FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Meme Generator
Student:	Bc. Ondřej Závodný
Supervisor:	Ing. Jaroslav Šmolík
Study program:	Informatics
Branch / specialization:	Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2023/2024

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 11 October 2022 in Prague.

Master's thesis

MEME GENERATOR

Bc. Ondřej Závodný

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Jaroslav Šmolík
May 4, 2023

Czech Technical University in Prague
Faculty of Information Technology
© 2023 Bc. Ondřej Závodný. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

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 Tables

List of code listings

Chtěl bych poděkovat především své snoubence a svému vedoucímu práce.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 4, 2023 .....

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

Mankind has developed many means of communication between one another. From drawing in caves, through inscribing on clay tablets, books, television and recently, memes.

The term “meme” was coined by Richard Dawkins in 1976 to describe small units of culture that spread from person to person by copying or imitation.[1] In the internet era, people usually refer to some sort of images with text as memes and their purpose is to be shared with other people.

There are many ways to create and share memes, and this thesis is focused on describing and implementing these in a web application. The application should allow users to create memes from ready to use templates, as well as creation of these templates. The creations should be easily sharable with other people.

The first chapter talks about existing meme generators and about a high-level overview of the implemented application.

The second chapter talks about all the in and outs of the implementation. Many frontend libraries were evaluated and then used or rejected. There is also a section about building the application using docker and deploying it using Docker compose.

The final chapter is denoted to testing, in this case end-to-end testing and usability testing.

Chapter 1
Analysis

1.1 Existing solutions

There are many solutions online that allow users to create memes from templates.[2] I decided to look at two websites, imgflip.com and iloveimg.com, and one mobile application, Mematic, to understand what users expect of such products and to identify flaws, that I could improve upon.

1.1.1 Popularity

Popularity is difficult to find for websites. Android applications have a rough number of downloads in the store, and we can use the number of reviews on both Android and iOS to get some idea of popularity. Websites do not have such options, but we can look at Google Trends, which shows how many times was each search term searched in Google. Since this is not at all comparable to the application metrics, websites and applications will not be compared between each other.

A graph from Google Trends is shown in image 1.1, 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[4] and 150.1 Thousand reviews on the App Store[5].

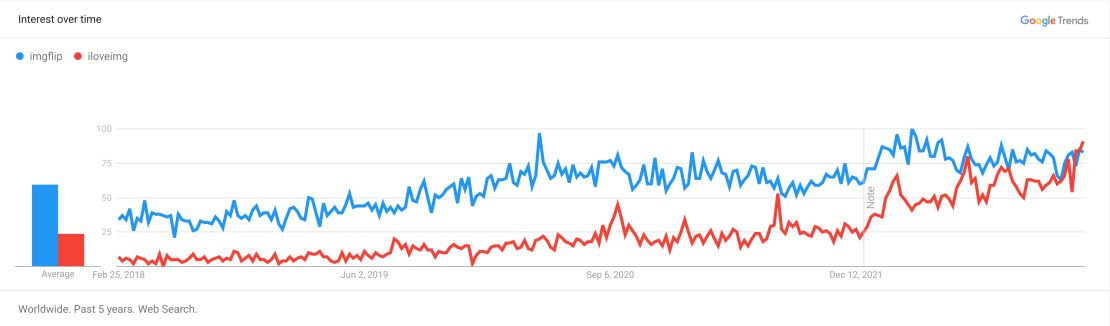


Figure 1.1 Google Trends for the search terms `imgflip` and `iloveimg` for the past 5 years. Taken on February 21. 2023.[3]

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 [6].

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.

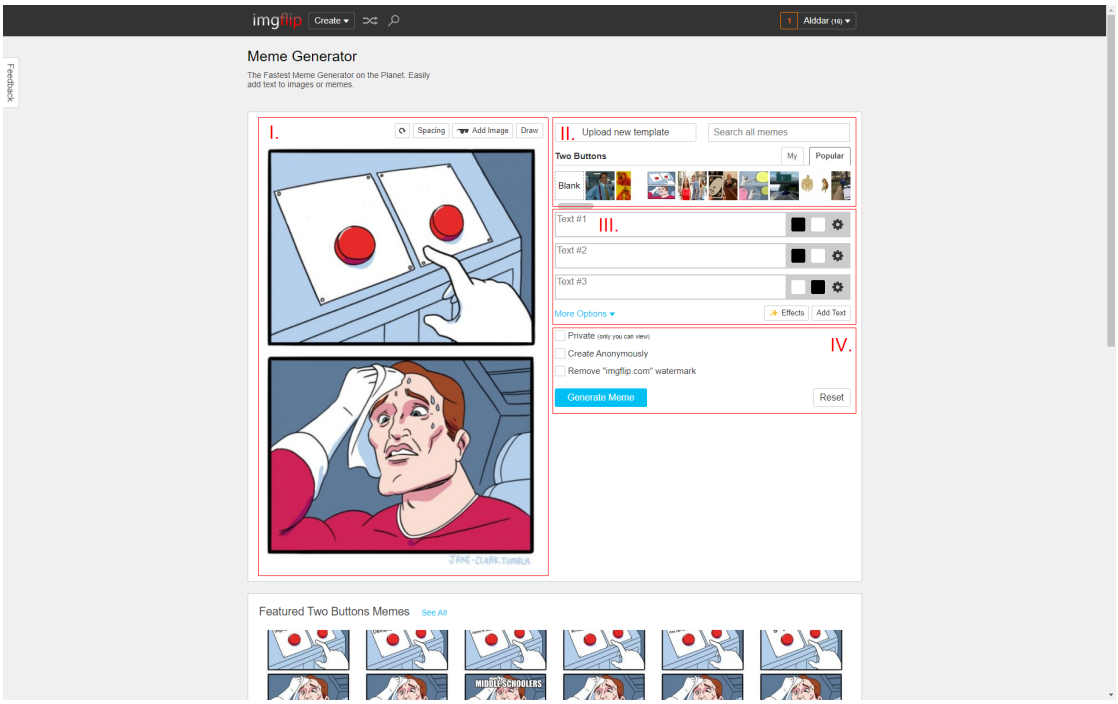
1.1.2.1.3 Section III.

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.

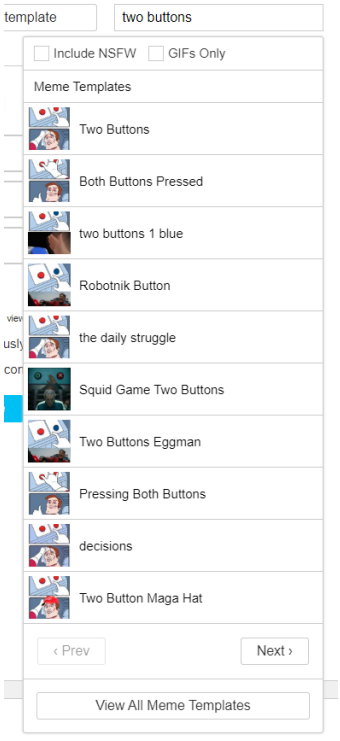
1.1.2.1.4 Section IV.

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 an imgflip.com watermark on the final generated meme, but that isn't available without getting a subscription (see subsubsection 1.1.2.4)

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

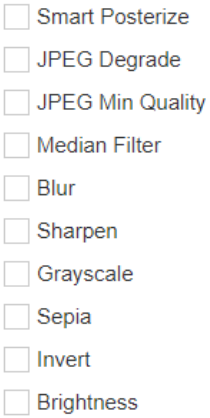


(a) Main Imgflip meme generator screen.

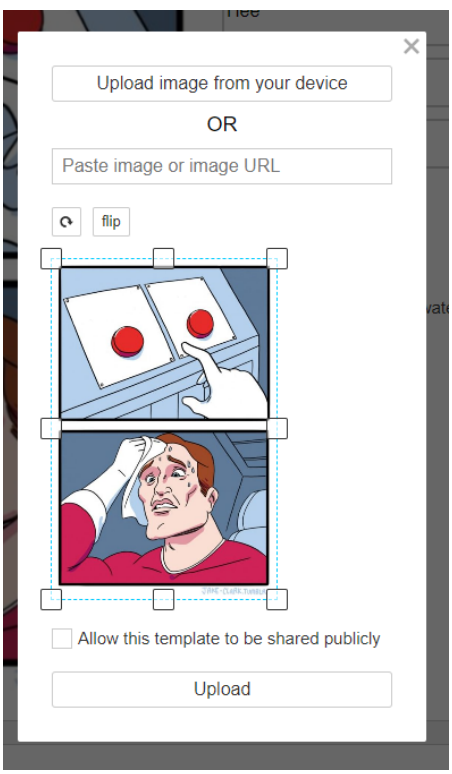


(b) Search dialog

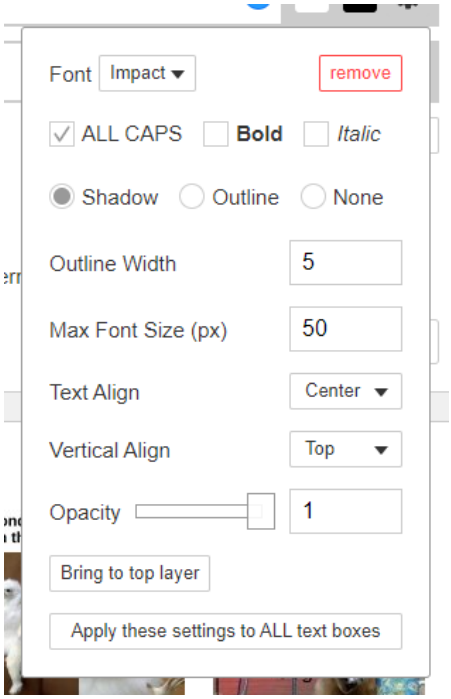
(c) Generated meme dialog.



(d) Available effects



(e) Upload dialog



(f) Text Options

Figure 1.2 Screenshots of Imgflip

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 a 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 a "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 [7].

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.

