and texts). Images have to firstly be uploaded using the `upload-image` API endpoint before the links can be sent using `tRPC` to the database. The function also handles the validation of required data and displays success or error messages using pop-up notifications.

2.5.7 Meme

```
type MemeState = {
    exportedMeme: null | string
    setExportedMeme: (exportedMeme: null | string) => void
    exportMeme: (meme: Blob, authenticated?: boolean) => void
}
```

This portion of the state is responsible for handling the export of meme images. It takes care of both authenticated and non-authenticated scenarios, ensuring that the exported meme image is appropriately stored in the `exportedMeme` state and saved in the database and image storage through the usage of `image-upload` API. Additionally, it provides error handling and user feedback through pop-up messages in case of any issues during the export process.

The `setExportedMeme` function is responsible for setting the `exportedMeme` variable to a provided value.

The `exportMeme` function initiates the process of exporting a meme image. It takes two parameters: `memeFile` (the meme image file) and `authenticated` (a boolean flag indicating whether the user is authenticated). The function begins by checking the application's current state.

If the background image is missing, it triggers a pop-up error message indicating that a meme cannot be created without a background.

If the user is authenticated, the function proceeds to upload the meme image file using the `upload-image` API endpoint. Upon successful upload, it makes an API call to create a meme with the uploaded image file and obtains the meme's image path. The `exportedMeme` state is then set to the meme's image path.

If the user is not authenticated, the image is not uploaded to the server. The `exportedMeme` state to a URL created from the provided `memeFile`. This URL can be used to display the meme image to the user without persisting it into the database.

2.5.8 Persisting state

To ensure that the user's progress with edited templates is persisted and retained even after a page refresh, the application leverages the `persist` middleware provided by Zustand. This middleware offers various options for storing and serializing state. In this case, the application serializes the state into JSON format and saves it to session storage. However, some properties are intentionally excluded from persistence as they are either unable to be serialized into JSON, like images, or they aren't needed on refresh, like popups.

2.6 Styles

In the realm of web applications, the integration of Cascading Style Sheet (CSS) is imperative for transforming a plain text-on-white background interface into a visually appealing and distinctive user experience.[32] Over time, various approaches have emerged for crafting styled components using CSS. One particularly promising method is exemplified by the Tailwind CSS library.[33]

Tailwind CSS, introduces a paradigm centered around utility classes. This approach offers immense convenience to developers. Instead of manually crafting extensive CSS files, developers can directly apply utility classes within their HTML markup. This streamlined workflow empowers developers to swiftly implement styles and layouts, enhancing development efficiency.

Moreover, the application under consideration leverages the DaisyUI library. DaisyUI aligns closely with Tailwind CSS but extends its capabilities by bundling additional functionalities. This synergy equips developers with an extensive toolkit for designing and styling components, reinforcing the application's visual appeal and usability.[34]

In essence, the combination of Tailwind CSS and DaisyUI within this application signifies a modern and efficient approach to web styling. It exemplifies the benefits of utility classes and the convenience of accessible styled components, ultimately contributing to a polished and user-friendly interface.

2.7 Components

This section describes all the notable components that are used throughout the application. Certain components play a central role in the entire application and are therefore organized within the dedicated `common` folder. These components are used on multiple pages and are therefore not tied to a specific page.

There is also a dedicated `layout` folder, which contains components that are used to define the layout of the application. These components are used on every page and are therefore not tied to a specific page.

The rest of the components are organized into folders based on the page they are used on.

2.7.1 Alert

The `Alert` component offers a convenient way to display informative messages to users while maintaining a consistent and visually appealing user interface. It can be easily integrated into various parts of the application where notifications or alerts are required, enhancing the user experience by providing clear and context-specific feedback.

These alerts can convey different messages, such as informational, warning, success, or error messages, to the user. The component is designed to be versatile, allowing developers to customize the type of alert and the content displayed within it.

The type of the displayed alert message is customizable by providing a `type` property. This property determines the visual style and icon associated with the alert, making it easier to convey the nature of the message to the user. The available types include 'info,' 'warning,' 'success,' and 'error.'

Internally, the component utilizes Scalable Vector Graphics (SVG) icons to represent the different alert types, which are fetched from the icons object based on the specified type. These icons are displayed alongside the alert message.

The component also applies CSS classes to style the alert based on its type. For example, it adds the alert-info class for informational alerts, alert-success for success alerts, alert-warning for warning alerts, and alert-error for error alerts. These classes help maintain a consistent visual design across the application.

