Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Project Report

on

"Cognifeed: Level You Up"

[Course Code: COMP 303]

(For partial fulfillment of III Year/ I Semester in Computer Engineering)

Submitted By:

Sailesh Dahal (10)

Sarayu Gautam (14)

Bishant Bikram Pant (32)

Aashish Pokharel (36)

Shirish Sigdyal (52)

Submitted To

Dr. Gajendra Sharma

Department of Computer Science and Engineering

Submission Date

March 12, 2020

Bonafide Certificate

This project work on

"**Cognifeed: Level You Up**"

is the bonafide work of

"Sailesh Dahal, Sarayu Gautam, Bishant Bikram Pant, Aashish Pokharel, and Shirish Sigdyal"

who carried out the project work under my supervision.

Project Supervisor

_____

Mr. Nabin Ghimire

Lecturer

Acknowledgment

In performing our project, we had to take the help and guideline of some respected persons, who deserve our greatest gratitude. The completion of this project gives us much pleasure. We would like to show our gratitude to **Mr. Nabin Ghimire, Lecturer, Kathmandu University** for giving us a good guideline for this project throughout numerous consultations. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in doing this project.

We would like to thank the Department of Computer Science and Engineering (DoCSE) and the whole university for providing us a chance to work on the project. Many people, especially our classmates and team members themself, have made valuable comments and suggestions on this project which inspired us to improve our project.

Sincerely,
Sailesh Dahal (10)
Sarayu Gautam (14)
Bishant Bikram Pant (32)
Aashish Pokharel (36)
Shirish Sigdyal (52)

Abstract

This semester we have built a mobile application that facilitates users by allowing them to select the topic of their desire and obtain related information collected from various websites daily. Our project named “Cognifeed: Level You Up” uses Flutter as a front-end framework, which uses Dart as the programming language and Node.js as server-side JavaScript runtime and MYSQL as database. The main objective of this project is to provide a platform for people who want to learn about something every day. This application is developed with the motive of providing users a quick refresher of the topics as per their selection. We have used web crawling tools to get the information. This application is targeted to anyone in search of information about any topic.

Keywords: *Crawler, Flutter, Node.js*

List of Figures

Abbreviations

Short-form	Full form
JS	JavaScript
MySQL	My Structured Query Language
SDK	Software Development Kit
OS	Operating System
HTML	Hypertext Markup Language
CSS	Cascading Style Sheet
DOM	Document Object Model
URL	Uniform Resource Locator
BLoC	Business Logic Component
API	Application Program Interface
JWT	JSON Web Token
JSON	JavaScript Object Notation
NPM	Node Package Manager
RAM	Random Access Memory
GB	Gigabytes
UI	User Interface
UX	User Experience

Table of Contents

Chapter 1: Introduction

This chapter discusses the background, objective and motivation related to this project.

1.1. Background

Living in a digital age, our lives have been fully occupied by gadgets and computers. Surfing the internet is regarded as an indispensable part of our work, entertainment and information gathering. Websites and social media have become a powerful tool to facilitate people, to convince them, and to share information. Information and data flow have been indispensable in today's world. Every minute we are being notified about something whether they are necessary for us or not.

We surf the internet for various purposes; mostly for getting quick information about a topic. The Internet is a cheaper means of gaining knowledge as compared to books. Also, people in 2020 are too busy to spend a significant amount of time surfing websites to websites for a quick look at a subject. Wikipedia and other websites have been the most useful sites for looking up information, but they provide very thorough details and contain other links for references that take extra effort to navigate. Most of us do not know which website consists of relevant information about the subject we are searching for. Still, we don't stop surfing the internet for gaining knowledge and knowing different things. The Internet is a vast sea of information and many of them are irrelevant to what we are searching for. Most of the time we end up scrolling different sites randomly without having an actual gain.

So, to make the best use of all the information and knowledge that these websites have collected and provide, we have created an application "**Cognifeed: Level You Up**" that saves the time to surf and provides a daily dose of relevant bits of information as per the selection of topics. This application provides an interface to the user where they can select the topic on which they would want to receive

information daily. Among the topics they have selected, our scrapper will search for different websites that contain those subject matters and will display the information scraped from those websites.

1.2. Objectives

The main objectives of this project are:

1. To provide a platform to get information extracted and refined from different websites.
2. To help users choose the information they want by providing categories to various topics.
3. To make the user resourceful about various subject matters.
4. To act as a quick refresher to a concept.

1.3. Motivation and Significance

We as students of Computer Engineering use the internet mostly for searching for information related to programming and technology. We have a hard time figuring out the sites that have relevant matters as per our needs. Also, we want to be updated on that information and technical knowledge. This becomes difficult with so many websites and limited time. Users like us face the same dilemma while searching for information. This brought the concept of making an application that updates users on subjects they are interested in by taking care of the surfing process and finding out the appropriate website that delivers the content that the user is searching for.
The content may be related to any topic. It may be a page from Wikipedia, a news article from CNN website, documentation from nodejs.org or any blog post article from HackerNoon.

We were inspired by the concept of microlearning and wanted to implement it uniquely. After the target was fixed, we started a discussion on its design, layout, features, and components. When we did some research on this idea and found that it had a notable potential, our team became confident that our idea of the project was worthy of acceptance and immediate response towards its actual development.

Chapter 2: Related Works

As the research for the project proceeded further, we went on to find some projects with goals like our own. Three such projects stuck out as noteworthy each of which is described in brief below.

2.1. Deepstash[4]

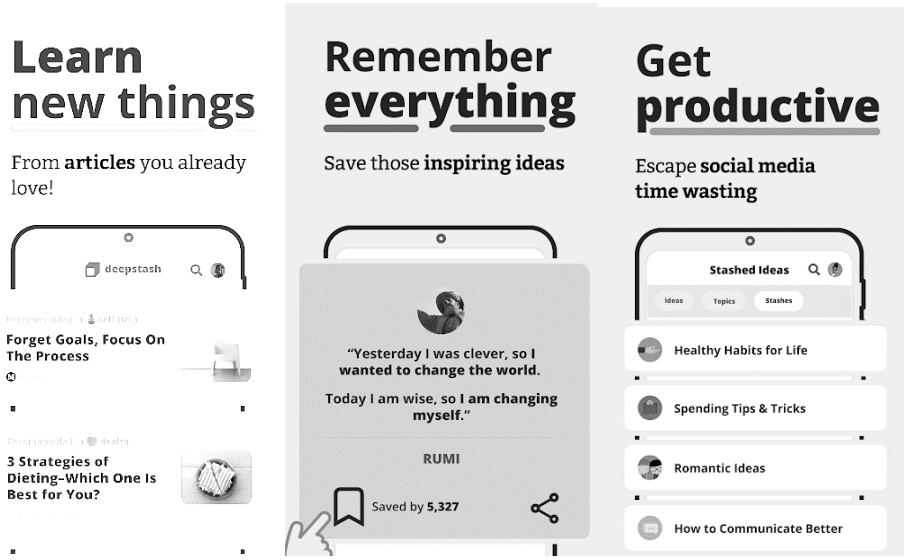

Figure 1: Deepstash

Deepstash [4] is an iOS and Android application created with the intent to provide curated, bite-sized information about any relevant topic from, according to their claim, any website on the web. Deepstash allows the user to create an account and pick at least 3 interests. The categories to choose from are wide including, but not confined to, self-help, motivation, and languages. After the selection of interests, Deepstash shows a dashboard with relevant content related to the interest selected previously. The application is advertised as the following in the Google Play Store: "Escape the negativity and time-wasting pull of social media platforms. Our goal is to help you towards self-improvement and self-care and to get you inspired, wiser and productive."

The key features of Deepstash are:

- It allows users to pick from a wide range of interests.
- It provides small bite-sized information to the users for a quick and easy read.
- Presents the information using colorful flashcards for easy digestion.
- Allows users to bookmark or “stash” certain ideas that they like for easy retrieval at a later moment.
- Presents content from multiple knowledge-related websites across the web.

2.2. Flipboard[7]

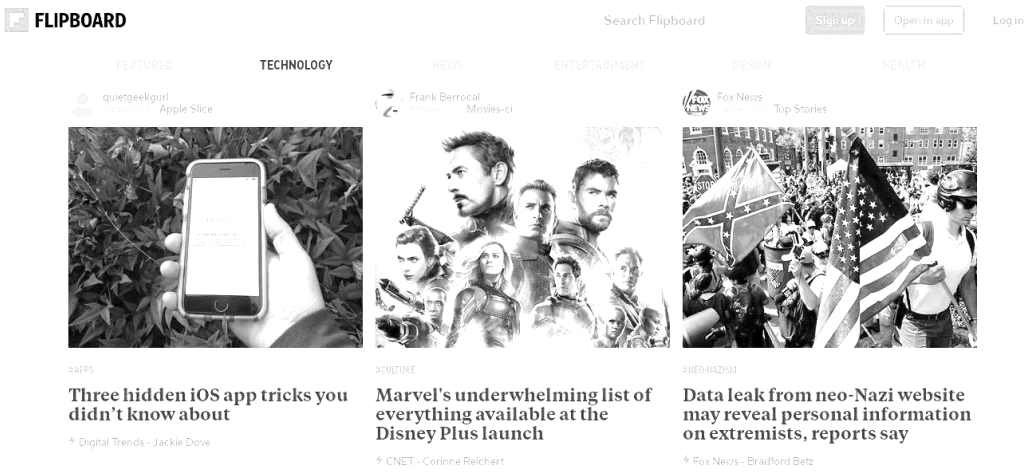

Figure 2: Flipboard

Flipboard[7] is a news and story aggregator website which curates and presents content from multiple news platforms, social media and photo-sharing sites. Flipboard, as the co-founder and CEO Mike Mucce states, “was founded as one place to find the stories for your day, bringing together your favorite news sources with social content, to give a deep view into everything from political issues to technology trends to travel inspiration.”

The key features of Flipboard are:

- It provides the latest stories and regular content updates and aggregations from multiple news and social media sources

- It allows the selection of multiple areas of interest.
- It provides category-based and provider-based selection and search functionality.
- It allows creations of “magazines”, where users can have content delivered to them regularly from the sources they choose.
- It provides “Related Stories” and “Featured” sections for stories that are relevant to the interest of users.

