MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Journaling Journey Web Application

License Thesis

Graduate Student
Hreniuc Paula

Supervisor
Supervisor: Assist. Eng. Cosmina IVAN

June 2020

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN OF FACULTY	HEAD OF DEPARTMENT
Prof. dr. ing. Liviu MICLEA	Prof. dr. ing. Rodica POTOLEA

Journaling Journey Web Application

License Thesis

1. **Graduate Student**: Hreniuc Paula
2. **Supervisor**: Supervisor: Assist. Eng. Cosmina IVAN
3. **Thesis contents**: Thesis presentation, supervisior evaluation, chapter 1, chapter 2, . . . , chapter n, References, Anexes, CD.
4. **Documentation place**: UTCN, Cluj-Napoca
5. **Thesis specifications' date**: . . . .
6. **Thesis termination' date**: . . . .

Signature of the Supervisor	Signature of the Graduate Student
Supervisor: Assist. Eng. Cosmina IVAN	Hreniuc Paula

June 2020

S

Author's Declaration Regarding the Autenticity of the License thesis

Subsemnatul *Hreniuc Paula*, legitimat cu *CI/BI* seria *XX* numărul *NNNNNN*, CNP *LLLLLLLLLLLL*, autorul lucrării *Journaling Journey Web Application* elaborată în vederea susținerii examenului de finalizare a studiilor de masterat la Facultatea de Automatică și Calculatoare, Departamentul Calculatoare, Specializarea *SSSSSSSS* din cadrul Universității Tehnice din Cluj-Napoca, sesiunea *June* a anului univeristar *20XX/20XX*, declar pe proprie răspundere, că această lucrare este rezultatul propriei mele activități intelectuale, pe baza cercetărilor mele și pe baza informatiilor obținute din surse care au fost citate în textul lucrării și în bibliografie.

Declar că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență sau disertație.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Cluj-Napoca
date

Signature of the Graduate Student
Hreniuc Paula

Abstract

Descrierea sumară a lucrării, în câteva fraze. Un site foarte util ce conține exemple LaTeXse găsește la `http://en.wikibooks.org/wiki/LaTeX`. Recomandăm crearea unui proiect în editoare Latex specializate (exemplu Kile pe Linux, TexnicCenter pe Windows) astfel încât compilarea/translatarea codului Latex în PDF să se facă din orice fișier editați la un moment dat (practic se va compila proiectul). De asemenea, dacă fișierul “thesis.bib” este inclus în proiect, există facilitatea de code-completion și la cite.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We are living some challenging times. As everyone faced staying at home for prolonged periods of time these days, constrained by the current pandemic, people were forced to find new ways to entertain themselves. They pursue new hobbies if they never had one, people draw, cook, talk and write, all for the sake of staying sane. The digital world has never been more active than in the last months. User interface matters a lot and makes the difference, it has the power to give the user a pleasant felling or the reverse of that. User interfaces and cool features make the user want to come back and use a platform that excels in these two areas, it has to be something that entices them, that offers them easy access to everything trending and creates the space for their creative ideas to take shape. These times allowed us all to reflect and ponder upon our own lives. Journaling helps people to free up their mind and they also want to save their ideas, tasks, plans, dreams and goals.

My personal motivation to create a journaling application started when i realised that a platform that would allow me not only to write text but also to add stickers, GIFs and YouTube videos on my journaling page, did not exist on the current market. I have always wanted to bring the creative ambitions of the world together with the more technical part.

Chapter 2 Project's Objectives and Specification presents, as the title of the chapter suggests, the objectives of the application at hand, also taking into consideration the quality aspects.

Chapter 3, called Bibliographic Study, talks about the concept of journaling it's advantages. Wikis, Blogging will also be described and analysed as those have been the roadmap to to journaling platforms as well. The role of journaling and its benefits will also be mentioned and also the elements of newness will be presented along with the reason they are so used and appreciated in todays's communication. These elements refer to Stickers, GIFs and Youtube videos.Similar systems present on the market will be analysed and compared looking at their features and design.

Chapter 4, Theoretical Background, presents a generic view over the problem statement, user needs and the system's main concept. The technologies used are described and the reason why they were chosen along with the functional and non-functional requirements. The high level view of the system will be provided through the conceptual architecture diagram.

In chapter 5 the more detailed implementation of the system is presented. Important pieces of code will be broken down and explained to give a better understanding of the system created and the way it works and functions. Also the front-end and back-end system architectures will be described. Important components will also be discussed and their importance.

Chapter 6 will focus on the way the application was tested and the results the user testing I organised in order to gain some feedback on the way the user interacts with the system created.

Chapter 7 presents the software and hardware resources needed in order to set up the system. Installation steps will be provided and also the navigation and the way the application works will be presented in this chapter.

In the chapter 8 we will make an overview of the all the results obtained, accomplishments, decision made that stood as a base when creating the project and the difficulties encountered. Also possible improvements will be mentioned here, how the application could be improved, what features could be added in future releases and what could have been done better.

Chapter 2

Project's Objectives

This chapter presents a general overview of the projects's functional and quality objectives.

The aim of this project is to create a web application that enhances the journaling experience. There are numerous applications already on the market that help users save their notes into a personal library and style their content but I have not found one that lets you integrate elements such as Youtube videos, GIFs or stickers onto the journaling page. Also, I have not found audio rendition of your own notes, which is something I wish to implement. Journaling in its traditional form does not offer you any of these features but neither do the digital platforms i looked up. My desire is to bring these elements together in this project.

The current application wishes to enable the user to add these elements of newness mentioned in the previous section. I always wished i could integrate those elements in my own writings and journaling pages as journaling is an activity I still use to practice. The current application will allow users to authenticated with Google and to start creating journaling pages for the current date. All journaling entries will be saved on the user's dashboard. Onto a journaling page, different elements can be added such as: YouTube videos, GIFs, stickers, images, text. The journaling pages can be printed out and also audio played aiming to offer the user a better reflect on his own ideas and thoughts.

2.1 Main Objectives

The application will :

- allow the user to authenticate with Google
- allow the user to create journaling pages

- allow the user to add GIFs, Stickers, YouTube video and images to the journaling page
- allow the user to audio rendition the text of his journaling pages
- be available as a web application
- be based on the client-server model
- be implemented in Javascript for the front-end part and java on the back-end
- will use MySQL as database to helps with storing the users content
- will make use of several APIs for fetching the desired assets
- will make use of React and Springboot

2.2 Quality Objectives

The following quality attributes will have to be taken into consideration when creating the application:

- to have an intuitive interface
- security of the user's personal data
- the responsiveness and performance of the user's request will also have to be taken into consideration

Chapter 3

Bibliographic Study

Chapter three focuses on similar systems that came as an inspiration to this project. What stood at the base of my desire to create such a web application, what is journaling and why it is important. Also I will mention what I saw as a limitation in the current systems available on the market. I will point out my opinion of why visual assets are of so much importance nowadays and reinforce my point by mentioning the work and of some studies of other researchers.

The following studies and articles represent the proof that people are interested and looking for means to express themselves in a more accurate way more than ever in this time and age.

In the first section 3.1 the difference between Wikis, Blogs and Journaling will be presented. Section 3.2 talks about the benefits of blogging and how this stands as a strong starting point for journaling applications. Then, section 3.3 presents the elements of newness that the current system tries to engulf along with the reason to why they are so important and about journaling as an online tool. The last section3.4 talks about some criteria by which journaling platforms can be analysed and about similar systems and the features they offer.

3.1 Wikis, Blogging and Journaling

There are many ways and tools that help people reflect, share their thoughts or just keep those ides for themselves. However, there are some tools that stood out once the virtual world had its boom. Their similarities would convince some to put them in the same basket. However a wiki is a collaborative tool, a website on which users are allowed to add, edit, collaborate and communicate. This platforms of course enabled group and research projects as well as team writing assignments. All lead to the key word collaboration.

Blogging came along more as a social media tool. Blogging meant that a user willingly and publicly shared with the world their 'personal' diary. Soon this diary become more of a tool for advertising and in some cases came down to pleasing the audience more than anything. It can be considered that it is an exchange between the audience and the blog owner. The blogger shares content, ideas and the audience responds. We can say this tool is based on constant feedback form both parties.

And then there are journals which are the most private and personal form of self reflection. Journals existed since humans started writing. Personal journals of great people in history were brought to light centuries after their death only to reveal incredible ideas, events, stories, secrets and facts. Certainly journaling is still going strong even today. Thought the most 'silent' means of writing, people are still looking and interested in personal journaling tools and platforms now that most things in our lives have been digitised. Phones and laptops are everyone's luxury and there are always ideas to be written down.There are many tools created for the purpose of journaling and we shall analyse some of them in the section 3.6.

People still want to have their own personal and private diary but that is just as powerful and cool looking as any other writing tool. Personal environments for self reflection should be the most flexible platform, having the best integration with other apps and services.

3.2 Learning from Blogging

Blogging and journaling though different at the first glance, their essence is quite the same. They are both journals, personal diaries. One is made available to the public while the other one is still hidden from the public view. Even though not much was written about journaling, most probably because of the private nature of this hobby, there are many things that have been observed about blogging and bloggers, the effect of writing and its benefits and how digital platforms made it all possible and even enhanced the experience. We can also learn about some characteristics of the users using these tools and the effects of writing and the motivation behind it.

In the article [2] the author points out that bloggers are concerned with everyday lives, be it theirs or others. Also the author talks about the fact that the audience the bloggers have can not be literally everyone. Though publicly available it is actually restricted to those who do have access to electricity, IT equipment, access to the internet and have the knowledge and skills required to find and take advantage of blogging.

Toppo [1] tell about an analysis that has been made that suggests that blogging has a powerful educational function. It develops the power of reflection of the individual, organising ideas, articulating ideas, interact with others and even helps you in defending your opinion.

Blogging developing so fast over the past years led to more definitions and labelings of what this space represents. It is so called citizen journalism, encyclopedics , videographers, diarists or hobbists. The article states, as what we may consider a conclusion, that blogging and all the process it implies may become a lifelong form of education.

[1]Toppo `http://www.usatoday.com/tech/news/2006-09-17-teacher-blogs_x.htm`

In her [3] case study, Moira Dunworth talks about one of the students involved in her study that confesed that blogging helped her sort out her thoughts, filter her reflections and learn a lot about herself in the process. Writing helped this student explore her feelings and responses. Moira also tells us how private reflections should not be lost. A hand-written journal would make the process more difficult and here comes the power of online tools and platforms. Despite the fact that the initial response of the other students involved seemed to have been dictated by their age they concluded in the end that using blogging as a reflective journaling tool had positive effects on their confidence and writing skills.

In another case study [5], it is clearly stated that paper-based journals are no longer providing the best opportunity for formative assessment. Blogging offers a solution to the many faults that came with paper-base journals. The elements of publicity and novelty seem to be of real importance here. Blogging had the ability to unlock something in the students involved in this study and to free up their writing. They referred to their writing sessions as a session of self expression. The entries written showcased humour, creativity as students became aware of the fact that they were writing for an audience. Though this study was conducted having in mind assessment as bottom line, the students involved said that the notes and ideas written stood more as a starting point for essays and reports and did not feel like an assignment.

Despite the fact that journaling does not imply the element of publicity the reason why I choose to develop such a system is that there are many platforms that allow users to publicly share ideas and contents in many forms and shapes, having great cross platform support, however there are few systems that offer the same support for private journals.

3.3 Emojies, GIFs, Stickers and YouTube

Text has been the primary means of communication for many years. Even though it may be hard to believe emoticons came on the scene as early as the late nineteen-eighties (1980s). Emoticons defined in a more technical way are keyboard symbols that, when used together, make a symbol that conveys a feeling.

For the languages that have an alphabet, text has been the second method of choice ,after speaking, when in came to communication. The languages that are based around pictograms and symbols to communicate, have a plus over the alphabet oriented once, but they still have to adapt to the new tendencies.

The use of emojis, GIFs, and stickers has turned into a science itself as the author says in his paper [4]. He calls these new tendencies, new frontiers. Companies are developing their own text to emoji, text to GIF, and text to sticker input systems and the interest for such fields of study can not be ignored.

Thus we can say that text alone can no longer be considered as primary means of communication. Text analysts and data scientists are forced to evolve to be able to analyse this new data and these new form of communication that has started to replace the words.

As computer power has drastically increased in the last decade, users are now able to add small images and short clips from videos into their conversations. These are called stickers and GIFs. When all these drastic changes made their way to the public, text could no longer be relied upon as the exclusive means to facilitate the communication process. Now, stickers and GIFs also need to be taken into consideration as part of the text analysis to make the analysis a correct and accurate one. It seems that there in no separation between text and these visuals assets. They have to be considered as one and need to be taken together.

The article by Steinmetz [9] is mentioned in this paper. It has been observed that thirty-six percent (36%) of the millennials ages between 18 and 34 use visual elements such as emojis, stickers, and GIFs in their communications. It has been observed that millennials feel these visual elements communicate their feelings in a more accurate way. Steinmetz also found that almost sixty-six percent (66 %) of millennials consider GIFs as better than words when talking about communicating emotions, while eighty percent (80 %) of them feel that text combined with pictures expresses more than text can communicate alone.

As about Youtube, the platform made its debute in 2005 and since then it only grew exponentially. The video library came to play a huge role in peoples lives and their self education. In his study[6] about creating Youtube language learning videos, Munassir admits the fact that there are little literature about Youtube and its role as a language learning tool.

Despite the fact that Youtube and its implications are poorly documented we have to look around. Most people use Youtube for learning, entertaining or informing themselves and it seams that people find this source of information more reliable and truthful than the news or mass-media. If in the past people used to keep extrapts and their favourite articles from newspapers and magazines, nowadays people save and share these videos. Youtube may be considered the new mass-media and everyone can be a little journalist.

3.4 Related Work

In my quest to find documents and papers talking about journaling as an online platform, I came across a study [7] whose aim was to see if an online journal will increase the power of self reflection among 6th grade boys.

In Zimmerman (2002) [8] the author talks about two main processes in self reflection. One it refers to the comparison of self-observed performances against a standard such as one prior performance, someone else's performance or even an absolute standard. The second process involves feelings of self satisfaction and positive affect regarding one's performance. These suggests that motivation is enhanced with increases of self satisfaction.

For the study a tool was developed , called AXEL. The website had a unique design that included images rather than text and allowed the user to customise their journal.

The had to log in and complete a journal entry containing by checking one to five on a series of statements pertaining to their behavioural and cognitive engagement. The students were able to change the background image, upload a .jps or .wav files and set goals which they were helped through the application to regularly reflect on the goals set.

The study concluded with success as AXEL was considered to be engaging and fun, allowing students to be reminded of their goals and helped them self reflect. The students enjoyed using the platform and proved to had a positive impact even on their learning curve and class work.

Foltyn Marek puts it very beautifully into words in his master thesis [1] stating that the difference between note taking applications and writing on paper is the ability of linking and referencing data. He says that these notes serve as the **digital brain** for digital data. And this can not be more true. As an online tool for writing and reflecting, saving and arranging you thought process in an accurate way that best describes the flow of event it is ideal. Foltyn continues talking in his paper about the many aspects a user must take into consideration when searching for such a tool and some of the criteria considered: the type of data that can be stored in those notes, how are they organised, the look and feel of the user interface and so on. Also the users work and habits are can not be overlooked.

The author identified some criteria by he choose to make his analysis of similar notes taking applications. He noticed four note workflows. The first one is the creative workflow which as the name says is focused on the creative aspect of the activities that can be performed. Some typical features of this type of workflow are text formatting, drawings and links. The second workflow is the data flow which refers to knowledge and the information stored: files, links, text. The last workflow is the sharing one. This workflow focuses on sharing notes and the ability to export them in different formats.

Another criteria by which the author analysed the systems is notes organisation. He discusses about two different approaches he encountered. One is tree organisation and the second, which I am also going to use in the implementation of my own system, **tag organisation**. A note can have multiple tags and thus a tag can belong to multiple notes subsets. Tag seem to give a more flexible approach of putting and retrieving those notes.

Markdown and user interface are another two criteria by which the systems were analysed. Markdown refers to text editing and formatting while user interface refers of course to the way the application looks and feels. The specific aspects that were thanking into consideration when thinking about the UI were if the elements helped users to stay focused on the note they were writing and if it was non-distracting highlighting the note and not necessarily the number of features the application had.

Below is a presentation of several platforms similar to the system I ended up creating. Those platform stood as foundation for my own application. I was able to analyse them, see their features, advantages and disadvantages, what they lacked and what they excelled at. Also the design aspects of those applications were considered and stood as a starting point when designing my own design and user interface.

- **Evernote**
  Evernote [2] was founded in 2007 by Stepan Pachikov and it aimed to tackle the issue

[2]https://evernote.com/

that technology started to create, that being an ever growing volume of information. The platform became very popular due to its features: syncing and organise, web clipper which enables saving articles and other interesting resources for later in your Evernote account. It also support templates, PDF and Doc Search, Search Handwriting which means you are able to search for keyword even through pictures or handwritten notes, document scanning and integration with other apps such as gmail, google drive and some more. The statistics show that 225 million users use Evernote's products today.

