DBMS Lab (L1)- Mini-Project
“Blogify”

Prepared by:

Hatim Murtuza (31124)

Jinesh Parakh (31126)

Vikrant Kashid-Patil (31130)

Submission Date:

05/12/2020

ABSTRACT

Blogs are informal articles written to show thought, leadership and expertise on a topic. They are a great way to generate fresh content on a website and provide a catalyst for email marketing, social media promotion to drive search traffic to your website. With the rising blogging culture, an efficient platform to provide a smooth experience of both reading and writing blogs is the need of time.

Hence, the main aim of this project revolves around creating a website/platform that would enable "bloggers" to enhance their blogging experience by exploring different blogs by various authors throughout the world and also share their ideas, experiences, etc. with the rest of the community. To incorporate this idea, we have provided a wide variety of features and functionalities. To list a few, the website provides the user freedom of creating their blog, updating their old blogs and also deleting them after logging in. To filter the various blogs updated on the website depending on the author or the blog tags. To attract users to explore more sections of the website, an interactive homepage featuring the 3 most recent blogs followed by other blogs on the next pages etc. have been provided.

The website uses SQLAlchemy from flask backed with the SQLite database to post and fetch the required data for the various functionalities and features. The frontend has been created using JavaScript, HTML, Bootstrap and CSS, while the backend has been written in python. The choice of this particular database allowed us to avoid redundancy in data. The choice of SQLAlchemy from flask backed with the SQLite database would also support further enhancement of this project with growing business needs because of its flexible data model.

This project being a basic prototype for a small model in Blog Management systems also leaves a lot of scope for future work. Hence, the technologies selected for its development would be apt to support the further additions to be made without any scalability or availability issues cropping up.

TABLE OF CONTENTS

Table 1. Table of Content

CHAPTER 1

INTRODUCTION

1.1 Introduction to Project

Blogs are informal articles written to show thought, leadership and expertise on a topic. They are a great way to generate fresh content on a website and provide a catalyst for email marketing, social media promotion to drive search traffic to your website. The emerging trend of blogs is a major factor that has influenced the idea of developing : "Blogify".

Keeping in mind the popularity of blogs, we designed a prototype to provide an online interface for exploring various blogs, and enhancing your experience.

1.2 Motivation behind the Project Topic

A blog is a piece of written content that is published on a web page or site. The topics of blogs can vary from person to person, or even business to business. The main purpose of blogs is to convey information in a way that is more informal or conversational than other long-form written content. A blog website is a site that is updated with new information on an ongoing basis. It normally consists of a collection of posts. Posts may be short, informal, controversial, or more professional.
Thinking on these lines, this project attempts to create a better physical experience for users, by facilitating creation, updation, deletion and reading blogs with ease.

1.3 Aim and Objectives of the work

Project aims to create an online platform to facilitate writing, reading, filtering and updating blogs to enable a hassle-free experience.

Project Objectives:

- To create a working prototype of a blogging website to write and read blogs
- To make use of various web technologies to create a platform for the easy and hassle-free exploration of new ideologies, concepts, etc. written in forms of blogs.

CHAPTER 2

SCOPE

2.1 Website Functionalities/Features Descriptions

The entire website is supported with a database connectivity to facilitate posting and fetching of data to and from the database. All the features are backed by a database which supports their functionalities. The website consists of an App bar in the header for smooth navigation through the entire website.

- **User Signup Page:**
  → A Flask Signup Form to gather data and Register A New User
- **User Login Page:**
  → A Flask Login Form to Login a Registered User
- **Home Page:**
  → The Home Page showcases all the existing Blogs in a paginated Fashion, with atmax 3 blogs on a page.
- **Create A New Blog:**
  → Any logged-in user can Create New Blogs using Markdown to enhance the readability and UX for other readers.
- **Creation of A New Tag For a Blog:**
  → If for any Blog, the user requires a New Tag which is not pre-existing in the database he/she can create it to his/her own disposal.
- **Update A Pre-existing Blog:**
  → Any logged-in User having a blog can Update the Blog for it's Title, Tags and Content.
- **Delete A Blog:**
  → The User who has a Blog of his own, can choose to delete it whenever required.
- **View All Blogs By A User:**
  → Clicking on any Username will display all the Blogs by that User in a paginated fashion with atmax 3 Blogs per page.
- **View All Blogs With A Specific Tag:**
  → Clicking on any Tag for a Blog Post will show all the Blogs posts having that specific tag in all the tags.