2.3. Google News[10]

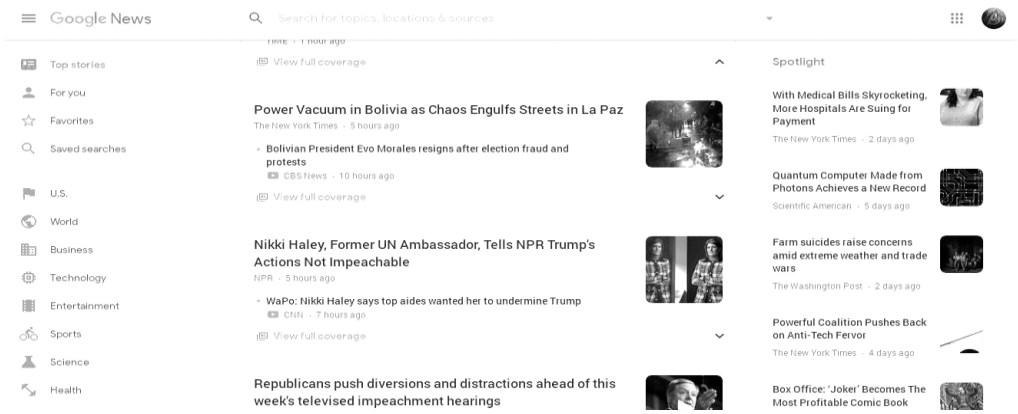

Figure 3: Google News

Google News [10] is a news indexing and aggregating application created as a separate service by Google. Its main objective is to separate and aggregate news stories from multiple news platforms in a single place, unlike the Google search engine which indexes all things web. It too, like the other aggregators discussed above, has categorical sorting and searching and features such as following certain categories of news or sharing specific news pieces.

The key features of Google News are:

- It indexes and aggregates news from all mainstream news-sources such as Bloomberg and CNN.
- It allows the categorical arrangement of news into categories such as “US”, “World”, “Science”, “Technology” and the like.
- It provides users with the features to favorite, follow and share news pieces.

- It provides multiple coverages (from multiple sources) for the same event.
- It identifies “Top Stories”, highlighting the latest and most relevant news articles.

Chapter 3: Design and Implementation

This chapter includes an explanation of the sequential procedure that we have performed during the project. It includes algorithms, flowcharts, system diagrams, and other analysis and design which gives a general idea about how we have approached different development procedures during the project.

3.1. Procedure

The procedure that we have followed throughout the completion of this application is enlisted below:

1. Plan and design the application.
2. Develop an application with the help of a software development kit, flutter and JavaScript runtime environment: Node.js.
3. Add user authentication and display information per the user to make the application user-specific.
4. Allow the user to select different categories on which they want to receive information.
5. Implement a web crawler to crawl different websites.
6. Design purifiers to extract snippets of information about the article specific to a website.
7. Maintain a server to spawn spiders.
8. Implement the idea in the frontend.
9. Write unit tests to ensure the operation of each unit of the application.
10. Test and debug to avoid errors present.
11. Finally, make the application usable in the real world.

As every project requires a certain level of planning, we started our project with proper planning and its design. Besides, we have selected some robust languages for development. After the completion of the prototype of our application, we have added features according to the needs of the user. Eventually, we tested and debugged the application for its implementation in the real-world scenario.

Cognifeed application is built upon following principles, components, and processes:

3.1.1. Child Process

Single-threaded, non-blocking performance in Node.js works great for a single process. But eventually, one process in one CPU is not going to be enough to handle the increasing workload of an application. In the case of our application, multiple spiders needed to be spawned to scrape multiple websites which compelled us to take advantage of multiple processes that node.js offers. Multiple child processes are executed through the server and each child process takes care of scraping and retrieving data from one website.

```
//Spider Instantiation.
;(function startServer(seeds) {
  let scraperChild

  try {
    scraperChild = fork("start-server.js", {
      cwd: path.join(__dirname, SCRAPER_DIRECTORY)
    })
  } catch (err) {
    return console.log(err)
  }

  process.on("SIGINT", () => {
    scraperChild.kill()
  })

  process.on("SIGTERM", () => {
    scraperChild.kill()
  })
})()
```

3.1.2. Spider

Web scraping is a way to grab data from websites without needing access to APIs or the website's database. Spiders, like their name suggests crawl on the web. Inside the spider is a class that we define which tells the scrapper what to do. For example, where to start crawling, the types of requests it makes, how to follow links on pages, and how it parses data. We can even add custom functions to process data as well, before outputting back into a file. In Cognifeed, Spiders are a crucial part of getting data from the web. Multiple Spiders are spawned to fetch data from multiple websites.

```
/**
 * Collect all links from within the seed url into a links collecton
 * @returns {LinksCollection}
 */
async getNewLinks() {
  try {
    this._html = await requestPromise.get(this._link.resolve())
  } catch (err) {
    if (err.error.code==="EAI_AGAIN"||err.error.code === "ENOTFOUND") {
     throw new Error(
       "Error fetching new links! Please check the internet connection"
      )
    }
    throw err
  }

  const $ = cheerio.load(this._html)
  $("a").each((i, e) => {
    if ($(e).attr("href") === undefined) return

    let baseURL = this._link.baseURL
    let path
    if ($(e).attr("href").startsWith("http")) {
      const url = new URL($(e).attr("href"))
      baseURL = url.origin
      path = url.pathname
    } else if ((path = $(e).attr("href")).includes("#")) return
    // Filter out relative URLs

    const newLink = new Link(baseURL, path)
```

```
      if (!this._link.matches(newLink))
        this._horizon = this._horizon.addLinks(newLink)
    })
    // eslint-disable-next-line no-undef
    return this._horizon
  }
```

3.1.3. Robots.txt Parser

Data scraping involves increasing the server load for the site that you're scraping, which means a higher cost for the companies hosting the site and a lower quality experience for other users of that site. Most sites have a file called robots.txt in their main directory. This file sets out rules for what directories sites do not want scrapers to access. A website's Terms & Conditions page will usually let you know what their policy on data scraping is. Before we try to obtain a website's data, we should always check out the website's terms and robots.txt to make sure we are obtaining legal data. When building our scrapers, we also need to make sure that we do not overwhelm a server with requests that it can't handle. We have implemented a feature to filter the links obtained from scrapping a website, to check if it is restricted by robots.txt or not.

```
class RobotsParser {
  /**
   * Check whether the given url is scrapable by the optional user-agent
   * @param {Link} link - The url to test for scrapability
   * @param {string} [userAgent = *] - The user-
agent to test for scrapability
   * @returns {boolean}
   */
  isDisallowed(link, userAgent = "*") {
    return this._parser.isDisallowed(link.resolve(), userAgent)
  }

  /**
   * Get the crawl delay of the url for a given user-agent
   * @param {string} [userAgent = *] - The bot to test for
   * @returns {number}
   */
```

```
  getCrawlDelay(userAgent = "*") {
    return this._parser.getCrawlDelay(userAgent)
  }

  /**
   * The constructor for RobotsParser class
   * @param {Link} link - The url to which the robots.txt string belongs
   * @param {string} robotsTXT - The robots.txt string of the url
   */
  constructor(link, robotsTXT) {
    /**
     * The link pbject for which to fetch the robots.txt document
     * @type {string}
     * @private
     */
    this._url = new Link(link.baseURL, "robots.txt").resolve()

    /**
     * The rotots.txt parser from "robots-parser"
     * @type {object}
     * @private
     */
    this._parser = roboParser(this._url, robotsTXT)
  }
}
```

3.1.4. Purifier

We obtain plain HTML after scraping a website by Spider. To obtain any useful information from that HTML, we need to purify/filter it. There are many tools specific to a programming language. In Node.js, we have used Cheerio. Cheerio is a fast, flexible, and lean implementation of core jQuery designed specifically for the server. It purifies the HTML by using jQuery Selectors implementing the DOM manipulation method. We have used Cheerio to obtain the title, description, image-URL and website's name from the article to display it as a snippet of the whole article.

```
class HackerNoonPurifier extends Purifier {
  /**
   *
   * @param {String} html
   * @param {Link} url
   */
  constructor(html, url) {
    super(html, url)
  }
  purify() {
    const $ = cheerio.load(this.html)
    this.title = $(".story-container .title")
      .text()
      .trim()
    this.description = ""
    $(".story-container .content .paragraph").each((i, e) => {
      if (i !== 0 && i < 5) {
        this.description += $(e).text()
      }
    })
    this.description = this.description.substr(0, 500)
    if (
      $(".story .content")
        .find("img")
        .attr("src") !== undefined
    )
      this.image_url = $(".story .content")
        .find("img")
        .attr("src")
  }
}
```

3.1.5. Unit Testing

Unit tests are the fundamental tests in an app testing strategy. By creating and running unit tests against the code, we can easily verify that the logic of individual units is correct. Running unit tests after every build help to quickly catch and fix software regressions introduced by code changes to the app. We have implemented unit testing in the server, spider and purifier classes to ensure that each unit is functioning properly before merging them during app integration.

```
const server = new Server()

describe("Server", function() {
  context("#start", function() {
    let server = new Server()
    it("should fail at the moment as server is not working", function() {
      expect(1).to.equals(2)
    })
  })

  context("#spawnSpider", function() {
    let spider = server._spawnSpider()
    it("should return a promise", function() {
      expect(spider instanceof Promise).to.be.true
    })
    it("should resolve to a Spider object", function() {
      spider.then(value => {
        expect(value instanceof Spider).to.be.true
      })
    })
  })
})
```

3.2. Database Tables

Table	Action	Rows	Type	Collation	Size	Overhead
article	Browse Structure Search Insert Empty Drop	6	InnoDB	utf8_unicode_ci	16 KiB	-
article_tag	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_general_ci	16 KiB	-
favourites	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8_unicode_ci	16 KiB	-
hidden	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16 KiB	-
profile	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_unicode_ci	16 KiB	-
tags	Browse Structure Search Insert Empty Drop	45	InnoDB	utf8mb4_general_ci	16 KiB	-
tag_website	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16 KiB	-
users	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_unicode_ci	16 KiB	-
user_tags	Browse Structure Search Insert Empty Drop	13	InnoDB	utf8mb4_general_ci	32 KiB	-
website	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16 KiB	-
10 tables	Sum	82	InnoDB	utf8mb4_general_ci	176 KiB	0 B

Figure 4: All tables

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	user_id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	email	varchar(150)	utf8_unicode_ci		No	None			Change Drop More
3	password	text	utf8_unicode_ci		No				Change Drop More
4	reset_token	varchar(255)	utf8_unicode_ci		Yes	NULL			Change Drop More
5	receive_notification	int(11)			No	1			Change Drop More
6	created_at	timestamp			No	current_timestamp()			Change Drop More
7	updated_at	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Change Drop More