2.7.2 Drag-Drop

The `DragDrop` component provides a user-friendly way to handle file uploads, particularly for images, while offering flexibility and ease of use within a web application. The component renders a form element with the appropriate styling and event listeners to support drag-and-drop functionality. It also includes the hidden file input element, a label for displaying instructions, and a button to trigger the file dialog for traditional file selection.

The component receives two properties: `actions` and `loading`. The actions prop is an object containing functions to handle image-related actions, specifically `setImage` and `setError`. The `loading` property indicates whether the component is currently in a loading state, so that the user does not upload multiple images at once.

The component uses local state to track whether a file is being dragged over it and to visually highlight the drop area when a file is being dragged.

2.7.3 Meme View

The `MemeView` component enhances the user experience by allowing users to view meme images and easily copy their URLs for sharing or embedding.

The component efficiently displays meme images provided through the src property. It handles image rendering and alt text, ensuring that the meme is visible and accessible to users.

The component includes a feature that allows users to copy the URL of the displayed meme image to their clipboard with a simple click. This feature is activated when the `canCopy` prop is set to `true`.

It dynamically constructs the complete image URL, taking into account various scenarios, so that it ensures that the URL is ready for display and copying.

2.7.4 Popup

The `Popup` component offers a clean and user-friendly way to present informative messages or alerts to users, allowing them to dismiss the message when they have read it. This feature is valuable for providing feedback or important notifications within a web application.

The component utilizes the `Alert` component, which is responsible for rendering the actual alert box. There is an option to dismiss the popup message, the component includes a close button represented by the `x` icon. When clicked, this button triggers the `removePopup` function, which is obtained from the application state.

2.7.5 Template Card

The `TemplateCard` component is designed for rendering template cards within a web application. It offers various capabilities for displaying template information and interacting with them.

The component supports two different sizes for template cards, large and small. This flexibility allows for the customization of card dimensions based on the specific design requirements.

Users can select a template and use it for customization by clicking on the card. This action is facilitated by the `chooseTemplate` function, which redirects the user to the creation page and sets the selected template by interacting with the application state.

When a template is editable, the component provides an option to delete the template using a trashcan icon button. Clicking this button triggers a delete operation, and upon successful deletion, a `refetch` function can be called to update the templates list.

The component incorporates hover effects to provide visual feedback to users. When hovered over, the card scales up slightly and may change its cursor to indicate interactivity.

2.7.6 Layout

The layout component serves as the cornerstone of the application's overall structure, defining the core layout elements that persist across all pages. Specifically, it establishes a fixed `navbar` at the top of the page and a `footer` at the bottom. Notably, the `footer` retains its position at the page's bottom, regardless of the amount of content between the `navbar` and the `footer`.

Additionally, the layout component is the container for the `popups`, ensuring their presence on every page while remaining entirely detached from the behavior of individual pages. This design choice guarantees the consistent availability of popups across the application, regardless of the specific page being accessed.

2.7.7 Navbar

The navbar serves as the central hub for navigation and user management within the application. Its dynamic behavior, responsive to user authentication, enhances the user experience, offering an intuitive and efficient interface for both new and authenticated users.

Authenticated users find their profile picture displayed in the `navbar`. This visual representation allows quick identification of the logged-in user. Unauthenticated users are presented with a login button, which redirects them to the login page.

The profile picture element includes a dropdown menu, accessible through a simple click. This menu offers user-specific options, including the link to their profile page and the logout button.

2.7.8 Footer

The footer is positioned at the page's bottom and serves as the concluding element, combining copyright information with project-related links to create a well-rounded user experience.

2.7.9 Import Dialog

This section describes the collection of components that work together to provide a comprehensive and user-friendly interface for importing and manipulating images. The section does not describe each of those components individually, but explains how they are used together to create the page. The components use the state to separate the business logic from the presentation logic. The components are designed to guide users through a multi-step process of importing, cropping, and editing an image, such as removing its background.

The `ImportImageDialog` component acts as the main container for the image import process. It uses a modal dialog structure, which appears over the content of the page. The dialog's visibility is controlled by the application state. It contains a step-by-step guide (import, crop, remove background) for the user to follow, rendering different components depending on the current step.