- **Update Account Information:**
  → Any Logged-In User can Update his/her account information viz. Email, Username and can add a Custom Profile Picture.
- **Password Reset Feature:**
  → Dynamically generated unique token with expiry time of 30mins are emailed to the registered User for a Password Reset Request.

2.2 Future Work

Designing a scalable and extremely efficient platform for a fully functional blogging website is a huge task. Currently having designed only a basic prototype for this system there is a lot of scope for future work.

The current module can be enhanced further to include a search bar to make it easy for our users to read blogs pertaining to their interests. To include the functionalities of like, share, comment and follow. The website will be linked to several social media platforms for the ease of our users. As these features require an actual working dataset, it was difficult to incorporate them in this project.

Other modules such as safety and security controls, maintenance management, data analysis models etc. can also be added to transform it into a fully functioning deployable blogging website.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Hardware Requirements

To access this web application there's only need of PC / Laptop with an integrated and updated web browser

a) Windows 10 / Ubuntu 20.04 LTS Operating System
b) 4 GB RAM
c) 512 GB HDD
d) Keyboard
e) Mouse etc.

3.2 Software requirements:

1. SQLite : SQLite is a relational database management system contained in a C library. In contrast to many other database management systems, SQLite is not a client–server database engine.

2. SQLAlchemy : SQLAlchemy is an open-source SQL toolkit and object-relational mapper for the Python programming language released under the MIT License.

