**Mini-Project Report**

**GupShup: The Discussion Forum**



**Group Members**

Rushabh Gandhi - 1911012

Kashish Harisinghani - 1911016

Harshit Jain = 1911017

# **Contents**

# 1 Introduction.

# 1.1 Problem definition

# 1.2 Scope of the project

# 2 Literature Survey

# 3 Project design

# 3.1 Block diagram / system diagram

3.2 Software Project Management Plan

3.3 Hardware and software requirements

# 4 Implementation and testing

# 4.1 Proposed system model implementation

# 4.2 Functions Implemented

# 4.3 User interface

# 4.4 Software Testing

# 4.5 GITHUB implementation

## 5 Working and Output

# 5.1 Conclusions

5.2 Scope of further work

# 

# 

# **Chapter 1**

# **INTRODUCTION**

**1.1 Problem Definition:**

# A college or an institute does not have a place where the students as well as the faculty can come together and can discuss ideas and thoughts. Sometimes the problems can be just solved by mere discussion with someone but the lack of resources, guidance, knowledge, and a suitable platform hinders this process of sharing and learning.

# Making a discussion forum for an institute where students and teachers can ask, and answer any of the questions asked by other members of the institute, also having some possibility to form a community of liked-minded people so that the questions can be answered according to the area of interest.

# **1.2 SCOPE:**

# 1. Login into the web portal using a unique id.

# 2. Adding blogs to the website.

# 3. Question answer-based forum.

# 4. Writing comments for other posts.

# 5. Answering a question.

# 6. Upvote and downvote.

# 7. Segregating the posts based on distinct categories.

# 8. Functionality to follow, block a person.

# 9. Functionality to bookmark a blog.

# Exclusions

# 1. Trust rating of user.

# 2. Functionality to share a post.

# 3. Private Chat functionality.

# 4. Forming groups for like-minded people.

# **Chapter 2**

# **LITERATURE SURVEY**

# <https://www.python-course.eu/>

# For learning some basic python to start with the project.

# <https://developer.mozilla.org/en-US/docs/Glossary/Python>

# For learning some python commands for with the project.

# <https://docs.djangoproject.com/en/3.1/>

# To get familiar with the Django syntax and learning the framework

# <https://getbootstrap.com/docs/4.0/getting-started/introduction/>

# For making GUI components

# <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

# To refer basic javascript for website responsiveness

# <https://tailwindcss.com/> <https://stackoverflow.com/>

# For making GUI components

# <https://www.geeksforgeeks.org/>

# To learn a few functionalities of python.

# <https://channels.readthedocs.io/en/latest/tutorial/part_2.html>

# For making the chat app in the website

Books Referred:

Django for APIs: Build web APIs with Python & Django

Django for Beginners: Build websites with Python and Django

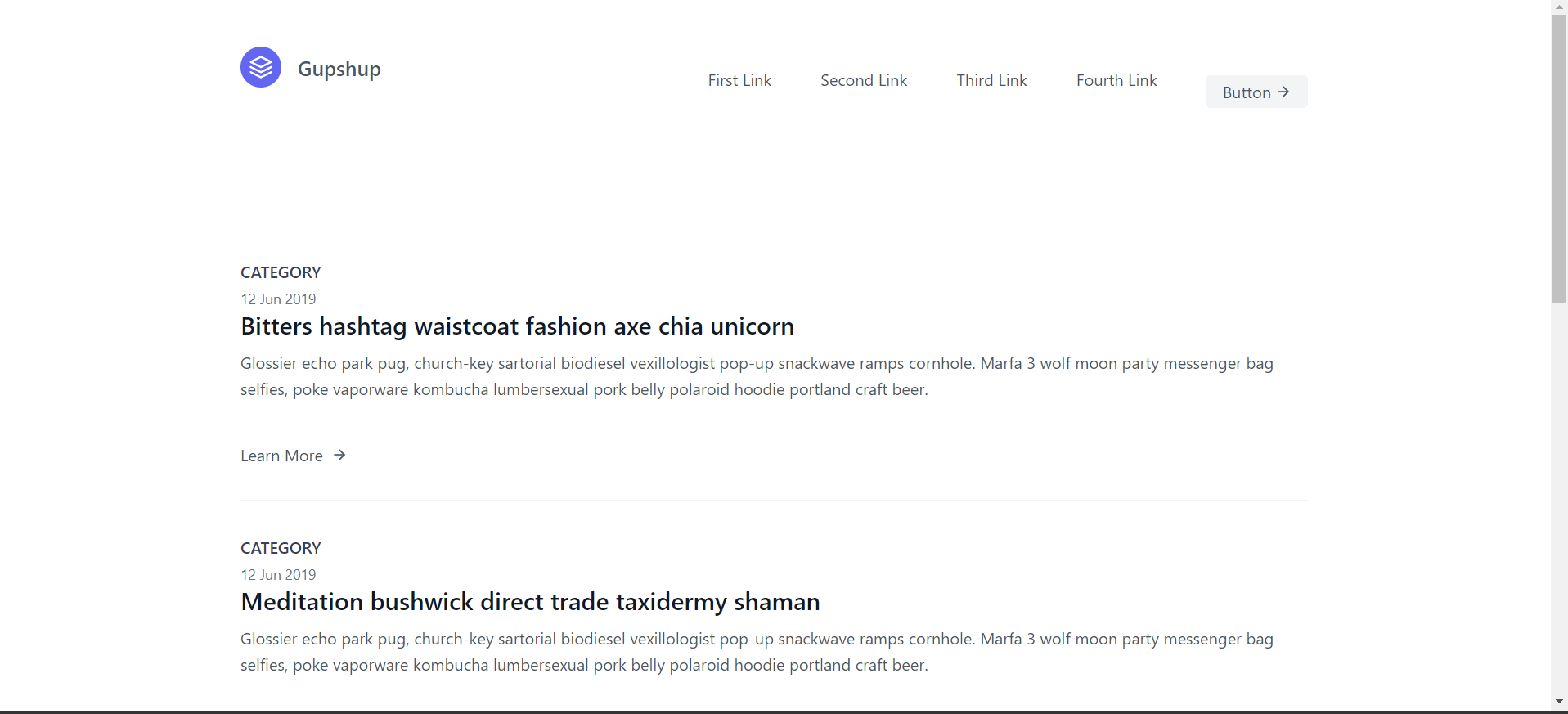
Django Design Patterns and Best Practices Django Unleashed