For many professionals and casual web users, Evernote has become the ultimate note-taking and archiving tool helping them collect information, collaborate with others and keep everything organised. [3]

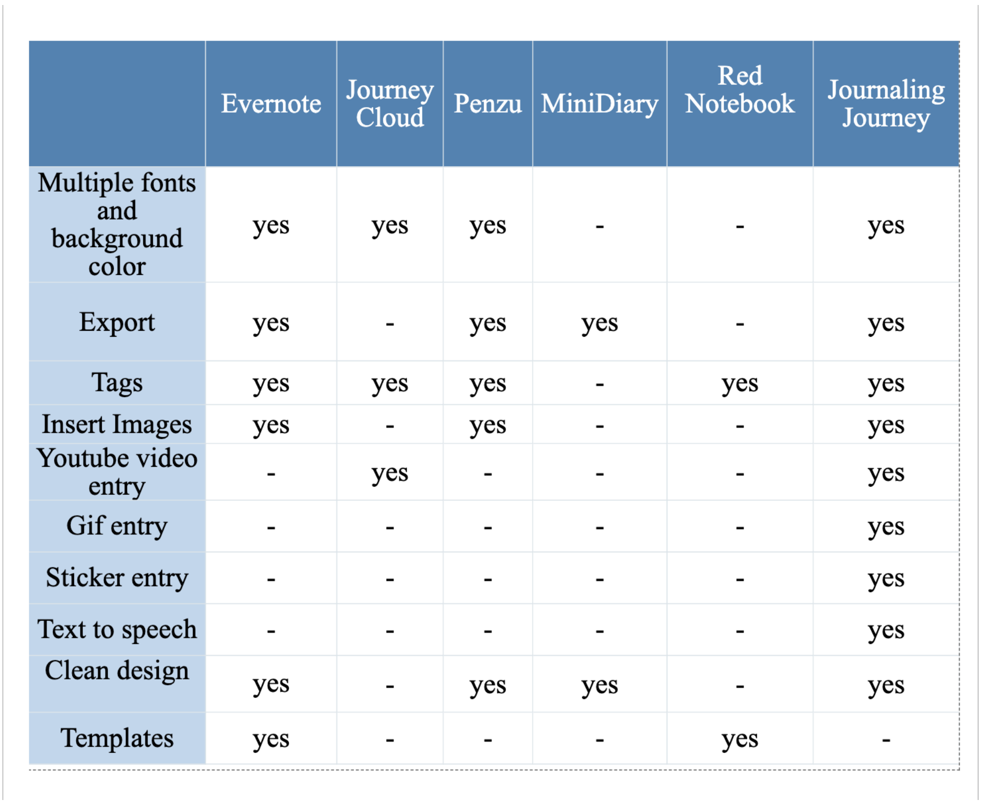
- **Journey.Cloud**
  This application [4] takes a more personal approach toward the users. The app claims to be your own motivational coach and trainer. The personal character of the application can be seen in some of its features such as a motivational prompt for each day and the ability to customise your journaling experience.

  Despite the fact that Journey Cloud does not enumerate the list of features it has and somehow it gives you a sense of surprise, Journey [5] is best known as the world's number 1 app for diary, journal and mood tracker. It has been downloaded over 3 million until today and many five star reviews. It was named by Google as both a best of 2018 award winner and Apple App Store's App of the Day 2019. Also it was picked as an "Editors' Choice" in 2019 by Google Play. Moreover it was ranked as the top lifestyle app and application worldwide in 2019. So we can not overlook all these positive feedback this app has received but learn from it.

- **Penzu**
  Penzu [6] was founded in 2008 in Toronto by Alexander Mimran, Simon Wilkinson and Michael Lawlor.[7] The application aims to help you focus on the writing and the the system to fill in the spaces in accomplishing this goal. Penzu has a premium version as well that comes with new features: fully customisable diary enabling the user to set custom covers, backgrounds and fonts, journal search and the so called military grade security which implies using 256-bit AES encryption.

- **MiniDiary**
  Though having less features MiniDiary [8] comes with a very simple, intuitive and clear design. The journal is encrypted with a locally saved password. Also the platforms

[3]https://www.lifewire.com/what-is-evernote-3485736
[4]https://journey.cloud/
[5]https://journey.cloud/press
[6]https://penzu.com/
[7]https://en.wikipedia.org/wiki/Penzu
[8]https://minidiary.app/



	Evernote	Journey Cloud	Penzu	MiniDiary	Red Notebook	Journaling Journey
Multiple fonts and background color	yes	yes	yes	-	-	yes
Export	yes	-	yes	yes	-	yes
Tags	yes	yes	yes	-	yes	yes
Insert Images	yes	-	yes	-	-	yes
Youtube video entry	-	yes	-	-	-	yes
Gif entry	-	-	-	-	-	yes
Sticker entry	-	-	-	-	-	yes
Text to speech	-	-	-	-	-	yes
Clean design	yes	-	yes	yes	-	yes
Templates	yes	-	-	-	yes	-

Table 3.1: Comparison of Similar Systems

makes it easy to import journals from other apps and to export your content in various formats. The application is free and open source. Some other features it offers is light and dark mode, full-text search and statistics.

- **RedNotebook**
  Despite the fact that RedNotebook [9] does not have a modern user interface it has quite a lot of useful features that other platforms do not have. The user can insert hashtags, format the text, insert images, files and links. The platform supports spell check, search as you type, the ability to backup to zip archive, templates, export to plain text, html or Latex.

A comparison of the systems I have taken into consideration before creating my own application can be seen in the table 3.4. The comparison is made based on the features each application offers. Where the feature is not existent a dash was placed.

At the first glance we can notice that none of the similar systems have as a feature the ability to add Youtube videos, GIFs or stickers to the journaling page.

[9]https://rednotebook.sourceforge.io/

Chapter 4

Theoretical Backgound

The aim of this chapter is to present the user needs and the features of the Journaling Journey Web Application in a high-level manner. The chapter will focus on the reason why such an application is needed in the Problem Statement section 4.1. In the next two sections 4.2 will talk about the users and their needs and also about the product overview. Lastly, but not least, we will go over the non-functional requirements and in more depth over the functional once in the Requirements section 4.4. The last two sections will go over the use cases 4.5, out of which three will be presented in details, and over the technologies used 4.6.

4.1 Problem Statement

In this time and age text alone is no longer considered an accurate means of communication. People like using visual expressions as they feel they communicate their feelings better. As Madewell Charles says [4]:

> *" text alone can no longer be counted upon as the primary means of communication and data scientists and text analysts must therefore evolve to even be able to analyse this new population of data containing emojis, GIFs, and stickers."*

The problem of being unable to fully express yourself and document your thoughts and ideas using the current platforms available on the market affects the people who use online platforms for writing and documenting. This impacts and affects the quality and accuracy of displaying the ideas and thought process of the individual. A successful solution would be to have the ability to introduce and add to journaling pages elements such as: GIFs, Stickers, Emojis, Youtube videos and images, that help you better express your feelings and put on the 'virtual paper' your mind's process.

This application aims to address and solve the limitations concerning the assets and content that can be added on a journaling page given an online platform. It is important to provide users with easy access to the elements they use or encounter on a regular basis and became part of the norm of the social media environment in recent years.

The target users are writers, bloggers, teenagers or adults who love journaling and documenting their ideas and thoughts, the Journaling Journey is a web application That enables you to add to your journaling page expressive graphics and assets as well as links and offers the possibility to listen to your own written content with the help of text to speech integration. Unlike any journaling application on the market (Evernote, Penzu, Mini Diary, etc) this product takes the social media elements that are used on most social media platforms nowadays and integrates them, offering easy access to these loved graphics and Youtube.

4.2 Users Needs

The target users of the application are bloggers, writers, people who enjoy writing, teenagers, adults, influencers. As the application works as a private personal journal. There is only one person involved in completing any of the tasks at any given moment. The main constraint of the application would be its connectivity to the internet. Making use of several APIs and services throughout the application we need internet connection to be able to get the data and the responses and functionalities we desire.

The user also needs to have a Google account to be able to authenticate themselves and use the application. This authentication method is the only available for now. Also the application will only be available on the web. In the future a react-native version can be created which will enable the app to run on both IOS and Android mobile devices.

1. There is the fundamental need of creating an account and tackling its security. There are many ways in which a user can create an account such as: creating an unique username and password associated with it, sign it with google, facebook or other social media platforms. The proposed solution for this project is Google Authentication. For this the user needs to have a google account or to create one.

2. Basic text editing is also of high priority. Keeping track of all this text edits and also managing them from a technical point of view. Similar systems allow a variety of text editing features much like a text editor such as Microsoft Word. However most focus on the most important once. The current implementation will support changing the font family and font size. Also the user will be able to change the background colour of the journaling page offering it a more personal touch.

3. Adding visual elements to the journaling is one the needs that i desire to fulfil. There are very few, if any similar systems available on the market right now offering the ability to add GIFs, stickers and YouTube videos integration. This project aims to integrate these endeared visual elements found on most social media platforms in this time and age.

4. The ability to audio play the written content of the page the user has written down is one of the needs that not everyone dreams of fulfilling. Not even the most popular journaling and notes taking applications have audio rendition. Allowing users to listen to their written contents and even choose the voice used for the audio helps the user giving him flexibility and a different perspective of his own writing.

5. There is also the need of exporting the pages to pdf format. Other platforms support this fas well. Some of them offer exporting in multiple document formats.

6. Sorting the journaling entries is something that needs to be considered and taken care of. Similar platforms sort their notes entries through the use of tags, hashtags or by date of creation. This app will use tags based classification. The user can add these tags when they are writing a journaling page in a dedicated area of the page.

4.3 Product Overview

Journaling Journey web application should provide the user with the functionalities that helps him add all the elements he wants, be it text or visual assets and give him the possibility to format this content. The application's aim is to give the user easy access to all the assets that he encounters on most social media platforms and bring them together on his own journaling page.

The systems architecture is presented in the figure 4.1. The client and the server are communicating through Rest requests. The server is of course communicating with the database. The client communicates with the Google server to authorise the login into the application and the server returns to the Client the user token and info.

This web application will add onto the existing features of other similar systems found on the market, having a slightly different approach. This will potentially attract new categories of users and enhance the writing experience.

4.4 Requirements

4.4.1 Functional Requirements

- **Feature 1**: The user can access the application by authenticating with Google
  The user can log in using his google account.

- **Feature 2**: The user can perform basic text editing and change the colour of the journaling page.
  The user is able to change the font family of the text, its size, make it bold or italic or change the colour of the journaling page background.

- **Feature 3**: The user can add GIFs to their journaling page
  The user can add as a visual asset gifs onto his journaling page.

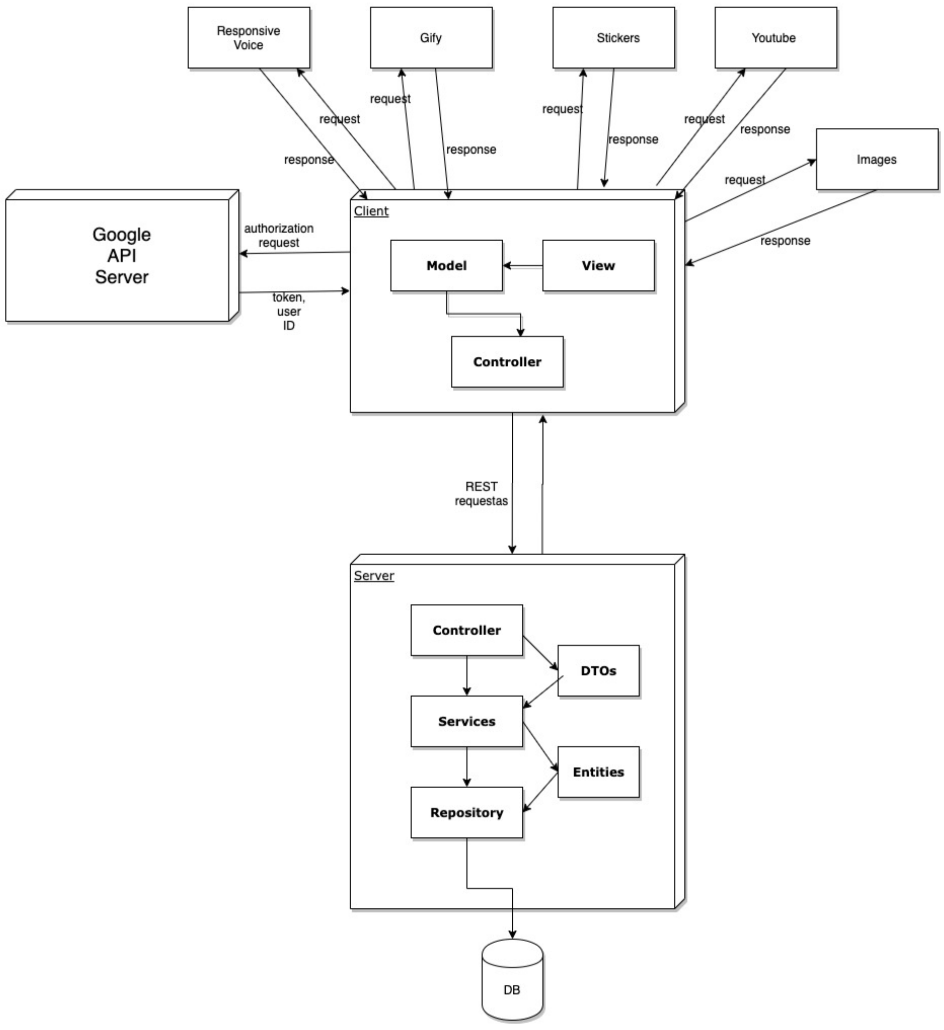


Figure 4.1: System architecture

- **Feature 4**: The user can add stickers to their journaling page
  The user can add as a visual asset stickers onto his journaling page.

- **Feature 5**: The user can add youtube videos to their journaling page
  The user can add as a visual asset youtube videos onto his journaling page.

- **Feature 6**: The user can add emojis to their journaling page
  The user can add as a visual asset emojis onto his journaling page.

- **Feature 7**: Text to speech of each journaling page
  The user will have easy access to the audio button on the journaling page he finds himself on. By pressing it, the text will be audio played for him by an automated voice. He can change the voice from the settings page.

- **Feature 8**: Export journaling page to pdf format
  There is also the possibility of exporting the journaling page. The only format that is supported for now is .pdf file format.

- **Feature 9**: A user can use tags to classify journaling pages
  The aim of this functionality is to allow the user to add tags to their journaling pages. This way the pages can be easily sorted on the dashboard. This offers flexibility and easy access.

- **Feature 10**: Search
  The system will permit searching for journaling pages using one of the two criterias:

  – tags
  – date

  Also while creating a journaling page the user will be able to search for gifs, sticker, images, youtube videos based on one of the two:

  – key words
  – title

- **Feature 11**: Settings tab
  In the settings tab the user is able to make the following operations:

  – change the voice for the text to speech rendition
  – delete his account content?

4.4.2 Non functional Requirements

1. **Usability:** We always have to keep in mind the usability aspect when creating the application. This way the aim is to create a simple and intuitive interface. A clear and simple design makes it easier for the user to navigate through the page and menus and get hold of all the functionalities available through good labelling of buttons and intuitive icons. Offering constant feedback, multiple options for an operation and a clean flow of events is key.

2. **Security:** This requirement refers to the ability of the system to protect the user's data and credentials. As we use Google Authentication, the security aspect of the application is solved by the Google's Service. Google authentication frees us from the task of storing user data and password directly into our database and also from the token management aspect.

3. **Scalability:** The aim is to make our application scalable so as to be able to support and adapt to as many users as possible at one given time. We want our users' experience to be a pleasant one, not lacking performance when multiple users are logged in at the same time and using the platform.

4. **Performance:** For the performance aspect of our application we will take into consideration the response time of our system. The response time for any operation made in our application that requires a lot of computation or fetching of data should not go beyond 2-3 seconds. For the rest of the operations the response time should take under 1 second.

4.5 Use Cases

All possible use cases are shown in figure 4.2 . Further we will examine in detail three of the use case. First Adding GIFs to the journaling page 4.3 , then the audio rendition of text on a Journaling Page 4.4 and lastly the filtering of the Journaling Entires by tag 4.5 .

4.5.1 Adding GIFs to the journaling page

This use case starts when the actor wants to insert on his journaling page as a visual asset a GIF.