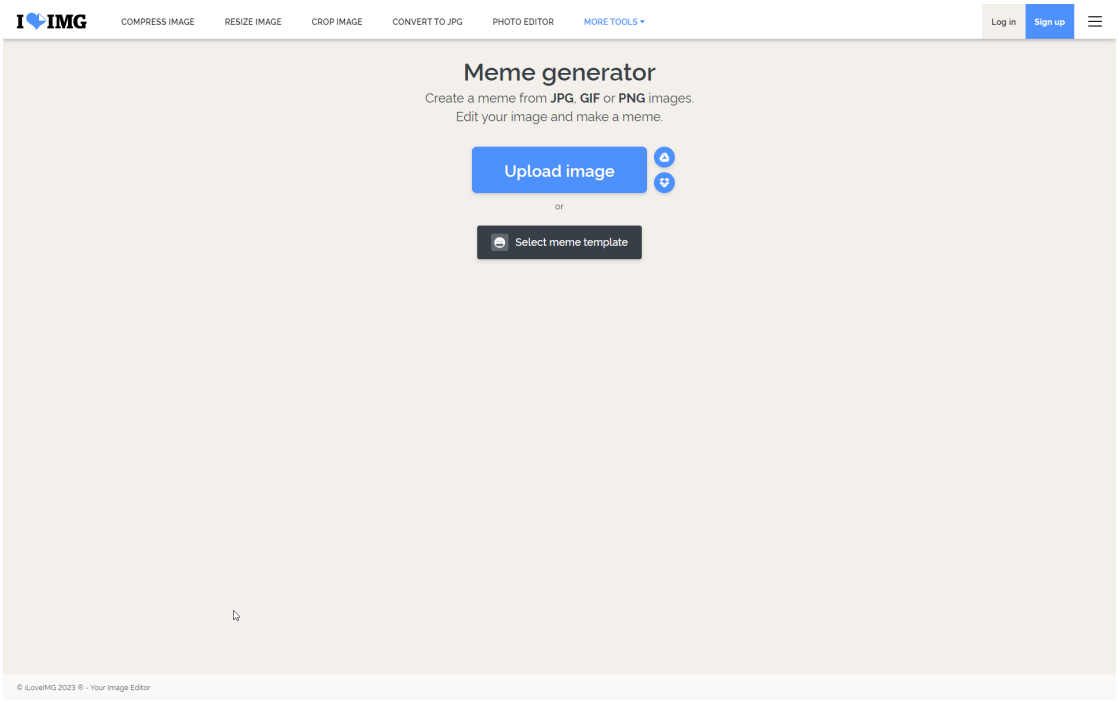
When the background is selected, the screen shown in image 1.3c appears. Here, the user can use the graphical editor to edit the meme. Every template comes with two default texts, one on the bottom and one on the top. The user can switch between the texts being positioned inside the image or outside 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.

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.

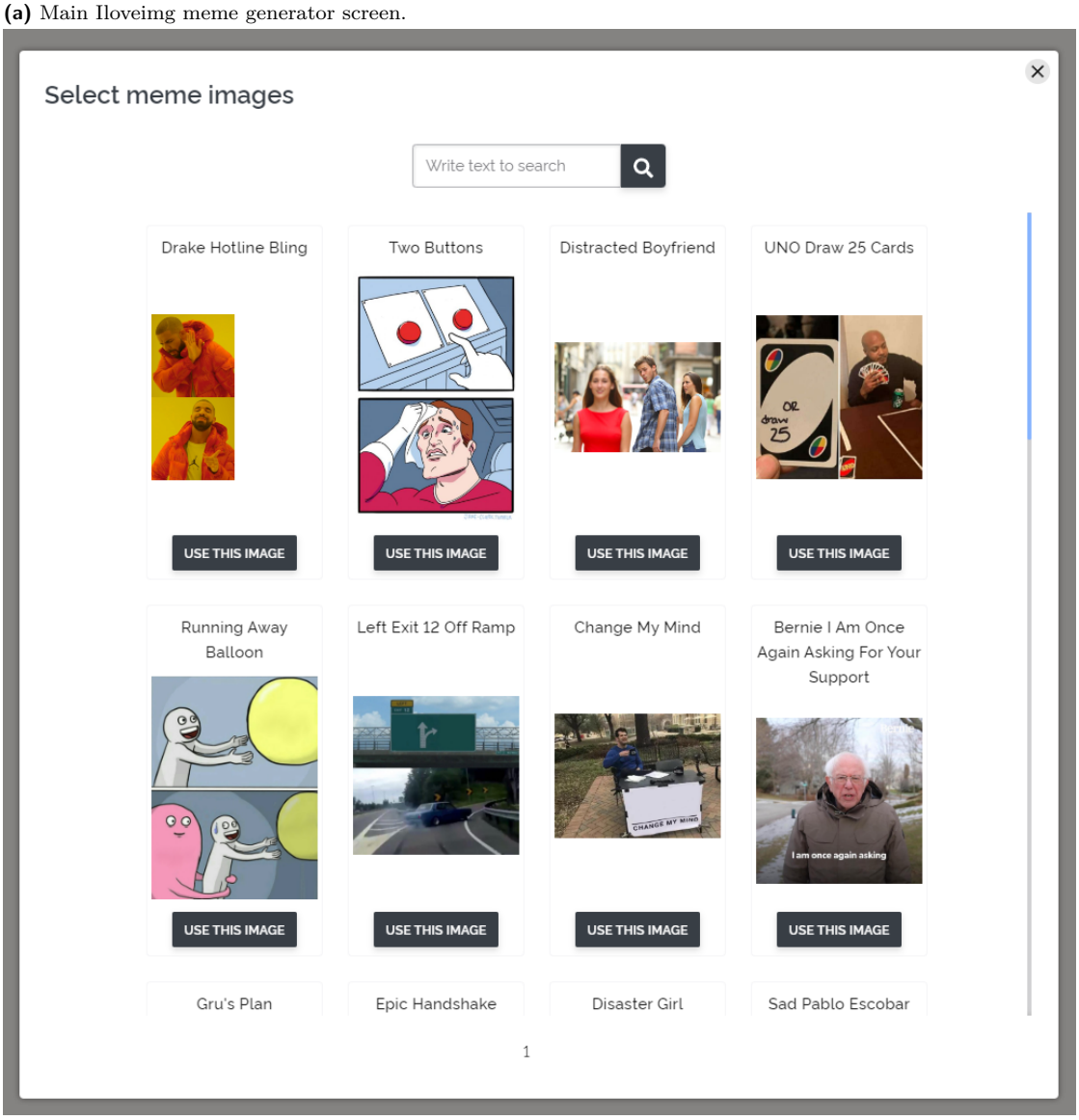
1.1.3.2 Downloading a meme

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

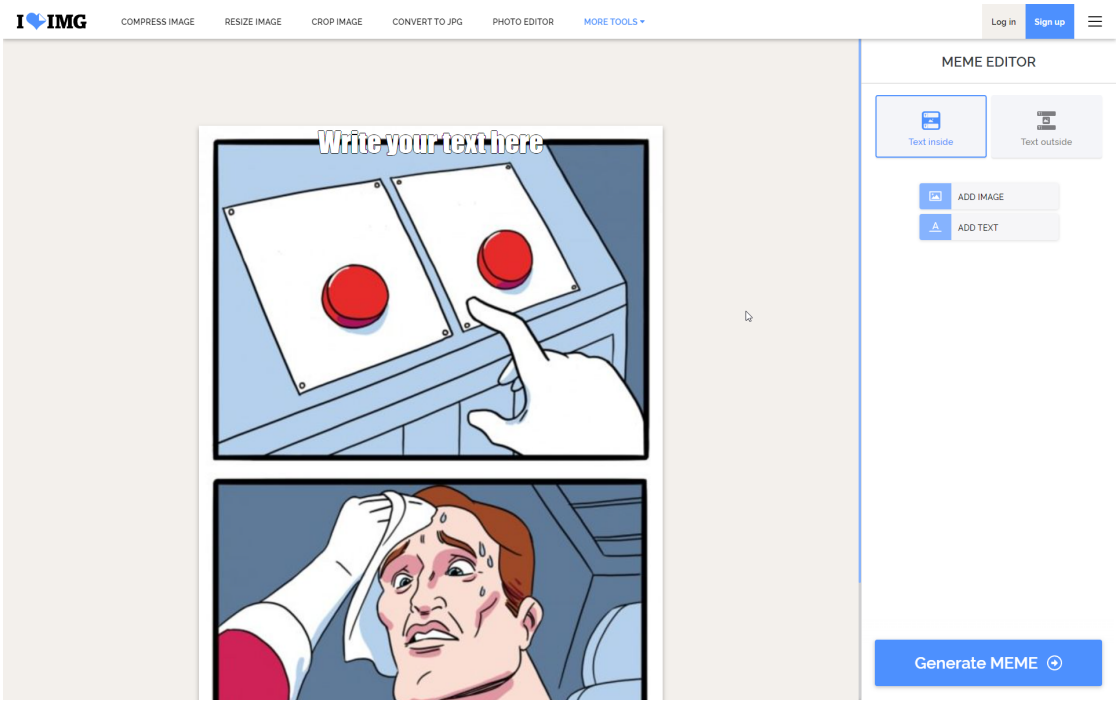
[3] https://www.iloveimg.com/meme-generator



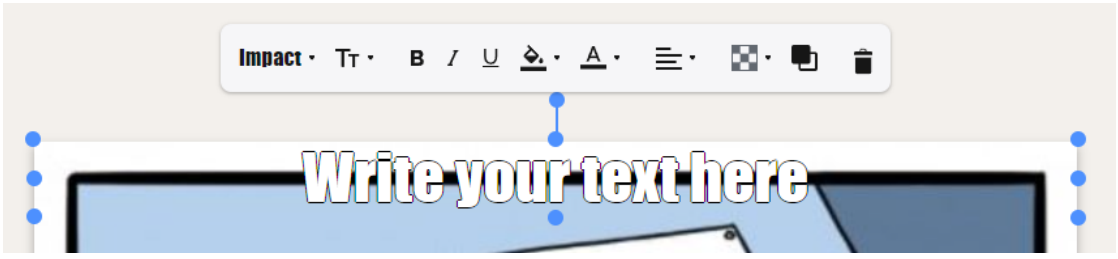
(a) Main Iloveimg meme generator screen.