3. Python - Flask : Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.

CHAPTER 4
DATA MODELING FEATURES

4.1. Use Case:

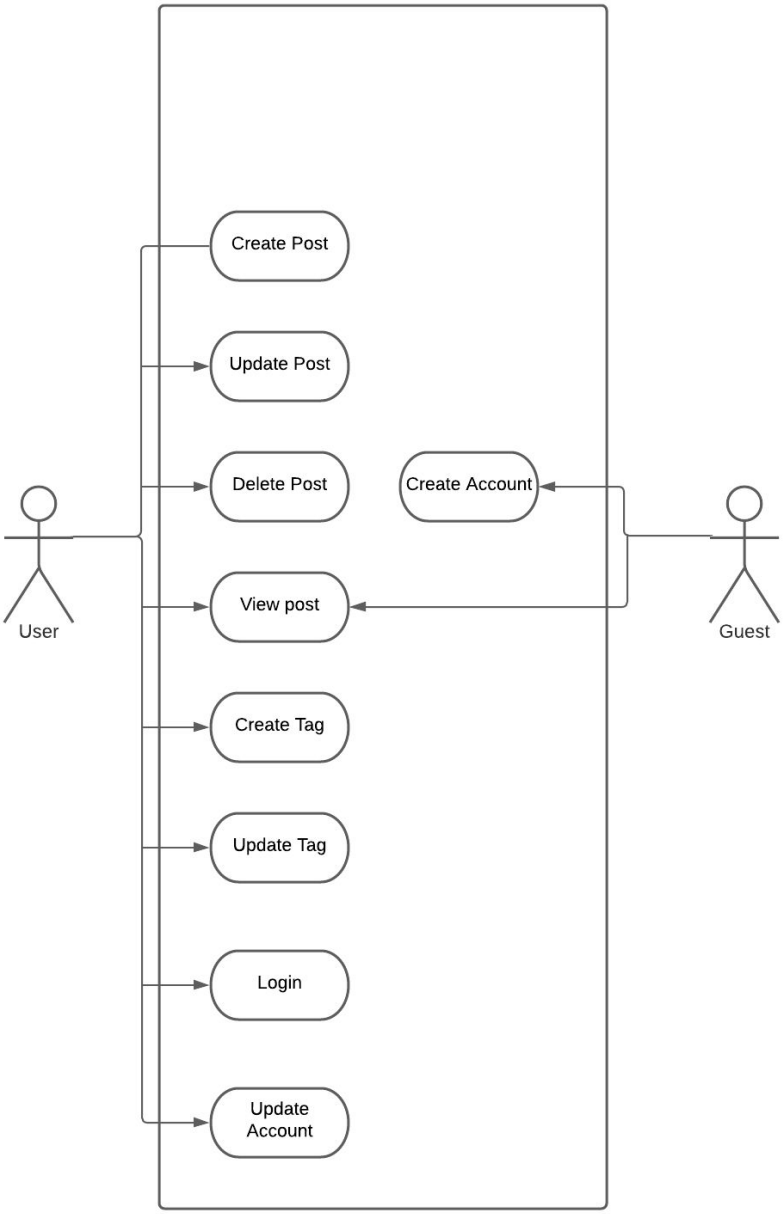

Figure 1: Use Case Diagram

Use case 1 - Login to app

Actor: User
Description: Only a registered user can login to the app to create, update or delete their posts and access the various features of the app that can change the database.

Use case 2 - Register

Actor: Guest
Description: A guest can create a new account if they wish to write a blog or make changes to the database.

Use case 3 - Update User Information

Actor: User
Description: Users can update details like their name, email address and profile picture from the accounts page when needed.

Use case 4: Create a post

Actor: User
Description: A registered user can write blogs using the markdown editor. To tend to specific groups that might like their post the user can add tags to each post.

Use case 5: View Post

Actor(s): User, Guest
Description: Any visitor to the app guest or user regardless can view blogs created by a user. As per their liking users can filter out blogs written by one or more authors. The app also allows posts to be filtered by tags which allows guests and users alike to view new posts in genres they love.

Use case 6: Update post

Actor: User
Description: Users can update their own posts using the update functionality of the app to correct mistakes or change the outlook of their blog. They can also change the tags or add new tags to the post as they feel necessary.

Use case 7: Delete post

Actor: User
Description: Users can delete their post by clicking on it and then clicking the delete button.

Use case 8: Create Tag

Actor: User
Description: Users can create new tags for new genres of posts they write making it easier for people to find posts in that genre by filtering them. This can be done while writing a new post and clicking on the "create tag" button.

Use case 9: Update Tag

Actor : User
Description: Users can update the posts linked to their blogs by selecting and deselecting them in the dropdown of tags while editing the post.

Features	Tables used
Create Post	post
Fetch Post	post
Update Post	post
Delete Post	post
Register User	user
User Login	user
Update User Details	user
Create Tags	tag
Add tags to post	post, tag, post_tag
Update tags of post	post, tag, post_tag

Table 2. Feature-Table Mapping

CHAPTER 5

DATABASE DESIGN

This project has been developed using SQLite software which is an embedded relational database.