Preconditions

1. The user is authenticated and authorised for this use case

2. The user created a new journaling page.

Postcondition

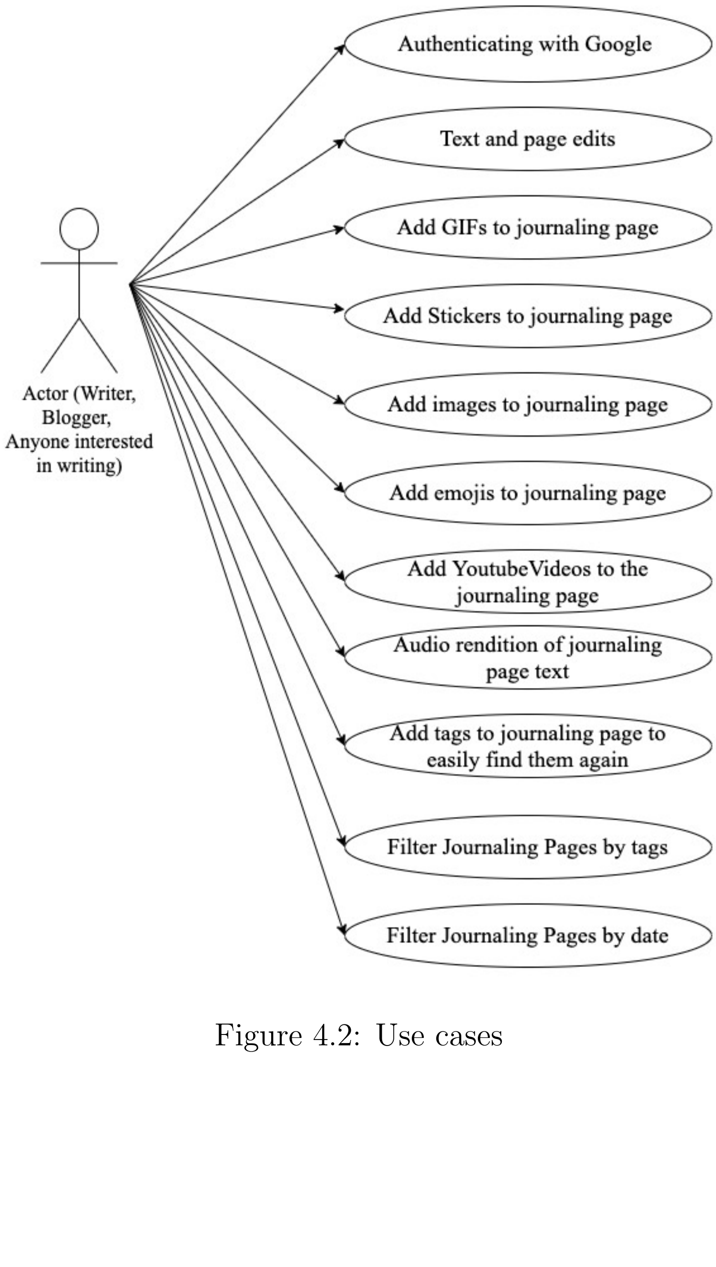

Actor (Writer, Blogger, Anyone interested in writing)
Authenticating with Google
Text and page edits
Add GIFs to journaling page
Add Stickers to journaling page
Add images to journaling page
Add emojis to journaling page
Add YoutubeVideos to the journaling page
Audio rendition of journaling page text
Add tags to journaling page to easily find them again
Filter Journaling Pages by tags
Filter Journaling Pages by date

Figure 4.2: Use cases

1. Scenario entities should remain in a consistent state if the Add gif operation has ended with success.

2. Scenario entities should not suffer any changes if the operation has not ended with success.

Basic flow:

1. The user creates a new journaling page

2. The user selects the menu tab allowing him to add a GIF to the page

3. System displays GIF's PopUp/selection window

4. The user enters the search keyword

5. The system displays GIFs based on the searched term

6. The actor selects a visual asset from the search results

7. The actor confirms the GIF selection has been made

8. The system displays the selected gif onto the page.

Alternative Flows

1. User aborts the add gif flow
   This alternate flow may occur at any moment after the add gif flow has started. This alternate flow may occur at one of the following steps: 2.1.3, 2.1.4, 2.1.5, 2.1.6.

   (a) The system remains unchanged, no gif will be displayed.

2. There is no results for the searched keyword

   (a) The system displays a message letting the user know there are no results for the searched keyword introduced.

3. The user confirms asset selection has been made even though no asset was selected

   (a) The system remains unchanged, no gif will be displayed.

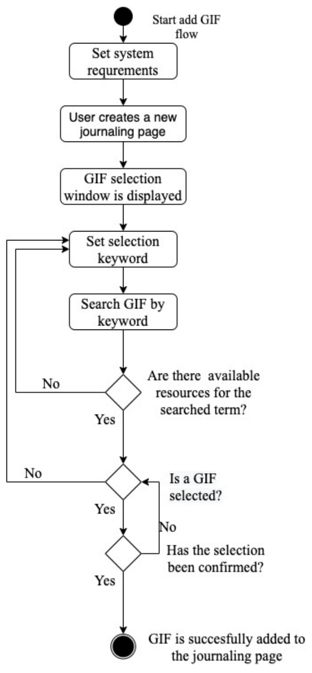

Start add GIF flow
Set system requrements
User creates a new journaling page
GIF selection window is displayed
Set selection keyword
Search GIF by keyword
Are there available resources for the searched term?
No
Yes
Is a GIF selected?
No
Yes
No
Has the selection been confirmed?
Yes
GIF is succesfully added to the journaling page

Figure 4.3: Use Case: Add GIF to Journaling Page

4.5.2 Audio Rendition of text on a Journaling Page

Preconditions

1. The user is authenticated and authorised for this use case
2. The user created a new journaling page
3. The user has written some text on his journaling page
4. The text must be written in english

Postcondition

1. Scenario entities should remain in a consistent state during and after the audio rendition has ended.

Basic flow:

1. The user selects from the menu tab the audio button
2. System starts the audio rendition of the text present on the page
3. The audio rendition of the content ends when the entire file has been spoken out

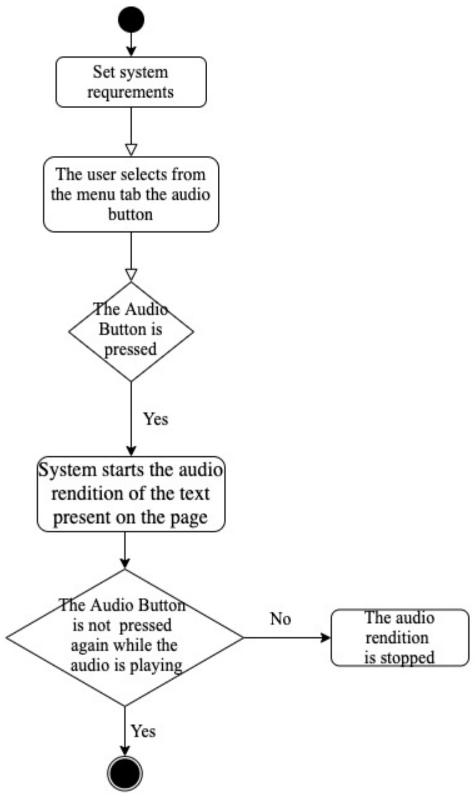

Set system requrements
The user selects from the menu tab the audio button
The Audio Button is pressed
Yes
System starts the audio rendition of the text present on the page
The Audio Button is not pressed again while the audio is playing
No
The audio rendition is stopped
Yes

Figure 4.4: Audio Rendition of Text

Alternative Flows

1. User stops the audio rendition by pressing the audio button again
   This alternate flow may occur at any moment after the audio has started.

   (a) The system remains unchanged.

2. The text on the page does not contain english words or intelligible words

   (a) The system will audio play the closest interpretation of the text or even letter by letter.

3. The page cointains no words

   (a) The system will have nothing to audio play as there is no text written on the page

   (b) The system remains unchanged

4.5.3 Filter Journaling Entries by tag

Preconditions

1. The user is authenticated and authorised for this use case
2. The user has tagged journaling pages in his history
3. The user finds himself on the dashboard tab

Basic flow:

1. The user enters the search tag in the search input
2. System displays on the dashboard only the list of the journaling pages that contain that tag

Alternative Flows

1. There are no journaling pages that contain the searched tag
   (a) The user will see a message letting him now there are no results for the searched tag
   (b) The system remains unchanged
2. The user erases the searched term from the input
   (a) The system displays on the dashboard all the users journaling pages history

4.6 Technologies and Methods

In this section the main technologies will be described along with their main advantages and the reason why some of them were chosen. First will be presented the technologies used on the frontend side followed by the once used on the backend side.

4.6.1 Front End

1. **Javascript**
   Javascript[11] is the language of the web browser. Because of the role it plays when it comes to the browser it has become one of the most popular programming languages in the world. Javascript in also the first lambda language that went mainstream. Even thought the name of the language recalls the Java language, at the roots Javascript has more in common with Lisp and Scheme, than it has with Java. Javascript is so to speak 'Lisp in C's clothing'. However this is also the reason why

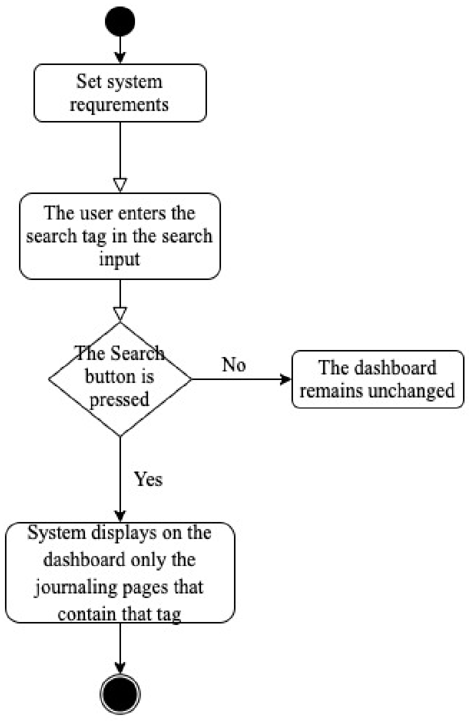

Set system requrements
The user enters the search tag in the search input
The Search button is pressed
No
The dashboard remains unchanged
Yes
System displays on the dashboard only the journaling pages that contain that tag

Figure 4.5: Filter Journaling Entries by Tag

Javascript is a remarkable powerful language. In Javascript has a powerful object literal notation which allows objects to be created simply by listing their components.

JavaScript[10] allows us to script the HTML and CSS content in web browsers, but it also allows us to define behaviour for those documents with event handlers.

To understand the role Javascript has in web applications we need to understand first that web browsers have surpassed their original role and they have become more like simple operating systems. In the same manner a traditional operating system allows you to organise on your desktop the icons that represent applications and files to run multiple applications in separate windows and defines lo-level APIs for networking graphics and saving files, a web browser displays multiple documents in different tabs and handles graphics and networking.

Javascript is the language of the browser. All modern web application make use of javascript and the once that do not, are most probably obsolete. For creating this application I had to get more familiar with javascript and take advantages of this versatile language.

2. **React**
   React[12] is a popular library created by Facebook used in creating user interface. The library deals with challenges associated with large-scale, data-driven websites.

However react does not impose any restrictions on the data architecture. Some reasons[1] for using react is its simplicity in approach. Through the state concept in react you tell the application how it should look like at any given moment in time and React will automatically manage all the necessary UI updates. React is also declarative which means that it knows to update only the part of the app that has been changed making it efficient. Lastly, React allows building reusable components. Components make code reusable and testing and separation of concerns is made easy.

The reason for choosing React above other possible alternatives for this project is the fact that React is a light weight library and gives the developer more control over what packages are added into the project. Angular, it's so to speak alternative, is a framework on the other hand. The files scope are better defined and it makes it easier for the developers to create components, however the developers do not get to dive as deep into javascript concepts.

3. **Redux**
Redux [2] is a predictable state container for JavaScript applications or simply said a state management tool. It comes as a handy addition to the applications as it helps with consistency, running in different environments and with testing. React is lightweight so developers do not have to worry about the application's asset size.

   With Redux the state of the application is maintained in a store which is made accessible to all components. The reason why Redux is needed with React is because this way states will no longer have to be lifted up the scope chain. Without Redux any communication between parent components and child components has to happen though the props and this one directional flow can become heavy and also can wight down the parent component. Redux greatly simplifies the app and makes it easier to maintain, debug, test and to persist the state.

   Mobx is another alternative to Redux and is mostly used for smaller projects. Even though I could have chosen Mobx, as it is also easier and straight forward to use, I chose Redux instead for the current project in order to experiment more with it and to better understand the way it works. Redux is also frequently used for larger projects and thus it is more probable to be encountered on real web applications.

4. **Axios**
Axios [3] is a Javascript library used to make HTTP requests from node.js or XML-HttpRequests from the browser that also supports the ES6 Promise API. By using axios we remove the need to pass the results of the HTTP request to the .json() method. Axios simply returns the data object you expect straight away. Additionally, any kind of error with an HTTP request will successfully execute the .catch()

[1]https://zhenyong.github.io/react/docs/why-react.html
[2]https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835/
[3]https://medium.com/@MinimalGhost/what-is-axios-js-and-why-should-i-care-7eb72b111dc0

block right out of the box. Consequently Axios is an improvement of the quality of life more than anything else. However making these small, incremental quality of life adjustments the workflow can considerably improve the development process and the speed of the implementation.

5. **Eslint**
   ESLint [4] is an open source JavaScript linting utility originally created by Nicholas C. Zakas in June 2013. Code linting is a type of static analysis that is frequently used to find problematic patterns or code that does not adhere to certain style guidelines. JavaScript, being a dynamic and loosely-typed language, is especially prone to developer error. Without the benefit of a compilation process, JavaScript code is typically executed in order to find syntax or other errors. Linting tools like ESLint allow developers to discover problems with their JavaScript code without executing it. ESLint is written using Node.js to provide a fast runtime environment and easy installation via npm. The primary reason ESLint was created was to allow developers to create their own linting rules. ESLint is designed to have all rules completely pluggable.

   ESLint statically analyzes your code to quickly find problems. Many problems ESLint finds can be automatically fixed. ESLint fixes are syntax-aware so there will not be errors introduced by traditional find-and-replace algorithms. Also ESLint can be customised by writing your own rules that work alongside ESLint's built-in rules. This allows ESLint to work the way you want it to for your specific project.

4.6.2 Back End

1. **Java**
   Java [5] was initially created by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991. The language was initially called "Oak," but was renamed "Java" in 1995.

   The main reason why Java is so popular is because it is platform independent [6]. This means that programs written in Java can be run on many types of computers. A Java program runs on any computer with a Java Runtime Environment, also known as a JRE, installed.

   Java is object-oriented which means that Java programs are made up of programming elements called objects. An object is a programming entity that represents either some real-world object or an abstract concept.

   Java is a widely popular language. Using it on this project was just another way of getting more accustomed to it an taking advantage of its object oriented concepts.

[4] https://eslint.org/docs/about/
[5] https://en.wikipedia.org/wiki/Java_(programming_language)
[6] https://www.dummies.com/programming/java/what-is-java-and-why-is-it-so-great/

2. **Spring-boot**
   The Spring Framework[13] was created in 2003 by Rod Johnson.The Spring Framework was the response to all the complexity that the J2EE specifications had at that time. We can say Spring is a complementary technology to Java EE. The Spring Framework integrates several technologies, such as Servlet API, WebSocket API, concurrency utilities, JSON Binding API, bean validation, JPA, JMS, and JTA/JCA. The Spring Framework supports the dependency injection and common annotation specifications that make development easier.

   So, Spring Boot [7] is used to build stand-alone and production ready spring applications. Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can simply run. You can get started with minimum configurations without the need for an entire Spring configuration setup. Spring Boot was designed having as goals to avoid complex XML configuration in Spring, develop a production ready Spring applications in an easier way, reducing the development time, running the application independently and offer an easier way of getting started with the application.

   The reason I chose the Spring-boot framework is due to its intuitive way of use and adaptability, as well as the ability to easily add dependencies. Using it, creating and setting up a back end project is straight forward as its author aimed and promised.

4.6.3 Database

MySQL [8] is a database management system that allows you to manage relational databases. It is open source software backed by Oracle which means you can change its source code to suit your needs. MySQL [9] is pretty easy to master in comparison with other database software like Oracle Database, or Microsoft SQL Server. Many of the world's largest and fastest-growing organisations including Facebook, Google and Adobe, rely on MySQL when it comes to their platforms.

Despite the fact that in this project i did not make much use of the relational features of the database I have chosen MySQL for the query capabilities and the performance advantage of that, its friendly interface and good support.

4.6.4 APIs

API [10] stands for Application Programming Interface. An API is a software intermediary that allows two applications to talk to each other. In other words, an API is the messenger that delivers the request to the provider and gets back to you the response.

[7]https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm
[8]https://www.mysqltutorial.org/what-is-mysql/
[9]https://www.mysql.com/why-mysql/
[10]https://blogs.mulesoft.com/biz/tech-ramblings-biz/what-are-apis-how-do-apis-work/

The APIs takes the load off the developers and allows them to be more productive. Developers can focus on the creating the application and the complexities it implies while outsourcing all of the commodity functionality to APIs.