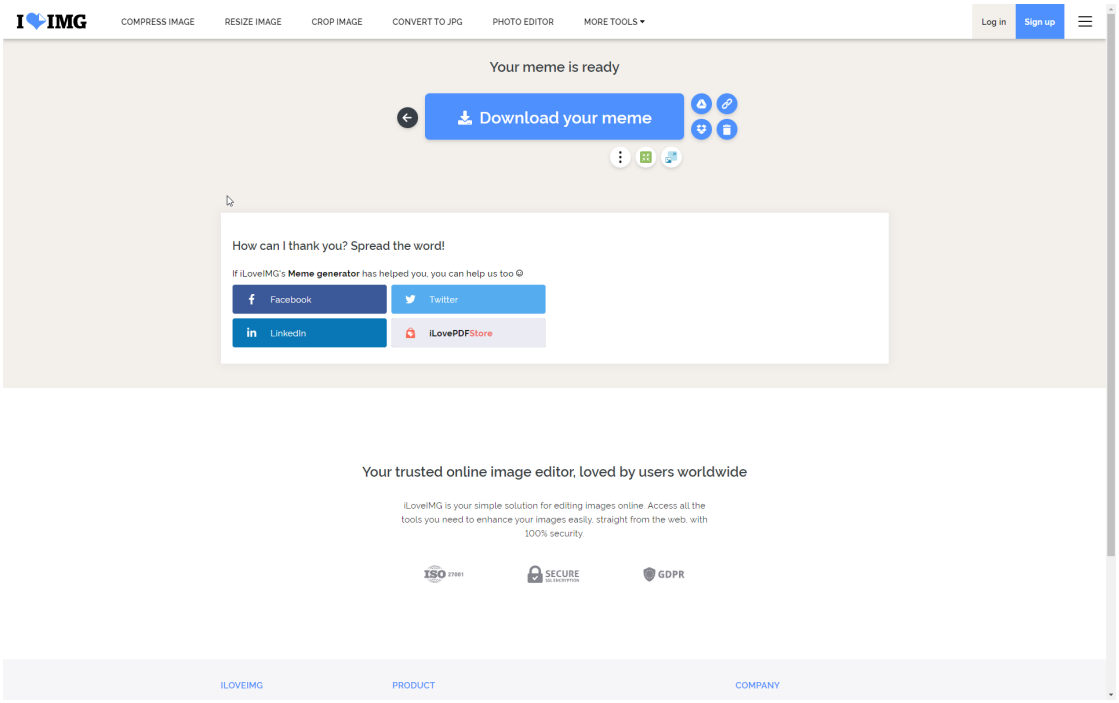
(b) Select meme dialog.



(c) Meme editor.



(d) Text editing tools



(e) Generated meme

Figure 1.3 Screenshots of Iloveimg

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 [7].

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 [8]

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 Database Schema

The database schema can paint the picture of what entities are present in the application and how they are saved to the database. The application uses Prisma () to manage the database schema using a `schema.prisma` file. The file contains the definition of the connection to the database, some generators to generate TypeScript code for the application and the definitions of all the tables seen in the image 1.5. This file is also used to create a database during the automated build (see section 2.11).

1.2.1 Authentication Tables

Tables `User`, `Account`, `Session` and `VerificationToken` are used to authenticate users and keep some information about them. All of them have their structure defined by the NextAuth.js library [9]. The `User` table stores information about users, like their name, email or image. A user can have multiple accounts connected.

The `Account` table represents a social login account, for example Facebook. Each user also has a session associated, which is used to keep them logged-in as they navigate the pages and refresh tabs.

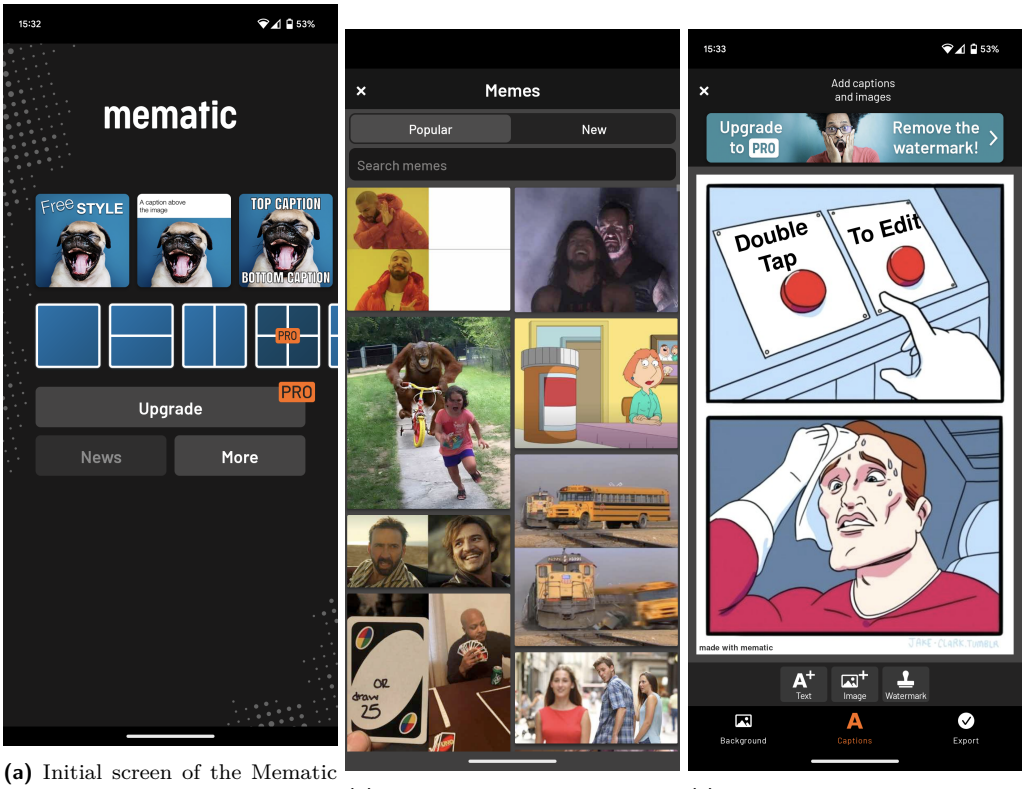
The `VerificationToken` table represents a token for passwordless login. When a user tries to log in using an email, they will receive an email with a verification token.

1.2.2 Image

The `Image` table represents an image. It has only one own property, which is `fileName`. That is the path, from which the image can be downloaded.

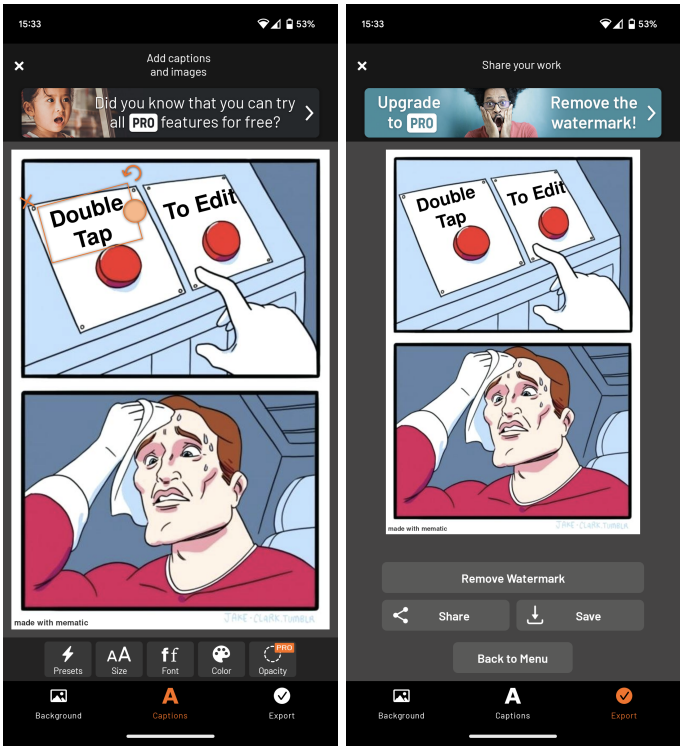
1.2.3 Template

The `Template` and `TemplateText` tables represent a template created by a user. Each template can have multiple texts. The `TemplateText` table stores information about a text, such as its value, its coordinates, size, or rotation. Each template has to belong to some user. Each template also has a background image, which is represented by the `Image` table.



(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

1.2.4 Meme

The `Meme` table represents a meme that a user has exported. It always has to belong to some user, and it also has an image, represented by the `Image` table.

1.3 High-level architecture

The application consists of five main components which communicate together.

1.3.1 Pages

The pages component serves the browser the statically generated frontend. This component contains all the static functionality and also all the code required to make the sites dynamic. The frontend then calls the API to get access to the database, image storage and authentication.

1.3.2 API

The API component of the application encapsulates all the communication between the browser and the other components. It provides procedures for the frontend to call and to receive back requested information.

1.3.3 Database

The database stores all the data relevant for the application except the images, which are stored in image storage.

1.3.4 Image Storage

The image storage component stores all the images used by the application, like the memes and the template backgrounds.

1.3.5 Authentication

The authentication component handles all authentication. It communicates with authentication providers or sends emails for email authentication. It also uses the database to store relevant data about users, accounts, and sessions.

1.4 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 [10]. The wireframes that were designed for this application are shown in the figure 1.7. The wireframes do not show how exactly the application looks at the end, but they provide an insight into the basic functionalities of the application.

1.4.1 Create Page

The `Create Page` is the main page of the application where the user will land at. The wireframes show it in two versions, one where a user has logged in to the application (image 1.7b and one where the user is anonymous (image 1.7a).