Figure 5: Users Table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	user_id	int(11)			No	None			Change Drop More
2	name	varchar(255)	utf8_unicode_ci		No	None			Change Drop More
3	phone	varchar(15)	utf8_unicode_ci		No	None			Change Drop More
4	bio	varchar(255)	utf8_unicode_ci		Yes	NULL			Change Drop More
5	image_url	varchar(255)	utf8_unicode_ci		Yes	public/images/profile.png			Change Drop More
6	website	varchar(255)	utf8_unicode_ci		Yes	NULL			Change Drop More
7	about	text	utf8_unicode_ci		Yes				Change Drop More
8	address	varchar(255)	utf8_unicode_ci		Yes	NULL			Change Drop More

Figure 6: Profile Table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	article_id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	title	varchar(255)	utf8_unicode_ci		No	None			Change Drop More
3	description	text	utf8_unicode_ci		No				Change Drop More
4	website	varchar(150)	utf8_unicode_ci		No	None			Change Drop More
5	image_url	varchar(255)	utf8_unicode_ci		No	None			Change Drop More
6	link_url	varchar(255)	utf8_unicode_ci		No	None			Change Drop More
7	view_count	int(11)			No	0			Change Drop More

Figure 7: Article Table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	tag_id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	tag_name	varchar(255)	utf8mb4_general_ci		No	None			Change Drop More

Figure 8: Tags Table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	user_id	int(11)			No	None			Change Drop More
2	tag_id	int(11)			No	None			Change Drop More

Figure 9: User-Tags Table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**article_id**	int(11)			No	*None*			Change Drop More
2	**tag_id**	int(11)			No	*None*			Change Drop More

Figure 10: Article-Tag Table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**user_id**	int(11)			No	*None*			Change Drop More
2	**article_id**	int(11)			No	*None*			Change Drop More

Figure 11: Favorites Table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**user_id**	int(11)			No	*None*			Change Drop More
2	**article_id**	int(11)			No	*None*			Change Drop More

Figure 12: Hidden Table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**website_id**	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
2	**link_url**	varchar(255)	utf8mb4_general_ci		No	*None*			Change Drop More

Figure 13: Website Table

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**tag_id**	int(11)			No	*None*			Change Drop More
2	**website_id**	int(11)			No	*None*			Change Drop More

Figure 14: Tag-Website Table

3.3. Database Entity Relationship Diagram

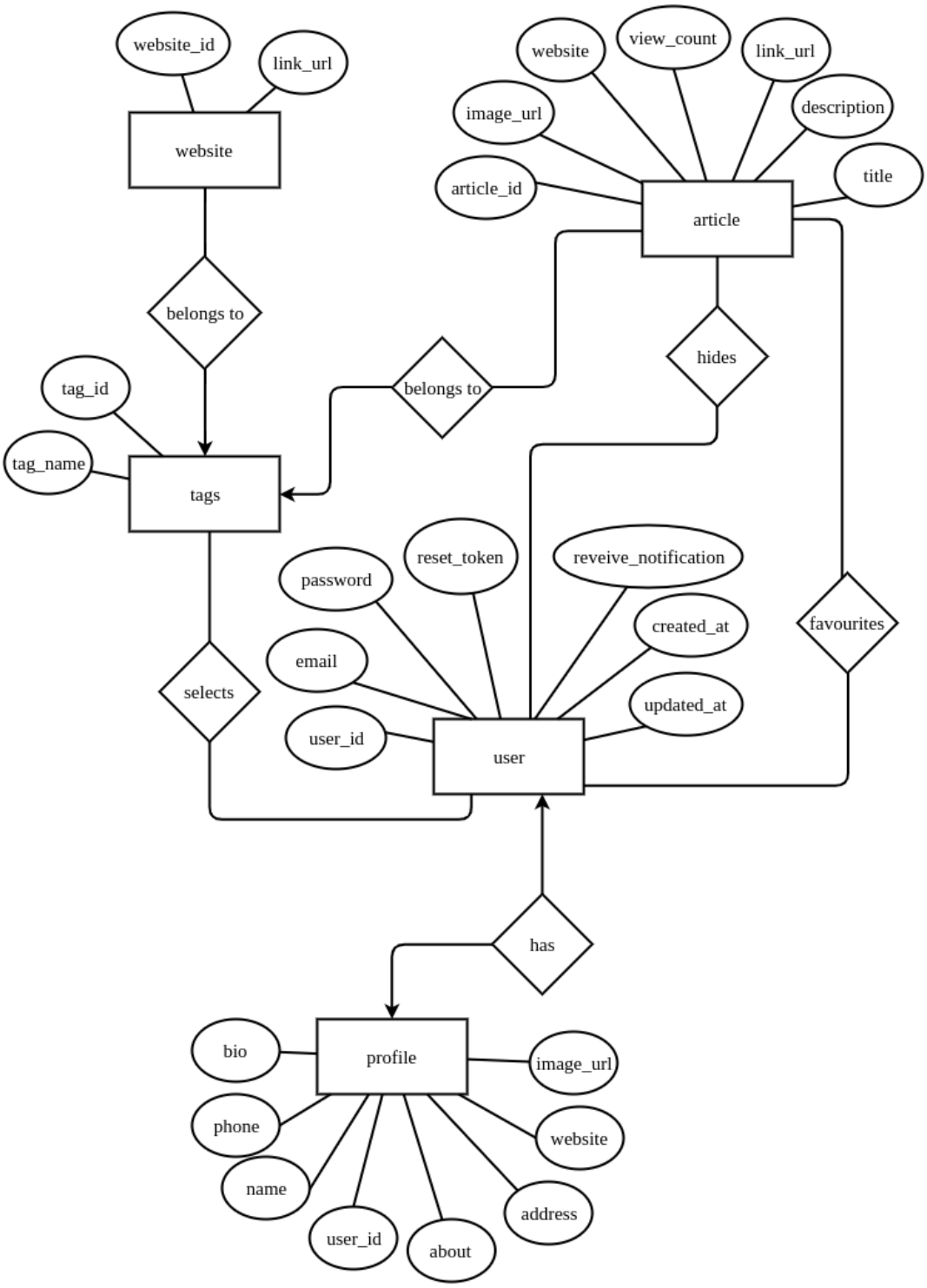

Figure 15: Entity Relationship Diagram

3.4. Flowchart

This includes flowcharts for the frontend and the scrapper of the application.