# **Chapter 3**

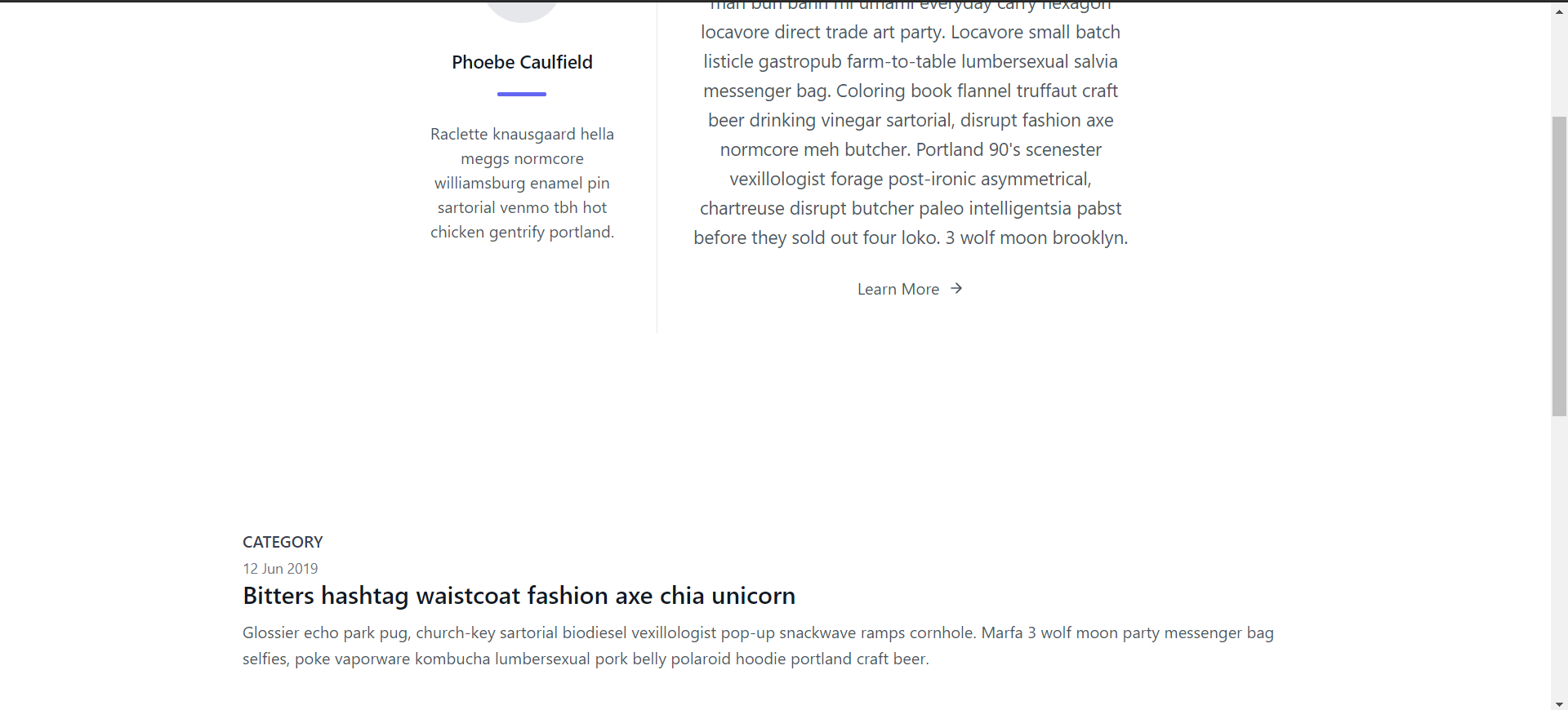
# **PROPOSED PROJECT DESIGN**

**Features**

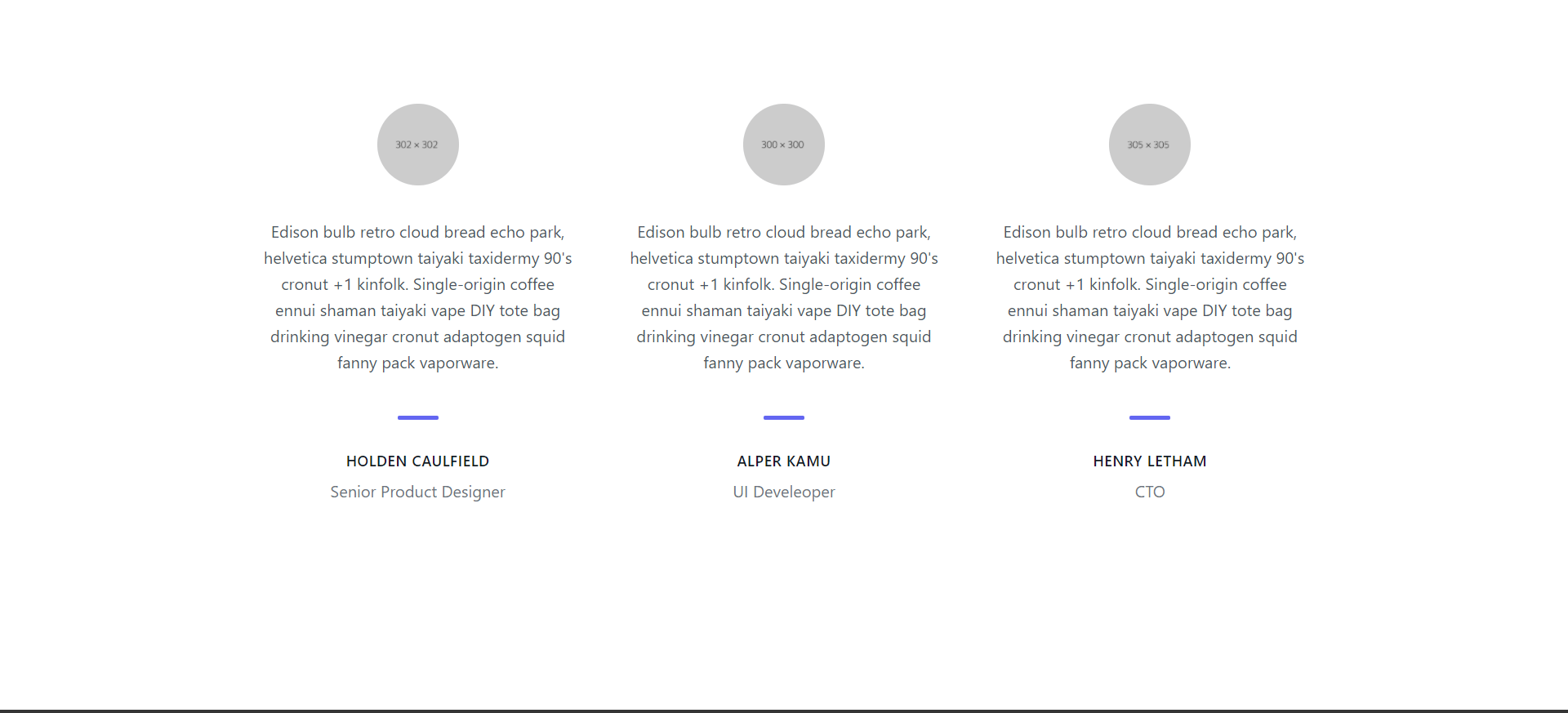
Home Page:



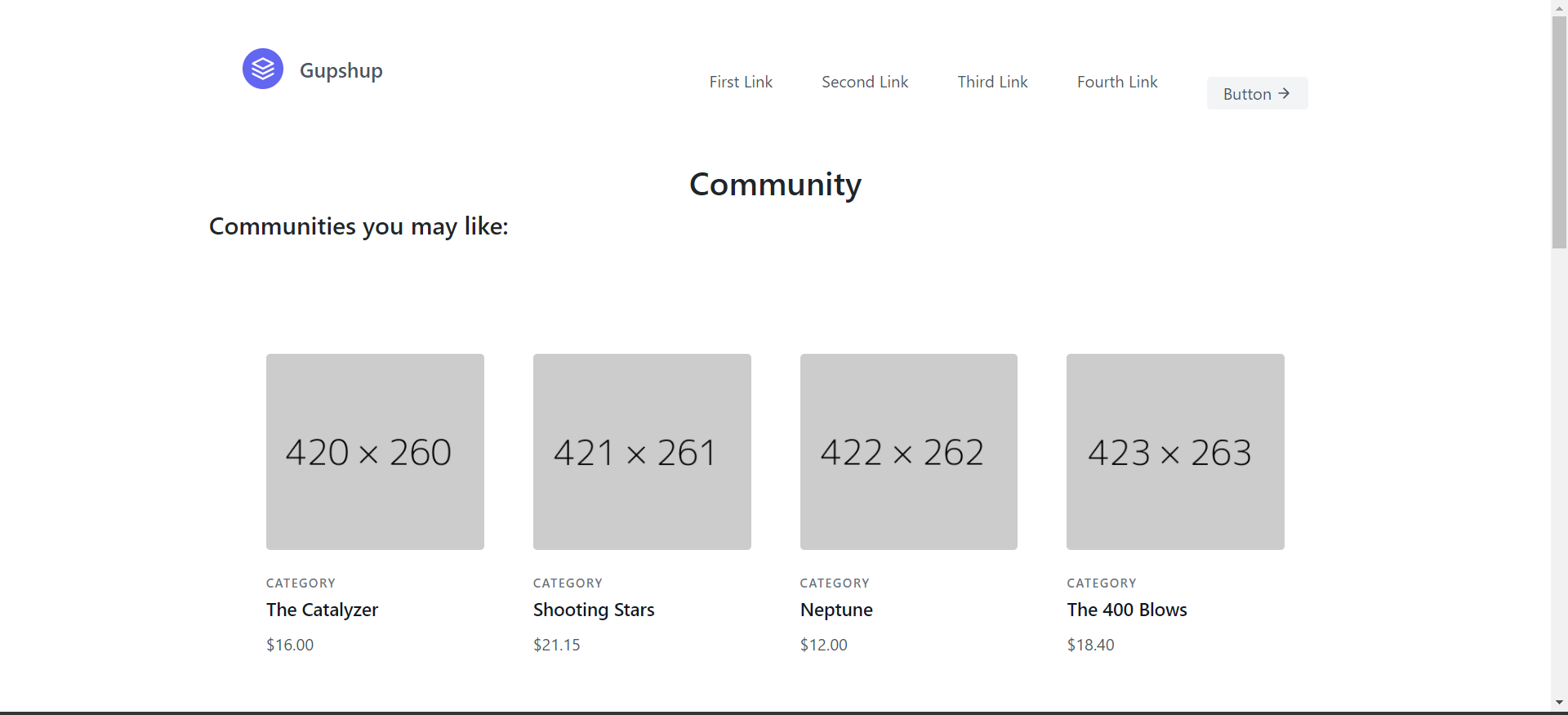
Profile:



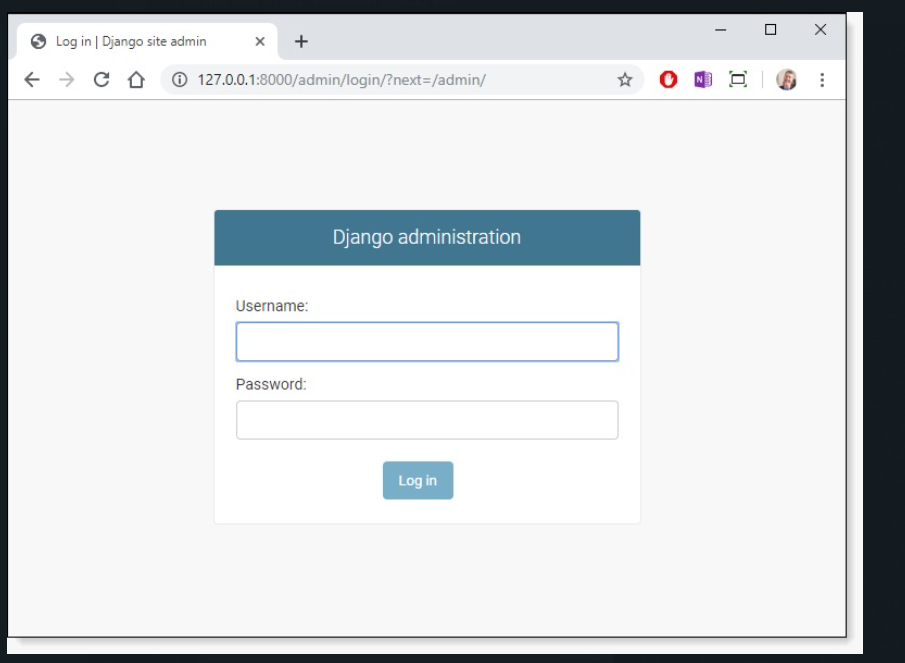
About:



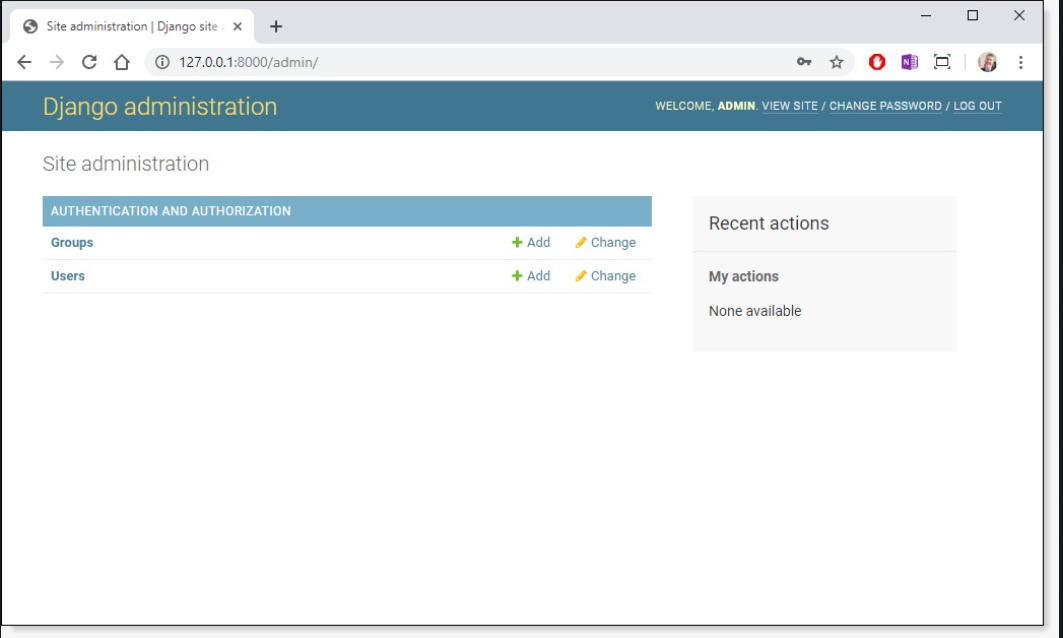
Communities:



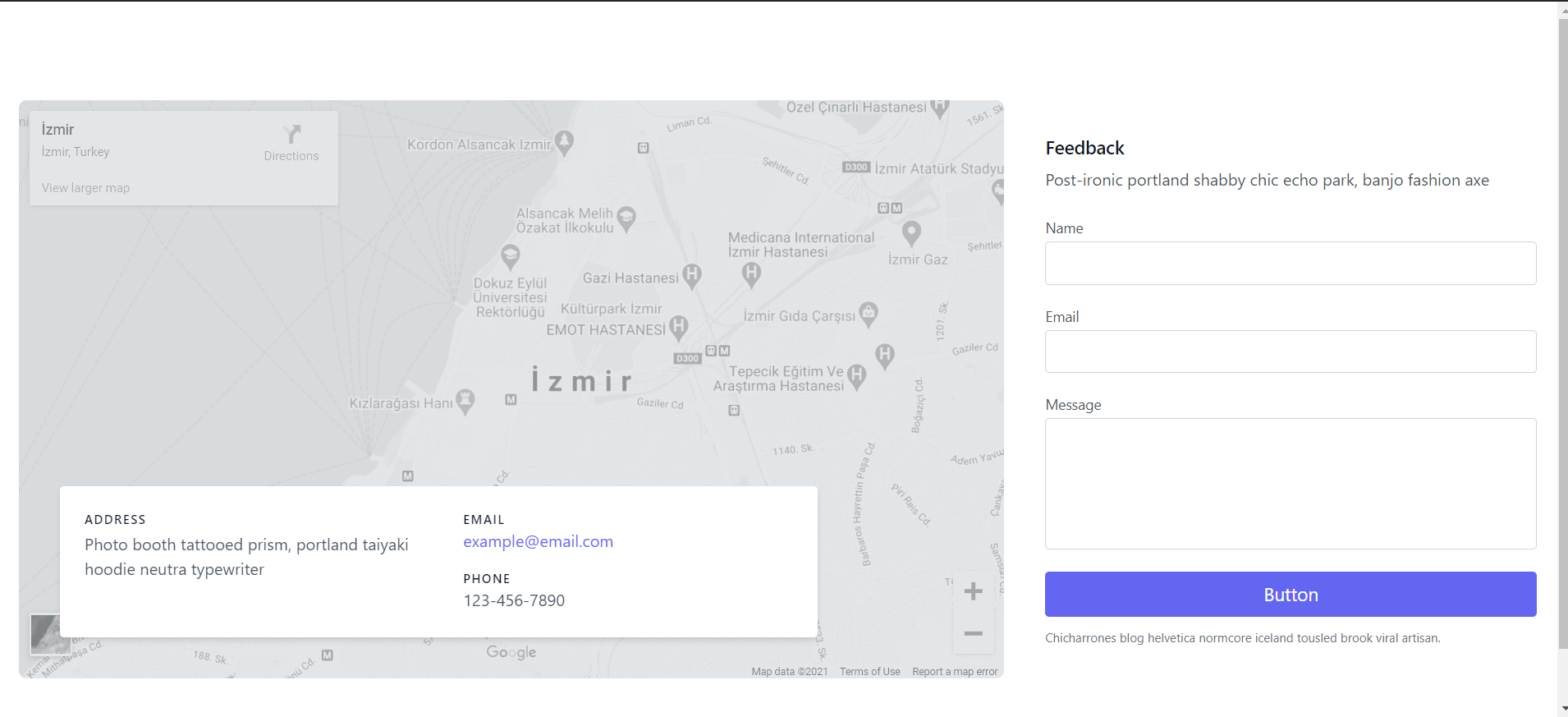
Admin Login:



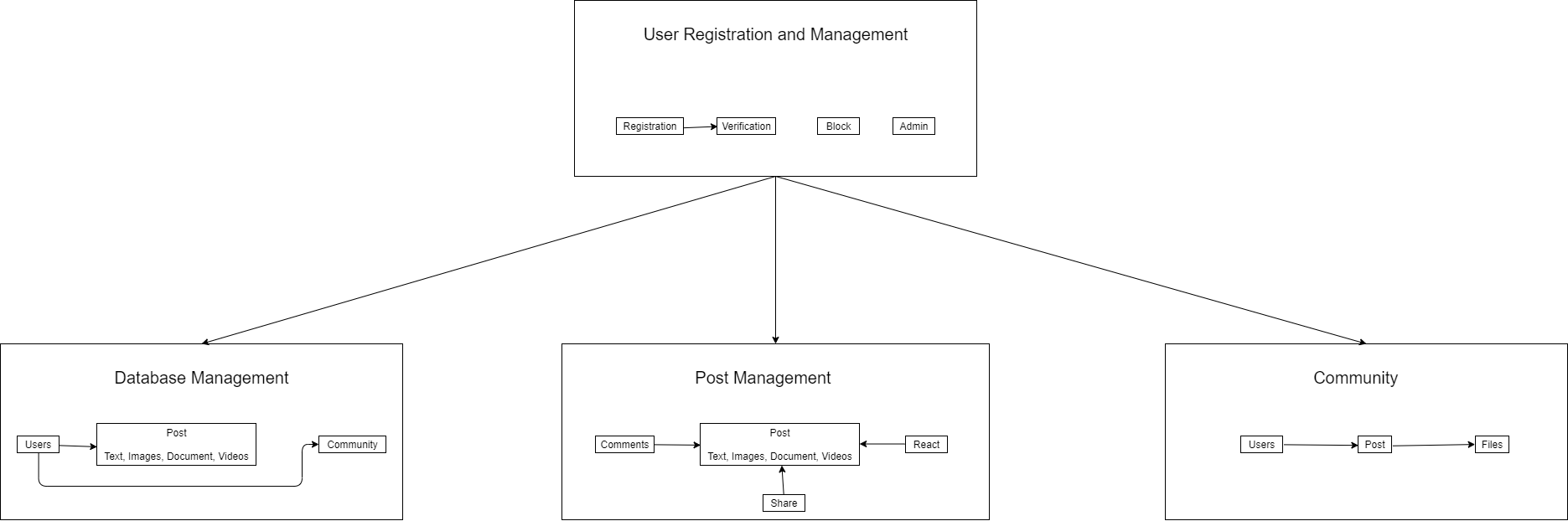
Admin Panel:



Contact:



**3.1 SYSTEM ARCHITECTURE**

****

**3.2** **SOFTWARE PROJECT MANAGEMENT PLAN**

**TimeLine Chart**

|  |  |
| --- | --- |
| **Activity** | **Duration** |
|  |  |
| Requirement Gathering  and Planning | Jan 18 - Jan 25 |
| Basic routes and UI design | Jan 26 - Feb 6 |
| Development of user app and user authentication, login and registration functionality | Feb 6 - Feb 22 |
| Development of post app | Feb 22-Feb 28 |
| Development of post app templates | Feb 20 – Mar 3 |
| Admin panel and database connectivity | Mar 3 - Mar 15 |
| Development of community chat app | Mar 15 - Mar 24 |
| Made Web-socket channels | Mar 24 - Mar 31 |
| Integration and Database connectivity | Apr 1 - Apr 7 |
| Bugs and error Fixed | Apr 8 - Apr 15 |
| Testing | Apr 15 -Apr 25 |

**3.3 SOFTWARE/FRAMEWORK USED**

1. VS code - Editor
2. SQLite3- Database
3. Django- Server side environment
4. Html, CSS, JavaScript – Frontend
5. Python
6. Docker and Web Sockets

**Backend:**

* *Django*- It is an open source, cross-platform runtime environment for developing server-side and networking applications. It is used to develop applications that run end-to-end in JavaScript i.e both client side and server side .
* *SQLite3*- SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day.
* *Python* - Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation.
* *Docker -* Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers.
* *Web Sockets -* Channels is a project that takes Django and extends its abilities beyond HTTP - to handle WebSockets, chat protocols, IoT protocols, and more. It’s built on a Python specification called ASGI.

Frontend:

* *Html* - It is used to create electronic documents (called pages) that are displayed on the World Wide Web.
* *CSS* - It is used for describing the presentation of Web pages, including colors, layout, and fonts.
* *Bootstrap 4*  - Bootstrap is a framework to help you design websites faster and easier. It includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels, etc.
* *Java script* - JavaScript is a client scripting language which is used for creating web pages. It is used when a webpage is to be made dynamic and add special effects on pages like rollover, roll out and many types of graphics.

TOOLS:

|  |  |
| --- | --- |
| **Tool** | **Description** |
| GitHub | GitHub, Inc. is a United States-based global company that provides hosting for software development version control using Git. Used for uploading the projects, such that it is simultaneously accessible to all members. |
| Vs code | Visual Studio Code is a code editor redefined and optimized for building and debugging modern web and cloud applications.Also it includes [Git](https://git-scm.com/) support in-the-box. |
| Docker | Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. |

**HARDWARE USED:**

Ram: 4GB or higher

Memory: 500 MB (for running the project only)

5 GB (for development – includes dataset and other files also)

Processor: Intel i3 or higher

Optional: The project works better if there is a GPU

# **Chapter 4**

# **IMPLEMENTATION AND TESTING**

# **4.1 Proposed System Model Implementation**

The website is created using Django which handles database internally using sqlite3. The models.py file for different apps created in the project contains all the different entities required in the website along with their fields.

Model: Post

Fields: title, content, author, date posted, category, likes

Model: Post Comment

Fields: post connected, author, comment, date posted

Model: Profile

Fields: user, image

The view.py file in the app handles the merging of backend and frontend for that app in the project.

**Our application structure is**

File manage.py => which runs the application and serves at port 8000.

Folder app => folder for all apps which includes: blog, users, chat

File app/urls.py=> which contains different categorised files for urls of all apps

File app/views.py => It contains all the functions for rendering the templates for the all apps.

Folder app/templates/ => It contains all the templates in the form of HTML files for that app.

Files app/models.py => This contains which specify different models for the database.

File setting.py=> This contains configuration values that the web app needs to work

File wsgi.py => The Web Server Gateway Interface is a simple calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language.

**Database Details**

1. Post Table:

title = models.CharField(max\_length=100)

    content = models.TextField()

    date\_posted = models.DateTimeField(default=timezone.now)

    author = models.ForeignKey(User, on\_delete=models.CASCADE)

    upvote = models.ManyToManyField(User, related\_name='upvote')

    downvote = models.ManyToManyField(User, related\_name='downvote')

    bookmark = models.ManyToManyField(User, related\_name='bookmark')

    category = models.CharField(

        max\_length=20,

        choices=Category\_choices,

        default='Miscellenous'

    )

1. **Post Comment Schema**

 post\_connected = models.ForeignKey(Post, related\_name='comments', on\_delete=models.CASCADE)

    author = models.ForeignKey(User, on\_delete=models.CASCADE)

    comment = models.TextField()

    date\_posted = models.DateTimeField(default=timezone.now)

    upvote\_comment = models.ManyToManyField(User, related\_name='upvote\_comment')

    downvote\_comment = models.ManyToManyField(User, related\_name='downvote\_comment')

1. **User Schema**

user = models.OneToOneField(User, on\_delete=models.CASCADE)

    image = models.ImageField(default='default.jpg',upload\_to='profile\_pics')

**Backend Details**

We have a search bar which on ‘input’ fetches ‘/api/products’ api route which has all the products as JSON response. After fetching we get the data as JSON on which we then apply our fuzzy search logic using Regex (Regular Expression).

Which helps user not only to search medicines by their name but also by disease name i.e. for e.g.: if person doesn’t know which medicine to take for fever then he can just type ‘fever’ in the search bar and appropriate medicine would be displayed if available in store.

**User** can add products to cart without logging in but is made to login while checkout.

User also gets confirmation email after the order is placed successfully.

For mailing we are using node mailer.

Also while signup user password is encrypted and stored in the database.

**Admin** has many functions which he can perform and view.

Admin side has a **Dashboard** which gives an overall view of the website in terms of Numbers like Total users, orders, Revenue etc.

Dashboard also has different **Analysis graphs** which may help the admin to analyze his sales and profit.

Admin can **Add a product** in the store, as well as **Edit** and **Delete** a product.

Admin can also view all the orders, **change status** of order like confirming and delivering order through simple buttons provided which reflects back in the database and gets updated on the user side so that the user is always kept updated.

All the admin routes are **secured** using middleware so that no user can access those routes and if tries to access then will be redirected to homepage.

**4.2 Functions Implemented :**

1. **register**

To register a new user to the platform

1. **verify**

This function send a verification mail to the user and finally activates the account on completion.

1. **Updateprofile**

It updates the user profile according to the changes made by the user in the profile section.

1. **postcreateview**

The function stores a new post to the database according to the feilds filled by the user and also checks for user authentication.

1. **Postlistview**

This function retrieves all the post from the database and displays it to the user.

1. **Category**

This function segregates to post according to the category.

1. **Bookmark**

This function fetches all the post that are bookmarked by a user and displays them.

1. **upvotedownvote**

This function manages all the upvotes and downvotes of a given post .

1. **Postdelete**

This function helps the user to delete his/her post.

1. **Postupdate**

This function helps the user to edit his/her post.