The `StepImport` component is responsible for the initial image import step. It offers multiple ways to import an image, which have their respective components. The component manages local state for the image, including loading states and error handling, and provides visual feedback like loading indicators and error alerts. It uses the `DragDrop` component from common components as well as utilizes `AiImport` and `ImportUrl` components. These components are specialized for importing images either through AI generation or from a URL. `AiImport` contains a text prompt that gets sent to the `generate-ai-image` API endpoint, while `ImportUrl` lets users import images via URLs. Both components manage their local state and handle errors, contributing to the robustness of the image import functionality.

The `StepCrop` component handles the image cropping functionality. It uses the `react-advanced-cropper` library to provide a user-friendly cropping interface. Users can select a portion of the imported image to crop, and this component manages the state of the cropped image. It provides buttons to confirm the crop or cancel the operation, integrating seamlessly with the overall image import workflow.

The `StepRemoveBg` component offers the functionality to remove the background from an imported image. It uses the `remove-background` API call to perform the background removal and manages the state of the image before and after the removal process. Users can choose between the original and the background-removed images. The component also provides a loading state to enhance user experience during processing.

2.7.10 Create

This section describes the collection of components that are used on the `create` page, which is designed for creating, editing, and managing meme templates. The components use the state to separate the business logic from the presentation logic.

The page features a central area where users can choose from a range of pre-defined meme templates or import their own custom backgrounds. The templates are requested using the `getInfiniteTemplates` API endpoint. This functionality is conveniently enabled through the usage of the React Query library.[35] The templates are displayed using the `TemplateList` component, which uses the `TemplateCard` component to render individual templates on the page.

A significant part of the page is dedicated to the meme creation process. This is facilitated through a canvas defined in the `Canvas` component, where users can add and manipulate text and images. The canvas is interactive, allowing for dragging, resizing, and rotation of elements, made possible through the integration of `Konva`, a canvas manipulation library. This provides users with a high degree of control over the design and layout of their meme, encouraging creativity and experimentation.

Adjacent to the canvas, a properties panel is present, represented by the `Properties` component, offering detailed control over the individual elements added to the meme. For text, options like font size, color, and outline can be adjusted. This panel is dynamically populated based on the elements present in the state. The panel has the option to export a meme, which is enabled by the `ExportMeme` component.

Authentication plays a crucial role in this page, as evidenced by the integration of `useSession` from `NextAuth`. This allows for features like saving templates and exporting memes to be gated based on user login status. Non-authenticated users are prompted to log in, ensuring a layer of security and personalization.

2.7.11 Templates

This section describes the components that display a template browsing interface. The `TemplateList` component serves as a container that manages state and fetches data using the `api`. There is a limited amount of templates displayed with the option to load more templates if available. This feature is enabled through the usage of the React Query library.[35]. The templates are displayed using the `TemplateCard` component, which renders individual templates on the page.

The `TemplateSearch` component is responsible for filtering templates based on a search query. It manages the search state locally and provides a search field where users can type their search query. The search query is then used to filter the templates displayed by the `TemplateList` component.

The component is displayed on two pages, one displayes all the available templates including public templates from other users, while the other displays only the templates created by the current user. The `TemplateList` component is used on both pages, but the `TemplateSearch` component is only used on the page that displays all the templates.

2.7.12 Memes

This section describes the components that display a list of all the memes created by the current user.

The `MemeList` serves as the main container, fetching and rendering memes in a grid layout with the ability to load more on demand using React Query.[35] If no memes are found, it shows a message and a link to encourage meme creation.

The `MemeCard` component provides an interactive element for each meme, offering a quick view and delete functionality. It receives meme data, and callback functions for handling meme view actions and refetching. It also includes a delete button, allowing users to delete a meme and then refresh the list using the `refetch` callback.

The `ViewModal` component utilizes the `MemeView` component from common components to display the meme in a larger format including the functionality to copy the meme's sharable URL to the clipboard.

2.7.13 Profile

The profile page displays the user's profile. The component visually represents the user's activity on the platform by displaying the number of meme templates and memes created, thus providing a quick statistical overview of the user's contributions. This functionality is enabled by `Next.js`' server side rendering to display the user's statistics. Server side generation means that the page does not request the statistics after the page is loaded, but rather the statistics are generated on the server and then sent to the client embedded onto the page.[36] The statistics are passed to the page as a property using `Next.js`'s `getServerSideProps` function.

The component presents a profile picture with an avatar component, accompanied by an edit icon. Clicking this icon opens the dialog for changing the profile picture, which uses the `DragDrop` component from common components for functionality. The user's name and email are prominently displayed, with the added functionality to edit the username with a text input field. Upon clicking the button, the component commits these changes to the database via the `updateUserName` mutation.