3.4.1. Front End Flow Chart

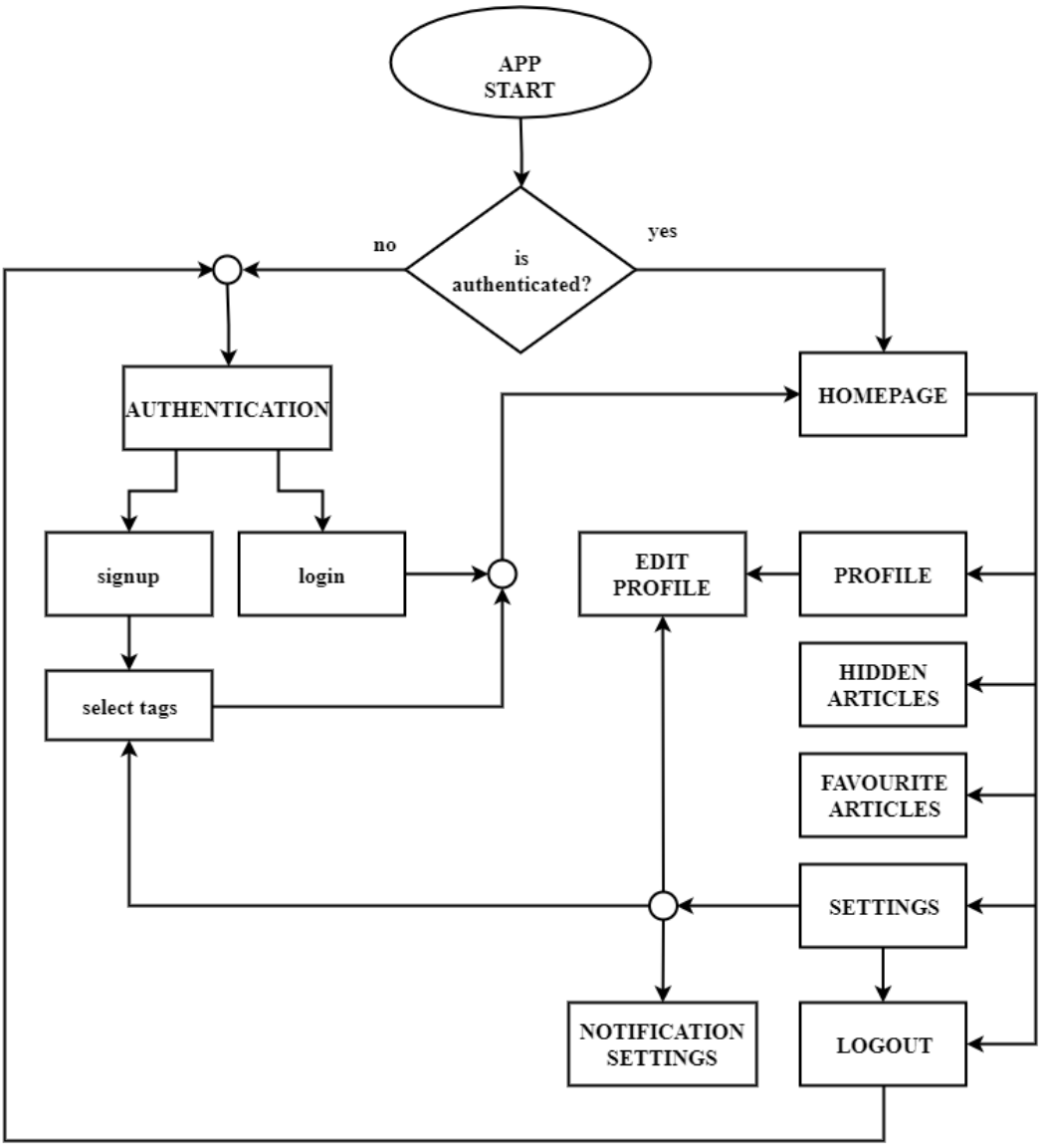

Figure 16: Front-end Flowchart

3.4.2. Scrapper Flow Chart

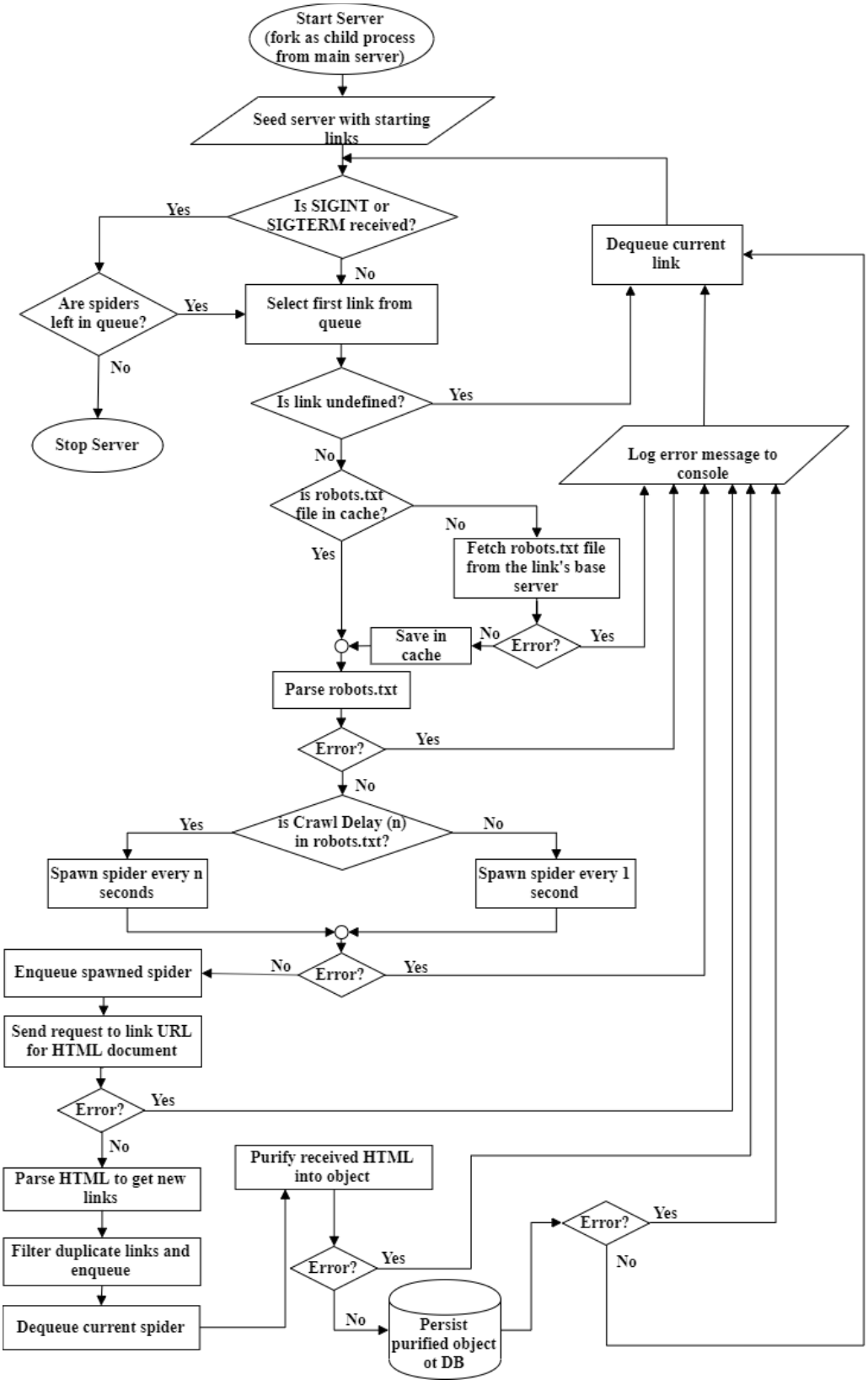

Figure 17: Scrapper Flowchart

3.5. Block Diagram

This includes API and application block diagrams.