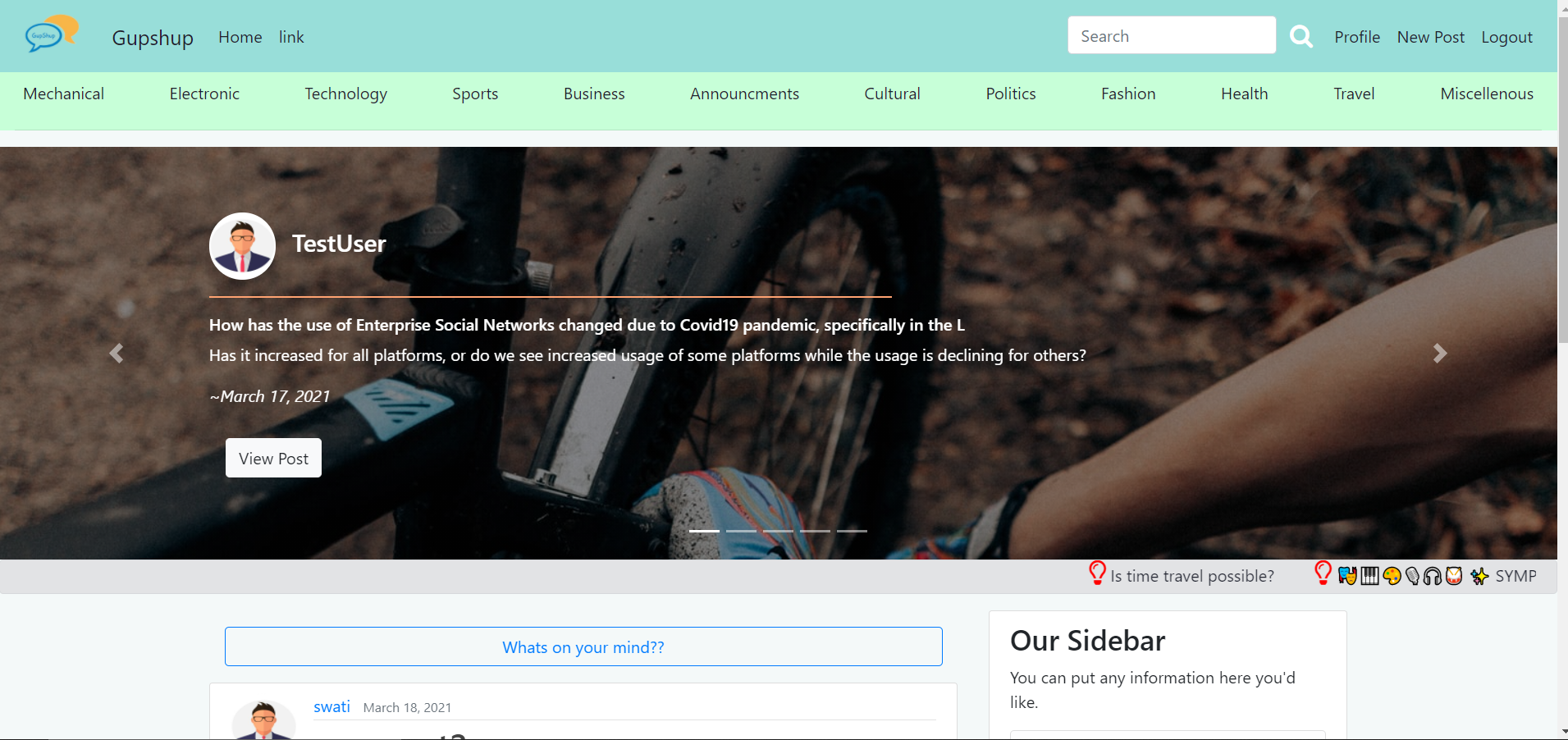
1. **Searchresult**

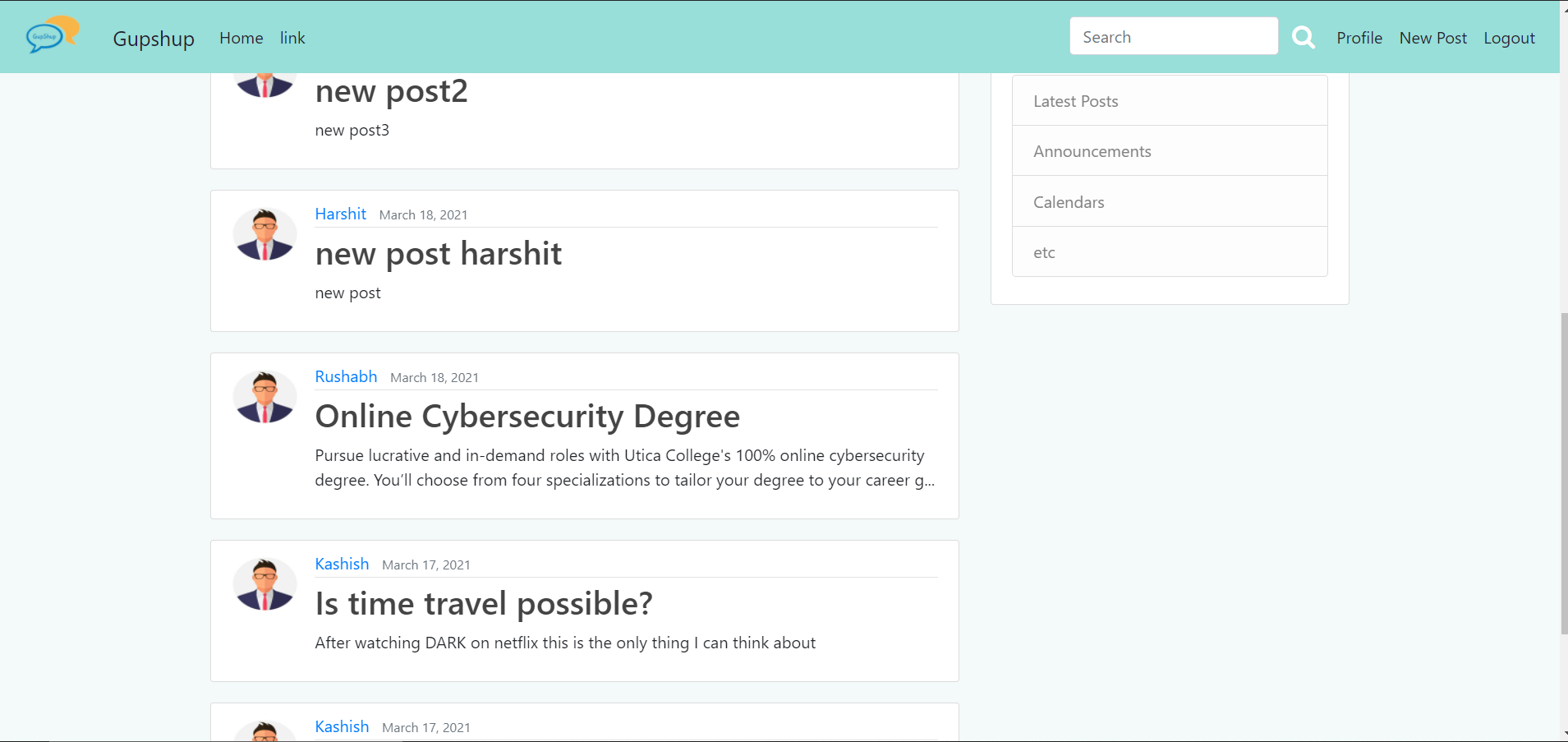
This function displays the search results searched by a user.