2.8 Build

This segment of the thesis delves into the intricacies of building and compilation processes within the context of containerized application deployment. The focal point of this discussion is the utilization of GitHub Actions, an advanced CI/CD platform that facilitates automated workflows. This tool is instrumental in the construction of the application into a Docker image, an essential step in ensuring the portability and scalability of the application across different computing environments.[37]

Furthermore, the thesis examines the subsequent stage of the deployment pipeline, where the Docker image, now encapsulating the application, is uploaded to Docker Hub. Docker Hub serves as a pivotal repository for managing Docker images, offering an efficient avenue for storing and sharing container images. This platform is a crucial component of the deployment process, as it enables the application to be deployed on a remote server, where it can be accessed by users.[38]

2.8.1 Dockerfile

The Dockerfile is a part of the implementation folder. It is a step-by-step instruction manual of the containerization process for the application.

The building process utilizes an optimization called a multi-stage build, which reduces the size of the images substantially[39]. The build process is split between multiple images referred to as stages, and only what is required is copied between them.

The initial stage, labeled as `deps`, short for dependencies, begins with a base image of `Node.js` and focuses on setting up the environment and installing necessary dependencies. Following this, the working directory is set, the application's package files are copied and `npm` dependencies are installed using `npm ci`, ensuring a clean, reproducible build process.[40]

The second stage, builder, continues from the base image and copies the entire application codebase into the Docker image. It inherits the node modules installed in the deps stage. Environment variables are set to facilitate the build process, including skipping configuration validation and defining the `DATABASE_URL` for database access. This stage executes a series of `npm` scripts which create and seed the database, build the application and prepare it for production deployment.

The final stage, runner, constructs the production image. A dedicated user is created for running the application, enhancing security by avoiding the use of a root user.[41] The built application, including public assets, SQLite database file, and compiled javascript files, is copied from the builder stage. The image also configures an `IMAGE_UPLOAD_DIR` directory as a volume for storing images, ensuring data persistence.[42] The container is configured to expose port 3000 and uses node server.js to start the application, making it ready for production deployment.

2.8.2 GitHub actions

GitHub Actions are used by providing a `.yml` file in the `.github/workflows` folder. The application contains a single workflow file `test-build-publish.yml`.

The workflow consists of two primary jobs: `test` and `push_to_registry`. The `test` job focuses on ensuring the quality and integrity of the application. It starts by checking out the repository's code, then proceeds to set up the environment for testing. This setup includes installing dependencies, creating and seeding a test database, and running both component and end-to-end tests. These steps ensure that every change pushed to the main branch does not introduce regressions or break existing functionalities.

The second job, `push_to_registry`, is dependent on the successful completion of the test job. This job is responsible for building and publishing the Docker image to Docker Hub. It begins with checking out the repository code, similar to the `test` job. The next steps involve logging into Docker Hub using credentials stored in GitHub secrets for security. The workflow uses third-party GitHub Actions for Docker, including `docker/login-action`[2] and `docker/metadata-action`[3], to handle the login and metadata extraction processes, respectively. The extracted metadata, such as tags and labels, are used in the subsequent `docker/build-push-action`[4]. This action builds the Docker image from the ./next context, using the docker file described above, and pushes it to Docker Hub, tagged and labeled as specified.

2.9 Deployment

The application uses docker-compose to orchestrate the deployment of a multi-container Docker application. This is done using the `docker-compose.yml` file[43], which is shown in listing 2.2.

[2]https://github.com/docker/login-action
[3]https://github.com/docker/metadata-action
[4]https://github.com/docker/build-push-action

The composition consists of two primary services: `caddy` and `app`, each serving distinct roles within the architecture.

The `caddy` service is configured to function as a reverse proxy for the application. It uses the official `caddy` image[5], which is a modern, powerful, yet easy-to-use HTTP web server with automatic HTTPS.[44] The `Caddyfile`, which contains the configuration for Caddy, is mapped from the host file system into the container, ensuring that the Caddy server within the container operates with the desired settings. The service is configured to expose ports 80 and 443, which are standard ports for HTTP and HTTPS traffic, respectively. This allows Caddy to handle incoming web traffic and effectively manage Secure Sockets Layer (SSL) termination and request routing.