3.5.1. API Block Diagram

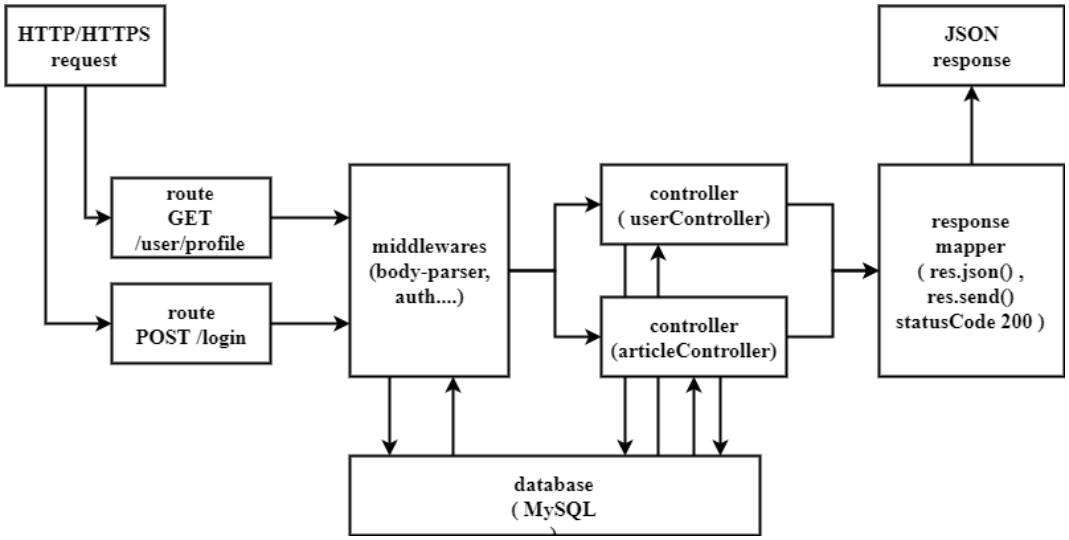

Figure 18: API Block Diagram

3.5.2. Application Block Diagram

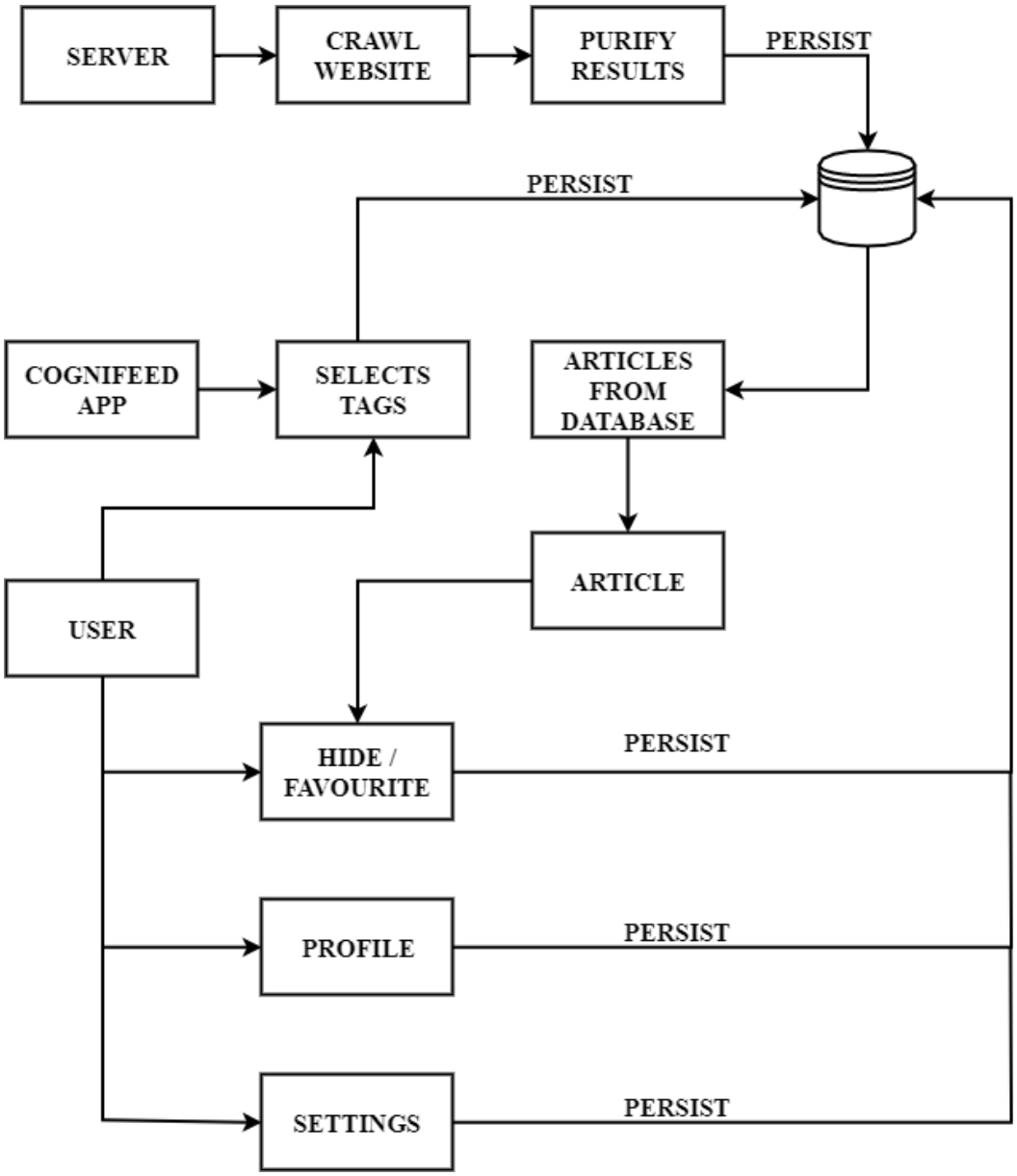

Figure 19: Application Block Diagram

3.6. Use Case Diagram

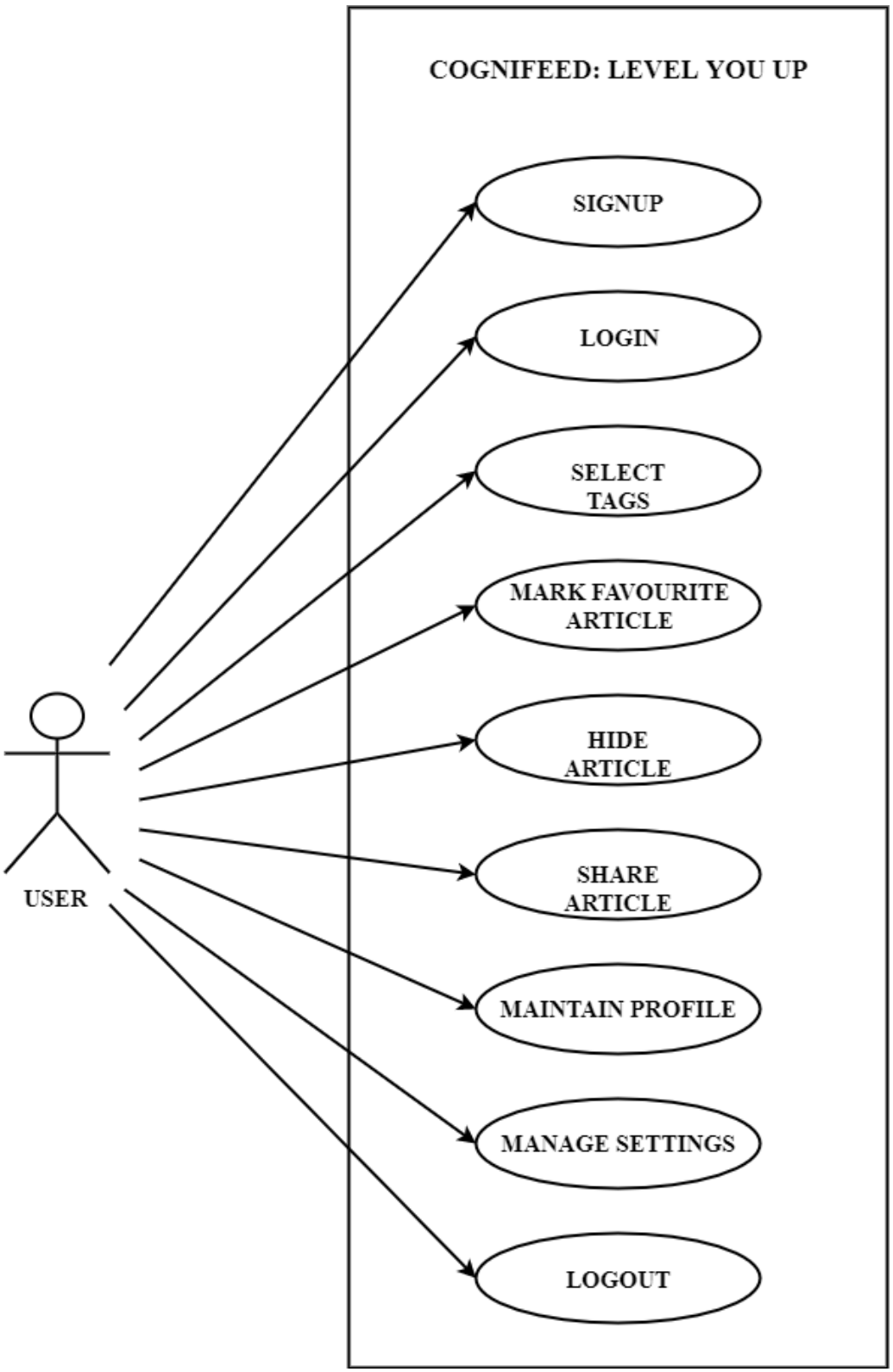

Figure 20: Use Case Diagram

3.7. System Requirement Specification

3.7.1. Software Requirement

It contains information about OS, programming platform and other APIs. It also contains information about the technique used to store and manipulate data.

3.7.1.1. Front End Tool

The front-end tools used for this project include Flutter.

3.7.1.1.1. Flutter SDK [9]

Flutter [9] is an open-source mobile application development framework created by Google. It is used to develop applications for Android and iOS. Flutter apps are written in the Dart language and make use of many of the language's more advanced features**.**

- **BLoC Architecture [1]**

BloC[1] is an architecture pattern introduced by Google. The idea behind it is to have separated components (BLoC components) containing only the business logic that is meant to be easily shared between different Dart apps.

3.7.1.2. Back End Tool

The back-end tools used for this project include Node.js, MySQL database management system.

3.7.1.2.1. Node.js [14]

Node.js [14] is a very powerful JavaScript-based framework/platform built on Google Chrome's JavaScript V8 Engine. It is used to develop I/O intensive web applications like video streaming sites, single-page applications, and other web applications.

- Cheerio [3]

Cheerio [3] parses markup and provides an API for traversing/manipulating the resulting data structure. It does not interpret the result as a web browser does. Specifically, it does not produce a visual rendering, apply CSS, load external resources, or execute JavaScript.

- Bcrypt.js [13]

The bcrypt [13] library on NPM makes it easy to hash and compare passwords in Node.

- JWT [2]

JSON Web Token (JWT) [2] is a JSON encoded representation of a claim(s) that can be transferred between two parties. The claim is digitally signed by the issuer of the token, and the party receiving this token can later use this digital signature to prove the ownership of the claim.

- Express [6]

Express [6] is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

3.7.1.2.2. MySQL [15]

MySQL [15] is an open-source relational database management system, enabling the cost-effective delivery of reliable, high-performance and scalable Web-based and embedded database applications.

3.7.1.3. Development Tools

These are the tools that we used to develop this application.

3.7.1.3.1. Visual Studio Code [12]

Visual Studio Code [12] is a lightweight but powerful source code editor that runs on your desktop and is available for Windows, macOS, and Linux. It comes with built-in support for JavaScript, TypeScript, and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, Go) and runtimes (such as .NET and Unity).

3.7.1.3.2. Gitlab [8]

GitLab [8] is a web-based DevOps lifecycle tool that provides a Git-repository manager providing wiki, issue-tracking, and CI/CD pipeline features, using an open-source license, developed by GitLab Inc.

3.7.1.3.3. Terminal Window

The Terminal is an interface that allows you to access the command line from the GUI.

3.7.1.3.4. draw.io [5]

draw.io [5] is a completely free online diagram editor built around JavaScript, that enables you to create flowcharts, UML, entity relation, network diagrams, mockups and more.

3.7.1.3.5. Android Studio [11]

Android Studio [11] is the official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.

3.7.2. Hardware Requirements

This project requires an Android OS with an Android version greater than 4.4. For the smooth running of the app, it is suggested that the device has RAM greater than 1 GB.

Chapter 4: Discussion on the Achievements

This chapter focuses on the challenges that we have experienced during this project. It also discusses any probable deviation from the objective, negative findings that were implemented.

4.1. Challenges

Scraping has always been misunderstood as an easy task; just to extract information from a website and display it in your application but extracting the right information from the right place is not easy as the internet has a plethora of information in a particular subject. So, here are some of the challenges we faced during the development of this application:

Concept-Based Challenges

- Many websites can provide us with information on a particular topic but the selection of best among them and extracting specific information wasn't easy.
- The design and implementation of the kind of UI / UX that we have planned for our application have been a challenging task.

Technical Challenges

- Learning new technology and implementing it in that short period is certainly difficult due to which we were compelled to scrap and extract information manually from the websites rather than using machine learning to do so.
- Due to a lack of knowledge of classifying the websites and managing several tags as response increased difficulty to some extent.
- Some websites responded with compressed content which was to be decompressed. It was very hard to diagnose how to solve the problem.
- And finally, juggling time between assignments, internals and the project was a mighty challenge as well.

4.2. Features

Despite all the aforementioned challenges, “Cognifeed” has come out to contain a few rather significant features for the users. Some of the important features are:

- **Curated information on different topic**s

The Internet is a treasure for information lovers, and we are often one google search away from getting primed with knowledgeable and factual tidbits. The main reason why websites like Facebook and Instagram have gained so much popularity is that they are a full package of information on social news to celebrity gossip, family tours and trips to health information. Our app is not like Facebook but has the same ability to engage people in not an addictive but productive way. It contains information that has been curated for the specific purpose of providing actual details, on topics that range wide and expansive. This feature is the most important one among its other specialties.

- **Multi-website surf results**

The most common way that we all use to get information on a topic is to google it. Then we visit multiple websites to see which one of them contains the particular information that we desire. With “Coginfeed”, you'll only need to select the topic that you are interested in and the rest of the manual task of surfing websites to websites will be done for you. The app provides the list of articles from multiple websites that are designed for containing that information, as per the tags you've selected. With the difficulty of surfing out of the way, one can enjoy the knowledge filled articles from multiple websites, at the expense of one click of a tag.

- **Push notification**

Notifications are made so that we don’t miss out on anything despite being busy. Cognifeed has an important feature of push notification which can be customized as per the user’s schedule. The user will get a notification daily from the app reminding them of articles from websites belonging to the category selected by the user. The time of the day at which notification arrives can be set by the user. A dose

of new articles will be provided daily so that the users can benefit from this app despite their busy schedules.

- **Save favorite articles**

Cognifeed also contains the feature to bookmark articles and save them for future reference. Users can mark an article they want to save as favorites. Then that article gets saved to the app's local database which can be referenced and viewed anytime by visiting the favorites section of the app.

- **Personalized selection of articles**

This app contains a feature to personalize your feed. When the user selects tags, he receives most of the article recommendations based on those tags plus articles related to other tags as a recommendation to follow those tags. Also, there is a feature to follow the tag related to a specific article to receive similar articles in the future. This can be done by clicking on the menu icon of a specific article and following the associated tag.

- **Feature to share, hide, and follow tags belonging to articles**

If the user wants to share an article, he may use the share feature included in the article display box to share the link of that article. There is also the feature to hide an article if the user doesn't like that article content or article of those kinds. Users can follow the tag related to an article if the user is interested in articles of that kind.

Chapter 5: Conclusion and Recommendation

Cognifeed is a great platform for people to get information on different topics which will surely help in making people more informed, knowledgeable and up to date. This application will improve the literacy of people, but it can surely help people to be primed about details and facts on different topics. While we worked hard to create this application in three months, due to time constraints there are certain limitations in the app and we expect to make some future enhancements in the coming days.

5.1. Limitations

Although we have tried our best to create a fully functional and user-friendly app, due to the time constraint of the project, there are still some significant gaping inconsistencies in it.

- The app doesn’t respond without the internet or offline.
- The selection of websites based on tags is done manually which made us miss many important websites belonging to a category.
- The credibility of a website belonging to a particular category/tag is doubtful and is prone to human error.
- A website may belong to multiple categories and, a category may have a subcategory. We couldn’t implement this feature in our application.
- The number of tags is limited to what we could come up with while categorizing different websites on the internet.
- We have scraped websites by the DOM manipulation method. The structure of the website including its HTML and CSS classes may change over time. Our application does not have the feature to address those changes.
- Due to storage constraints, we have implemented a low-level server that spawns only a limited number of spiders at an instance to crawl the website which won't be feasible for maintaining a large number of tags and websites related to those tags.
- The implementation of search features is not very efficient in our application.