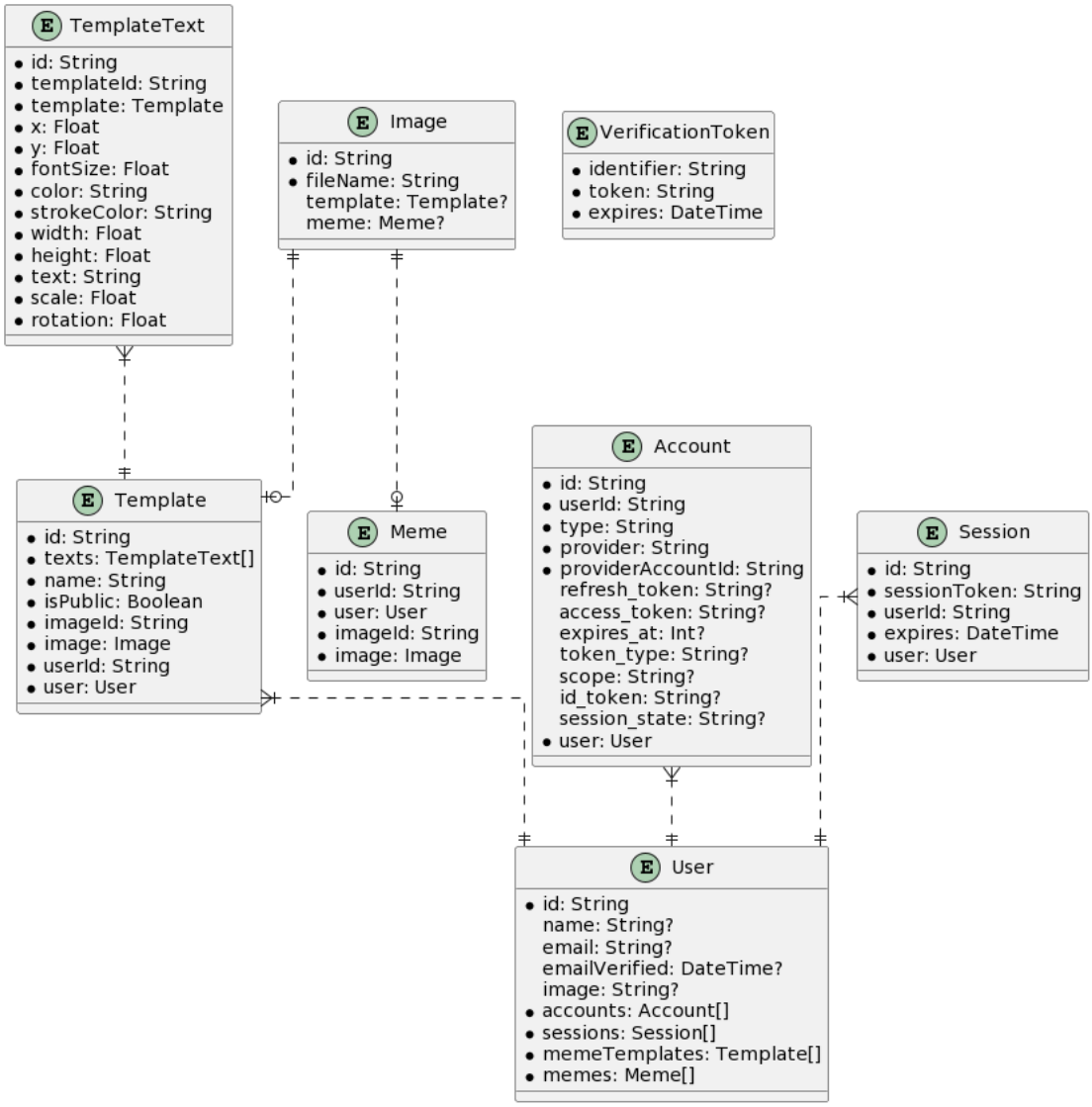


Figure 1.5 Database schema.

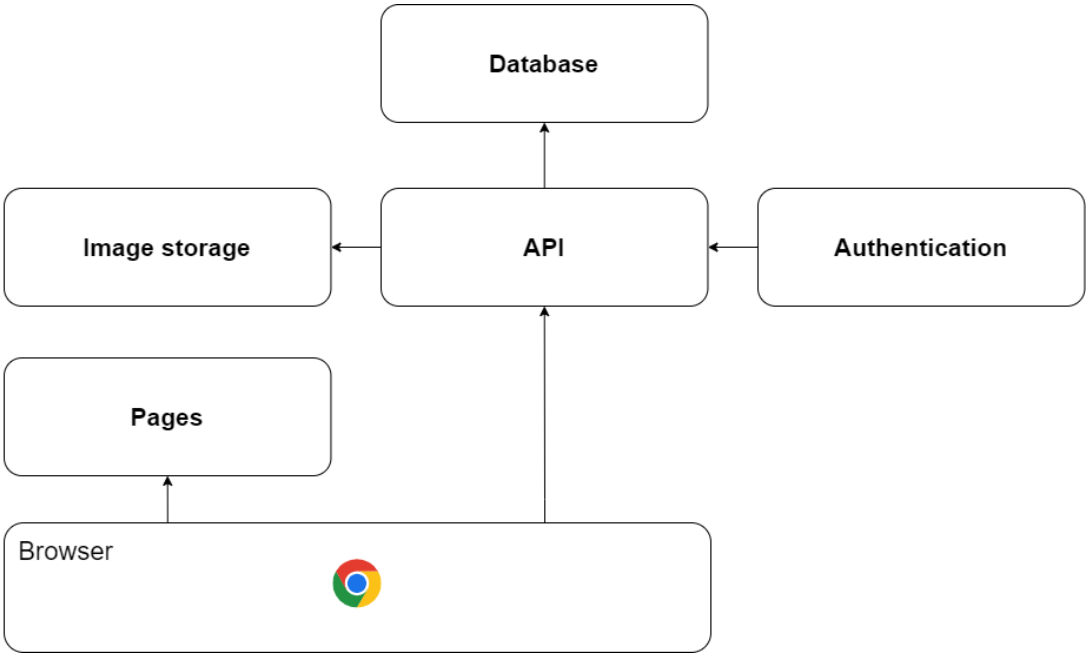


Figure 1.6 High-level architecture overview.

The main part of the page is mostly the same for both types of users. On the top there is a selection of templates with an option to import a custom background to create a new template. Under that are the two parts that are used to create the meme.

To the left is the graphical section. It has the template background as the main part and all the user-created texts are rendered in front of the background.

The right section is used to modify some other attributes of the texts, like the colors, or extra properties. There is also an option to remove the texts. Under the text options are the actions that the user can make. An anonymous user can only export the meme, which would open the exported meme page. A logged-in user can also save the template, which results in the save template page.

1.4.2 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.

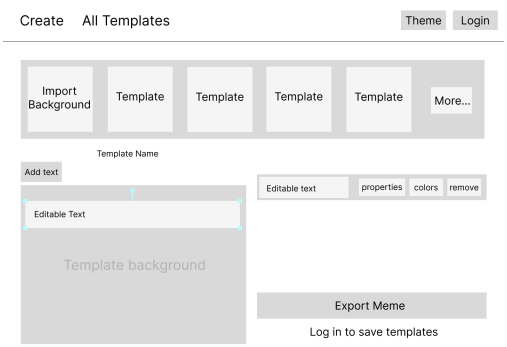
Clicking any of the displayed templates redirects the user to the `Create page` and loads the clicked template into the editor.

1.4.3 My Templates Page

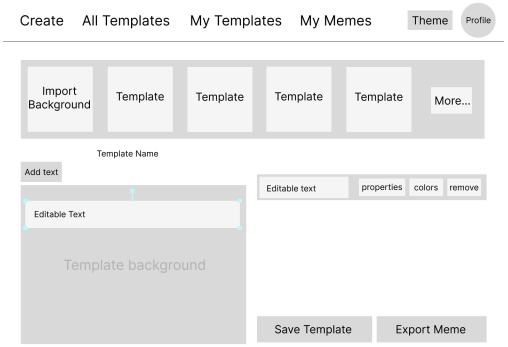
The `My Templates Page` is only available to logged-in users. Here, the users can view all the templates they created and saved. There are two options for each template, opening it in the `Create Page`, or deleting it.

1.4.4 My Memes Page

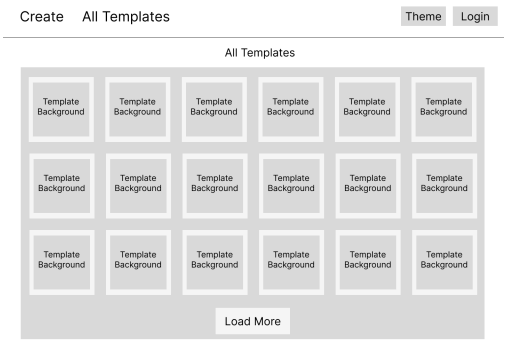
The `My Memes Page` has a similar design to the `My Templates Page`, but instead of a list of templates, it shows a list of memes. There is an option to open the meme's details, which would show a larger version of the image and the link to the meme's image.



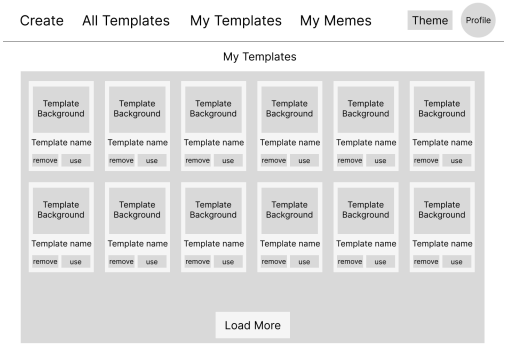
(a) Create page, anonymous user.



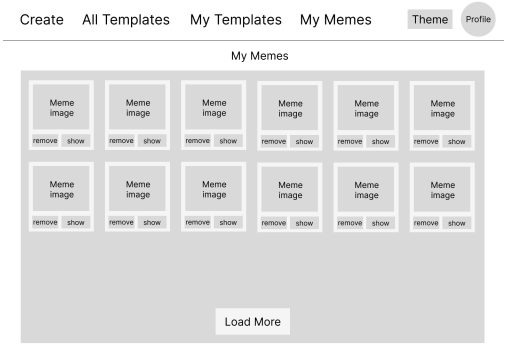
(b) Create page, logged-in user.



(c) All templates page.



(d) My templates page.



(e) My memes page.

Figure 1.7 Wireframes of the application.

Chapter 2
Development

This chapter explains how the application was developed. It covers the libraries that were used and how all the parts of the application tie together. It also explains how the library and framework selections were made.

2.1 Initial implementation

The initial implementation consists of two parts, a backend and a frontend. This initial implementation was difficult to work with and to deploy, which is why it was completely replaced by a new implementation. Many of the libraries that were used during the initial implementation ended up being used by the new implementation as well.

2.1.1 Backend

The backend is made using the express library. The types of the different entities are defined by the zod library, and tRPC is used to create type-safe communication between the frontend and the backend.

2.1.2 Frontend

The frontend uses the React library for rendering and the zustand library for state management. There are many libraries that handle state management in the React framework, mainly redux, redux-toolkit, jotai, recoil, zustand and others. After careful evaluation of each of these libraries, zustand seemed the easiest to use and provides important features out of the box, like immer integration and persistance (see section 2.8)

2.1.3 Why was it scrapped

There are reasons why this version of a semi-developed application was not an ideal candidate for this thesis.

2.1.3.1 Backend/Frontend separation

Because the backend and the frontend were two completely separate applications during development, they both had to be run separately, which is difficult to use for local development. There are also some parts of the applications which are shared anyway, like code formatting tools,

typescript configuration and compilation, or static code analysis tools. Sharing configurations of these tools between the applications is possible to do automatically with the introduction of extra dependencies, but that adds other inconveniences.

2.1.3.2 Authentication

The NextAuth library is, in my opinion, easier to use than the libraries available for express. NextAuth sets up all the pages required for authentication and also all the API endpoints with almost no configuration [11].

2.2 Application design

2.3 The T3 Stack

The `T3 Stack` is a web development stack focused on simplicity, modularity, and full-stack type safety. The development team that made the t3 stack also created a tool called `create-t3-app`. This tool creates a minimal version of a Next.js application with optional extra tools. These optional extras are: tailwind, Prisma, tRPC and NextAuth. All of these are used in the implementation of the application.

When running the `create-t3-app` command, the CLI offers interactive selection of options. The first question the tool asks is whether TypeScript or JavaScript should be set up. TypeScript is my preferred language for these types of scenarios because it provides compile-time type safety, better auto-complete and many other features [12]. The second question is about enabling extra packages to be installed and configured. The options are: NextAuth, Prisma, tailwind and tRPC. All of these useful for this project, as will be explained in later sections. All the other questions are left at the default option.