1. RapidApi, [11] is the world's largest API Marketplace, is used by over one million developers to find, test, and connect to thousands of APIs. The APIs are embedded in your application and you are able to track usage of all the APIs.

2. Giphy is an animated GIF search engine. The giphy API [12] offers your app access to the largest GIF and Sticker library in the world. The Giphy API implements a REST-like interface. Connections can be made with any HTTP enabled programming language. The Giphy API also implements CORS, allowing you to connect to Giphy from JavaScript / Web browsers on your own domain.

3. Unsplash API [13] is a simple API for embedding free high-resolution photos from Unsplash. The API is a modern JSON API that enables easy manipulation of data and offers it a good structure. The API is fast, flexible, free to use and easy to embed.

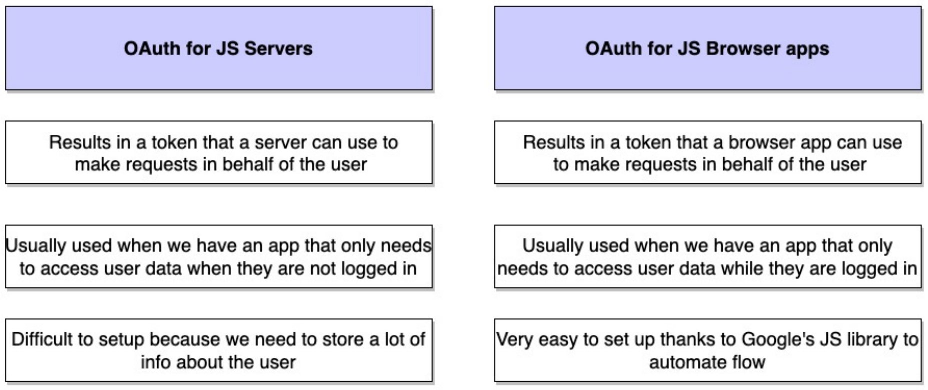
4. Responsive Voice [14] is a HTML5-based Text-To-Speech library designed to add voice features across all smartphone, tablet and desktop devices. It supports 51 languages through 168 voices and has no dependencies.

4.6.5 Google Authentication

Google APIs use the OAuth 2.0 protocol for authentication and authorization. Google supports common OAuth 2.0 scenarios such as those for web server, client-side, installed, and limited-input device applications

In figure 4.6 the comparison between OAuth for JS Servers versus OAuth for Browser apps is shown. It can be seen that the OAuth set up for browser much easier than for the Servers where many configurations must be made.

[11] `https://docs.rapidapi.com/docs/what-is-rapidapi`
[12] `https://rapidapi.com/giphy/api/giphy/details`
[13] `https://unsplash.com/developers`
[14] `https://responsivevoice.org/api/`



OAuth for JS Servers	OAuth for JS Browser apps
Results in a token that a server can use to make requests in behalf of the user	Results in a token that a browser app can use to make requests in behalf of the user
Usually used when we have an app that only needs to access user data when they are not logged in	Usually used when we have an app that only needs to access user data while they are logged in
Difficult to setup because we need to store a lot of info about the user	Very easy to set up thanks to Google's JS library to automate flow

Figure 4.6: OAuth Server side vs OAuth Browser side

Chapter 5

Implementation Details and Design

This chapter focuses on the implementation details, system architecture and code organisation. Classes, functions and APIs will be described in more detail. The more difficult parts encountered during the implementation and the way these were tackled will be mentioned and explained. Other important details will be specified as well.

5.1 General system architecture

The systems is implemented as a Three-Tire architecture. This architecture is a client-server architecture that is actually split in three different layers. The first layer is the **presentation** layer that has to do with user-interface and the logic needed on the frontend side of things. The second layer is the **bussiness** one, that deals with data manipulation and communication between the first layer and the last one called **data access** layer which is concerned with database access.

The main feature of this architecture is the fact that each layer is kept on a different platform, structuring the project and making it more efficient and clean as the logic is not clustered in one single place. Also, another benefit of this architecture is the fact that at any moment if one of the layer is wanted to be replaced or upgraded and kept up to date with the newest technology evolution, this change will not affect the other two layer. Of course each layer can be structured and subdivided as well. We will continue in the next sections talking about each one of these 'tires' and how they were implemented and structured in this application.

5.1.1 Frontend architecture

The **presentation layer** is the highest level of the architecture. This layer is represented by a graphical user interface, in the case of our application, a web page. This interface offers the user the possibility to visualise, modify and actually add data to the database. At this level of architecture design details can be added and layouts, which

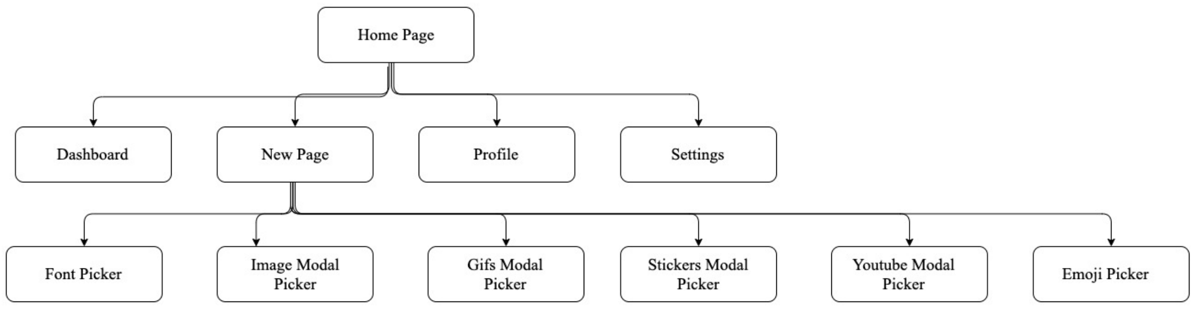


Figure 5.1: Logical Way of Accessing the Components of the Application

help with customising and personalising the application. This layer is represented in the Journaling Journey web application by the client side, implemented in React, utilising the Javascript programming language.

In figure 5.1 the logical way of accessing the components of our application is presented. After logging in we have access to four components dashboard, new page, profile and settings; and from the New Page component we have access to several other components identified as pickers: font, images, gifs, stickers, youtube video and emoji pickers.

5.1.2 Backend architecture

Between the presentation layer of the tier architecture and the data access layer there is an intermediate layer called the business layer. This layer implements and controls the functionality of the system through data processing. The presentation layer takes the user requests and maps them to functions that can actually access the database. Also, at this level the reverse functionality is implemented as well and it refers to the transport of data from the database to the presentation layer when this is operation is requested. All the layer implementation comes down to Controller and Service classes and the JPA Repositories developed and put together through the help of the Spring Boot Framework.

In the database, data is being saved consistently in the corresponding tables. Data can be created, visualised, filtered, updated or deleted. The database layer communicates with the business layer as we have mentioned before, by receiving from this layer queries requests and it gives back to it the extracted collections of data.

In figure 5.2 the packetdges structure can be easily seen and the way they are dependent on each other.

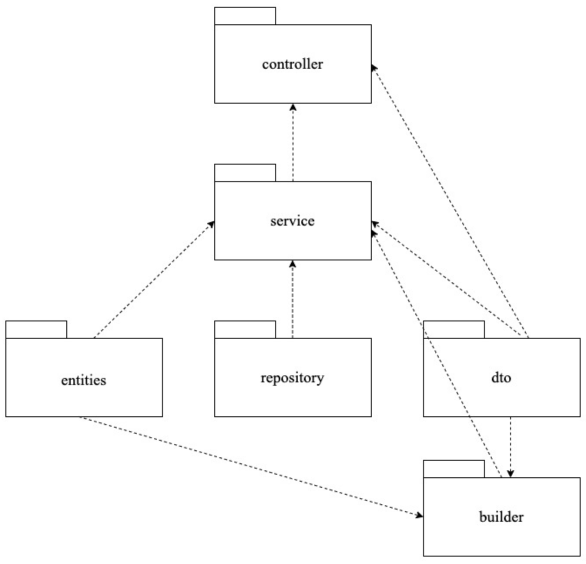


Figure 5.2: Packages structure

5.2 Frontend implementation

The frontend side was implemented using Javascript and React library. For setting up the project the command **npx create-react-app my-app** was run which created our application directory and set up the development environment. This application started enables using the latest Javascript features and optimises the application for production. Node JS is required to be installed on the running machine. Under the hood React uses Babel and WebPack, the first deals with making sure the javascript code is understood by all browsers and the second one is a module bundler. Another important file is the package.json file, so called package manager, all the packages added in the project are visible in this file.

Directory structure and file naming is very important in any project as it can make the developers lives easier or harder depending on the approach taken. Even this simple aspect of file organisation and naming must be carefully be taken into consideration. Firstly there are the public and src directories that are created when creating the react-app. The index.html file is found in the public directory and represents the dom. The src folder structure is created from scratch when starting the project. The index.js file that represents the stating point when opening the application (ex. the Home page) can be left outside any directory along with the constants file. There can be multiple constants files in larger projects and is such case a constants directory can be of course created.

Eslint was added to the project as well along with some custom rules. The dependencies can be seen in the package.json. How the rules are set up in the project can be seen in the in the .eslintrc.js. Bellow we can see some the rules used:

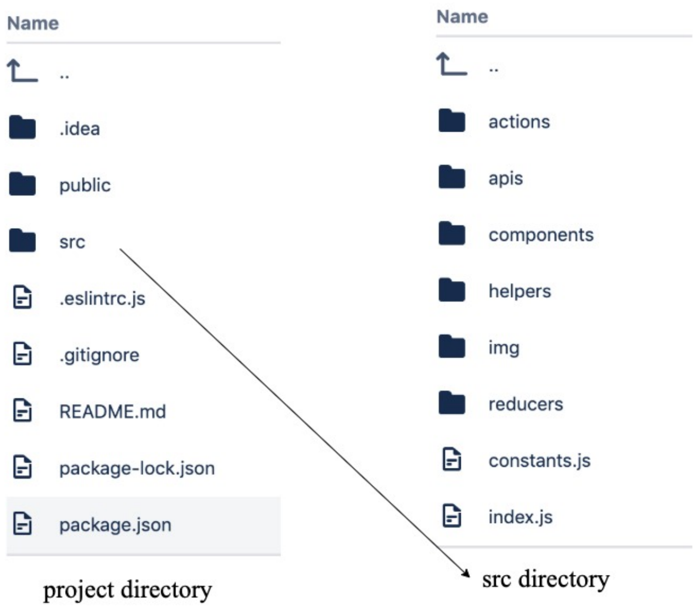


Figure 5.3: Directory Structure

```
"rules": {
    "arrow-body-style": ["error", "as-needed"],
    "arrow-parens": ["warn", "as-needed", { "requireForBlockBody": true
    ↪  }],
    "arrow-spacing": "warn",
    "comma-dangle": ["warn", "always-multiline"],
    "comma-spacing": ["error", { "before": false, "after": true }],
    "comma-style": ["error", "last"],
    "computed-property-spacing": ["warn", "never"],
    .
    .
    .
    "react/jsx-sort-props": ["error", { "ignoreCase": false }],
    "react/jsx-tag-spacing": "warn",
    "react/prop-types": [1, { skipUndeclared: true}],
    "react/require-default-props": "warn",
    "semi": ["warn", "always", { "omitLastInOneLineBlock": true }],
    "space-before-blocks": "error",
    "space-before-function-paren": ["warn", "always"],
    "space-in-parens": ["warn", "never"],
    "space-infix-ops": "error",
    "spaced-comment": ["error", "always"],
        "template-curly-spacing": "warn",
        "max-len": ["warn", {
            "code": 120,
            "tabWidth": 2,
            "ignoreComments": true,
            "ignoreTrailingComments": true
        }],
    },
```

For **styling** the components of the application I used Semantic UI. This helps with the html structure and styles the element by the class given to them. To integrate it the following link was added to the header section of the index.html file found in the public directory:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
    ↪ semantic-ui/2.4.1/semantic.min.css" />
```

Talking about **components**, in React there are two methods of creating components, creating classes or functional components. Most of the components in the application are classes as their complexity is higher and allow the use of lifecycle methods. Functional components focus on returning a piece of jsx and as we will see in this section jsx mimics html.

In the case of classes the .js file contains all the the imports needed for the current component at the top of the file, followed by the class declaration. Each class extends the Component class from the react library. Inside any class first we can identify the constructor that contains the call to reference the props of the upper class. Also, the state of the current class is written here, along with the state's initial values. After the constructor, lifecycle methods are usually written, however not all class components need to use those methods. Though such a class can be converted into a functional component, the use of classes or functional components is mainly a matter of choice for each developer. Between the constructor part and the render function all the handlers and additional functions are written. If the file becomes too heavy by containing too many functions, a helper file is created and pieces of code are moved to this file and imported from it back into the component file where they are needed .

Some of the main components created for this application are the Dashboard component, JournalPage and the PopUps. Starting with the **Dashboard** this component refers to the left navigation menu. On the right, based on the selected tab on the left, the specific component is displayed. The top navigation bar still remains, similar to the one present on the Home page. There are two operations handled by the top navigation bar, one is LogOut and the other routes the user to the home page. Looking at the render function of the Dashboard component we can see that it contains the top navigation component called InAppNavBar, the additonal jsx structure of this component and the links and routes for the tabs. A Link element is set on each of the tabs and has specified as attribute the path with which it is associated. For example the 'New' tab link looks like

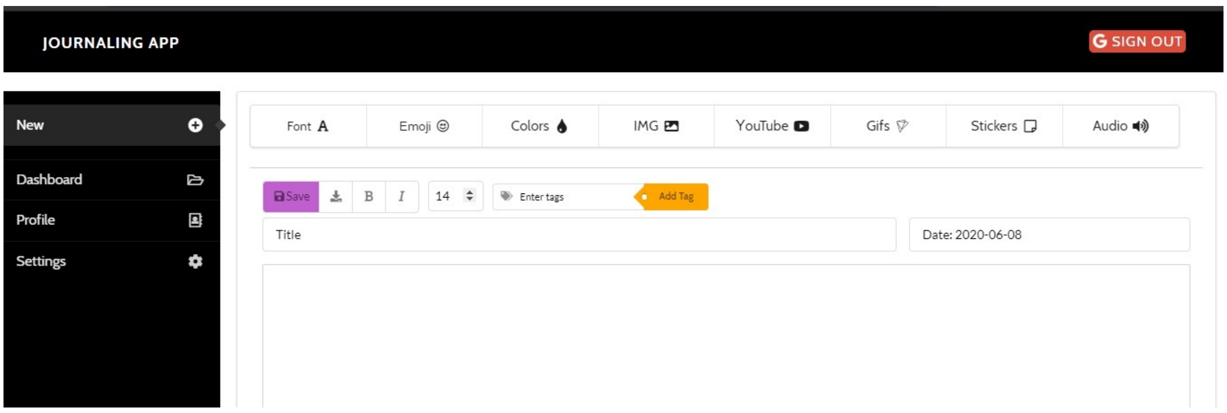


Figure 5.4: New Page Tab

```
<Link to="/my-journal/new" className="item" id='first-item' onClick={
   ↪ this.onNewLink}>
   New
   <i className="plus circle icon"/>
</Link>
```

The link is specified on the tab button and the routing is specified in jsx section where the routing will happen. This means that on the right depending on the tab pressed different components will be rendered. Depending on the routing made the appropriate component will be dispalied on the right. For the 'New' tab 5.4 the routing looks like this :

```
<Route exact path="/my-journal/new"
      render={(props) => <JournalPage
                    {...props}
                     pageData={pageData}
                     onPageSave={this.onPageSave}
                     voicePicked={voicePicked}
                    />}
 />
```

Route, Link, BrowserRouter as Router, Redirect are elements taken from react-router-dom library and to add them to the project we have to install react-router-dom library via npm.

Going on to the **JournalPage**, this component contains a large portion of the logic and brings together the main features we have intended to implement. Because of its initial size some logic was extracted in the journalPage.helper.js file that was placed in the component's directory along with the css file. The component has a big state as to keep track of all the changing elements present on the page such as: page content, font size, background color picked, tags set for the current page and many more. In the **componentDidMount()** lifecycle method there are some initial state setting for the current date and the pageData. If the pageData is not an empty object then it means

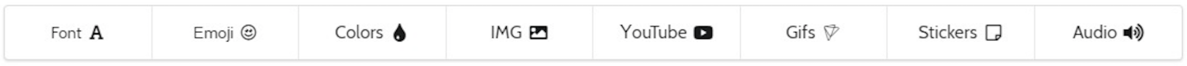


Figure 5.5: Journal Page Tool Bar

that from the Dashboard a journal entry was clicked upon in order to be edited and all is data in being sent to this component to be displayed. There is also the voice for the audio Rendition that needs to be set and if there is nothing already received from the pageData then it means we are creating a new page from scratch and not editing an existing one, then the voice set in the settings tab will be used for audio rendition.