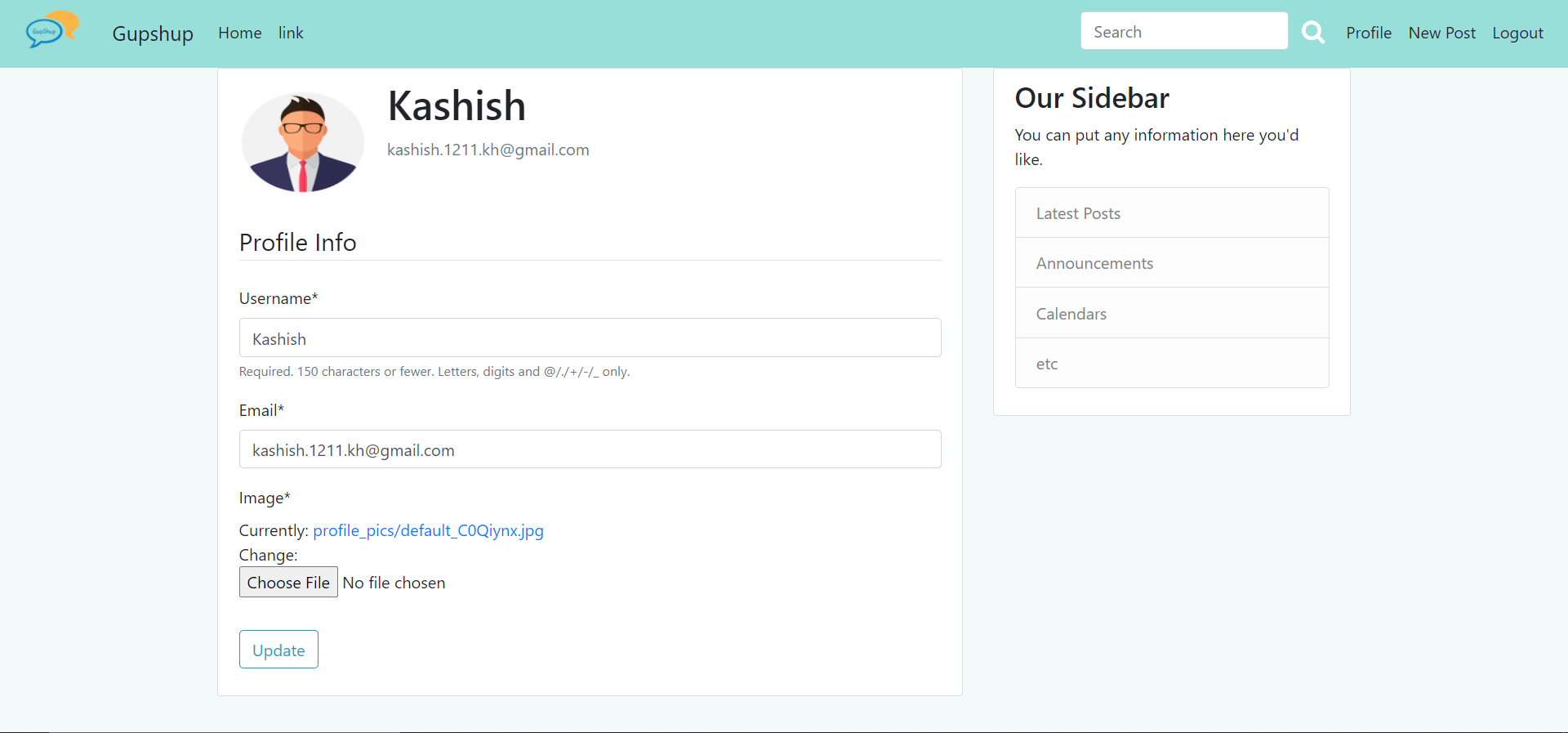
**4.3 User interface**

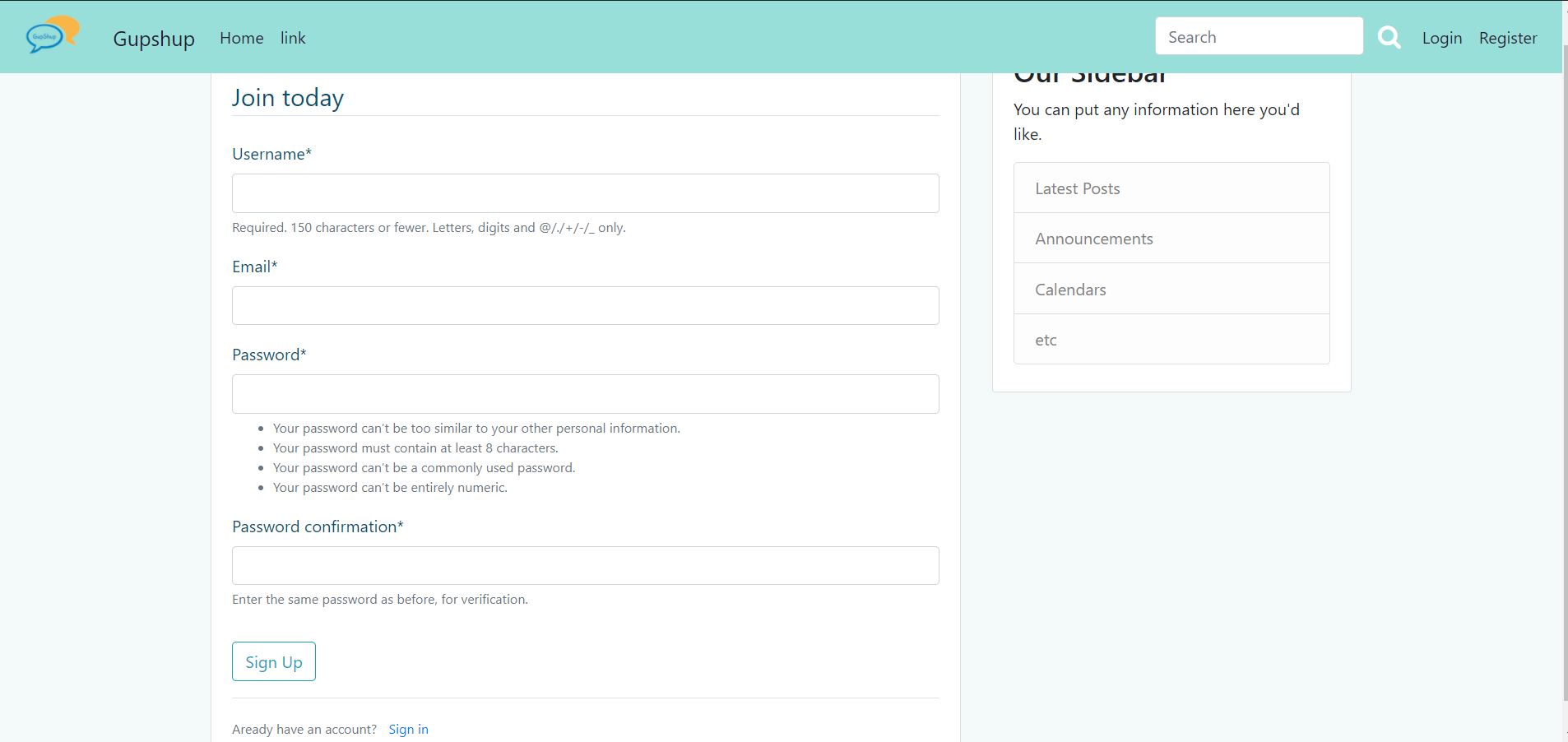
**User side:**

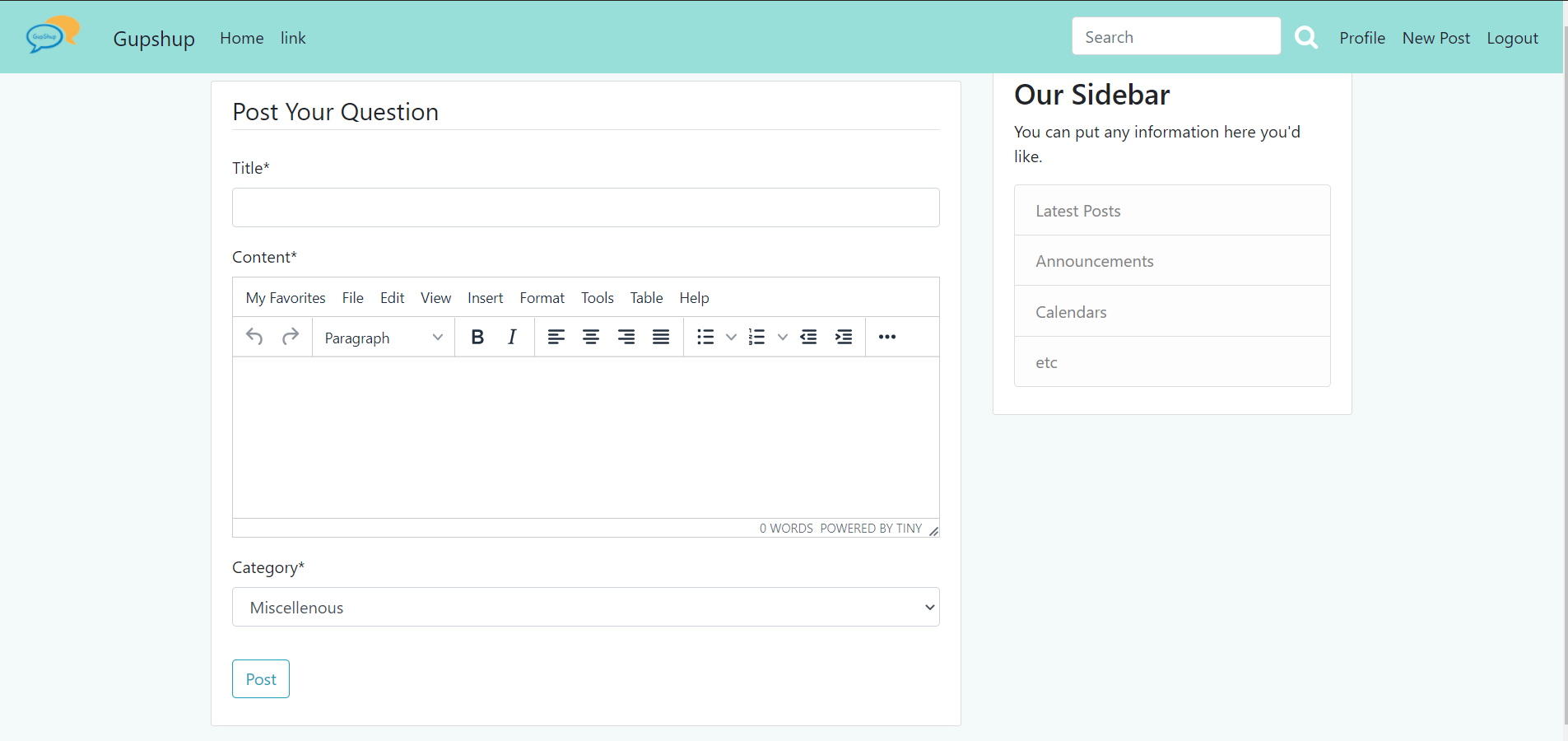
**User interface:**

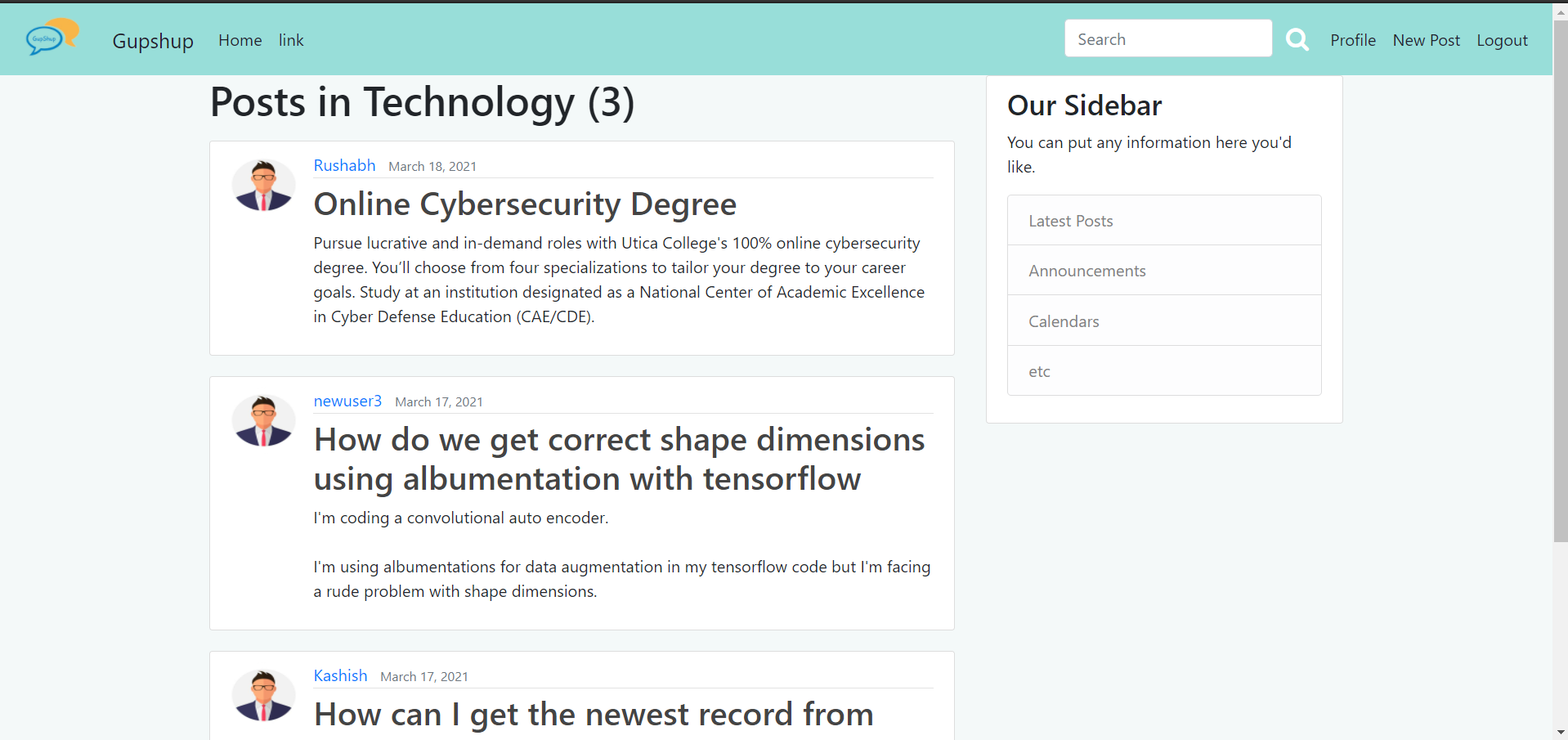
****

****

****





****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Cases for GUPSHUP | | | | | | |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | **Purpose/Objective:** |  | To Validate the functionality of the website | |  |  |
|  | **Pre requisites:** |  | Website, Browser, Internet Connection | |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **Sr.no** | **Objective** | **Procedure** | **Expected Results** | **Actual Result** | **Pass/Fail** | **Remarks** |
|  |
|  | **Registration** |  |  |  |  |  |  |
| 1 | Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only. | username:Arjun13 email:arjun@somaiya.edu password:admin@123 confirm password: admin@123 | data gets added in data base. | data gets added in data base. | Pass |  |  |
|  |  |  |  |  |  |  |  |
| 2 | Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only. | username:Arjun# email:arjun@somaiya.edu password:admin@123 confirm password: admin@123 | data gets added in data base. | data gets added in data base. | Fail |  |  |
|  |  |  |  |  |  |  |  |
| 3 | The password can’t be too similar to your other personal information. | username:Arjun13 email:arjun@somaiya.edu password:admin@123 confirm password: admin@123 | proper error message should be displayed if condition failed | data gets added in data base. | Pass |  |  |
| 4 | The password can’t be too similar to your other personal information. | username:Arjun13 email:arjun@somaiya.edu password:Arjun13 confirm password: Arjun13 | proper error message should be displayed if condition failed | data doesn't get added in data base. | Fail |  |  |
| 5 | The password must contain at least 8 characters. | username:Arjun13 email:arjun@somaiya.edu password:admin@123 confirm password: admin@123 | proper error message should be displayed if condition failed | data gets added in data base. | Pass |  |  |
| 6 | The password must contain at least 8 characters. | username:Arjun13 email:arjun@somaiya.edu password:admin confirm password: admin | proper error message should be displayed if condition failed | data doesn't get added in data base. | Fail |  |  |
| 7 | The password can’t be a commonly used password. | username:Arjun13 email:arjun@somaiya.edu password:admin@123 confirm password: admin@123 | proper error message should be displayed if condition failed | data gets added in data base. | Pass |  |  |
| 8 | The password can’t be a commonly used password. | username:Arjun13 email:arjun@somaiya.edu password:password confirm password: password | proper error message should be displayed if condition failed | data doesn't get added in data base. | Fail |  |  |
| 9 | The password can’t be entirely numeric. | username:Arjun13 email:arjun@somaiya.edu password:admin@123 confirm password: admin@123 | proper error message should be displayed if condition failed | data gets added in data base. | Pass |  |  |
| 10 | The password can’t be entirely numeric. | username:Arjun13 email:arjun@somaiya.edu password:12345 confirm password: 12345 | proper error message should be displayed if condition failed | data doesn't get added in data base. | Fail |  |  |
| 11 | To check for a valid username if changes is available or not. | username:Arjun13 email:arjun@somaiya.edu password:admin@123 confirm password: admin@123 | proper error message should be displayed if condition failed | data gets added in data base. | Pass |  |  |
| 12 | To check for a valid username if changes is available or not. | username:kashish email:arjun@somaiya.edu password:admin@123 confirm password: admin@123 | proper error message should be displayed if condition failed | data doesn't get added in data base. As username already exist | Fail |  |  |
|  | **Login** |  |  |  |  |  |  |
| 13 | To check whether the user is email verified | username:kashish password:admin@123 | proper error message is displayed | data gets retrieved from data base. | Pass |  |  |
| 14 | To check whether the user is email verified | username:kashish password:admin@123 | proper error message is displayed if email is not verified. | data gets retrieved from data base. | Fail |  |  |
|  |  |  |  |  |  |  |  |
| 15 | Check the username and password is valid | username:kashish password:admin@123 | proper error message should be displayed if condition failed | data gets retrieved from data base. | Pass |  |  |
| 16 | Check the username and password is valid | username:kashish password:admin@123 | proper error message should be displayed if wrong credentials are entered | data gets retrieved from data base. | Fail |  |  |
|  | **Posts** |  |  |  |  |  |  |