The tool initializes a new Next.js application and installs all the dependencies that were selected. It also provides boilerplate code to make everything work together.

There are also many other dependencies, that will be highlighted in the next sections.

2.4 Application diagram

The architecture diagram shown in image 1.6 is an overview of what the final deployed application looks like. The entry point of the application is a reverse proxy, which is used to easily enable and use the Hypertext Transfer Protocol Secure (HTTPS) protocol. Without it, the application would have to rely on Hypertext Transfer Protocol (HTTP), which is less secure [13]. The proxy forwards all the requests to the main Next.js application.

The application consists of statically generated pages and an Application Programming Interface (API). The pages are served to the client and the API are used for communication between the client and the server. Additionally, there is an Object-relational mapping (ORM), which is facilitated by the Prisma library, that creates a layer between the backend and the database to make the database easier to work with.

The application uses external services for some functionality. All the template images that are uploaded by users and also the generated meme images are stored inside Object Storage in Oracle Cloud. Three authentication providers are used to simplify the login and registration process.

2.4.1 Prisma

The Prisma library is used to interact with the database. The main file is `schema.prisma`, which specifies the database structure (see section 1.2). Prisma is able to update the database to make

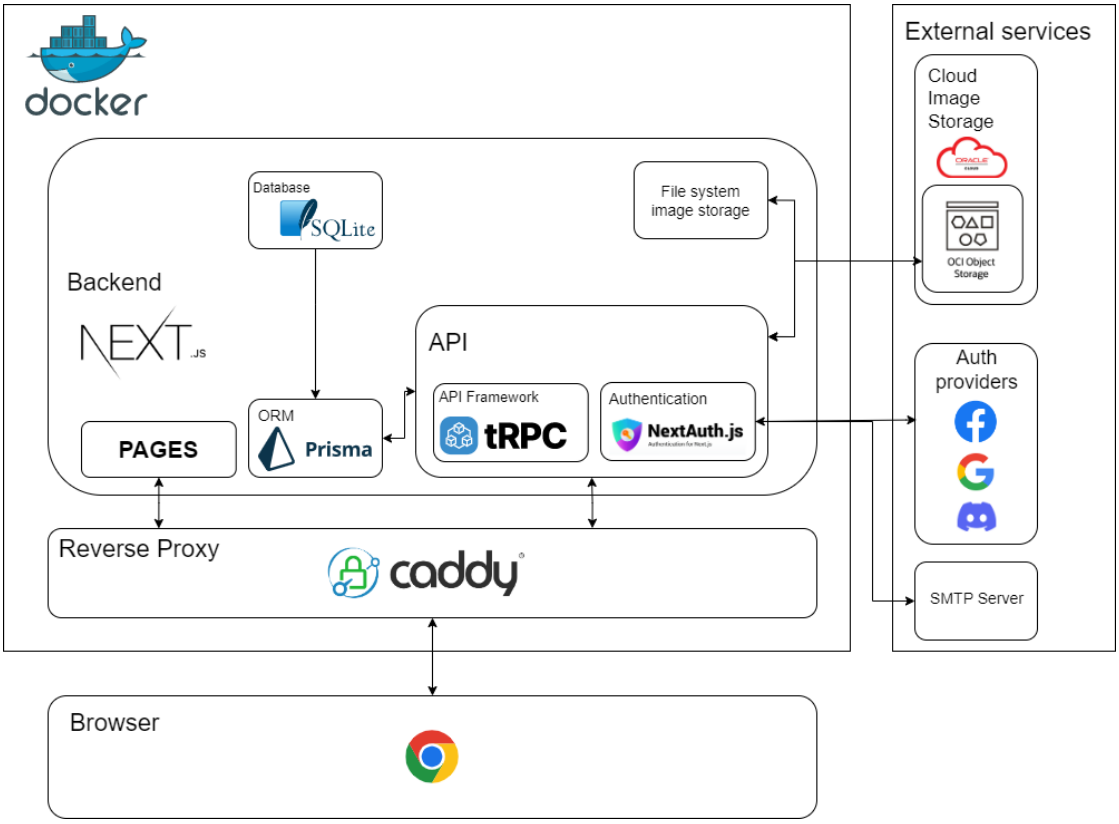


Figure 2.1 Application diagram.

it in sync with the schema, which is used during the build process (section 2.11).

The Prisma library uses the `schema.prisma` to generate TypeScript definitions not only for all the database tables, but also all the operations that are possible to do with these tables using the ORM.

2.5 Authentication

The `create-t3-app` tool set up and configured the NextAuth authentication library for Next.js. The configuration of NextAuth is done in the `server/auth.ts` file. It uses the Prisma adapter to connect to the database. There have been tables set up in the `prisma.schema` according to the NextAuth documentation [9]. This includes tables Account, Session, User and VerificationToken, see section 1.2.

The authentication has four different options for logging in, which are described in subsection 2.6.2.

2.6 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 `create-t3-app` tool sets up a `env.mjs` script, which defines the types of all the environment variables using zod so that they can be validated at the launch of the application.

2.6.1 Database

The database is configured using the `DATABASE_URL` variable. There is not an option to change the database provider, it can only be SQLite. The URL has the following format `file:file-location`[14].

2.6.2 Authentication

NextAuth expects two environment variables to be set up [11]. The first one is `NEXTAUTH_URL`, which is supposed to be set to the canonical URL of the website. The second one is `NEXTAUTH_SECRET`, which is a string used to encrypt and has the JWT, and to hash email verification tokens.

The default NextAuth configuration was also extended to make the authentication providers configurable by environment variables. There are four authentication providers that can be enabled by configuring their respective environment variables.

2.6.2.1 Email

When the email provider is set up, the user can log in using an email. When the user fills in their email, the application sends them a link that will log them into the application. The email provider has two configuration variables. The `EMAIL_SERVER` variable contains the SMTP server, that NextAuth will use to send emails. NextAuth uses the Nodemailer library under the hood to send emails, so the format of the `EMAIL_SERVER` environment variable can be found in their documentation [15]. The `EMAIL_FROM` variable specifies the email address from which the emails will be sent from.

2.6.2.2 OAuth Providers

NextAuth has many built-in OAuth providers [16]. This application has the option to configure three of them, Facebook, Google, or Discord. Each of these providers has a `[PROVIDER]_CLIENT_ID` and a `[PROVIDER]_CLIENT_SECRET` environment variable. Information on how to set up these providers is available in the NextAuth documentation [16]. Any of these providers will be enabled if both of their respective environment variables are set.

2.6.3 Image storage

The application has to save two types of images somewhere, which are the template backgrounds and the generated memes. Two image storage options are offered, either Oracle Cloud Object Storage or local file system storage. If the `USE_OCI` environment variable is `false`, the file system storage will be used. If the variable is `true`, Oracle Cloud will be used.

2.6.3.1 File system

If file system storage is enabled, the images will be stored in the `public/image` folder because the `public` folder is served by the Next.js web server.

2.6.3.2 Oracle Cloud Object Storage

The second option is to use the Oracle Cloud Infrastructure (OCI). The first step to configure OCI was to generate an API key. That was done using the Oracle Cloud Console [17]. The generated API keys were placed into a `.oci` configuration folder, along with a config file, which was provided by the Console and has the following structure:

```
[DEFAULT]
user=...
fingerprint=...
tenancy=...
region=...
key_file=...
```

All the fields except `key_file` were prefilled by the Oracle Cloud Console, and the `key_file` field was filled with the location of the generated API key.

The environment variable configuration has four variables. The first one is `OCI_BUCKET_NAME`, and it specifies the bucket that will be used for the images. `OCI_USER` and `OCI_PASSWORD` are for the credentials that will be used to log in to Oracle Cloud. And finally, the `OCI_CONFIG_PATH` contains the path to the config file in the `.oci` folder mentioned earlier.

2.7 API

2.7.1 tRPC

TRPC is a library that simplifies the development of fully type safe APIs.

2.7.1.1 Alternatives

2.7.1.1.1 Sharing type definitions

2.7.1.1.2 OpenAPI

2.7.2 Image upload

Since tRPC only supports sending JSON, which cannot include files[18], it is impossible to use tRPC to upload images to the backend. For that, the `image-upload` API endpoint was set up in Next.js. The API uses a library called multer, which is a node.js middleware for handling multipart/form-data and can be used for uploading files [19]. Multer can be used with Next.js easily by connecting it with the next-connect library. Next-connect allows the usage of middleware in Next.js API handlers.

Multer expects a configuration object as a parameter at its creation [19]. The application has two different versions of the configuration object for the two different options of storing images. It uses the correct object based on the environmental variable `USE_OCI` described in subsection 2.6.3

The API handler implementation receives a file object as part of the request argument [19], from which it can access the path property with the path to the uploaded file. The path property for the file system implementation has the format of `public/image/[filename]`, but the frontend expects the path to be without the public directory to be resolved correctly by Next.js [20]. The handler uses regular expressions to match if the path string contains the public directory and remove it if it does. There is also a call to `replaceAll` on the path string, where the character is replaced by the `/` character for Windows file systems, which uses different directory separators [21].

2.7.2.1 File system

The file system configuration object has an `engine` property that is made using the `multer.diskStorage` library function. The function takes in a configuration object with the type `multer.DiskStorageOptions`, that has two optional properties, both of which are specified.

The first property is the destination function, which provides a folder where the uploaded files should be stored. For the purposes of this application, all the files are stored in the `public/image` folder.

The second property specifies the filename function, which returns the filename that is used for the file stored on the file system. The application generates a random filename using the uuid library [22] and finds out what the file extension is based on the `mimetype` property on the uploaded file object supplied by multer. It uses the `extension` method from the mime-types library.

2.7.2.2 Oracle Cloud Object Storage

The OCI storage configuration object has an `engine` property that has the type of `multer.StorageEngine`. The storage engine has two required properties, `_handleFile` and `_removeFile`.

2.7.2.2.1 _handleFile

The `_handleFile` function is the function that saves the file to the OCI Object Storage. The file name is created the same way as in subsubsection 2.7.2.1. The function uses the official OCI TypeScript Software Development Kit (SDK) available on GitHub [23]. It uses the `getNamespace` function to get the namespace in Oracle Cloud. Then the `PutObjectRequest` object is constructed. It has four properties, the `namespaceName`, which is returned by the `getNamespace` function, `bucketName`, which is specified by the `OCI_BUCKET_NAME` environment

variable, `putObjectBody`, which is supplied by the file argument from multer and the `objectName`, which is the randomly generated filename with its original extension.

The `PutObjectRequest` is passed to the `oci.putObject` function, which persists the object into the object storage. If the function succeeds, the `path` property is updated with the full url of the saved object and the `fileName` property is updated to the generated filename.