5.1. ER Diagram

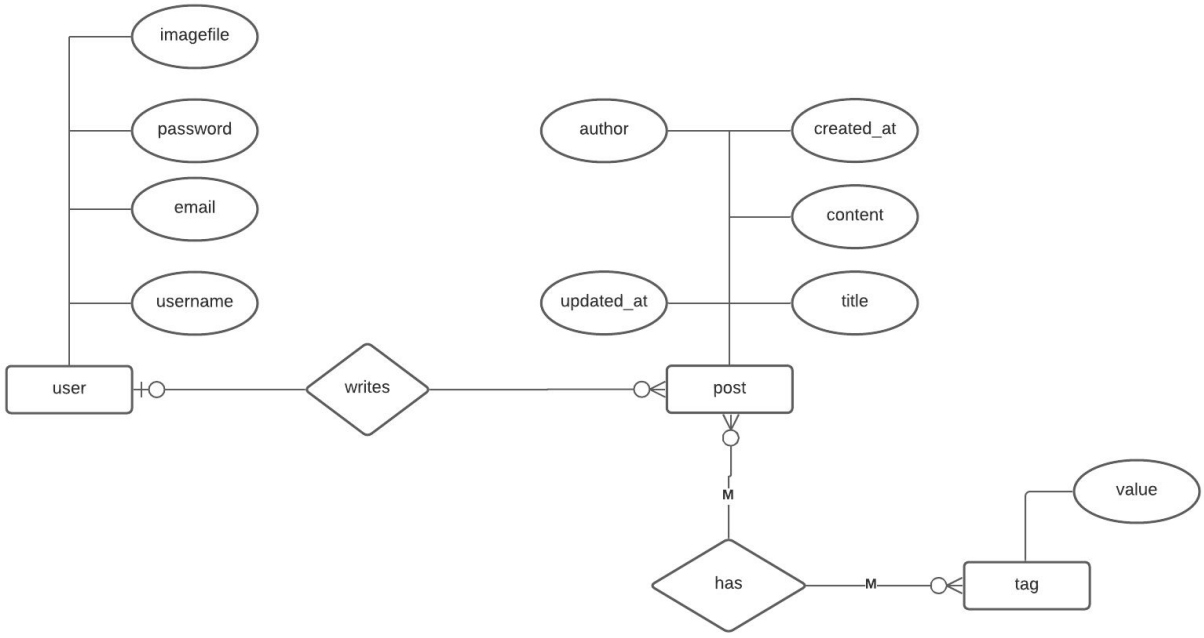

Figure 2: Entity Relationship Diagram

CHAPTER 6

RELATIONAL DATABASE SCHEMA

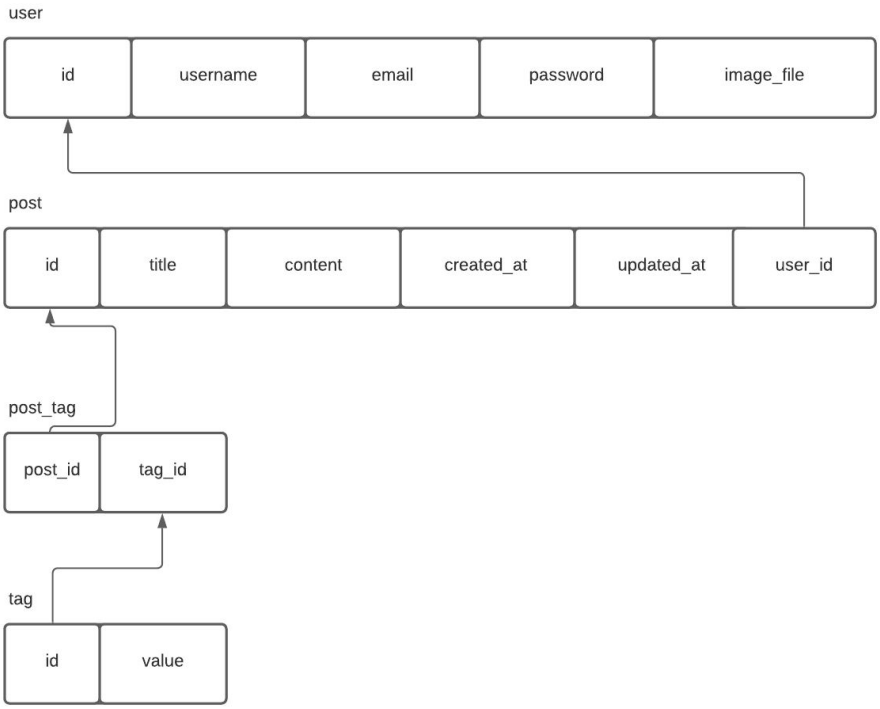

Figure 3: Schema Diagram

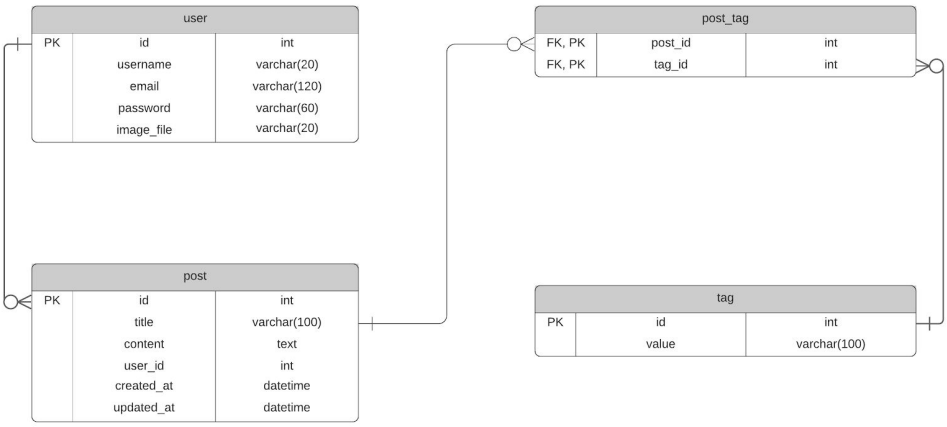

Figure 4: Class Diagram

CHAPTER 7

DATABASE NORMALIZATION

Table name: user
State: Second Normal Form
Reason:

- No multi-valued attribute. Intersection of each row and column has only one value.
- No partial dependencies. Only one unique attribute to identify each tuple is present and no redundancies.

Proof:
The user table does not have any multi-valued attribute and the intersection of each row and column has only one value. Hence, it is in First Normal Form.

The user table has no partial dependencies as all the columns of the table are dependent on the `id.` Hence, the user table is in Second Normal Form.

In the users table `image_file` is a functional dependency of the user, forming a transitive relation as shown:

```
Id -> email -> image_file
```

which transitively makes `image_file` a transitive dependency of the user.
Hence, the user table is not in the Third Normal Form.

Table name: post
State: Second Normal Form
Reason:

- No multi-valued attribute. Intersection of each row and column has only one value.
- No partial dependencies. Only one unique attribute to identify each tuple is present and no redundancies.

Proof:
The post table does not have any multi-valued attribute and the intersection of each row and column has only one value. Hence, it is in First Normal Form.

The post table has no partial dependencies as all the columns of the table are dependent on the `id.` Hence, the user table is in Second Normal Form.

In the post table `content` is a functional dependency of the `title`, forming a transitive relation as shown:

```
id -> title -> content
id -> title -> updated_at
```

which transitively makes `content` a transitive dependency of post.

Hence, the post table is not in the Third Normal Form.

Table name: tag
State: Third Normal Form
Reason:

- No multi-valued attribute. Intersection of each row and column has only one value.
- No partial dependencies. Only one unique attribute to identify each tuple is present and no redundancies.
- No transitive dependencies. No non-primary-key attribute is transitively dependent on the primary key.

Proof:

The tag table does not have any multi-valued attribute and the intersection of each row and column has only one value. Hence, it is in First Normal Form.

The tag table has no partial dependencies as all the columns of the table are dependent on the `id`. Hence, the user table is in Second Normal Form.

In the tag table, there are no functional dependencies forming a transitive relationship, hence it is in Third Normal Form.

Table name: post_tag
It is a join table created to normalize the many to many relation between tags and posts.