```
  componentDidMount() {
    const { pageData, voicePicked } = this.props;
    const date = getCurrentDate();
    this.setState({creationDate: date});
    this.setState({pageData: this.props.pageData});
    if(pageData) {
      const tagsReceived = pageData.tags.split(',');
      this.setState({
        pageColor: pageData.colorCode,
        fontSize: pageData.fontSize,
        voiceType: pageData.voiceType,
        creationDate: pageData.date.slice(0,10),
        title: pageData.title,
        tags: tagsReceived,
        pageContent: pageData.contentPage,
        textToSpeech: pageData.contentTextOnly,
      });
    }
    else {
      if(voicePicked) {
        this.setState({voiceType: voicePicked});
        console.log(voicePicked);
      }
    }
  }
```

The journalPage component contains a the top a tool bar 5.5 with corresponding buttons that will allow you to add the assets available (GIFs, images, emojis, YouTube), font and the button that enables audio rendition is present there as well. All these elements are added of course to the editable area on the screen. I will talk about how this editable area works.

```
<ContentEditable
```

```
    className='apply-font'
    disabled={this.state.isPageEditable} // use true to disable editing
     html={pageContent}  // innerHTML of the editable div
     innerRef={this.contentEditable}
     onBlur={this.sanitize}
     onChange={this.setPageContent} // handle innerHTML change
     style={{ ...styleBorder, minHeight: 350, backgroundColor:`${pageColor
       ↪ }`, fontSize, lineHeight: 1 }}
     tagName='article'
/>
```

The disable prop manages if page is editable or not, the default is set to false, always enabeling editing when creating a new page. The html property contains the html code and structure for the pageContent at any given moment. The setPageContent function is triggered each time something changes in the content editable area and the pageContent state is updated. This pageContent state is not changed only when writing but also when adding visual elements.

When a visual element is added on the page, a pop up is opened, a visual asset is selected, and on modal closing confirmation the asset chosen is appended to the the pageContent state. The asset chosen is wrapped in html elements and looks in the end like a string. The ContentEditable understands and displays html elements.

```
  handleCloseGifModal = () => this.setState({ isGifModalOpen: false });
  handleGifConfirmation = (gif) => {
    this.setState({ gifPicked: gif });
    const currentGif = `<img alt=${gif.alt} src=${gif.src}/>`;
    this.setState({pageContent:`${this.state.pageContent}${currentGif}`});
    this.handleCloseGifModal();
  };
```

The last important component to be described is the popup window 5.6 used when picking the assets. For this component I made use of Button, Header, Icon and Modal elements from the semantic-ui-react library. The popup is opened when on the Modal element the 'open' property is set to 'true'. At the bottom of the popup there are two buttons, on to cancel the operation and consequently close the popup, and another one that confirms the operation and then closes the popup. Of course the operation I am referring to for this particular popup is Gif search and selection. The popup contains a scrollable area were gifs are displayed for the searched term. This list of gifs to be displayed corresponds to the following line of code:

```
<GifList gifs={this.state.gifs} handleGifPicked={this.
   ↪ handleGifPicked}/>
```

GifList is a component itself that receives as props the gifs and a function handleGifPicked which will allow the parent component (gifPopUp) to know about the gif picked. The

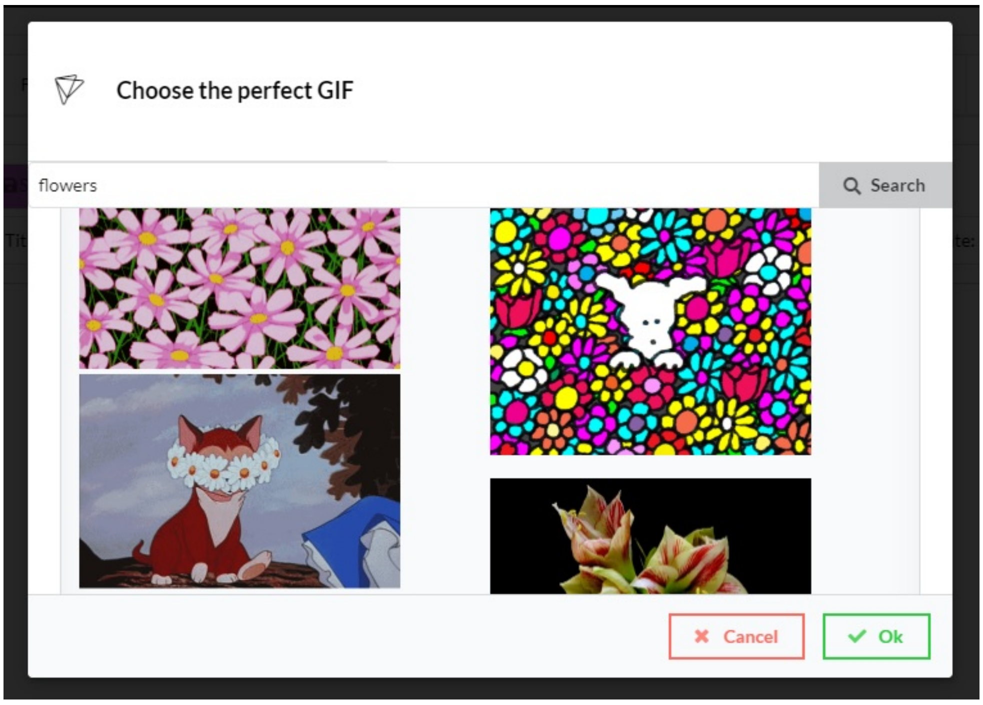


Figure 5.6: Gif Popup

gifs sent as props are the response.data.data of the request made to the gif api in the componentDidMount lifeCycle method of the current component.

In the GifList component we have to map over the list of gifs received as props and actually display each of these gifs. The map function goes over each element and applies some logic to it, in our case for each element it return a GifCard meaning the styling and the actual way a single gif will be shown on the interface. The end result of the map function is a brand new array. This list received as props is actually an array of objects. From each object element that represents the data about a gif we want to take the url, height and width in order to properly display the gif.

```
const GifList = (props) => {
  const gifs = props.gifs.map(gif =>
        <GifCard
               gif={gif}
               handleGifPicked={props.handleGifPicked}
               key={gif.id}
        />);
  return <div className='gif-list'>{gifs}</div>;
};
```

The GifCard component receives as props all the data about the gif, a unique key and a function handleGifPicked which will be the same function received from the parent component GifPopUp and was passed it down up to this level. From the render method of the GifCard component it will be returned a div on which we will able us to dynamically set the appropriate hight and width for each gif received. Inside the div there is a img element on which we will set the url and all the other attributes necessary for it.

```
  render() {
    const { gif } = this.props;
    return (
      <div style={{ gridRowEnd: `span ${this.state.spans}`}}>
        <img
          className='gifCard'
          alt={gif.title}
          ref={this.gifRef}
          src={gif.images.downsized_large.url}
        />
      </div>
    );
  }
```

The popup declaration is put in the JournalPage component as we want to have the control to open or close the popup at this level. On the JournalPage we make the decision to add or not a gif on the page. The GifPop up component is declared as follows in the JournlPage component:

```
<GifPopUp
      handleCloseConfirmationModal={this.handleCloseGifModal}
      handleYesConfirmation={this.handleGifConfirmation}
      isOpen={isGifModalOpen}
 />
```

There are several **APIs** used in the application but I will describe how the setup was made for only one of the APIs, the Rapid API that helps us get the GIFs. The others were set up in the same manner and respecting the same principles. The only thing that differs between them is the header, base url and parameters but all these are specified in the documentation of each API. The project contains an api directory in which the axios requests for each API is created in a separate file. The axios request for the Rapid API looks like this:

```
export default axios.create({
  baseURL: 'https://giphy.p.rapidapi.com/v1/gifs',
  headers: {
    "x-rapidapi-host": "giphy.p.rapidapi.com",
    "X-RapidAPI-Key": KEYS.RAPID_API_GIFS
  }
});
```

When making a request using axios.create we pass a as argument an object. The first element to be specified is the base url or endpoint, from where the gifs will be taken. The second element is the header where we have to specify the host and a unique access key. For getting a valid key we have to create an account on the Rapid API website and a key will be given to us upon creation.

The actual call to the api to get the gif happens in the gifPopUp component. The gifPopUp contains an input field where the search term is inserted. On 'Search' button press the request is triggered. For the call to be complete we of course need to specify the searched term just written by the user, an additional api key that rapid api requires for the request to be made and the type of request that is being made. This key is specified in the API documentation along with the order in which these parameters need to be written. Here is the code snippet that illustrates the above:

```
onSearchSubmit = async () => {
  const { searchedTerm } = this.state;
  const response = await rapidGifs
    .get('/search', {
      params: { q: searchedTerm, api_key: 'dc6zaTOxFJmzC' },
    });
  this.setState({ gifs: response.data.data});
};
```

In the response constant the request's response is stored. As mentioned previously we have to add some additional logic to our request, we specify the fact that the request is a 'search' request and as parameters we specify an object having as q element the search term and as api key the key mentioned in the documentation. We can see that the gifs state is actually set not to response but to response.data.data, due to the fact that the response returned by the API is a very large object with many details. However, we only need to access the list of urls of the gifs corresponding to our searched term and that is found at response.data.data.

Google OAuth and **Google Services** such as the one needed for **Youtube** required significantly more setup. First of all we need to go to the google api console [1] to create a new project and follow the steps prompted there as google wants to know where and with what purposes their APIs and services will be used. After creating the project we will add credentials for the YouTube API and we will enable OAuth consent screen. Documentation for both OAuth and Youtube can be found at google for developers page [2].

For the Youtube access we need an authorisation key which we got from the google console after creating the project and setting up the credentials. The process of making a request to search and fetch the desired youtube videos is the same as in the case of APIs. We use an axios request in which the base urls and parameters sent with the request are

[1]https://console.developers.google.com/
[2]https://developers.google.com

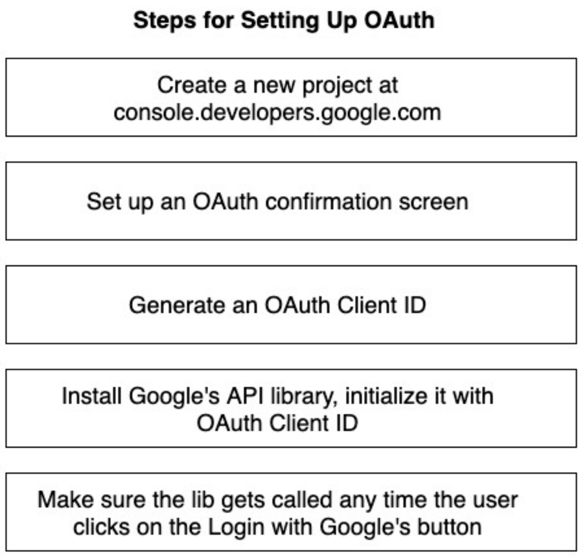

Steps for Setting Up OAuth
Create a new project at console.developers.google.com
Set up an OAuth confirmation screen
Generate an OAuth Client ID
Install Google's API library, initialize it with OAuth Client ID
Make sure the lib gets called any time the user clicks on the Login with Google's button

Figure 5.7: Setting up OAuth

specified looking up the documentation. When making the actual get call we have to also specify the request type, a search in our case as we want to fetch a list of youtube videos, and as a parameter the searched term.

For setting up Google OAuth the following script needs to be added to the index.html file found in the public directory:

```
<script src="https://apis.google.com/js/api.js"></script>
```

I have also created a GoogleAuthButton component and inside it the logic and connection to the Google OAuth was added. In the componentDidMount() lifecycle method we will initialise the gapi client with the client id that was generated for us when creating the credentials for the OAuth service for our project. We need to specify a scope and because we are not using any other google service we stick to the email scope which gives us back basic information about our user. For this initialisation a Promise is used, which means that after the initialisation of the gapi happenes, on the 'then' branch we will take the authentication instance of our user, we will update his authentication state (logged in / logged out) in our global application store and we will let the auth instance listen for any state changes.

```
componentDidMount () {
  window.gapi.load('client:auth2', () => {
    window.gapi.client.init({
      clientId: KEYS.GOOGLE_CLIENT_ID,
      scope: 'email',
    }).then(() => {
```

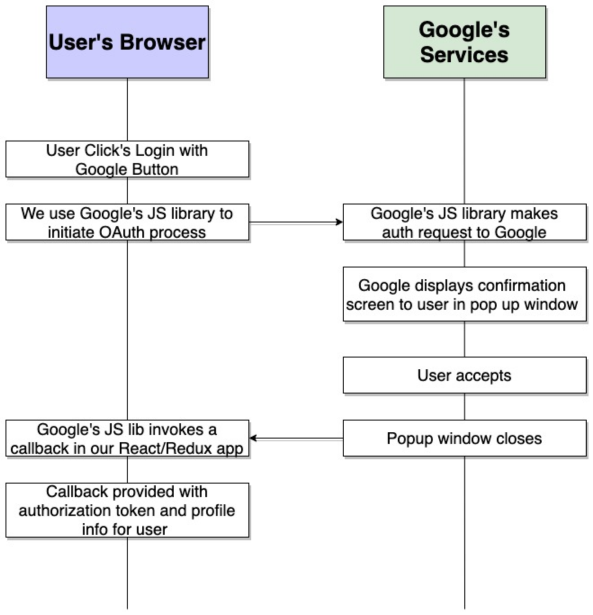

User's Browser
Google's Services
User Click's Login with Google Button
We use Google's JS library to initiate OAuth process
Google's JS library makes auth request to Google
Google displays confirmation screen to user in pop up window
User accepts
Popup window closes
Google's JS lib invokes a callback in our React/Redux app
Callback provided with authorization token and profile info for user

Figure 5.8: OAuth flow

```
        this.auth = window.gapi.auth2.getAuthInstance();
        this.onAuthChange(this.auth.isSignedIn.get());
        this.auth.isSignedIn.listen(this.onAuthChange);
      })
    });
    this.setState({LO: this.props.triggerLO})
  }
```

The global state of the application I mentioned previously is nothing else but the Redux store. To be able to make use of Redux I had to add the the redux and react-redux libraries to the project via npm. As a good practice when using Redux, I created an actions and a reducers folder. The actions folder contains a file with all the actions made through redux. Each action has a type and a payload, though the last one is not mandatory. In the reducers folder there is a file named authReducer.js. This file contains a function that receives as parameters the initial state and an action. Based on the action given an updated version of the initial state is created and returned from the function. There is a second file in the reducers directory simply called index.js. Here all the created reducers should be combined and put together in a single compact object. However I created only one reducer, therefore only one reducer is present in the combineReducer method. After all these are set, in order to access the redux store inside a component we have to go to the components file and write outside the class the mapStateToProps function:

```
const mapStateToProps = (state) => {
  return { isSignedIn: state.auth.isSignedIn };
};
```

This method will, as it name states, map the redux state to the current components props. Then, we export the component using the connect method from react-redux library which will make the connection of the component with the redux store.

```
export default connect(mapStateToProps, { signIn, signOut })(
    ↪ GoogleAuthButton);
```

5.3 Backend and database architecture

In this section we will present the backend structure and the things that were implemented on this side of the application which represents the server, through the use of the Spring Boot framework.

In the resources directory there is the application.properties file where the application's connectivity configurations are set.The ports on which the database and the server will run on are also specified here. In the pom.xml the dependencies needed for the backend are specified. These dependencies refer to integration with the mysql database, hibernate, spring-boot and any plugins necesarry.

```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
</dependency>

<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>6.0.17.Final</version>
</dependency>

<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>4.3.8.Final</version>
</dependency>
```

The only **entity** present on our back end side is the JPage class and the class name stands for journaling page. The entity, table, id, column and all the other annotations found inside the JPage class helps to the actual creation of the jpage table into our mysql database. The table will contain all the details about the journaling page created, from styling to content, so that we are able to reconstruct it when the user accesses his account again and wants to visualize his past entries.

DTOs : JPageDTO, JPageViewDTO,
Builder: JPageBuilder, JPageViewBuilder :

```
public static JPageDTO generateDTOFromEntity(JPage jpage){
       return new JPageDTO(
               jpage.getId(),
               jpage.getGoogleId(),
               jpage.getColorCode(),
               jpage.getFont(),
               jpage.getFontSize(),
               jpage.getVoiceType(),
               jpage.getDate(),
               jpage.getTitle(),
               jpage.getTags(),
               jpage.getContentPage(),
               jpage.getContentTextOnly());
   }

   public static JPage generateEntityFromDTO(JPageDTO jpageDTO){
       return new JPage(
               jpageDTO.getId(),
               jpageDTO.getGoogleId(),
               jpageDTO.getColorCode(),
               jpageDTO.getFont(),
               jpageDTO.getFontSize(),
               jpageDTO.getVoiceType(),
               jpageDTO.getDate(),
               jpageDTO.getTitle(),
               jpageDTO.getTags(),
               jpageDTO.getContentPage(),
               jpageDTO.getContentTextOnly());
   }

}
```

The JPageRepostitory interface extends the JpaRepository which enables us to make use of the full API it contains and its operations which refer to CRUD operations and also sorting of data from the tables of the database. JpaRepository allows us to easily write queries such as the once shown below. The **getAllByGoogleIdAndTagsContaining** query will, as the name says, get all journaling pages with a specific tag that belong to the user that is made the request. For more specific queries **deletePage** can be taken as an example, in this query the question mark values will be replaced by the parameters sent in the same order they were passed, first googleId and second pageId in our case.