2.7.2.2.2 __removeFile

The `_removeFile` function is called in case an error happens during handling of the `_handleFile` function [19] and it is supposed to remove the object from object storage. Here the approach is very similar to the `_handleFile` function, but instead of the `oci.putObject` function, it uses the `oci.deleteObject` function.

2.7.3 tRPC

TRPC uses a concept of routers to divide API endpoints [24]. The application specifies two routers, `template` and `meme`, which handle the two parts of the communication between frontend and backend.

2.7.3.1 Template router

The template router handles all the communication regarding templates. It contains six procedures.

2.7.3.1.1 createTemplate

The `createTemplate` procedure is used to create templates. It is a protected procedure, so it can exclusively be used by authorized users. The zod definition of the input looks like this:

```
z.object({
    name: z.string(),
    isPublic: z.boolean(),
    fileName: z.string(),
    texts: z.array(
        TemplateTextSchema.omit({ id: true, templateId: true })
    ),
})
```

The `name` property of the template object is its name and `isPublic` specifies, whether the created template should be public. The `fileName` property contains the path to the template background image, which was uploaded using the image upload API endpoint (see subsection 2.7.2). The `texts` property contains the array of texts included in the template. It has the type defined by the database schema (see section 1.2) but with properties `id` and `templateId` omitted.

The procedure persists an image into the database with the `fileName` property, then it persists the template object and finally persists all the texts.

The procedure returns the template object with extra properties being the image and the texts.

2.7.3.1.2 updateTemplate

The `updateTemplate` procedure is used to update templates. It is a protected procedure, so it can exclusively be used by authorized users. The zod definition of the input looks similar to the `createTemplate` procedure:

```
z.object({
    template: TemplateSchema.omit({ imageId: true, userId: true }),
    texts: z.array(
        TemplateTextSchema.omit({ id: true, templateId: true })
    ),
})
```

It contains the `template` property with `imageId` and `userId` omitted because these will be filled by the backend. The `texts` property is exactly the same.

First, the procedure retrieves the template that it is updating by the `template.id` property from the database. If the template doesn't exist or the user who created the template is different from the currently logged-in user, then the procedure returns an error.

Then, all the texts that belong to the original template are all deleted and the template is updated. Because the application does not support changing an existing template's background, the `imageId` of the updated template is kept the same. After the template is updated, all texts from the input are persisted again.

The procedure returns the updated template with the updated texts.

2.7.3.1.3 deleteTemplate

The `deleteTemplate` procedure is used to delete templates. It is a protected procedure, so it can exclusively be used by authorized users. The input of this procedure is an object that has one property, `id` of type `string`.

First, the procedure retrieves the template that is supposed to be deleting. It checks, that the template exists and that the user who is deleting it is the one who created it. If not, the procedure returns an error.

Then, the texts, template, and image are deleted. The procedure does not handle the deletion of the image from the image storage.

2.7.3.1.4 getTemplate

The `getTemplate` procedure is used to get a single meme template. The input for this procedure is a string with the id of the template that is supposed to be retrieved. The template can only be retrieved if its `isPublic` property is true or if the template was created by the user who is accessing it; otherwise, the procedure returns an error. The return type of the procedure is the template with the image and texts field.

2.7.3.1.5 getInfinateMemeTemplates

The `getInfinateTemplates` procedure is used to get a variable number of meme templates. Infinite queries are a concept from react-query [25]. It uses a cursor-based pagination, where each query returns the items it finds and also a `nextCursor` property, which is an id of the item that would come after these, which can be used to get the next page of items. The input object looks like this:

```
z.object({
    fetch: z.enum(['fromUser', 'public', 'all']),
    limit: z.number().min(1).max(100),
    cursor: z.string().nullish(),
    searchString: z.string().optional(),
})
```

The `cursor` property is the id of the object that the query should start from. The Prisma database client supports cursor pagination [26], which means the cursor from the input can be directly passed to Prisma.

The `limit` property tells the query how many results it wants to fetch. It has to be between the number 1 and 100.

The `searchString` property is used for searching, and it is optional. If it is provided, the results will be filtered by the search string. To make the search more useful, the search is split into words (by using the space character) and each word is matched separately.

The `fetch` property which can be one of three values, `"fromUser"`, `"public"` or `"all"`. If the property is set to `"fromUser"`, only the templates that belong to the currently logged-in user will be returned. If the property is set to `"public"`, only the public templates will be returned. If the property is set to `"all"`, it is the same as if `"fromUser"` and `"public"` were set simultaneously.

2.7.3.2 Meme router

Meme router contains all the communication regarding memes. It has similar procedures to the template router.

2.7.3.2.1 createMeme

The `createMeme` procedure is the procedure that allows creating meme. It is a protected procedure. It takes an object as an input with a single property, `fileName`. The `fileName` property is the path to the image of the meme which was uploaded using the image upload API endpoint. The procedure persists the image path into the database and then persists the meme. It returns the meme with an extra property, `imagePath`, which contains the path to the image.

2.7.3.2.2 deleteMeme

The `deleteMeme` procedure is the procedure that allows creating meme. It is a protected procedure. The input for this procedure is an object with a single property, `id`, which is the id of the meme that is supposed to be deleted. If the meme does not exist, or it was not created by the user who is trying to delete it, the procedure returns an error.

The procedure deletes the image and the meme from the database, but does not delete the image from the image storage.

2.7.3.2.3 getInfiniteMemes

The `getInfiniteMemes` works very similar to the `getInfinateMemeTemplates` procedure defined in paragraph 2.7.3.1.5. It gets a list of memes with a given limit and cursor from the database and also returns the `nextCursor` property.

2.8 State

State Management on the frontend is one of the problems that the t3 stack doesn't solve. There are many libraries for the React framework that offer some form of state management solutions, from elementary libraries to complex libraries solving multitudes of problems. The most common library is Redux. All the libraries have some version of the conceptually same building blocks, which are State, Actions, Reducers and Effects.

The main idea of these libraries is to create a single object, that represents the whole state of the application. This object is always immutable, which means that the object cannot be modified in after it is created.

The state object and its actions are used to encapsulate all the ways of how the state can be modified. State can only be changed inside actions. All the components of the application can only read the state and call actions.

This section also describes all the objects and actions that the state has, which have been grouped by their functionality.

2.8.1 Upload Image Helper

The `uploadImageHelper` function is a function, that takes an image as a parameter and uploads that image using the image upload API defined in subsection 2.7.2. It creates a new `FormData` object, which can be used to construct a set of key/values pairs representing a form [27]. The image is added to this set and then the `FormData` object is passed to the fetch function, which calls the image upload API endpoint. The result is parsed into JSON and checked, whether it contains the path property, which would be the path to the uploaded image. This path is returned, otherwise, an error occurs.

2.8.2 Background State

The `background` object has a type of `Blob | null` and contains the background image of the template. The `Blob` type represents a blob, which is a file-like object of raw data [28]. If no background has been selected, `null` is used.

There is also a corresponding `setBackground` function of type `(img: Blob) => void`, which takes a `Blob` and sets the `background` variable to the argument.

2.8.3 Popup State

Popups are used to inform the user about various events that are happening in the background. When a popup is created, it shows up in the bottom-right corner of the page. A popup has the following type:

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

The `PopupState` type definition looks like this:

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

2.8.3.1 popupCounter

This is used to assign ids to individual popups. Each time a new popup is created, this counter is incremented by one to guarantee uniqueness.

2.8.3.2 popups

The `popups` variable holds the individual popups in an array.

2.8.3.3 removePopup

The `removePopup` function gets an id of the popup that is supposed to get removed as a parameter and removes it from the state.

2.8.3.4 addPopup

The `addPopup` handles the assignment of the id and the incrementing of the `popupCounter` variable, the id can be omited from the argument to the function. The function also handles the removal of the popup from the popups array after a period given by the constant `POPUP_TIMEOUT`. It uses the `setInterval` function where the callback uses the `removePopup` function.

2.8.4 Dialog State

The application contains three dialogs, which can be either open or closed. A dialog is of the following type

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

The `DialogState` type looks like this:

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

For each dialog type, there is a `boolean` variable that determines whether the dialog is open or not. There is also a `setDialog` function, which, when called, sets the dialog passed as the first argument to the state passed as the second argument.

2.8.5 Text State

The text state has all the behavior related to changeable texts, that are a part of the template. It has the following type:

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

The `FTemplateText` type is defined as foolows, where `TemplateText` is a type generated from the database schema, section 1.2

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

The type of the `FTemplateText` (F stands for frontend) specifies, that each text has every property from the database schema, but properties `id` and `templateId` are optional. Properties `id` and `templateId` are generated when the text is saved to the database on the backend, so they are not known at the time of text creation, so they have to be optional.

2.8.5.1 addText

The `addText` function creates an empty text and adds it to the `texts` object under a key that is randomly generated using the `uuid` library.

2.8.5.2 deleteText

The `deleteText` deletes the text at the key passed as the argument.

2.8.5.3 updateText

The `updateText` replaces the text at the key passed as the first argument with the new text passed as the second argument.

2.8.6 Template State

The template state stores the information about the currently edited template. It has the following type:

```
type TemplateState = {
    template: Pick<Template, 'isPublic' | 'name'> & Partial<Template>
    updateTemplate: (updates: Partial<Template>) => void
    useTemplate: (id: string) => Promise<void>
    saveTemplate: (overwrite: boolean) => void
}
```

2.8.6.1 template

The `template` property stores the currently edited template. It is based on the `Template` type generated from the database schema described in section 1.2. It has the `isPublic` and `name` properties as required, and all the other properties from the `Template` type are optional.

2.8.6.2 updateTemplate

The `updateTemplate` function is used to update the template. It takes in a partial `Template` object as an argument and sets them to the state.

2.8.6.3 useTemplate

The `useTemplate` function is for selecting a template for editing. It gets the id of the template as an argument. First, the function gets the template using the `getMemeTemplate` procedure. Then it uses the `getImageFromUrl` method to download the image file. Then it sets the `template` property to the template received from the `getMemeTemplate` procedure, sets the `texts` property to the `texts` property from the received template and updates the `background` property to the image downloaded using `getImageFromUrl`.

2.8.6.4 saveTemplate

The `saveTemplate` function is used to save templates. The function has two main branches, one if the user decided to overwrite the template, and the other one if the template is considered a new template.

2.8.6.4.1 Overwrite is true

If the overwrite parameter is true, the `template.id` property has to be set, otherwise overwriting does not make sense. This if branch of the function calls the `updateTemplate` API procedure with the contents of the `template` and `texts` properties. There is also a popup on success and a popup on error.