- There is no system to recommend the articles/tags as per the current selection of tags, and articles marked as favorite by the user.
- The app system is not as secure as we wanted it to be.

5.2. Future Enhancements

Based on the limitations of the project, we intend to make these enhancements to the project to make it more deployable in the market.

- Make the app respond both offline and online.
- Use appropriate methods to purify websites to extract meaningful content which will work dynamically even if the structure (HTML) of the websites gets changed.
- Use a strong server that is efficient and can spawn multiple spiders at a time without being down.
- Implementation of an effective article search and recommendation system.
- Incremental improvements to the styles, interactivity and data security of the web application.
- Possible ports to desktop and web applications.
- Use machine learning (Unsupervised Learning) to classify websites to different tags that are generated as per the content of the website. This will enhance the credibility of categorization and will improve the selection and number of tags.

References

1. Angelov, F. (2019, 9 19). *Bloc*. From BLoC: https://bloclibrary.dev/#/
2. Auth0. (2020, 1 15). *JSON Web Tokens - jwt.io*. From JWT: https://jwt.io/
3. cheeriojs. (2019, 12 21). *cheerio | Fast, flexible, and lean implementation of core jQuery designed specifically for the server.* From Cheerio.Js: https://cheerio.js.org/
4. Deepstash. (2019, December 12). *Deepstash App: Self Improvement & Daily Motivation*. From Deepstash: https://deepstash.com/
5. draw.io. (2020, 3 10). *Untitled Diagram - draw.io*. From draw.io: https://www.draw.io/
6. Express. (2019, 11 18). *Express - Node.js web application framework*. From Express: https://expressjs.com/
7. Flipboard . (2019, 11 18). *Flipboard - Personalized for any interest*. From Flipboard: https://flipboard.com/
8. Gitlab. (2019, 11 25). *The first single application for the entire DevOps lifecycle - GitLab | GitLab*. From Gitlab.
9. Google. (2019, 8 16). *Flutter - Beautiful native apps in record time*. From Flutter : https://flutter.dev/
10. Google. (2019, 10 11). *Google News*. From Google News: https://news.google.com/?hl=en-US&gl=US&ceid=US:en
11. Google Developers. (2019, 11 10). *Download Android Studio and SDK tools | Android Developers*. From Android Studio: https://developer.android.com/studio
12. Microsoft. (2019, 11 5). *Visual Studio Code - Code Editing. Redefined*. From Visual Studio Code: https://code.visualstudio.com/
13. npm. (2019, 12 14). *bcryptjs - npm*. From npm: https://www.npmjs.com/package/bcryptjs
14. OpenJS Foundation. (2019, 10 17). *Node.js*. From Node.js: https://nodejs.org/en/
15. Oracle. (2019, 11 19). *MySQL :: MySQL Documentation*.

Appendix

A.1. Gantt Chart

Week	1	2	3	4	5	6	7	8	9	10
Planning & Preparation	■									
Work Division	■									
Coding		■	■	■	■	■	■	■		
Debugging & Beta-Testing								■	■	
Documentation									■	■