```
public interface JPageRepository extends JpaRepository<JPage, Integer> {
    . . .
    List<JPage> getAllByGoogleIdAndTagsContaining(String googleId, String
       ↪ tag);
    . . .
    @Modifying
    @Query(value ="delete from JPage u where u.googleId like ?1 and u.id =
       ↪  ?2")
    void deletePage(String googleId, Integer pageId);
}
```

In the JPageService class we need to make use of the JPageRepository interface and for that a private final instance of the JPageRepository is made inside it. In this service class there is a method for each operation to be performed on the database. These methods return the data from the database, through the use of the DTOs previously discussed, in an object form that is more manageable and easier to be further transmitted. There are three annotations used here, the Service one which marks the class as being a service provider, the Autowired annotation that wires the Repository interface to the current service class and the last one the Transactional annotation which refers to the fact that the operation to be performed inside the method the annotation belongs to is a one sided database operation, in our case the deletion of a page.

```
@Service
public class JPageService {
    private final JPageRepository jpageRepository;

    @Autowired
    public JPageService(JPageRepository jpageRepository) {
        this.jpageRepository = jpageRepository;
    }
    . . .
    public List<JPageViewDTO> findAllByGoogleIdAndTag(String googleId,
       ↪ String tag){
        List<JPage> jpages = jpageRepository.
           ↪ getAllByGoogleIdAndTagsContaining(googleId, tag);

        return jpages.stream()
                .map(JPageViewBuilder::generateDTOFromEntity)
                .collect(Collectors.toList());
    }
    . . .
    @Transactional
    public int deleteByPageId(String googleId, Integer pageId){
        this.jpageRepository.deletePage(googleId, pageId);
        return pageId;
    }
}
```

The JPageController class is the last layer through which a method has to go through for it to be ready for usage and the endpoints to be exposed. The backend side once running it is exposed on the localhost:8080 and the **GetMapping** annotation is used to specify the rest of the url where the result for that method will be available at. For example the result of the **findByGoogleIdAndTag** method will be exposed at the address `localhost:8080/pages/byGI&tag/{googleId}/{tag}` where the googleId and tag variables will be specified by the once requesting the result. Inside the method we actually call the service method corresponding to this operation and return as result all the pages that match the values specified. The controller class makes use of the service class as we have seen and for that a private final instance of the JPageService class is declared at the beginning the controller class. The **DeleteMapping** annotation works in a similar way only that the mapping is different. To perform this operation need to access the `localhost:8080/pages/delete/page/{googleId}/{pageId}` address

```
@RestController
@CrossOrigin
@RequestMapping(value = "/pages")
public class JPageController {

    private final JPageService jpageService;

    @Autowired
    public JPageController(JPageService jpageService) {
        this.jpageService = jpageService;
    }
   . . .
    @GetMapping(value = "/byGI&tag/{googleId}/{tag}")
    public List<JPageViewDTO> findByGoogleIdAndTag(@PathVariable("googleId
        ↪ ") String googleId, @PathVariable("tag") String tag){
        return jpageService.findAllByGoogleIdAndTag(googleId, tag);
    }
    . . .
    @DeleteMapping(value = "/delete/page/{googleId}/{pageId}")
    public int deletePageOfUser(@PathVariable("googleId") String googleId,
        ↪  @PathVariable("pageId") Integer pageId){
        jpageService.deleteByPageId(googleId, pageId);
        return pageId;
    }
}
```

From the front end point of view to access this endpoints and trigger the request we had to create in the src/apis/data folder, the **pageApi.js** file in which all functions that perform a request on the journaling page table are written here. For the **getPagesBy-GoogleIdAndTag** we make a new request where the first parameter is the url where the result of our request will be and we said previously at the controller, this url will be `localhost:8080/pages/byGI&tag/{googleId}/{tag}` where googleId and tag will be parameters passed when calling the method. Below is the snippet of code illustrating what was just explained.

```
function getPagesByGoogleIdAndTag (userGoogleId, tag, callback) {
  let request = new Request(HOST.backend_api + endpoint.get_pages + 'byGI&
     ↪ tag/' + userGoogleId + '/' + tag, {
    method: 'GET',
  });
  RestApiClient.performRequest(request, callback);
}
```

Where HOST is a constant imported from src/apis/host.js file

```
export const HOST = {
  backend_api: 'http://localhost:8080',
};
```

and the endpoints starting value are mentioned in the same file as the request methods:

```
const endpoint = {
  get_pages: '/pages/',
  post_pages: '/pages/',
};
```

5.4 Deployment solutions

For the deployment part I will present the possible approaches for it. The first approach would be to use **Docker**. This consists in creating an image for each part of the application.

Docker uses a client-server architecture [3]. The Docker client talks to the Docker daemon, which does the harder part of building, running, and distributing the Docker containers created. The environments that are run on Docker are hosted in isolation. Containers are launched from Docker images, adding a read-write layer on top of the image as well as a network interface and IP address. The Dockerfiles consists of a list of commands listed in the order in which they are wanted to be executed as this process

[3]https://dzone.com/refcardz/java-containerization?chapter=2

will be done automatically. Those instructions specify the operating system, application artifacts, data volumes, exposing the ports to be used, as well as the commands to run when launching a Docker container.

There should be two dockerfiles in our case, one for the backend and one for the frontend. The docker compose file will know to make use of those docker files and any additional configuration, such as the once necessary for the database exposure, will be added in it. In those Dockerfiles the step by step instructions are mentioned which showcase how the application is going to be run once the docker compose file is executed. This means that if the project is run for the first time the back end will be run first and exposed on the port specified with the .jar used in the project. As for the dockerfile on the front end side this will perform the **npm install** and **npm start** commands. A good example of the how docker compose would end up looking is shown bellow:

```
version: "3"
services:
  backendnew:
    image: "backend"
    build:
        context: \\here comes the path to the backend
        dockerfile: Dockerfile
    ports:
      - "8080:8080"
    depends_on:
      - mysqldb

  mysqldb:
    image: "mysql:latest"
    ports:
          - "3306:3306"
    environment:
          - MYSQL_ROOT_PASSWORD= \\password
          - MYSQL_DATABASE=journalingjourney
    volumes:
          - db_data:/var/lib/mysql

  frontendnew:
    image: "frontend"
    build:
        context: \\here comes the path for the front end directory
        dockerfile: Dockerfile
    ports:
      - "3000:3000"
```

However we want to offer scalability to our system so deploying it using **AWS**[4] **(Amazon Web Services)** is a much more desired solution in this case. AWS provides a seamless and cost-efficient solution and support for a variety of architectures making it a very versatile tool to deploy your application.

We need to acknowledge and point out the advantages of using AWS and how it makes the deployment easier and more efficient. First of all AWS is cost efficient as it can provide additional servers when needed by specifing and adjust the capacity. Also scalability solution are provided to handle unexpected traffic peaks. AWS uses an automatic scaling approach based on actual traffic trends. Moreover it offers caching within the application which means it caches frequently used information and therefore imporves the performance of the application. AWS offers both relational and NoSQL database infrastructure which just adds to show the flexibility of this service. The differences between the AWS Cloud architecture and the traditional hosting model is that AWS can automatically scale the web application fleet on demand to handle changes in traffic. In addition, AWS enforces a more secure model, in which every host is locked down. An analysis of traffic between hosts needs to be made beforehand in order to decide on the ports that will be necessary to be opened.

Amazon web services also has a suite of cloud services to support every single part of software development cycle. S3[5] is one of those services that is very straight forward. Moreover there is a cli tool built by a community member for easily deploying react applications to the Amazon S3 service called s3cmd, making the process even more easy.

Also AWS Elastic Beanstalk[6] is another service provided by Amazon that automatically handles the deployment once you upload the code. As for the database Amazon RDS[7] can be used which supports MySQL Community Edition versions 5.5, 5.6, 5.7, and 8.0.

[4]https://d1.awsstatic.com/whitepapers/aws-web-hosting-best-practices.pdf
[5]https://blog.bitsrc.io/8-react-application-deployment-and-hosting-options-for-2019-ab4d668309fd
[6]https://aws.amazon.com/elasticbeanstalk/
[7]https://aws.amazon.com/rds/mysql/

Chapter 6

Tests and Results

In this chapter we will present the methods used to test our application, what was used for it and how did we write those tests. Also an analysis of the user testing made on this application will be discussed along with other testing methods such as manual and performance testing.

6.1 User Testing

For the current project a did a small scale user testing in which five users were involved. Of course the users had to know basic english to understand the buttons and tabs labels and were interesting in writing in general which made them part of the targeted users for this application. During the testing the users were asked to perform different actions and go through several basic flows of the application. At the end of the testing each user was asked to grade each criteria mentioned in table 6.1 with a grade from 1 to 5, where 5 represented the highest level of satisfaction.

Overall the as we can see from the table, the users were satisfied with most of the criteria that required evaluation. However considering the small number of users that took part in the testing we have to consider even the smallest dissatisfaction with some of the features in order to think of further improvements and predict possible risks.

We notice that the areas that require attention are : speed, navigation and adding a visual asset to the page. However when looking at the grades given to the 'adding an asset to the page' we can see that only one user graded this with 3 while all the others gave a maximum grade. When the user was asked why he gave this grade he said the reason was the inability to start a search for an asset on 'Enter' key press. After the user testing this was made available, the users being able to hit enter now and search for the desired assets.

The last two more serious criteria to analyse are speed and navigation. Speed can be achieved by a more strategic structure of the code and rendering of the components on the front end side while concerning navigation a more intuitive design might help. Also

Criteria	User 1	User 2	User 3	User 4	User 5	Avarage
Navigation	5	5	4	4	5	4.6
Ease of use	5	5	4	5	5	4.8
Design	5	5	5	5	5	5
Intuitive UI	5	5	5	4	5	4.8
Speed	4	4	5	4	5	4.4
Login process	4	5	5	5	5	4.8
Adding an asset to the page	5	5	3	5	5	4.6
Saving the page	5	5	5	5	5	5
Adding a tag to the page	5	5	5	5	4	4.8
Going to the settings section	5	5	5	5	5	5

Table 6.1: User Testing

am initial presentation tour the first time you log in would be helpful for the user to get accustomed with the app's features.

6.2 Manual Testing

Manual testing was performed throughout the development of the application. This was achieved by manually following a sequence of steps and introducing data in different fields as necessary for performing a certain task. To perform a successful manual testing the requirements of the system must be thoroughly understood and the expected behaviour to be well known by the one performing the testing. Manual testing ensures the application offers a satisfactory user experience and that the expected quality for such an application is met. In case bugs are found those are noted down and a fix is required in order to achieve the desired outcome and behaviour.

In table 6.2 a test case is presented in detail in order to get a better understanding of the flow required for such a testing.

6.3 Functional Tests

Functional testing's main goal is to test the functionalities of the system and its quality. This is also so called 'black box' testing as we test the functions of the application by giving appropriate inputs and checking that the output is the expected one.

No	Action	Expected result
1	Opening the webpage	The home page is visible
2	Click the Login button from the right corner of the navigation bar	The Google login confirmation pop up appears
3	The user fills in the credentials for his Google Account	User is redirected dashboard
4	User clicks on the New button from the left menu in order to create a new page	The page edit mode is displayed on the left
5	User clicks on the GIFs button	GIFs searching modal is displayed
6	User inserts a keyword in the input area and presses Enter	GIFs found corresponding to the searched word are displayed inside the modal
7	User selects the desired GIF	On hover the gifs have pink background
8	User presses the Ok confirmation button	Modal closes and the selected gif appears on the page

Table 6.2: Manual Testing - Adding visual assets on the journaling page

6.3.1 Frontend testing

For the frontend I used **jest** framework for testing. Jest was already configured when we created our react-app project so there is no further configuration needed. Tests are written in files with the .spec.js extension identifying these files as test files. All tests are run by executing the **npm test** command in the console, opened at the location of our frontend project.

Tests begin with a describe function that contains as first parameter a string and as the second one a function. The string denotes the title of the test suit and helps us group tests in scopes. Inside the describe's function the test are written. Test can be written using **it** or **test** functions as they both mean the same thing. I chose to wrote the tests using the 'test' function declaration. The function contains two parameters similar to the describe function. The first parameter is the test title while the second parameter is a function that contains the test body.

The expect function helps us validate that different values match certain conditions depending on what we are testing. Usually what is worth testing on the front end are the functions and peaces of code that contain specific login and more complex computations as well as certain state changes.

```
describe('helper journal page', () => {
  test('removeTag', () => {
    const tags = ['cat', 'dog', 'flowers'];
    const result = ['dog', 'flowers'];
    expect(removeTag('cat', tags)).toEqual(result);
  });
  test('getDateToStringFormat', () => {
    expect(getDateToStringFormat()).toEqual('2020-06-29');
  });
});
```

6.3.2 Backend testing

To be able to write tests on our Spring Boot application we had to add some dependencies in the pom.xml file. Most developers use spring-boot-starter-test module. This module contains the following libraries: JUnit 4, Spring Test, Spring Boot Test, AssertJ, Hamcrest, Mockito, JSONassert and JsonPath.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
</dependency>
```

For the backend side some unit tests were written in order to verify the JPageService methods that imply operations on the database JPage table. Bellow are two tests, the first one aims to verify if the insertion of a journaling page is performed correctly while the second one checks that given a user id and a tag, the specific page is successfully retrieving the pages of that user containing that also contain the given tag, this way performing a filtering of his pages by tag.

```
@Test
public void insertDTOGood() {
    Date date = new Date();
    JPageDTO jpageDTO = new JPageDTO();
    jpageDTO.setGoogleId("11111");
    jpageDTO.setColorCode("#080787");
    jpageDTO.setFont("Comic Sense");
    jpageDTO.setFontSize(16);
    jpageDTO.setVoiceType("UK English Man");
    jpageDTO.setDate(date);
    jpageDTO.setTitle("Test");
    jpageDTO.setTags("flowers, unittest, memories");
    jpageDTO.setTags("What a beautiful day ! <img src='' />");
    jpageDTO.setTags("What a beautiful day !");
    Integer id = jpageService.insert(jpageDTO);
    JPageViewDTO newjpage = jpageService.findJPageById(id);
    assert(!jpageDTO.equals(newjpage));
}

@Test
public void findPageByTag() {
    Date date = new Date();
    JPageDTO jpageDTO = new JPageDTO();
    jpageDTO.setGoogleId("11111");
    jpageDTO.setColorCode("#080787");
    jpageDTO.setFont("Comic Sense");
    jpageDTO.setFontSize(16);
    jpageDTO.setVoiceType("UK English Man");
    jpageDTO.setDate(date);
    jpageDTO.setTitle("Test");
    jpageDTO.setTags("flowers, unittest, memories");
    jpageDTO.setTags("What a beautiful day ! <img src='' />");
    jpageDTO.setTags("What a beautiful day !");
    Integer id = jpageService.insert(jpageDTO);
    JPageViewDTO newjpage = jpageService.findJPageById(id);
    List<JPageViewDTO> result = jpageService.findAllByGoogleIdAndTag("
       ↪ 11111", "unittest");
    List<JPageViewDTO> resultF = new ArrayList<JPageViewDTO>();
    resultF.add(newjpage);
    assert(!resultF.equals(result));
}
```

6.4 Performance Tests

Performance testing focuses on analysing the speed, response time, reliability and resource usage of an application. This type of testing helps us identify potential bottlenecks and defects in our system. Performance testing wants to determine the speed of the application under test, its scalability which means the maximum users supported at the same time and also its stability under a variety of loads.

In the case of the current system the performance varies depending on the request performed and also on the hardware capabilities of the machine on which the system is run on. The system was tested on a Dell machine, Intel(R) Core(TM) i7-4510U CPU, 2.60GHz, with 8GB RAM and a Windows 10 (64-bit) operating system. The time taken by a request made upon the database between 2 and 4 seconds. The requests for fetching the visual assets such as videos, images, gifs and stickers take between 3 and 6 seconds. Concerning loading components, the longest time to load is taken by the journaling page component as it may take up to 10 seconds to load in its entirety. Similarly the login process is prolonged as the application needs to connect to google services and authorise the credentials.

Chapter 7

User Manual

This chapters includes the necessary installation steps needed to set up the Journaling Journaling web application. Steps about how to run the application will also be provided along with a walkthrough of all the app's functionalities and their correlation to the user interface.