2.8.6.4.2 Overwrite is false

If the template should not be overwritten, the function uses the `uploadImageHelper` function described in subsection 2.8.1 to upload the image using the image upload API endpoint. Then, it calls the `createTemplate` API procedure with the following properties, `fileName`, which is the path returned from the `uploadImageHelper` function, and the `texts`, `name` and `isPublic` properties, which are a part of the `template` object from the state.

2.8.7 Meme State

The meme state is the part of the state which contains behaviors related to exporting memes. It has the following type definition:

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

2.8.7.1 exportedMeme

The `exportedMeme` property contains the URL of an exported meme. If no meme has been exported, the value of the property is `null`.

2.8.7.2 setExportedMeme

The `setExportedMeme` function sets the `exportedMeme` property to the value of its argument.

2.8.7.3 exportMeme

The `exportMeme` function is used to export a meme. The `memeFile` argument is the file with the meme that is supposed to be exported. The behavior of this function is different, based on whether a user is logged in or not, which is indicated by the authenticated parameter.

2.8.7.3.1 User is logged in

If the user is logged in, the function uses the `uploadImageHelper` function to upload the image using the upload image API endpoint. Then it uses the `createMeme` procedure to create a meme, passing in the path returned from the `uploadImageHelper` as the `fileName` argument. The successfully created meme is then set to the `exportedMeme` property in the state.

2.8.7.3.2 User is not logged in

If the user is not logged in, the function sets the `exportedMeme` property to a URL to the passed in `memeFile` argument, using the `URL.createObjectURL` function [29].

2.8.8 Persisting state

The application needs to somehow persist state so that the user can refresh the page without losing progress with the edited templates. Zustand provides a `persist` middleware, which can be attached to the store object. The `persist` middleware has multiple options of where to store state and how to serialize it [30]. The application serializes the state to JavaScript Object Notation (JSON) and saves it to session storage. Because the background image property cannot easily be converted to JSON, it is excluded from persistence, along with `popups` and `popupCounter`, which are not needed after page reload.

2.9 Styles

Any web application requires Cascading Style Sheet (CSS) to look different from just black text on a white background. There have been many developments in how developers make styled components using CSS. One of the most promising approaches in my opinion is a library called tailwind[1]. It uses an approach of utility classes, which is very convenient to use. Instead of writing CSS, the developer adds utility classes directly to .

This application is using the daisyUI[2] library to get access to styled components made using the tailwind library. DaisyUI looks very similar to tailwind, but it contains more functionality bundled together.

2.10 Next.js

Next.js is an open-source JavaScript web framework for React that ships with a rich set of features out of the box, such as server-side rendering, static site generation, and incremental static regeneration [31]. Next.js is the entry point of the application. It provides the scripts to build the application and to run the local development server. The main building block of a Next.js application are pages, which are stored in the `pages` folder. Next.js uses filesystem-based routing, which means that every file in the `pages` folder corresponds to a route accessible from the browser.

2.10.1 The API routes

There are two directories with API routes generated by the `create-t3-app` tool. These API routes are used for tRPC and for NextAuth.

Then there is the image-upload file with the image upload API route, which is described in subsection 2.7.2.

2.10.2 Common Components

Some components are used throughout the whole application, so they are placed inside the `common` folder.

2.10.2.1 Alert Component

The `Alert` component is a wrapper for the daisyUI alert component. The alert has four variations, `info`, which is blue, `success`, which is green, `warning`, which is yellow and `error`, which is red.

[1] https://tailwindcss.com
[2] https://daisyui.com

The component gets its desired variation through props. There is also an icon for each variation, which is in the form of a SVG.

2.10.2.2 MemeView Component

The `MemeView` component is a dialog with the details of a meme. It gets two props passed in, `src` and `canCopy`.

The `src` property contains the URL to the image if the meme that it is displaying.

The `canCopy` property determines whether the URL to the image should be displayed with the copy button next to it.

2.10.2.3 Popup Component

The `popup` component displays the popup. It uses the `removePopup` function from the application state to enable the functionality of clicking the close button.

2.10.2.4 TemplateCard Component

The `templateCard` component is the template card, that is shown on the `Create Page` or the `All Templates Page`. If the template card is clicked, it calls the `useTemplate` function of the application state. It has five properties of in the props.

2.10.2.4.1 editable

`editable` specifies, whether the delete button should be displayed.

2.10.2.4.2 selected

The `selected` property is used on the `Create Page` to indicate, that this is the currently selected template. It adds a colorful glow to the template card.

2.10.2.4.3 size

The `size` property controls whether the big version or the small version of the component gets displayed.

2.10.2.4.4 template

The `template` property contains the template that is being displayed.

2.10.2.4.5 refetch

The `refetch` property is used to indicate to the parent component, that the template card got deleted, so that the parent component can update the list of templates.

2.10.3 Layout

The layout component specifies the main layout of the application. It sets a navbar to the top of the page and a footer to the bottom. The footer will always be at the very bottom of the page, even if there is not enough content to fill the area between the navbar and the footer.

The layout component also contains the popups because these need to be present on all other pages and are independent of other page's behavior.

2.10.3.1 Navbar

Navbar is the top menu of the application. It contains the main navigation component with links to all the pages. It also contains the login button in case a user is not logged in, or it displays the profile picture of the currently logged-in user. The profile picture has a dropdown menu that has a button to log out.

2.10.3.2 Footer

The footer is the bottom section of the page. It contains the copyright information as well as relevant links for the project.

2.10.4 Pages

2.10.4.1 index.tsx

The index page contains the `Create Page`. Because the page contains dynamically loaded content, like the selected template, it cannot be statically generated nor server side rendered. The `next/dynamic` function from the Next.js library is used to get around this issue by loading the `components/create/create` component only on the client.

The `Create Component` is the most complicated component. At the top, it contains a list of template cards. Then there is the TemplateCreate component, which encapsulates the template creation. That is then divided into four main building blocks.

2.10.4.1.1 Canvas

The `Canvas` component contains all the functionality of the graphical editor. It uses the react-konva library, which uses the konva[3] library under the hood. The konva library is a wrapper of the canvas API exposed by the browser [32].

The `Canvas` component first sets up a stage, within which it sets up a layer. It uses the `useHtmlImage` hook to transform the background that is present in the state into a format that is acceptable by the konva `Image` component, which renders it onto the canvas.

Then, each text from the state gets transformed to a `CanvasText` component, which renderes texts.

The `CanvasText` component uses the konva `Text` component to render a text into the background. The `Text` component also gets a `Transformer` component attached, which handles the moving, resizing and rotating of the text element.

2.10.4.1.2 Properties

The `Properties` component has the extra properties of texts behavior. Like `Canvas`, it has a corresponding `PropertiesText` component, which renders a single text.

It also contains the `Save Template` and the `Export meme` buttons. There is also an alert which notifies an anonymous user, that they cannot save templates unless logged in.

2.10.4.1.3 TemplateSaveDialog

The `TemplateSaveDialog` renders the dialog to save a template. It relies on the `updateTemplate` function of the application state to set properties like `isPublic` or `name`. When the user confirms the dialog, it calls the `saveTemplate` function.

[3] https://konvajs.org

2.10.4.1.4 ImportDialog

The `ImportDialog` contains all the functionality required to import a custom background for the template. There are two different methods of uploading an image.

The first method uses the URL input, where the user can pass an URL to an image that is available on the internet. There is also a possibility to paste an image directly from the clipboard when the user has the input focused.

The second method is to use the upload section. It has two ways of interacting, either the user can drag and drop a file directly from their file system which is done by listening to the `onDragEnter`, `onDragLeave`, `onDragOver`, and `onDrop` events, or click the area and get a normal upload file popup, which is handled by the `onChange` event.

When an image is uploaded, it appears under the inputs. Here the user can crop the image, which is done using the react-advanced-cropper[4] library.

2.10.4.2 all-templates.tsx

The `All Templates` page contains a list of all the templates. It uses the `TemplateList` component, which uses the `TemplateCard` component to render individual templates on the page. The template cards are not editable on this page, so the `editable` parameter is set to `false`. The `fetch` parameter is set to `"all"`. It uses the `getInfiniteTemplates` Infinite Query [25] to fetch templates.

2.10.4.3 my-templates.tsx

The `My Templates` uses the same `TemplateList` component as the `All Templates` page. The setup is the same, but the `"fetch"` parameter is set to `"fromUser"` and the `editable` parameter is set to `true`.

2.10.4.4 my-memes.tsx

The `My Memes` page contains all the logged in user's memes. It uses the `MemeList` component. It uses the `getInfiniteMemes` Infinite Query [25] to fetch memes. There is also a `MemeCard` component, which is very similar to the `TemplateCard` component.

The `MemeCard` component shows the image of the template. If the user clicks the meme card, the `MemeView` component is displayed with the clicked meme. It also has a delete button, which calls the `refetch` method, same as in the `TemplateCard` component.

2.11 Build

This section is about building and compilation. The application is built into a Docker image using GitHub Actions[5]. The image is pushed to the Docker Hub[6].

2.11.1 Docker

Docker is used to make bundle the application into a docker image, which can be launched in any environment that supports docker. This image is uploaded to the Docker hub image repository. Docker uses what is called a Dockerfile to build images, which is a step-by-step instruction set on how to build the image. The Dockerfile is described in subsubsection 2.11.1.1

[4] https://advanced-cropper.github.io/react-advanced-cropper/
[5] https://github.com/features/actions
[6] https://hub.docker.com

2.11.1.1 Dockerfile

The Dockerfile is a part of the implementation folder and describes how the image containing the application is built [33]. The Dockerfile used for this thesis is a modified Dockerfile from the official Next.js GitHub repository [7].

The building process utilizes an optimization called a multi-stage build, which reduces the size of the images substantially [34]. The build process is split between multiple images, and only what is required is copied between the stages. The first stage of the build is the dependencies stage, where all the dependencies required for the build are installed. The second stage is the build stage, where the Next.js application is built. This step also creates and seeds the database that is included in the docker image. The last stage is the run stage, where the image that will actually run the application is set up. All the required files are copied and permissions are set up with a new user and a new group.

2.11.2 GitHub actions

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. Developers can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production [35].

All the workflows of the application are specified in the `.github/workflows` folder. There is only one workflow specified, which uses the defined Dockerfile and builds the application into a Docker image, which it then pushes to the Docker Hub. The workflow is triggered on every push to the main branch.

The workflow contains one job with four steps.

2.11.2.1 Checkout

The first step uses the checkout action, which checks out the repository on the runner, so that all the code is available to use for the next steps.

2.11.2.2 Login to Docker Hub

The second step uses the login-action[8] which is published int the Docker team's GitHub repository. This action logs the user into Docker Hub using a username and password. These are saved as Encrypted Secrets and are a part of the GitHub repository. The secrets are encrypted by GitHub to increase security.