The app service represents the core application. It uses a the docker image created in the automated build step, which it downloads from Docker Hub. The service is also part of the main network, as defined at the end of the docker-compose file, allowing the `caddy` container to forward requests to the `app` container. Additionally, a volume is mounted to /images in the container, which is used for storing uploaded images.

Environment variables are set for the app service, configuring the application's runtime environment as described in the Configuration section.

Code listing 2.1 The Caddyfile for the reverse proxy.

```
// setup a single endpoint on the desired
// domain and reverse proxy to the app container
diplomka.zavodny.net {
    reverse_proxy masters-app:3000
}
```

Code listing 2.2 The docker-compose.yml file

```
    version: '3.1'
    services:
      # caddy is used as the reverse proxy
      caddy:
        container_name: masters-caddy
        image: caddy:latest
        volumes:
          # map the Caddyfile from the host file system into the docker container
          - './Caddyfile:/etc/caddy/Caddyfile'
        ports:
          # expose ports 80 and 443 for http/https communication
          - '80:80'
          - '443:443'
        # containers need to be on the same network to see each other
        networks:
          - main
      app:
        container_name: masters-app
        image: docker.io/ozavodny/masters-thesis:main
        networks:
          - main
        volumes:
          # map the volume for image storage
          - [PATH_TO_IMAGES]:/images
        environment:
          # set the application configuration
          - PORT=3000
```

[5]https://hub.docker.com/_/caddy

```
          - DATABASE_URL=file:./db.sqlite
          ...
    networks:
      main:
```

Chapter 3

Testing

This chapter focuses on the testing aspect of the application. It describes the testing process, including the tools used and the types of tests performed. Two automated testing metodologies were used: component testing, end-to-end testing. The chapter also describes the usability testing process and the results of the test.

The automated testing within this project was carried out using the Cypress testing framework. Cypress stands out as a versatile and powerful tool in the realm of software testing, offering comprehensive end-to-end and component testing capabilities. Notably, it facilitates these tests through actual browser environments, ensuring a high degree of accuracy and realism in simulating user interactions.[45]

3.1 End-to-end testing using cypress

This section describes automated testing of the user profile functionality. The test focuses on two critical features: changing a user's username and updating the profile picture.

The test suite, named `Profile test` initiates with a before block that outlines the steps for setting up the testing environment. This includes navigating to the home page of the application, triggering the login process, and waiting for authentication to complete. The use of `cy.intercept` to monitor network requests indicates a check for successful authentication. This setup ensures that tests run under authenticated user conditions.

The test case simulates user actions such as navigating to the profile page, entering a new username in the input field, and saving the change. The test verifies the functionality by checking if the username displayed on the profile page reflects the new name. This test ensures that the username change feature works as expected and updates the user interface accordingly.

The second part of the test focuses on the functionality to update a user's profile picture. It uploads a new image file and confirms that the new image is previewed correctly. It then saves the change and checks if the dialog box closes as expected, indicating successful completion of the process.

3.2 Component testing using cypress

This section describes automated testing of the common components of the application. The test suites associated with each component are located in the `src/components/common` folder alongside the respective components.

The tests focus on the functionality of the components, ensuring that they work as expected. The tests simulate user actions such as clicking buttons, entering text, and uploading files. The tests verify the functionality by checking if the user interface reflects the expected changes. This ensures that the components work as intended and that the user interface is updated accordingly.

3.3 Usability test

This usability test was concluded with the goal of better understanding the user's perspective and possibly finding ways to improve the application. Another goal of this test was to figure out the strengths and weaknesses of the application.

3.3.1 Criteria for participants

Users should have at least a general understanding of memes.

3.3.2 Participants

- 24-year-old student - Technical
- 26-year-old person - Technical
- 50-year-old person - Non-Technical

3.3.3 Testing Setup

Participants were given 40 minutes to complete their tasks. The testing assistant was with them the whole time and instructed them to comment on their steps and thoughts. After they completed the tasks, they were encouraged to have additional comments and ask follow-up questions and were given additional information if needed. Users participated in testing willingly with the understanding that their information would be anonymous.