CHAPTER 8

GRAPHICAL USER INTERFACE

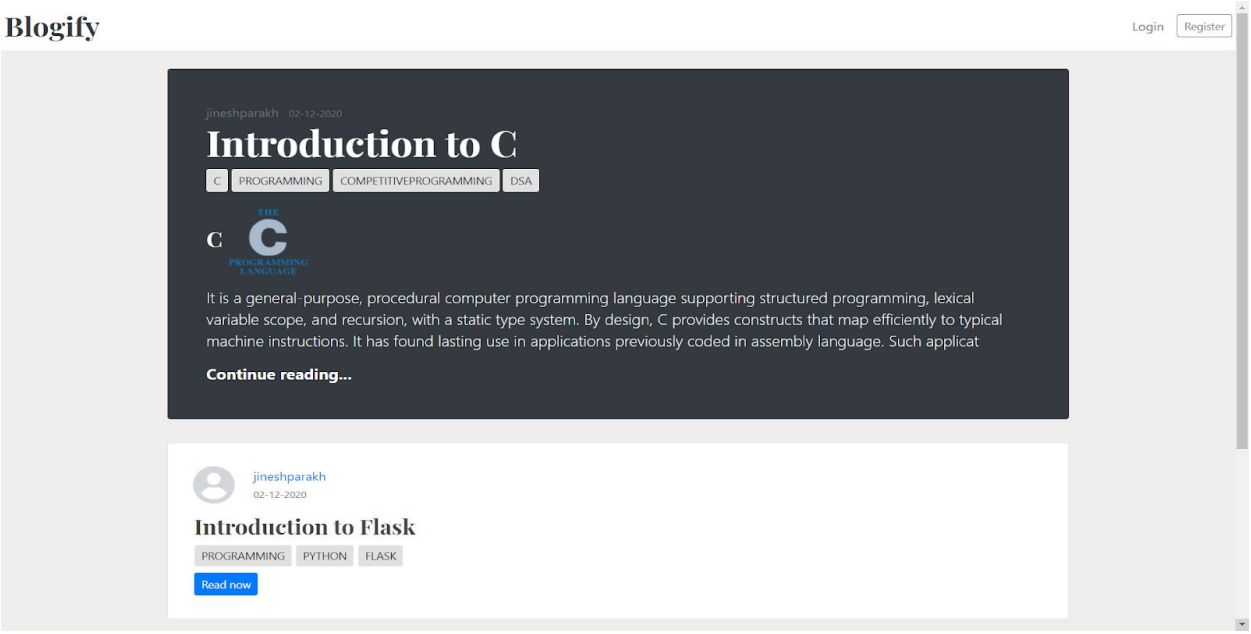

Figure 5: Home Page of Blogify