2.11.2.3 Metadata

After the login step, the third step accesses the metadata using the metadata-action[9] from the Docker team's GitHub repository, such as tags and labels, for the Docker image of the thesis from Docker hub. This metadata is then used for the next action.

`Build and Publish` The final action, the build-publish-action[10] from the Docker team's GitHub repository, builds and pushes the Docker image. It has four inputs specified. Tags and labels are passed from the previous steps. It needs to have the context specified, which is not only the location of the Dockerfile but also the working directory in which then Dockerfile is built. After the image is built, the action pushes the image to Docker Hub, where it is accessible for the Deployment process.

[7] https://github.com/vercel/next.js
[8] https://github.com/docker/login-action
[9] https://github.com/docker/metadata-action
[10] https://github.com/docker/build-push-action

```
// version of the docker-compose
version: '3.1'
// the docker containers
services:
//caddy is used as the reverse proxy
  caddy:
    container_name: memebro-caddy
    image: caddy:latest
    volumes:
    // map the Caddyfile from the host file system into the docker container
      - './Caddyfile:/etc/caddy/Caddyfile'
    // expose ports 80 and 443 for http/https communication
    ports:
      - '80:80'
      - '443:443'
    // containers need to be on the same network to see each other
    networks:
      - main
  // the application container
  app:
    container_name: memebro-app
    // use the image published to Docker hub
    image: docker.io/ozavodny/masters-thesis:main
    networks:
      - main
    volumes:
    // mount the .oci configuration directory
    // to enable Oracle Cloud Image Storage
      - './.oci:/app/.oci'
    environment:
    // set environment variables
      - DATABASE_URL="file:./db.sqlite"
      ...
networks:
  main:
```

Figure 2.2 The docker-compose.yml file

2.12 Deployment

The section is about the deployment of the application. It is deployed on a VM running the Debian operating system. There is a `docker-compose.yml` file to help with the deployment of the individual docker containers. Docker-compose specifies the structure of deployed docker containers and their properties. The `docker-compose.yml` file is shown with comments in the listing ??. The `Caddyfile` required for the reverse proxy to work is displayed in listing ??.

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

Figure 2.3 The Caddyfile for the reverse proxy.

Chapter 3
Testing

3.1 End-to-end testing using cypress

Cypress is a testing framework which offers e2e testing using a browser. The application has a single test setup which follows the following test scenario.

3.1.1 Test scenario

- Click the login button in the upper-right corner
- Type in your email
- Click submit
- Find the email that the application has sent and click on the login link
- Click the `Import custom background` button
- Input an URL to your favorite meme background
- Click `Use` and then `Confirm`
- Click the `Add Text` button
- Fill in some text on the right
- Click the `Save Template` button
- Fill in some template name
- Click the `Save` button
- A popup with the text `Meme template saved.` should appear.

3.1.2 Mocking the SMTP server

Because it is difficult for cypress to access an email inbox, the application connects to a mocked Simple Mail Transfer Protocol (SMTP) server launched by cypress. This is done using the ts-smtp-test library. Cypress then has access to the received email via the `getLastEmail` task.

3.2 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.

3.2.1 Criteria for participants

Users should have at least a general understanding of memes.

3.2.2 Participants

- 24-year-old student – Technical
- 25-year-old person – Technical
- 50-year-old person – Non-Technical

3.2.3 Testing Setup

Participants were given 30 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.

3.2.4 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. Create a new template with a new background picture 5. Create a meme from the new template 6. Delete a created meme from memory 7. Change the new template

3.2.5 Results

3.2.5.1 Task 1

All users were able to complete this task

3.2.5.2 Task 2

All users were able to complete this task

3.2.5.3 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

3.2.5.4 Task 4

All users were able to complete this task. User 3 had a hard time finding the feature and had to be helped after asking

3.2.5.5 Task 5

All users were able to complete this task

3.2.5.6 Task 6

All users were able to complete this task

3.2.5.7 Task 7

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

3.2.6 Found problems

The application lagged from time to time.

3.2.7 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.

3.2.8 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.

3.2.9 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 4 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.

3.2.10 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

The goal of this thesis was to look into how people create memes on the internet and create a new solution with the desired functionalities.

The first chapter of the thesis is denoted to analysis, which covers the evaluation of existing solutions to this problem, and designing a high-level overview of a possible custom solution.

The second chapter describes all the parts of a developed application, how they were made, and how they communicate together. It focuses on the graphical elements as well as the functional elements of modern web development. Many interesting libraries were evaluated and most of them ended up being used in the implementation. The resulting application does not have as many features as the other evaluated solutions, but it delivers the wanted results with simplicity. The last part of this chapter described how the application is built and how it is deployed into production environment.

The last chapter was denoted to testing. A modern end-to-end testing framework was used to smoke test the basic functionality of the application. The chapter also includes a detailed analysis of a usability testing session, which was conducted with actual potential users of the application.

The implementation accomplished all the goals set forth by the assignment, but there are areas that could be improved. There are many features of other existing solutions that were described in the analysis chapter, which could be implemented into this application.

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]. 2023-04-20. [visited on 2023-04-20]. Available from: `https://filmora.wondershare.com/meme/best-free-meme-maker-online.html`.
3. TRENDS.GOOGLE.COM. *Google Trends* [online]. 2023-02-21. [visited on 2023-02-21]. Available from: `https://trends.google.com/trends`.
4. 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`.
5. 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`.
6. LLC, Imgflip. *Meme Generator* [online]. 2023. [visited on 2023-03-27]. Available from: `https://imgflip.com/memegenerator`.
7. ILOVEIMG. *Our features* [online]. 2023. [visited on 2023-03-22]. Available from: `https://www.iloveimg.com/features`.
8. ILOVEIMG. *Compare plan features* [online]. 2023. [visited on 2023-03-22]. Available from: `https://www.iloveimg.com/pricing`.
9. ORBÁN, Balázs. *@next-auth/prisma-adapter* [online]. [N.d.]. [visited on 2023-04-29]. Available from: `https://authjs.dev/reference/adapter/prisma`.
10. HAMM, Matthew. *Wireframing Essentials*. PACKT, 2014. ISBN 1849698546.
11. ORBÁN, Balázs. *Options* [online]. 2023. [visited on 2023-04-29]. Available from: `https://next-auth.js.org/configuration/options`.
12. UZAYR, Sufyan bin. *TypeScript for Beginners: The Ultimate Guide*. 1st ed. CRC Press, 2022. ISBN 1032067586, ISBN 9781032067582.
13. ORZACH, Yoram. *Network Protocols for Security Professionals: Probe and identify network-based vulnerabilities and safeguard against network protocol breaches*. Packt Publishing - ebooks Account, 2022. ISBN 1789953480, ISBN 9781789953480.
14. PRISMA DATA, INC.. *SQLite* [online]. 2023. [visited on 2023-04-21]. Available from: `https://www.prisma.io/docs/concepts/database-connectors/sqlite`.
15. REINMAN, Andris. *SMTP TRANSPORT* [online]. 2023. Available also from: `https://nodemailer.com/smtp/`.

16. ORBÁN, Balázs. *OAuth* [online]. 2023. [visited on 2023-04-23]. Available from: https://next-auth.js.org/configuration/providers/oauth.
17. ORACLE. *Required Keys and OCIDs* [online]. 2023. [visited on 2023-04-15]. Available from: https://docs.oracle.com/en-us/iaas/Content/API/Concepts/apisigningkey.htm.
18. GITHUB, INC. *Is there a way to handle multipart/form-data requests?* [Online]. 2023. [visited on 2023-03-21]. Available from: https://github.com/trpc/trpc/discussions/658.
19. YAAPA, Hage. *Multer* [online]. 2023. [visited on 2023-04-01]. Available from: https://www.npmjs.com/package/multer.
20. VERCEL, INC. *Static File Serving* [online]. 2023. [visited on 2023-04-12]. Available from: https://nextjs.org/docs/basic-features/static-file-serving.
21. GILLES 'SO- STOP BEING EVIL' HTTPS://SUPERUSER.COM/USERS/42315/GILLES-SO-STOP-BEING-EVIL. *Why does Windows use backslashes for paths and Unix forward slashes?* [Online]. 2023. [visited on 2023-04-12]. Available from: https://nextjs.org/docs/basic-features/static-file-serving.
22. KIEFFER, Robert. *uuid* [online]. 2023. Available also from: https://www.npmjs.com/package/uuid.
23. ORACLE. *Oracle Cloud Infrastructure SDK for TypeScript and JavaScript* [online]. 2023. [visited on 2023-04-13]. Available from: https://github.com/oracle/oci-typescript-sdk.
24. JOHANSSON, Alex. *tRPC* [online]. 2023. [visited on 2023-04-13]. Available from: https://trpc.io/docs.
25. LINSLEY, Tanner. *Infinite Queries* [online]. 2023. [visited on 2023-04-15]. Available from: https://tanstack.com/query/latest/docs/react/guides/infinite-queries.
26. PRISMA DATA, INC. *Pagination* [online]. 2023. [visited on 2023-04-16]. Available from: https://www.prisma.io/docs/concepts/components/prisma-client/pagination.
27. MOZILLA FOUNDATION. *FormData* [online]. 2023. [visited on 2023-04-16]. Available from: https://developer.mozilla.org/en-US/docs/Web/API/FormData.
28. MOZILLA FOUNDATION. *Blob* [online]. 2023. [visited on 2023-04-28]. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Blob.
29. MOZILLA FOUNDATION. *URL: createObjectURL() static method* [online]. 2023. [visited on 2023-04-29]. Available from: https://developer.mozilla.org/en-US/docs/Web/API/URL/createObjectURL_static.
30. HENSCHEL, Paul. *SQLite* [online]. 2023. [visited on 2023-04-20]. Available from: https://github.com/pmndrs/zustand.
31. 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.
32. LAVRENOV, Anton. *Getting started with react and canvas via Konva* [online]. [N.d.]. [visited on 2023-04-22]. Available from: https://konvajs.org/docs/react/index.html.
33. DOCKER INC. *Dockerfile Reference* [online]. 2023. [visited on 2023-04-21]. Available from: https://docs.docker.com/engine/reference/builder/.
34. GHOSH, Saibal. *Docker Demystified : Learn How to Develop and Deploy Applications Using Docker*. Draft2Digital, 2021. ISBN 9789389845877. Available also from: http://gen.lib.rus.ec/book/index.php?md5=ACBA7471BE582BAC13BFB39907DDF446.
35. GITHUB, INC. *GitHub Actions Documentation* [online]. 2023. [visited on 2023-04-23]. Available from: https://docs.github.com/en/actions/.

Obsah přiloženého média

```
masters
├── impl ..... zdrojové kódy implementace
└── thesis.pdf ..... text práce ve formátu PDF
```