Figure 21: Gantt chart

A.2. Workflow Chart

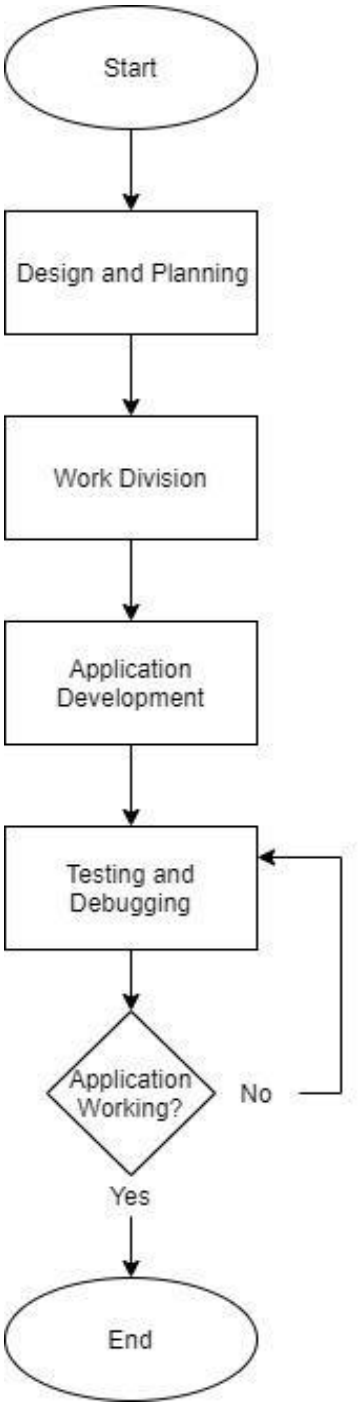

Figure 22: Workflow Chart

A.3. Screenshots

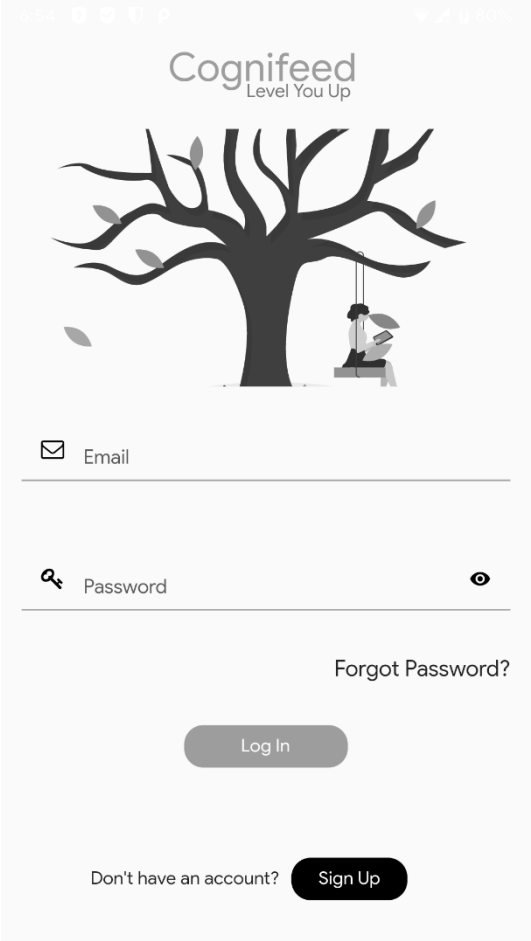

Figure 24: Login Page

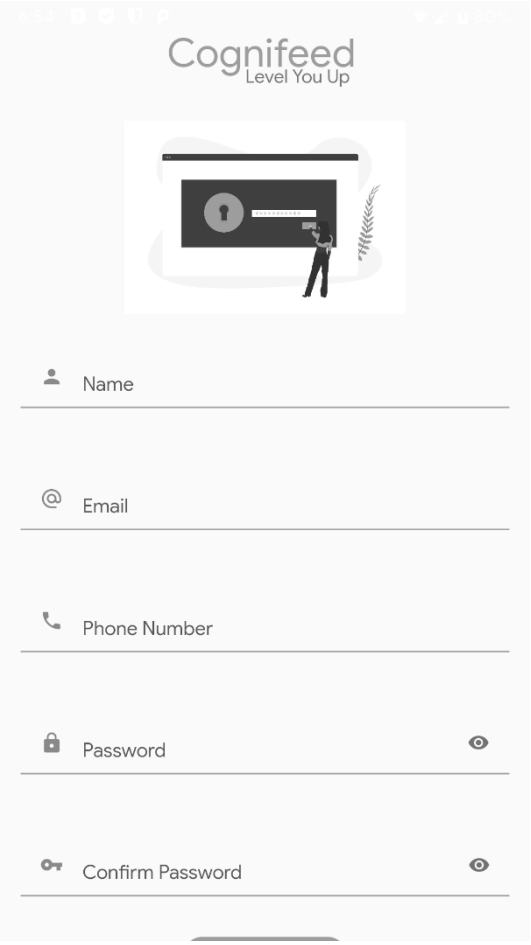

Figure 23: Signup Page

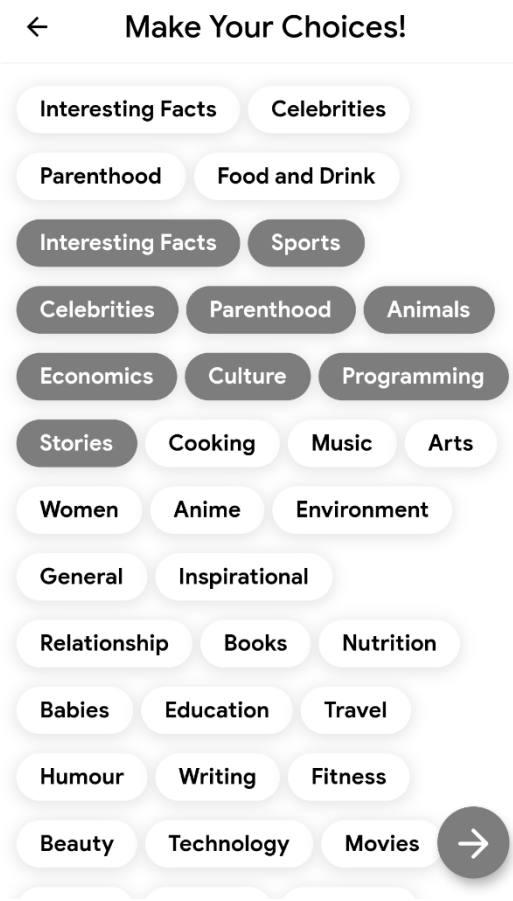

Figure 25: Onboarding Page

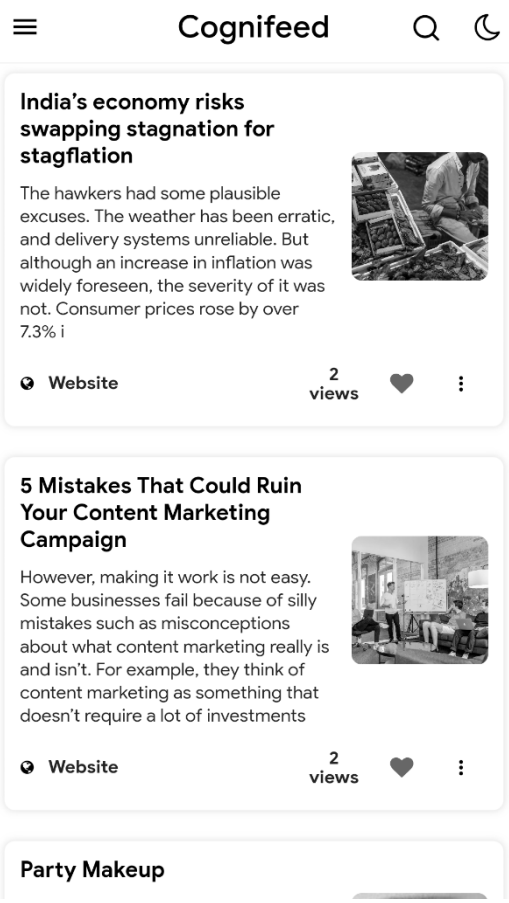

Figure 26: Home Page

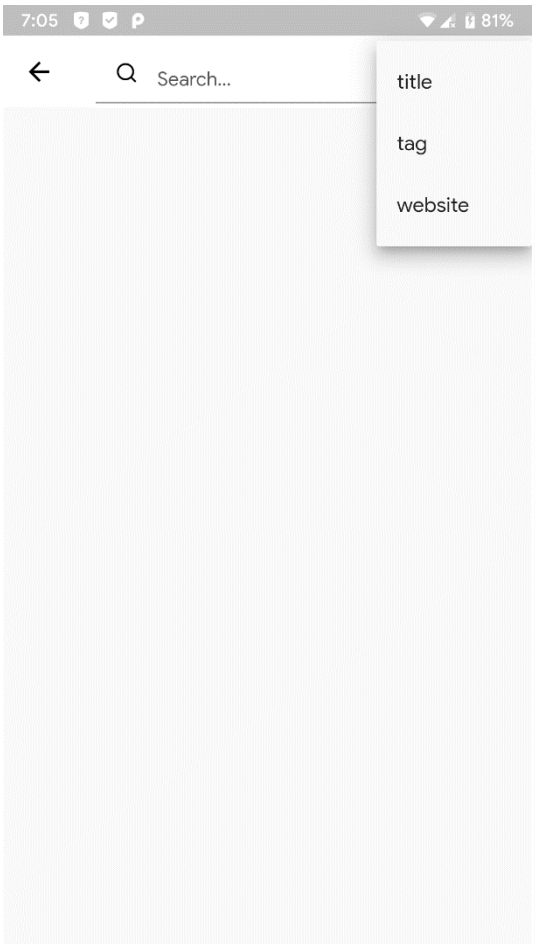

Figure 27: Search Page

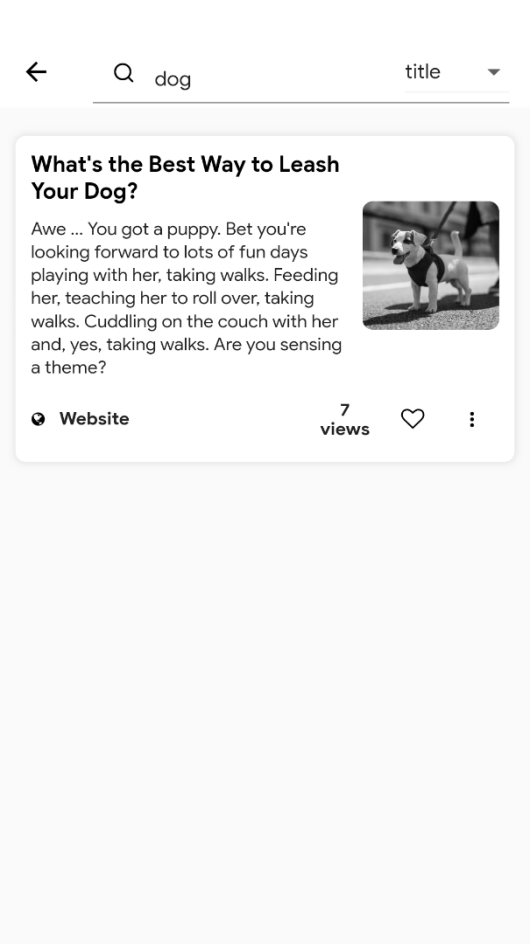

Figure 28: Search Result

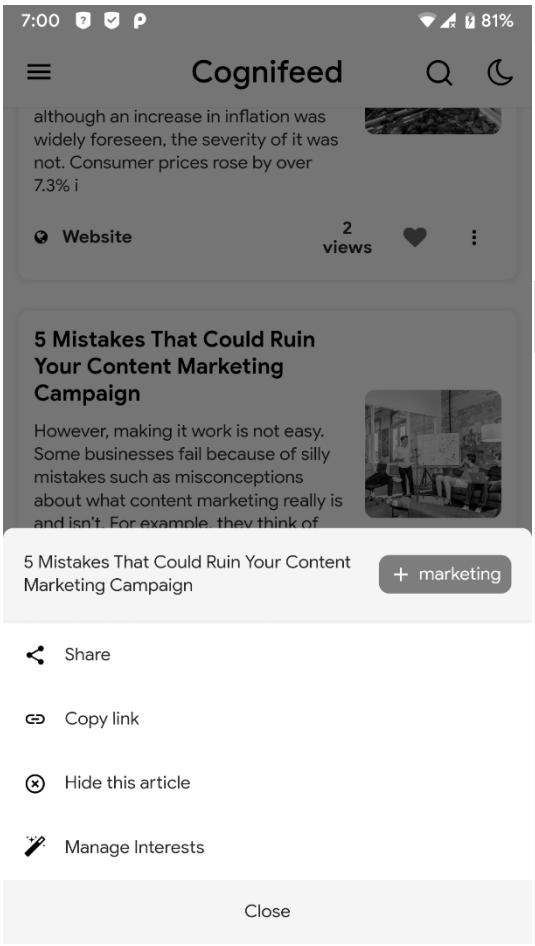

Figure 29: Bottom sheet

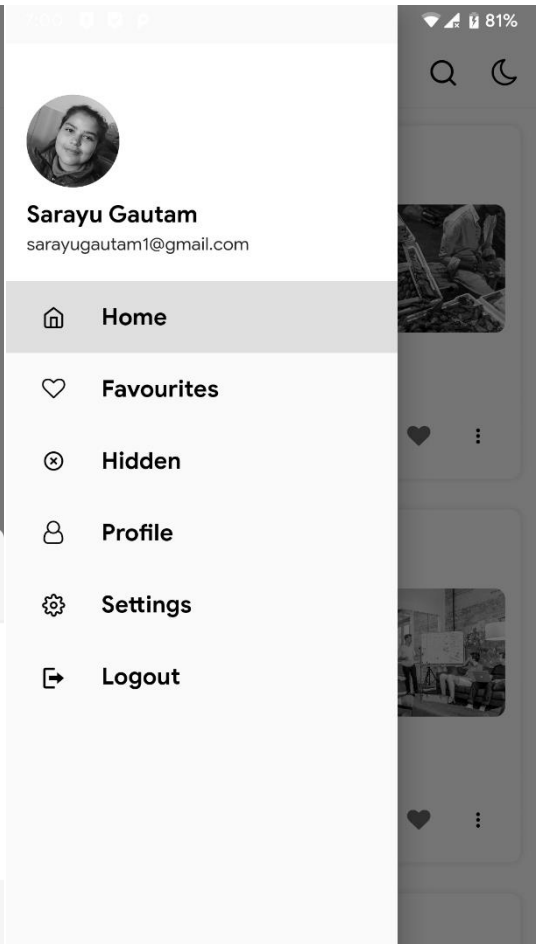

Figure 30: Drawer

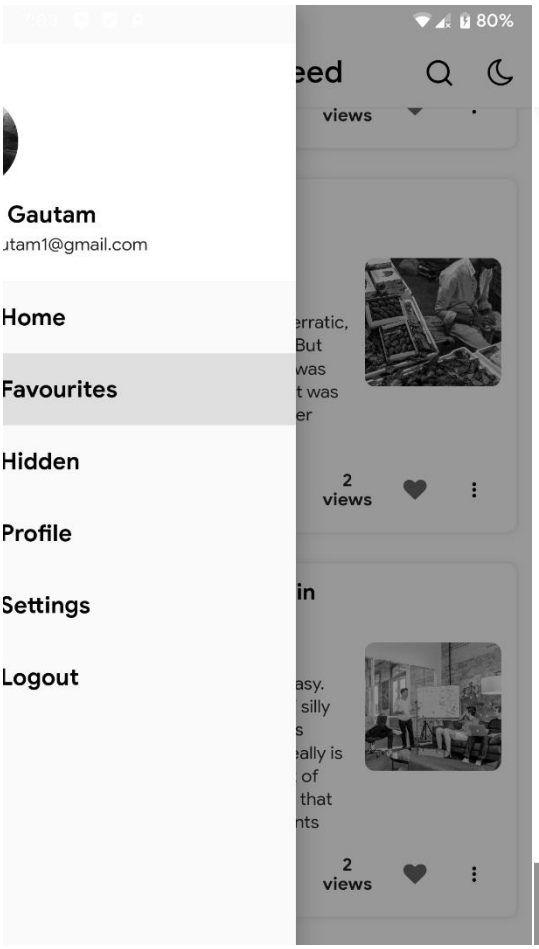