| 17 | To check whether a user is verified | click on post button | proper error message is displayed | data gets retrieved from data base. | Pass |  |  |
| 18 | To check whether a user is verified | No button displayed | proper error message is displayed | data doesn't get added in data base. | Fail |  |  |
|  |  |  |  |  |  |  |  |
| 19 | To check whether Post field is empty | Title: Test title Content:Sample content category: technology | proper error message is displayed | data gets added in data base. | Pass |  |  |
| 20 | To check whether Post field is empty | Title: Test title Content:Sample content category: | proper error message is displayed | data doesn't get added in data base. | Fail |  |  |
|  | **Update and Delete Posts** |  |  |  |  |  |  |
| 21 | To check if the post has the same author as the logged in user | To open post detail | The button appears only when the condition is satisfied | data gets added in data base. | Pass |  |  |
| 22 | To check if the post has the same author as the logged in user | post detail : button doesn't appear | The button appears only when the condition is satisfied | data doesn't get added in data base. | Fail |  |  |
|  | **Update User Profile** |  |  |  |  |  |  |
| 23 | To check for valid email id if changed belongs to somaiya domain | username: Kashish email:kashish@somaiya.edu photo:sample.jpg | proper error message should be displayed | data gets added in data base. | Pass |  |  |
| 24 | To check for valid email id if changed belongs to somaiya domain | username: Kashish email:kashish@gmail.com photo:sample.jpg | proper error message should be displayed | data doesn't get added in data base. | Fail |  |  |
|  |  |  |  |  |  |  |  |
| 25 | To check for a valid username if changes is available or not. | username: Kashish email:kashish@somaiya.edu photo:sample.jpg | proper error message is displayed | data gets added in data base. | Pass |  |  |
| 26 | To check for a valid username if changes is available or not. | username: arjun(which already exist) email:kashish@somaiya.edu photo:sample.jpg | proper error message is displayed | data doesn't get added in data base. | Fail |  |  |
|  |  |  |  |  |  |  |  |
| 27 | Check the username if changed is valid | username: arjun email:kashish@somaiya.edu photo:sample.jpg | if the condition fails, error message is displayed | data gets retrieved from data base. | Pass |  |  |
| 28 | Check the username if changed is valid | username: #arjun email:kashish@somaiya.edu photo:sample.jpg | if the condition fails, error message is displayed | data doesn't get added in data base. | Fail |  |  |
|  | **Upvotes/ Downvotes** |  |  |  |  |  |  |
| 29 | To check whether a user is logged in | if logged in | proper error message is displayed | data gets retrieved from data base. | Pass |  |  |
| 30 | To check whether a user is logged in | if not logged in | proper error message is displayed | data doesn't get added in data base. | Fail |  |  |
|  |  |  |  |  |  |  |  |
| 31 | To check is the post is already upvoted or downvoted by the logged in user | press upvote/downvote button | The buttons are highlighted according to the data retrieved | data gets added in data base. | Pass |  |  |
|  | **Answers** |  |  |  |  |  |  |
| 32 | To check whether a user is logged in | If logged in | proper error message is displayed | data gets retrieved from data base. | Pass |  |  |
| 33 | To check whether a user is logged in | If not logged in | proper error message is displayed | data doesn't get added in data base. | Fail |  |  |
|  | **Bookmarks** |  |  |  |  |  |  |
| 34 | To check whether a user is logged in | If logged in | proper error message is displayed | data gets retrieved from data base. | Pass |  |  |
| 35 | To check whether a user is logged in | If not logged in | proper error message is displayed | data doesn't get added in data base. | Fail |  |  |
|  | **Miscelleneous** |  |  |  |  |  |  |
| 36 | To check whether a category of the post is "announcement" | category : announcement | The post gets added to the marquee | data gets added in data base. | Pass |  |  |
| 37 | To check whether a category of the post is "announcement" | category : bussiness | The post gets added to the marquee | data doesn't get added in data base. | Fail |  |  |
| 38 | For Live Chat: To check whether chats get stored after refreshing the page | click on chat button | Does Not store previous chats | data does not get added in data base. | Fail |  |  |
| 39 | To check whether a post opens when clicked on heading in marquee. | On clicking the post heading in the marquee | Post details page opens | data gets fetched from database | Pass |  |  |
| 40 | To check whether category page opens when clicked on its name on the secondary navigation bar. | On clicking the heading "Mechanical" | Post with category having the name "mechanical" opens. | data gets fetched from database | Pass |  |  |