The software required for the application will be the following:

- InteliJ IDEA version 2019.2.3
- Java version 1.8.0
- MySQL Workbench version 8.0.17 CE
- NodeJS version 10.16.3

As for the hardware required, the application being a web application it needs to run in a browser and so any device that supports a web browser and is also capable to support the software mentioned previously, is suitable for the running and setting of the application.

7.1 Installation of the necessary resources

In order to correctly run the web application we need to install and configure the following software tools that we have also enumerated in the previous section.

1. Installing the Java kit The Java JDK can be downloaded from the following link `https://www.oracle.com/java/technologies/javase-downloads.html` from where Java SE 8u251 will be downloaded.

2. Installing InteliJ IDE The installation of the InteliJ IDE can be achieved by running the downloaded executable specific for your machine from the following web address `https://www.jetbrains.com/idea/download/`.

3. Installing NodeJS The NodeJS installer for all machines can be found at the following address `https://nodejs.org/en/download/`.

4. Installing MySQL Workbench For the MySQL database Workbench the installer for Windows machine can be found at the following address `https://dev.mysql.com/downloads/windows/installer/` and for the other operation systems at the following address `https://dev.mysql.com/doc/workbench/en/wb-installing.html`. Other in depth installation steps necessary can be found at the provided links as well. Mainly the custom installation is all that is needed and the installation path can be kept the default one suggested by the installer.

7.2 Locally cloning the project's repository

The project and it's source code can be found in its entirety in the BitBucket repository that can be accessed at the following link `https://bitbucket.org/paula_hreniuc/journalingapp/src/master/`. The repository can be downloaded or cloned by copying HTTPS and running the command **git clone** followed by the url copied previously.

7.3 Running the project

To start the backend first we need to import the project in InteliJ IDE and we do this by selecting "File -> New Project -> Project from existing source" and then selecting the location where the project repository was cloned or downloaded and selecting the back-end folder. Then we need to run the SpringDemoApplication class. At this point the back end is running.

To start the front end we need to go in the project's directory and into the journaling-app directory where the front end part is. To start it we need open up a terminal at this exact location and run the command **npm start**. Depending on what browser you have set as default on your computer the application will open in that specific browser at the address "http://localhost:3000".

7.4 User Manual

1. Home Page

   When the application first starts the user will see the home page 7.1 which is for presentation purposes and it is an introduction to the app, describing its main features and aim. The home page has a navigation bar that is sticked at the top. Pressing the links from the bar will smoothly scroll down the page to that specific section referred to in the navigation. We can observe the about section in the figure 7.2. To

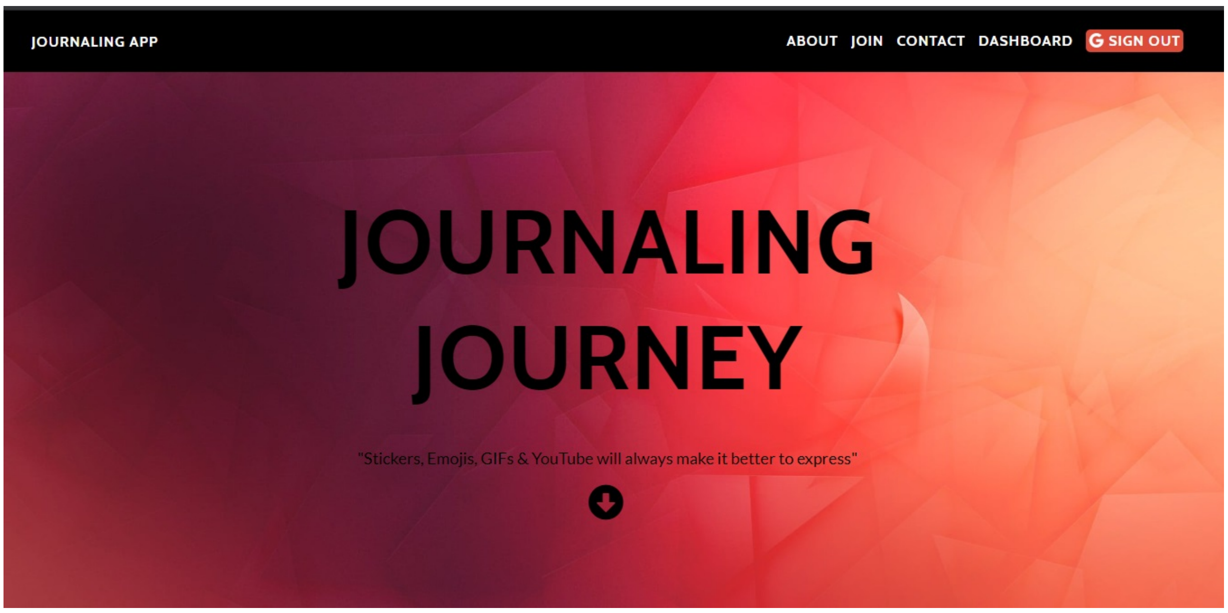

JOURNALING APP
ABOUT
JOIN
CONTACT
DASHBOARD
SIGN OUT
JOURNALING JOURNEY
"Stickers, Emojis, GIFs & YouTube will always make it better to express"

Figure 7.1: Home Presentation Page

reach it we can either press the About link from the top navigation or simply scroll down to it.

2. Login

   When first running the application and the user is not yet signed in the top right button will have the ”Sign in with Google” label as the figure 7.3 shows. Once the user is signed in the button labelling will turn into ”Sign out”.

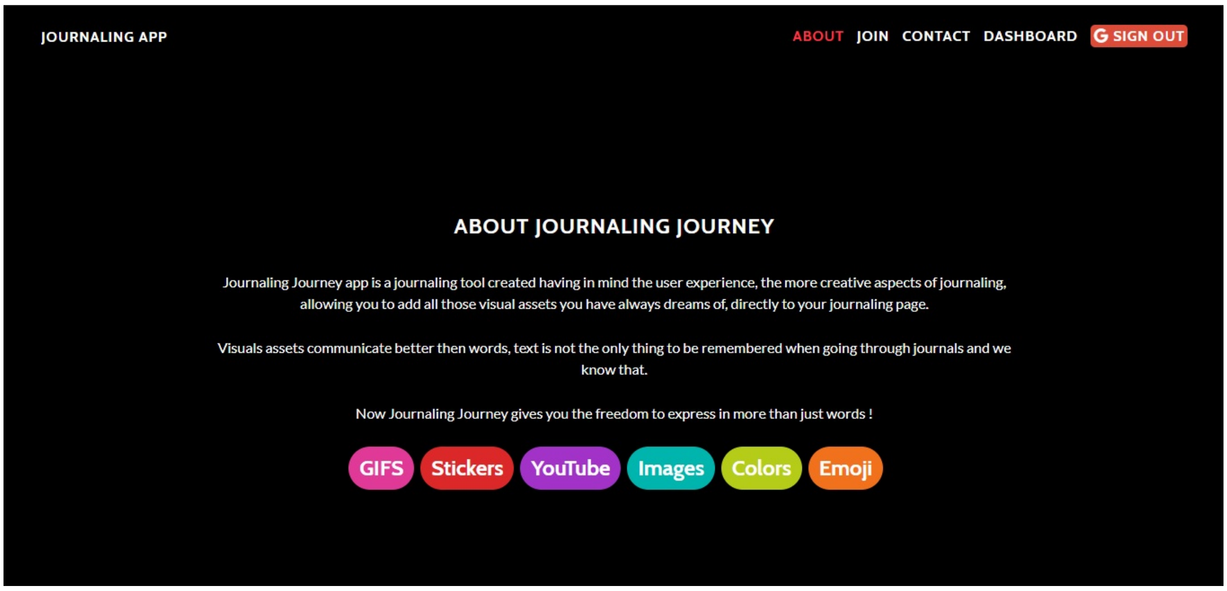
3. Dashboard

   Once the user is logged in he is redirected to the dashboard view 7.4 Here he can view the entries he already had saved, can perform searches by tag and date using the top search field and can select specific pages to be deleted or edited.

4. Creating a new page

   Using the left menu we can navigate to the New Page View by 7.5 pressing the ”New” link. Once in this view we can see that there is a top menu 7.6 that gives quick access to adding visual assets and to audio rendition of the text on the page. Speaking of text, it can be seen that there is also a text area where all the writing and assets can be added, along with a second menu in between the first menu bar and the text area. This second menu 7.7 is for more specific actions such as adding tags to the page, saving the page to your library and formatting portions of text.

   An image with each of the pickers from the top menu is attached to this user manual so they can be easily identified: the font and emoji pickers 7.8, the color picker 7.9,


JOURNALING APP

ABOUT JOIN CONTACT DASHBOARD SIGN OUT

ABOUT JOURNALING JOURNEY

Journaling Journey app is a journaling tool created having in mind the user experience, the more creative aspects of journaling, allowing you to add all those visual assets you have always dreams of, directly to your journaling page.

Visuals assets communicate better then words, text is not the only thing to be remembered when going through journals and we know that.

Now Journaling Journey gives you the freedom to express in more than just words !

GIFS Stickers YouTube Images Colors Emoji

Figure 7.2: About Section On The Home Page

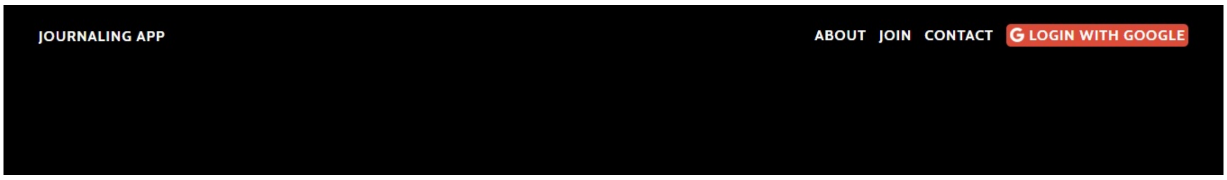
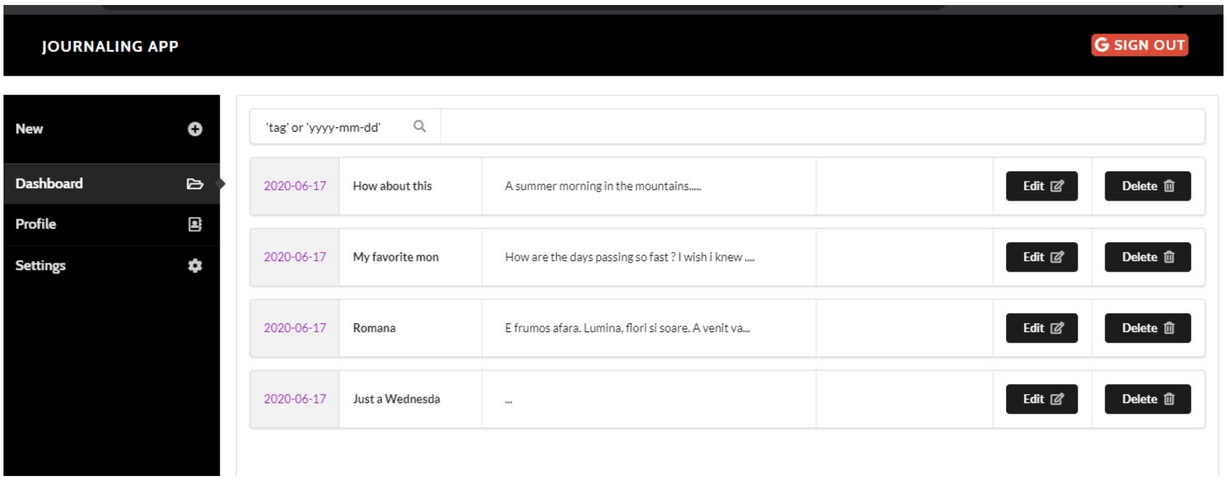


Figure 7.3: Login Button


JOURNALING APP

SIGN OUT

New

Dashboard

Profile

Settings

'tag' or 'yyyy-mm-dd'

2020-06-17	How about this	A summer morning in the mountains.....	Edit	Delete
2020-06-17	My favorite mon	How are the days passing so fast ? I wish i knew	Edit	Delete
2020-06-17	Romana	E frumos afara. Lumina, flori si soare. A venit va...	Edit	Delete
2020-06-17	Just a Wednesda	...	Edit	Delete

Figure 7.4: Dashboard Tab

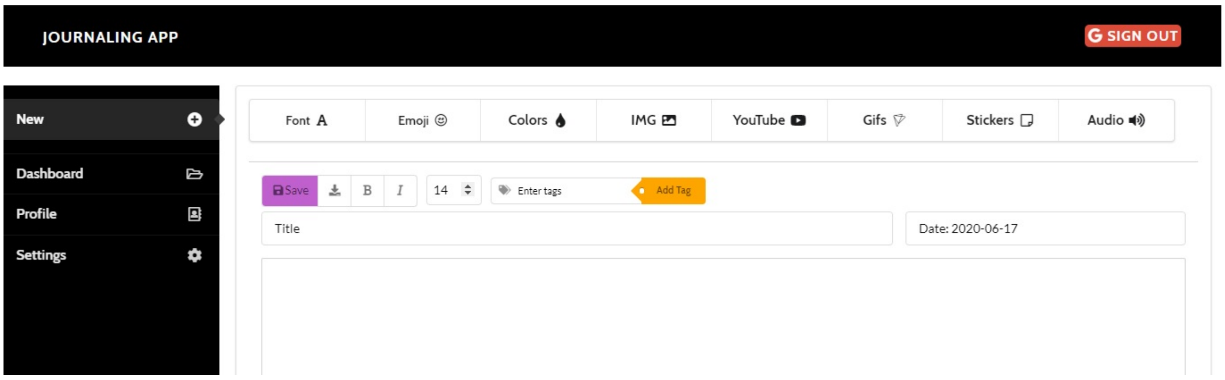


Figure 7.5: New Tab

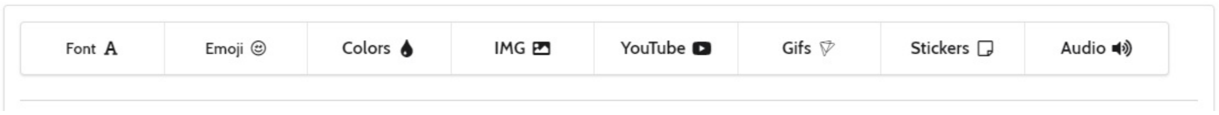


Figure 7.6: Menu bar

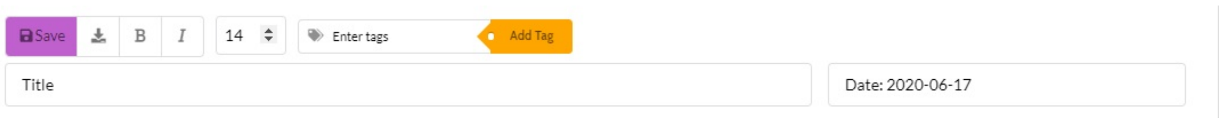


Figure 7.7: Easy access tools

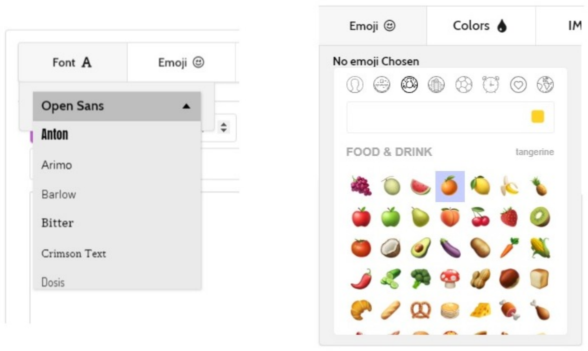


Figure 7.8: Fonts and Emojis Pickers

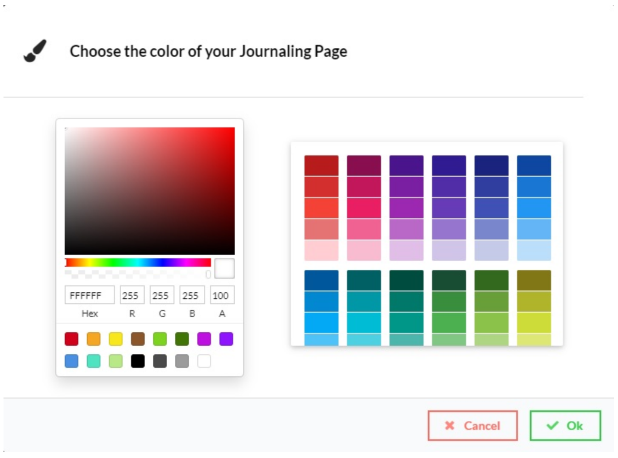


Figure 7.9: Colors Picker Menu

the youtube picker 7.12, the gifs picker 7.10 and finally the sticker picker 7.11.