3.3.4 Tasks

Users were given 7 tasks to complete:

1. Create a meme from an already existing template
2. Change the page to the light theme
3. Log in
4. Change your username
5. Create a new template with a new uploaded picture (picture provided)
6. Add AI generated picture of a cat without background into the new template
7. Create a meme from the new template
8. Delete a created meme from memory
9. Change the new template by adding picture trough URL (URL provided)

3.3.5 Results

- Task 1 - All users were able to complete this task
- Task 2 - All users were able to complete this task
- Task 3 - All users were able to complete this task. User 1 was the only one that chose to use Discord as a sign-in method. User 2 managed to have their link expire by sending it twice and clicking on the wrong one
- Task 4 - All users were able to complete this task
- Task 5 - All users were able to complete this task. User 3 had a hard time finding the feature and had to be helped after asking
- Task 6 - All users were able to complete this task
- Task 7 - All users were able to complete this task
- Task 8 - All users were able to complete this task
- Task 9 - User 3 did not manage to complete this task, they did not understand that saving a changed owned template will give them the option to override the saved one, they also ran out of time

3.3.6 Found problems

The application lagged from time to time.

3.3.7 Good feedback

Participants liked the design of the application, and they found it understandable and easily usable. Two of them also mentioned plans to use the application for their meme creation needs. The application was called modern looking. The dark theme of the application was appreciated. Participant 2 liked the search function in All Templates. Participant 1 liked that the site does not use passwords, as they do not view it as a safe way of verification. All participants enjoyed the AI generation and deleting backgrounds of pictures.

3.3.8 Bad feedback

Two participants did not perceive positively the lack of a password and the need to use Discord or email again on potentially repeated login. Participant 3 found the changing of existing template not intuitive.

3.3.9 Conclusion

The site was better received by younger participants, this was not a surprising fact as they are the targeted group. The usability test had overall positive results. The confusion of user 3 during task 5 led to a change in the site. The window for importing custom backgrounds became more highlighted. The possible inclusion of a password was considered, but this change was purposefully not implemented, as the belief of Participant 1 was seen as stronger.

3.3.10 Disclaimer

The group of participants may not represent every possible user of the application. Every person has their own biases (e.g. using or not using passwords) and this can possibly affect usability testing results.

Chapter 4
Conclusion

This thesis embarked on a journey to explore the dynamics of meme creation on the internet and to develop a novel solution with different functionalities.

The first chapter of this thesis is dedicated to a comprehensive analysis. This segment meticulously evaluates existing meme-creation solutions, laying the groundwork for the conception of a custom solution. This process is grounded in the established design principles of software engineering. The focus of this chapter is twofold: to critically assess the current landscape of digital meme creation and to pave the way for an innovative, user-centric solution.

Proceeding to the second chapter, the thesis delves into the architecture and functionalities of the developed application. This discussion focuses both on the graphical and functional aspects of modern web development. The application's implementation was formed by a careful evaluation and selection of various libraries, with an emphasis on usability and simplicity. The latter part of this chapter provides a detailed explanation of the application's build process and deployment strategy.

The final chapter is devoted to testing, where a state-of-the-art end-to-end and component testing framework is employed to validate the basic functionality of the application. This section is not limited to technical testing but extends to a thorough usability testing session, conducted with prospective users. This approach provides invaluable insights into the application's user experience and interface design.

The newly developed application aligns closely with the functionalities observed in the analyzed solutions, indicating a strong adherence to industry standards and user expectations. However, a key differentiator that emerged during this comparison is the application's unique feature to remove backgrounds from images. This functionality is notably absent in the other solutions and it is not just a technical enhancement but also a strategic move to foster user creativity. This capability significantly enriches the user experience and potentially broadens the application's appeal and usability.

In conclusion, while the implementation successfully meets the initial objectives set by the thesis, it also opens avenues for further enhancement. The thesis identifies several features from existing solutions, as outlined in the analysis chapter, that hold potential for future integration. This recognition of possible enhancements not only underscores the application's current efficacy but also highlights its capacity for evolution and adaptation in the rapidly changing landscape of internet meme creation.