| 41 | To check whether category page opens when clicked on its name on the secondary navigation bar. | On clicking the heading "Technology" | Post with category having the name "Technology" opens. | data gets fetched from database | Pass |  |  |
| 42 | To check live chat opens on clicking the chat button in a particular category | On clicking the chat button on "Technology" page | Chat page with category having the name "Technology" opens. | Channel is created | Pass |  |  |
| 43 | To check if chat button appears on category page | On clicking the heading "Technology". The button appears if the user is logged in. | Chat page with category having the name "Technology" opens. | Channel is created | Pass |  |  |
| 44 | To check if chat button appears on category page | On clicking the heading "Technology". The button doesn't appear if the user not is logged in. | The chat button doesn't appear. | Channel is not created | Fail |  |  |
| 45 | To check whether category page opens when clicked on its name on the Quick access sidebar. | On clicking the heading "Technology" | Post with category having the name "Technology" opens. | data gets fetched from database | Pass |  |  |
| 46 | To check if bookmark button appears on the quich access sidebar. | If the user is logged in the button appears. | The button appears | Appropriate output is shown | Pass |  |  |
| 47 | To check if bookmark button appears on the quich access sidebar. | If the user is not logged in then button doesn't appear. | The button does not appear | Appropriate output is shown | Fail |  |  |
| 48 | To check if the new post page open when clicked on the button"click here to post" | Click on the "click here to post" button | If the user is logged in the new post page opens. | Appropriate output is shown | Pass |  |  |
| 49 | To check if the new post page open when clicked on the button"click here to post" | Click on the "click here to post" button | If the user is not logged in the login page opens. | Appropriate output is shown | Fail |  |  |
| 50 | To check whether a post opens when clicked on "view post" on the carousel. | On clicking the "view post" on the carousel. | Post details page opens | data gets fetched from database | Pass |  |  |

#### **4.5 GITHUB Implementation:**

# **Link of GITHUB project:** [**https://github.com/kashish1211/GUPSHUP**](https://github.com/kashish1211/GUPSHUP)

# **Implementation details:**

# ***1.*** **Using *clone* command:**

# ⮚ **Cloning an Empty repository already created by us on GitHub.**

# **Command:** git clone: https://github.com/kashish1211/GUPSHUP

# 

# ***2.*** **Using *add* command:**

# a) **Command:** git add .

# '.' refers to all the files which have changes

# git add filename(filename is the file you edited)

# ***3.*** **Using *commit* command:**

# **Command:** git commit -m "any message"

# ***4.*** **Using *push* command:**

# To push your changes to github repo.

# **Command:** git push origin master

# 

# ***5.*** **Using *pull* command:**

# To pull any changes to your local project made by other collaborators.

# **Command:** git pull

# ***6.*** **Using *status* command:**

# It shows the state of your working directory and helps you see all the files which are untracked by Git, staged or unstaged.

# **Command**: git status

# 

# **Chapter 5**

# **5.1 CONCLUSION**

Thus, in this project we have successfully understood and implemented the concepts of Django, Python, HTML, CSS, Bootstrap, JavaScript, Docker and Web Sockets

We understood various concepts Django including the running a server, rendering templates, error handling, creating web-sockets channels and using docker for our live chat feature, storing data to the database and eventually fetching it in the HTML templates. We also learnt how to make a responsive website which is compatible with both desktop as well as mobile device.

# **5.2 FURTHER WORK**

* Currently the website has a live chat feature which can be modified to facilitate private chat as well.
* There is also a scope to add a feature of calculating the trust rating of a user based on the accuracy of his posts and answers/comment

# **REFERENCES**

# <https://www.udemy.com/course/python-and-django-full-stack-web-developer-bootcamp/>

# <https://www.python-course.eu/>

# <https://developer.mozilla.org/en-US/docs/Glossary/Python>

# <https://docs.djangoproject.com/en/3.1/>

# <https://getbootstrap.com/docs/4.0/getting-started/introduction/>

# <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

# <https://tailwindcss.com/> <https://stackoverflow.com/>

# <https://www.geeksforgeeks.org/>

# <https://channels.readthedocs.io/en/latest/tutorial/part_2.html>

Books Referred:

Django for APIs: Build web APIs with Python & Django

Django for Beginners: Build websites with Python and Django

Django Design Patterns and Best Practices Django Unleashed