5. Profile tab

   By navigating to the profile tab 7.13 the user can visualise the information he is logged in with. This information is readonly and can not be edited.

6. Settings tab

   In the settings section 7.14 the user is able to change the voice which will be reading the text for the audio rendition or delete his account. These are the two main settings changes available for the user at this point.

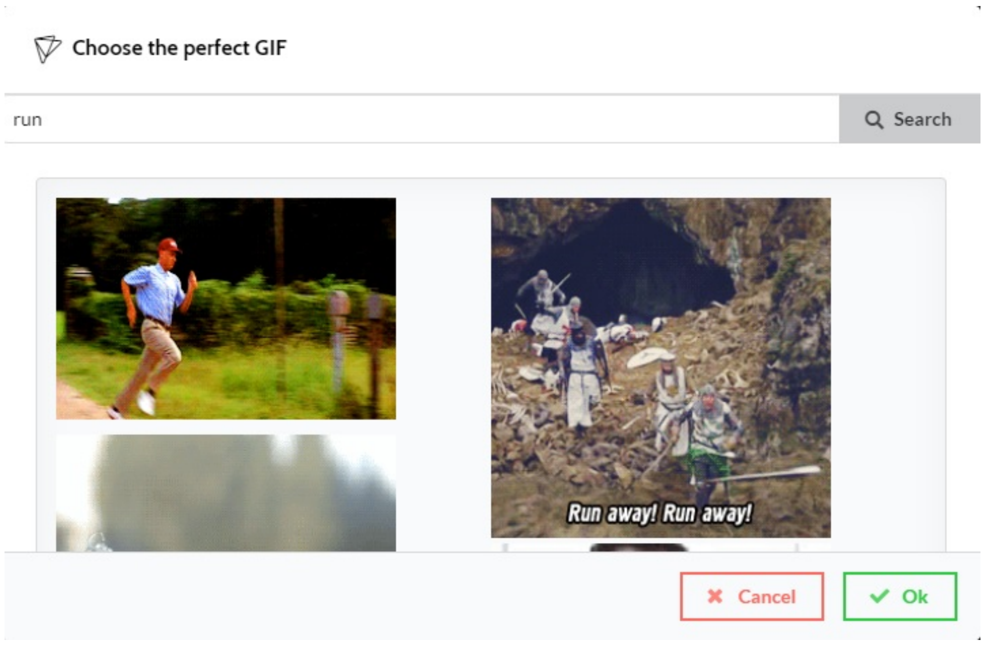


Figure 7.10: Gif Picker Menu

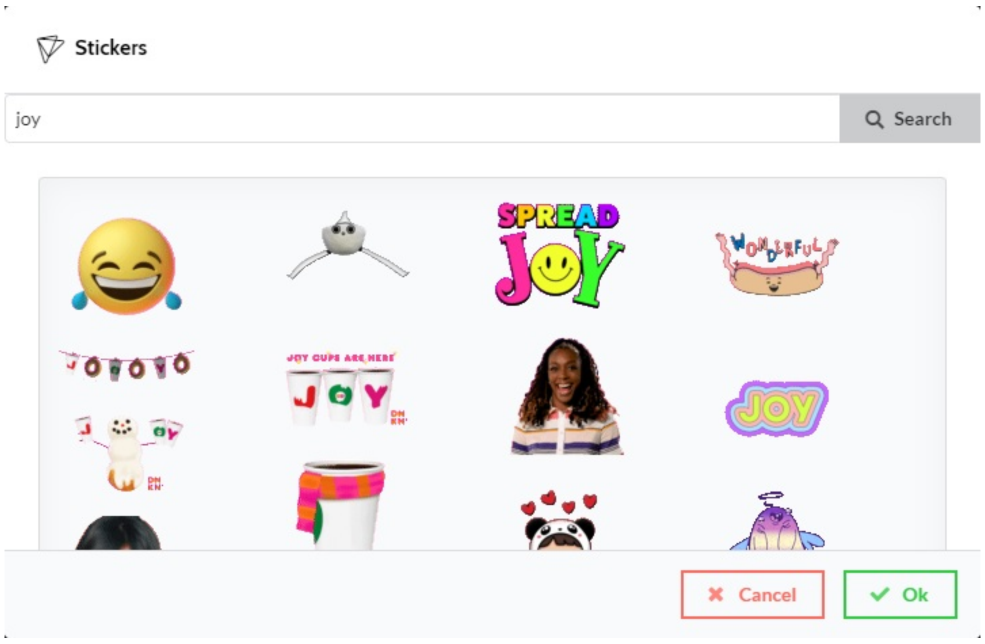


Figure 7.11: Sticker Picker Menu

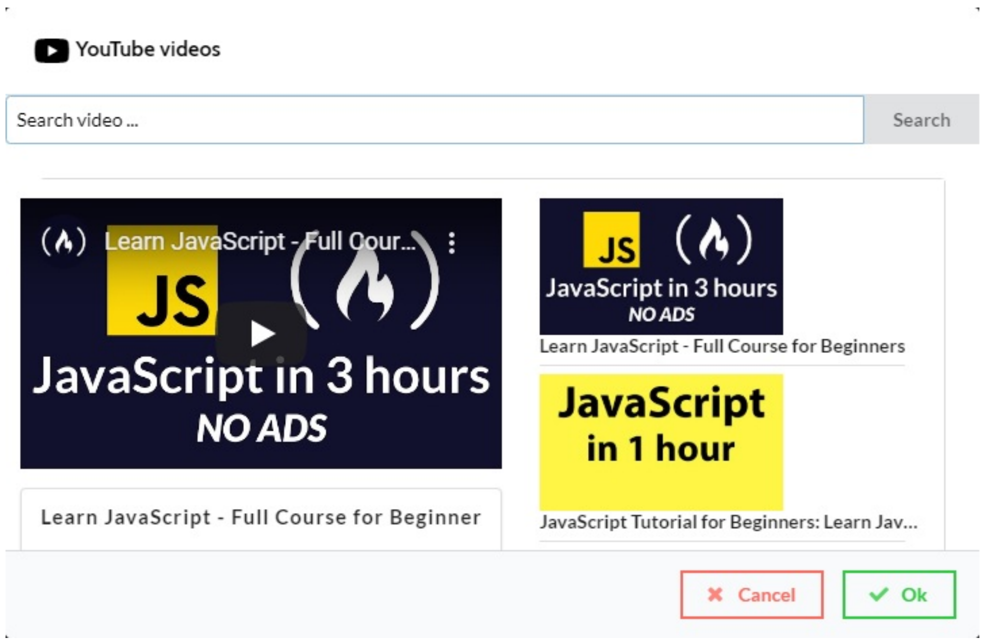
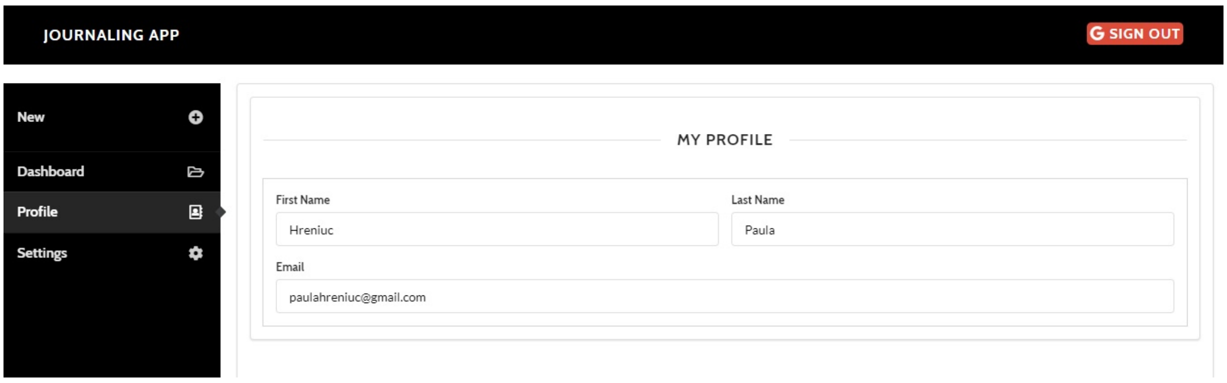

YouTube videos
Search video ...
Search
Learn JavaScript - Full Cour...
JS
JavaScript in 3 hours
NO ADS
Learn JavaScript - Full Course for Beginner
Learn JavaScript - Full Course for Beginners
JavaScript in 1 hour
JavaScript Tutorial for Beginners: Learn Jav...
Cancel
Ok

Figure 7.12: Youtube Picker Menu

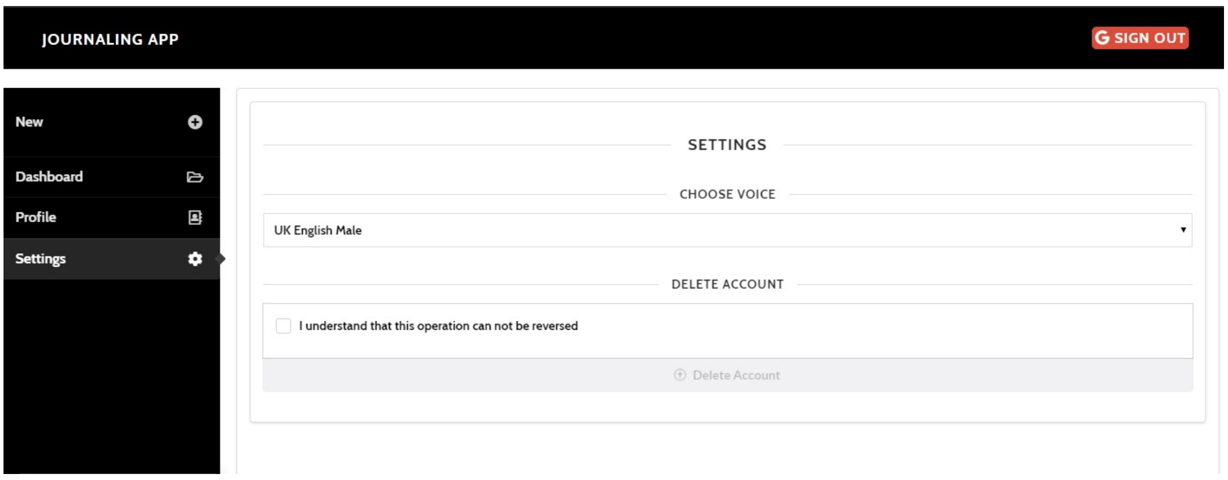


First Name: Hreniuc

Last Name: Paula

Email: paulahreniuc@gmail.com

Figure 7.13: Profile Tab



SETTINGS

CHOOSE VOICE

UK English Male

DELETE ACCOUNT

I understand that this operation can not be reversed

Figure 7.14: Settings Tab

Chapter 8

Conclusions

8.1 Reached Objectives

Looking back on all the stages of bringing this system to life we can say that the initial set objectives were achieved. We created a brand new journaling application and brought an element of newness to it. This refers to the ability to add the famous visual assets found in most social media platform nowadays such as facebook, whatsapp, instagram, viber and many more and those are gifs, stickers, emojis and above those, images and direct access to Youtube for searching videos. Add these elements were made available to be added onto the journaling page and saved for further access. Text to speech audio rendition was made available in order to give the user the possibility to listen to their own written content.

Along the way I got to dive deeper into React and Javascript concepts and saw just how powerful the front end can be and how many things can be integrated on this side of the application. I had to made use of Spring Boot and MySQL database in achieving the full implementation and to separate the scopes of the system.

We performed different types of testing on the resulting system but also throughout the development of the application, which enabled the identification of issues, flaws and possible bottlenecks in performance and therefore to improve. Testing also offered an objective perspective of the things that can be improved on changed and those are mentioned in the following section as ideas for a possible future iteration of the application.

8.2 Further Improvements

Everyone knows that to all existing applications there is still place for improvement and refinement. We learn from mistakes, users feedback and we also have to adapt to the always changing trends. Bellow is a list with some of the things that stood out as points of improvement after the user testing and at the end of this stage of the implementation.

- Create an account from scratch

The current implementation supports creating an account by loging in using your google account. This may come as a limitation to the application as some people do not have a google account or simply do not want to use it for creating an account on the Journaling Journey application. Therefore as a further improvement users should be offered the possibility to create an account by using an email of their choice and a brand new password or at least allow them to login using one of the other famous platform such as Facebook.

- Templates

  Templates are one the features that is present in another journaling application on the market. Those templates come in handy for those that prefer following a certain pattern when writing but would also offers a sense of creativity, diversity and newness to every user who might try to use them. Such templates can keep our users more engaged with the app as templates become more and more innovative.

- Themed backgrounds of the journaling pages

  Currently the for the background of the journaling page the user can select a color, however enabling the user to choose a pattern for his background would offer him even more control and will offer him more freedom to express himself.

- Font color

  The ability to color the text would be another plus along with more powerful text editing and formatting.

- Multiple deletes

  Sometimes multiple entries are wanted to be deleted by a user and the current implementation only allows you to delete one journaling entry at a time. Enabling multiple deletes will offer the user more flexibility and ease of use.

- Speed

  Concerning speed there are some components refactoring that can be made in order to prevent unnecessary rerenders of certain components. Also organising the code and keeping components as lightweight as much as possible helps in achieving a higher speed. However the main speed issues come from unnecessary rerenders, unnecessarily complex logic and request made in less strategic places. Refactoring takes what is already in place but refines and improves it.

- React Native version for IOS and Android support

  React Native is closely related to React, so by using React native for mobile the mobile implementation of the app will not only be much easier but will also support IOS as well as Android devices.

Bibliography

[1] Bc. Marek Foltyn, “Shere - Notes and Document Management Application,”, *Master’s thesis. Czech Technical University in Prague, Faculty of Information Technology*, 2018. Available at: https://dspace.cvut.cz/bitstream/handle/10467/76231/F8-DP-2018-Foltyn-Marek-thesis.pdf?sequence=-1&isAllowed=y.

[2] Carol Azumah Dennis, “Blogging as public pedagogy: creating alternative educational futures,”, *International Journal of Lifelong Education*, TLED 1000408, Volume 34, Issue 3, 24 January 2015. Available at: https://www.tandfonline.com/doi/full/10.1080/02601370.2014.1000408

[3] Moira Dunworth, “Blogging as a reflective journaling tool,”, *Journal of Practice Teaching and Learning*, page 6-21, 2016. Available at: https://www.researchgate.net/publication/270482013_Blogging_as_a_reflective_journaling_tool

[4] Madewell Charles, “New Frontiers in text analytics: Emojis, GIFs and Stickers,”, 2018. Available at: https://www.academia.edu/35973415/New_Frontiers_in_Text_Analytics_EMOJIs_GIFs_and_STICKERS

[5] Steven Curtis, Barrie Axford, Alasdair Blair, Caroline Gibson, Richard HugginsPhilippa Sherrington, “Placement blogging: the benefits and limitations of online journaling,”, ELiSS, ISSN: 1756-848X, Vol 1, Issue 3, April 2019. Available at: https://www.academia.edu/28360582/Placement_blogging_the_benefits_and_limitations_of_online_journaling

[6] Munassir Alhamami, “Observation of YouTube language learning videos(YouTube LLVS),”, *Teaching English with Technology*, 13(3), 3-17, Available at: https://www.academia.edu/188887/Using_an_Online_Journaling_Tool_to_Promote_Elementary_Students_Self_Reflection.

[7] Chris Campbel, Craig Deed, “Using an online journaling tool to promote elementary students self reflection,”, *Readings in Education and Technology: Proceedings of ICICTE*, 2008. Available at: https://www.academia.edu/188887/Using_an_Online_Journaling_Tool_to_Promote_Elementary_Students_Self_Reflection

[8] Zimmerman, Barry, “Becoming a Self-Regulated Learner: An Overview. Theory Into Practice,”, Vol 41, pages 64-70, 2002. Available at: https://doi.org/10.1207/s15430421tip4102_2

[9] Steinmetz K, “Forget Words, a Lot of Millennials Say GIFs and Emojis Communicate Their Thoughts Better Than English,”, 2017. Available at: https://time.com/4834112/millennials-gifs-emojis/

[10] David Flanagan, “JavaScript: The Definitive Guide,”Mike Loukides, 1005 Gravenstein Highway North, Sebastopol, O’Reilly Media, 2011.

[11] Douglas Crockford, “JavaScript: The Good Parts,”Simon St.Laurent, 1005 Gravenstein Highway North, Sebastopol, O’Reilly Media, 2008.

[12] Alex Banks, Eve Porcello, “Learning React: Functional Web Development with React and Redux,”Allyson MacDonald, 1005 Gravenstein Highway North, Sebastopol, O’Reilly Media, 2017.

[13] Felipe Gutierrez, “Pro Spring Boot 2: An Authoritative Guide to Building Microservices, Web and Enterprise Applications, and Best Practices,”Mark Powers, 233 Spring Street, 6th Floor, New York, NY 10013, Apress Media, 2019.