Bibliography

1. SHIFMAN, Limor. *Memes in digital culture*. The MIT Press, 2014. MIT press essential knowledge. ISBN 9780262525435.
2. MATTISON, Ollie. *Top 12 Best Meme Makers Online for FREE* [online]. 2023-12-14. [visited on 2023-12-21]. Available from: `https://filmora.wondershare.com/meme/best-free-meme-maker-online.html`.
3. SANDY. *Top 5 Best Meme Makers - DIY Funniest Memes Online* [online]. 2023-06-06. [visited on 2024-01-01]. Available from: `https://www.flexclip.com/learn/meme-generator.html`.
4. BAKER, Luke. *Top 12 Best Meme Makers Online for FREE* [online]. 2023-09-30. [visited on 2024-01-01]. Available from: `https://www.pocket-lint.com/best-meme-generator-app`.
5. YOUCAM, Team. *5 Best Free Meme Maker Apps for iPhone & Android in 2023* [online]. 2023-10-05. [visited on 2024-01-01]. Available from: `https://www.perfectcorp.com/consumer/blog/photo-editing/best-meme-maker-apps`.
6. CAPCUT. *15+ Best Meme Maker You Should Have a Try in 2023* [online]. 2023-12-15. [visited on 2024-01-01]. Available from: `https://www.capcut.com/resource/best-meme-maker`.
7. TRENDS.GOOGLE.COM. *Google Trends* [online]. 2023-02-21. [visited on 2023-02-21]. Available from: `https://trends.google.com/trends`.
8. PLAY.GOOGLE.COM. *Google Play - Mematic* [online]. 2023-02-21. [visited on 2023-02-21]. Available from: `https://play.google.com/store/apps/details?id=net.trilliarden.mematic`.
9. APPS.APPLE.COM. *App Store - Mematic* [online]. 2023-02-21. [visited on 2023-02-21]. Available from: `https://apps.apple.com/us/app/mematic-the-meme-maker/id491076730`.
10. LLC, Imgflip. *Meme Generator* [online]. [N.d.]. [visited on 2023-03-27]. Available from: `https://imgflip.com/memegenerator`.
11. ILOVEIMG. *Our features* [online]. [N.d.]. [visited on 2023-03-22]. Available from: `https://www.iloveimg.com/features`.
12. ILOVEIMG. *Compare plan features* [online]. [N.d.]. [visited on 2023-03-22]. Available from: `https://www.iloveimg.com/pricing`.
13. KARL WIEGERS, Joy Beatty. *Software Requirements 3*. 3rd Revised edition. Microsoft Press,U.S, 2013. Developer Best Practices.
14. HAMM, Matthew. *Wireframing Essentials*. PACKT, 2014. ISBN 1849698546.

15. BABU, Rajesh. *A Quick Guide To Choosing A Robust Frontend Tech Stack* [online]. 2022. [visited on 2024-01-04]. Available from: `https://medium.com/@rajeshdavid/a-quick-guide-to-choosing-a-robust-frontend-tech-stack-1f47c0458f08`.
16. *Stack Overflow Developer Survey 2023* [online]. 2023. [visited on 2024-01-04]. Available from: `https://survey.stackoverflow.co/2023/`.
17. DAKOWICZ, Jakub. *WHAT IS NEXT JS AND WHY SHOULD YOU USE IT IN 2024?* [online]. 2023. [visited on 2024-01-04]. Available from: `https://pagepro.co/blog/what-is-nextjs/`.
18. RIVA, Michele. *Real-World Next.js: Build scalable, high-performance, and modern web applications using Next.js, the React framework for production.* Packt Publishing, 2022. ISBN 9781801073493.
19. AGARWAL, Avneesh. *Why and how to get started with Next auth?* [online]. 2021. [visited on 2024-01-05]. Available from: `https://medium.com/geekculture/why-and-how-to-get-started-with-next-auth-61740558b45b`.
20. ISRAEL. *Redux vs Zustand vs Context API: Their Pros, Cons, and Usage* [online]. 2023. [visited on 2024-01-05]. Available from: `https://bootcamp.uxdesign.cc/redux-vs-zustand-vs-context-api-their-pros-cons-and-usage-d3bcbb79ab6a`.
21. ORUC, Ugur. *4 Ways We Use Prisma to Speed Up Development* [online]. 2022. [visited on 2024-01-05]. Available from: `https://www.oakslab.com/story/4-ways-we-use-prisma-to-speed-up-development`.
22. DODDS, Kent C. *Why you should probably be using SQLite* [online]. 2023. [visited on 2024-01-05]. Available from: `https://www.epicweb.dev/why-you-should-probably-be-using-sqlite`.
23. PRISMA DATA, INC.. *Prisma Documentation* [online]. [N.d.]. [visited on 2023-04-21]. Available from: `https://www.prisma.io/docs`.
24. PRISMA DATA, INC.. *SQLite* [online]. [N.d.]. [visited on 2023-04-21]. Available from: `https://www.prisma.io/docs/concepts/database-connectors/sqlite`.
25. ORBÁN, Balázs. *Options* [online]. [N.d.]. [visited on 2023-04-29]. Available from: `https://next-auth.js.org/configuration/options`.
26. ORBÁN, Balázs. *OAuth* [online]. [N.d.]. [visited on 2023-04-23]. Available from: `https://next-auth.js.org/configuration/providers/oauth`.
27. REINMAN, Andris. *SMTP TRANSPORT* [online]. [N.d.]. [visited on 2023-05-01]. Available from: `https://nodemailer.com/smtp/`.
28. GITHUB, INC. *Is there a way to handle multipart/form-data requests?* [Online]. 2021. [visited on 2023-03-21]. Available from: `https://github.com/trpc/trpc/discussions/658`.
29. PIRES, Mateus. *Is base64 encoded image uploading a bad practice?* [online]. 2022. [visited on 2024-01-04]. Available from: `https://stackoverflow.com/questions/55436601/is-base64-encoded-image-uploading-a-bad-practice`.
30. JOHANSSON, Alex. *tRPC* [online]. [N.d.]. [visited on 2023-04-13]. Available from: `https://trpc.io/docs`.
31. REFINE DEVELOPMENT INC. *How to use Zustand* [online]. [N.d.]. [visited on 2024-01-07]. Available from: `https://refine.dev/blog/zustand-react-state/#getting-started-with-zustand`.
32. W3SCHOOLS. *What is CSS?* [online]. [N.d.]. [visited on 2024-01-07]. Available from: `https://www.w3schools.com/css/css_intro.asp`.