Figure 31: Favorite Article Page

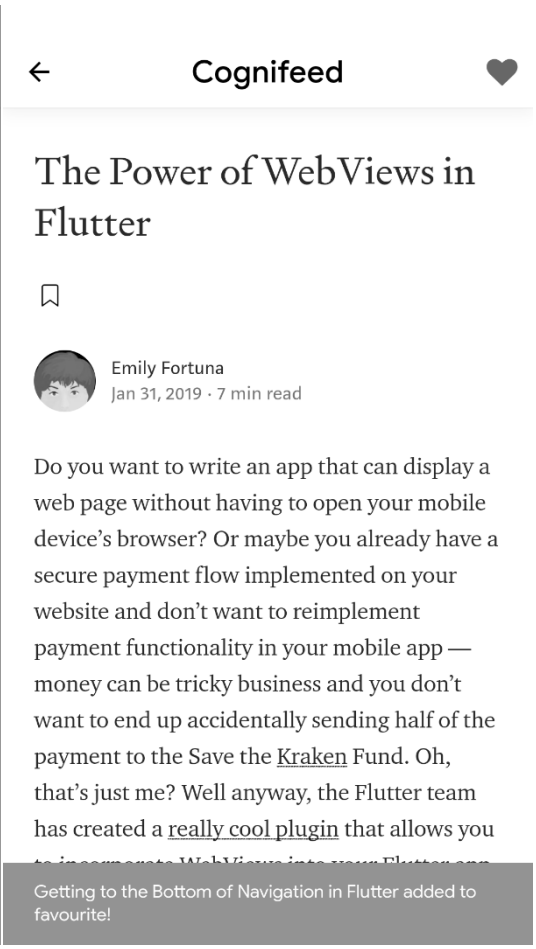

Figure 32: Article Preview

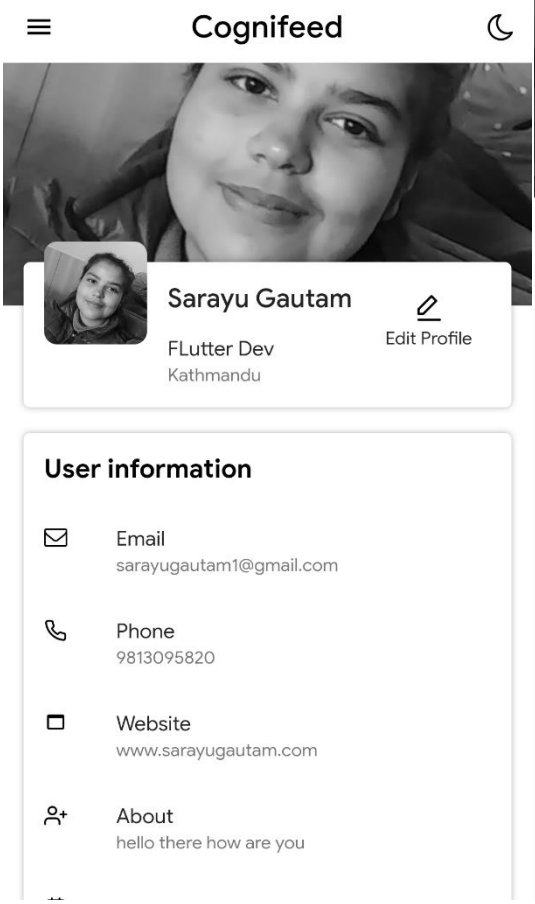

Figure 33: Profile page

Figure 34: Edit profile

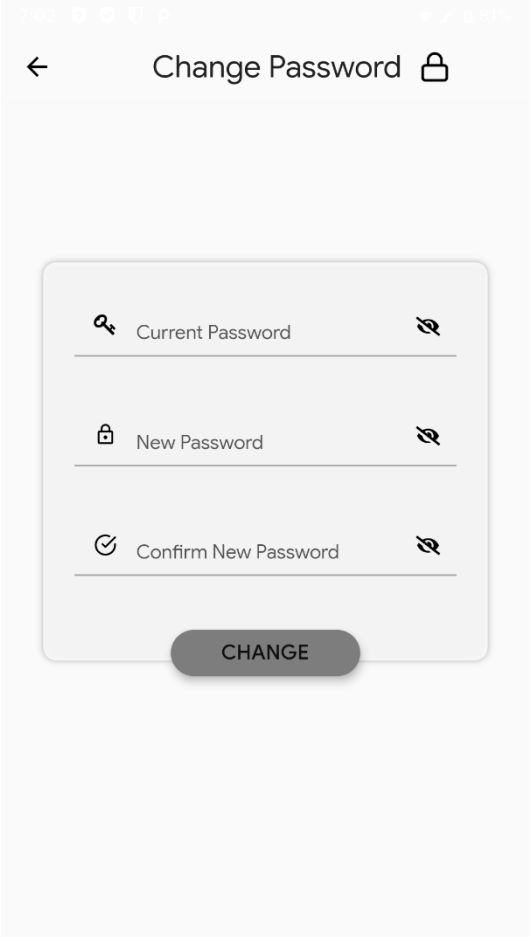

Figure 35: Change password page

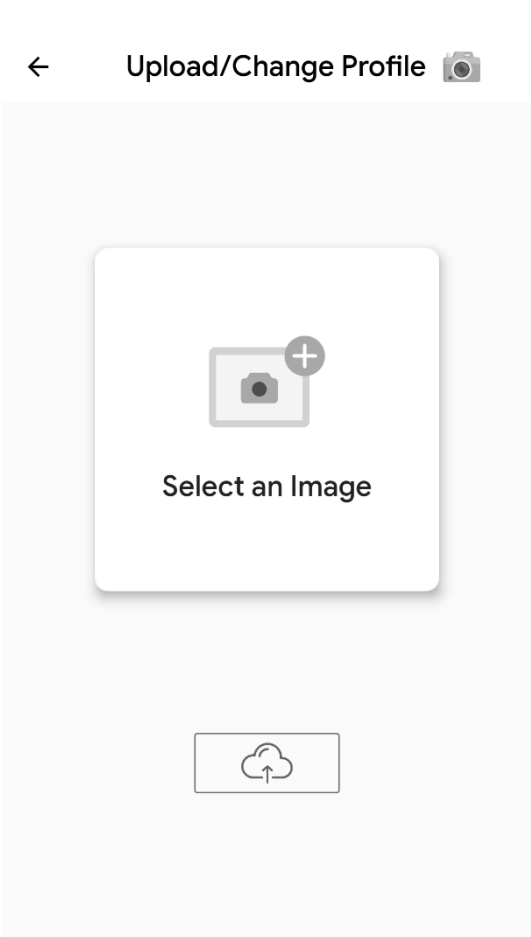

Figure 36: Image Upload page

Figure 38: Crop and compress new image

Figure 37: New image uploading

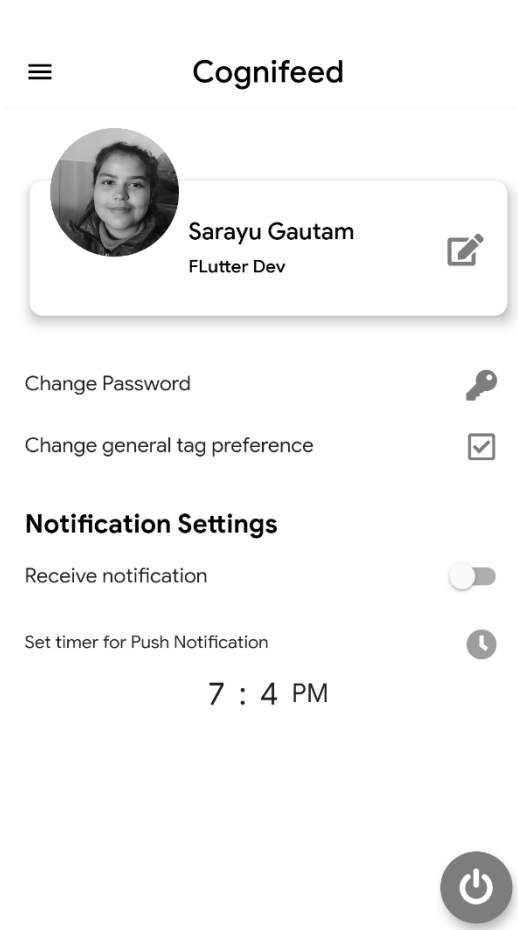

Figure 39: Settings Page

Figure 40: Select time page

Figure 42: Code request for resetting password

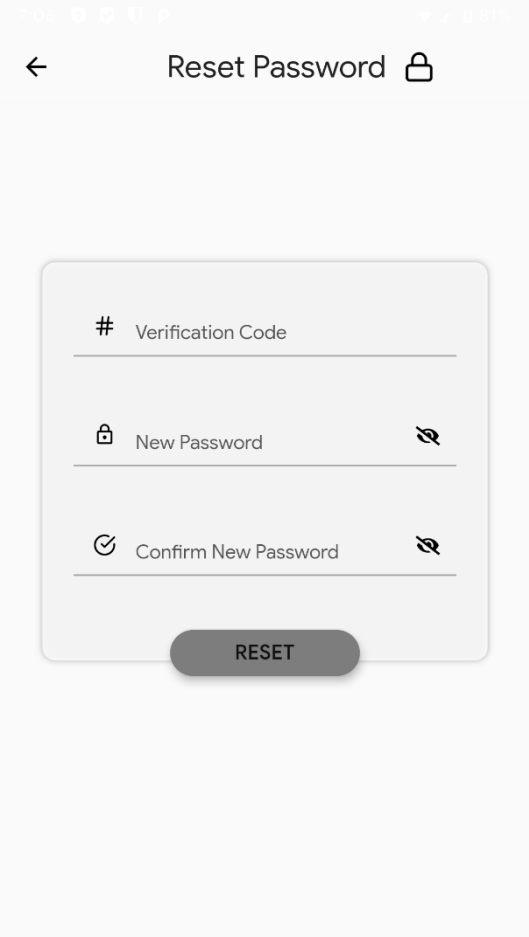

Figure 41: Reset password with reset code

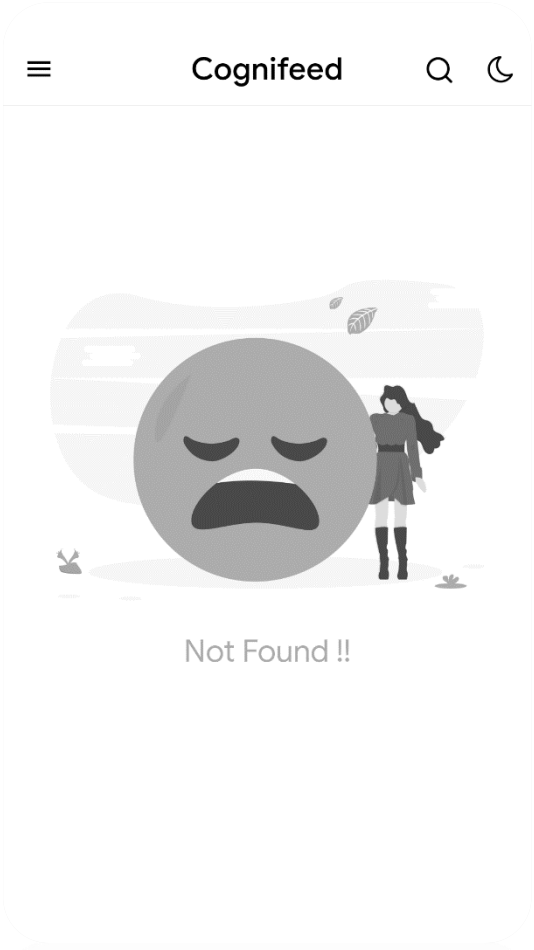

Figure 43: Not found 404 response