33. ABBA, Ihechikara. *How to Use Tailwind CSS to Rapidly Develop Snazzy Websites* [online]. 2023. [visited on 2024-01-07]. Available from: `https://kinsta.com/blog/tailwind-css/#:~:text=Tailwind%20CSS%20is%20best%20used,about%20creating%20your%20design%20systems.`.

34. SAADEGHI, Pouya. *What is daisyUI? (and other questions I get asked a lot)* [online]. [N.d.]. [visited on 2023-09-10]. Available from: `https://daisyui.com/blog/what-is-daisyui/`.

35. LINSLEY, Tanner. *Infinite Queries* [online]. [N.d.]. [visited on 2023-04-15]. Available from: `https://tanstack.com/query/latest/docs/react/guides/infinite-queries`.

36. VERCEL, INC. *Static Site Generation (SSG)* [online]. [N.d.]. [visited on 2024-01-02]. Available from: `https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation`.

37. SCHMITT, Jacob. *Docker image vs container: What are the differences?* [Online]. 2024. [visited on 2024-01-02]. Available from: `https://circleci.com/blog/docker-image-vs-container/`.

38. DOCKER INC. *Docker Hub, The world's largest container registry* [online]. 2023. [visited on 2024-01-02]. Available from: `https://www.docker.com/products/docker-hub/#:~:text=Docker%20Hub%20is%20a%20container,repos%20for%20teams%20and%20enterprises.`.

39. GHOSH, Saibal. *Docker Demystified : Learn How to Develop and Deploy Applications Using Docker*. Draft2Digital, 2021.

40. KARRYS, Luke. *npm-ci* [online]. [N.d.]. [visited on 2024-01-02]. Available from: `https://docs.npmjs.com/cli/v10/commands/npm-ci`.

41. MCCARTY, Scott. *Understanding root inside and outside a container* [online]. 2019. [visited on 2024-01-02]. Available from: `https://www.redhat.com/en/blog/understanding-root-inside-and-outside-container`.

42. DOCKER INC. *Volumes* [online]. [N.d.]. [visited on 2024-01-03]. Available from: `https://docs.docker.com/storage/volumes/`.

43. DOCKER INC. *Docker Compose overview* [online]. [N.d.]. [visited on 2024-01-03]. Available from: `https://docs.docker.com/compose/`.

44. ZEROSSL GMBH. *Automatic HTTPS* [online]. [N.d.]. [visited on 2024-01-03]. Available from: `https://caddyserver.com/docs/automatic-https`.

45. TESTINGXPERTS. *Cypress Automation: 7 Key Benefits to DevOps-driven Businesses* [online]. 2023. [visited on 2024-01-05]. Available from: `https://www.testingxperts.com/blog/cypress-automation#:~:text=Cypress%20automation%20offers%20numerous%20benefits,their%20applications%20function%20as%20intended.`.

Contents of the attached media

```
masters
├── impl ........................................ Implementation source code
└── thesis.pdf .................................. Thesis thext in PDF format
```