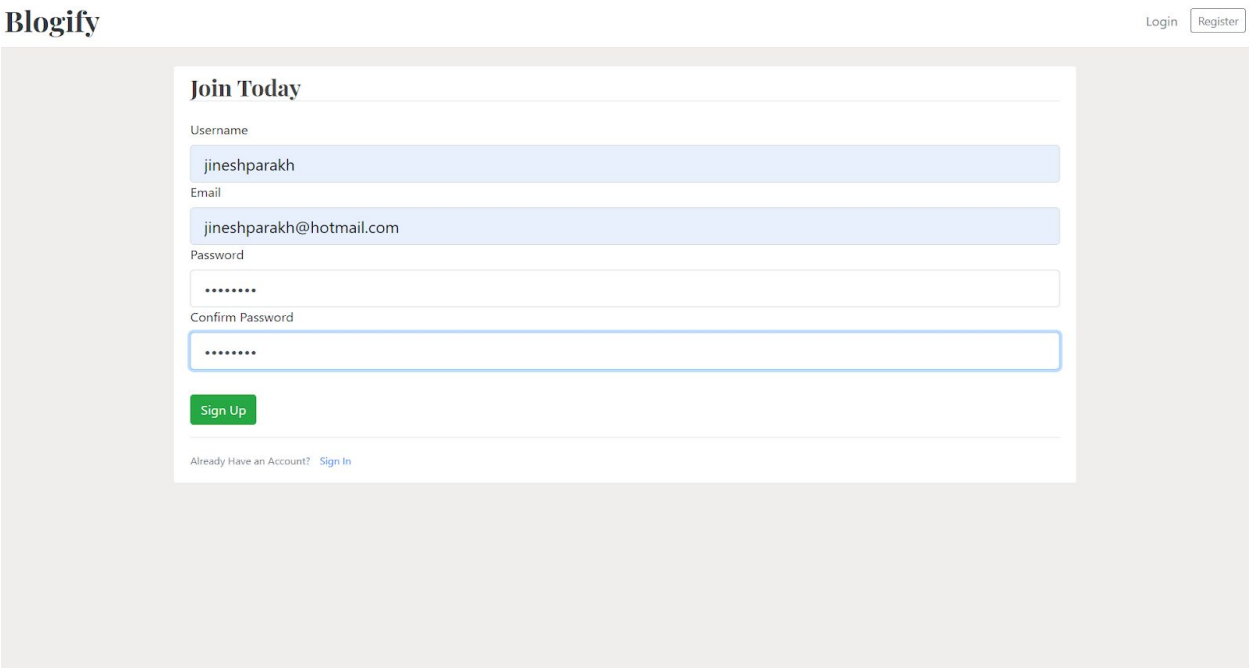

Figure 6: Registration Functionality

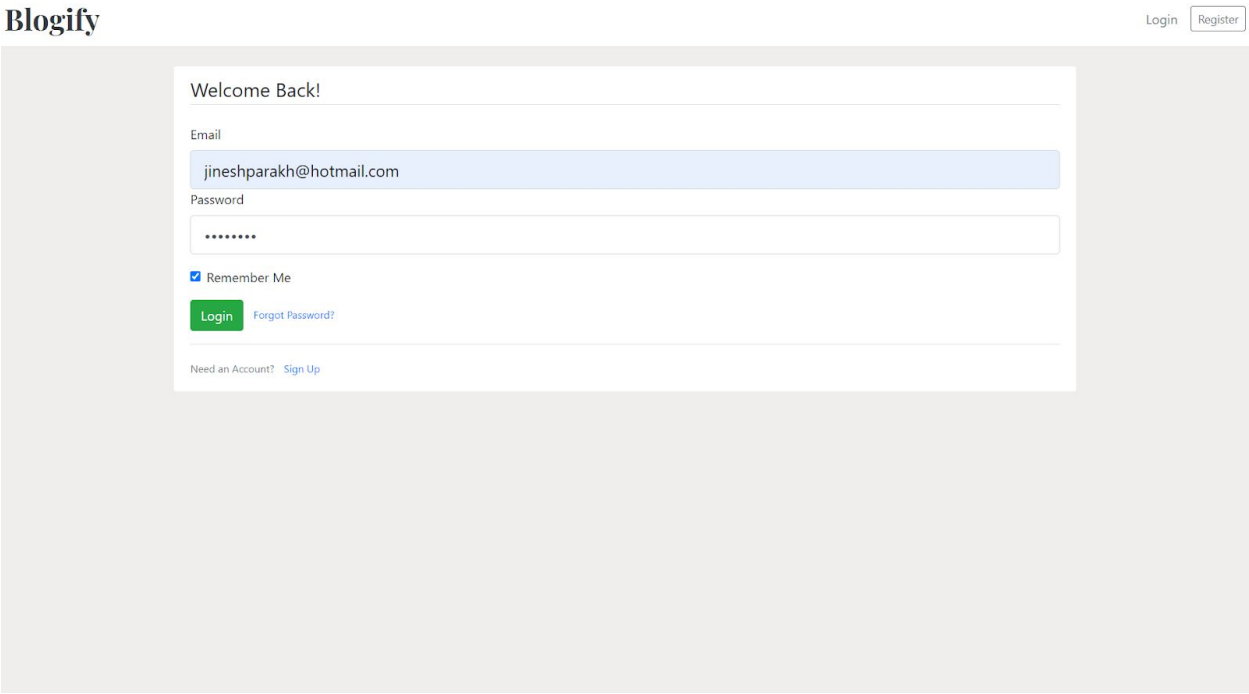

Figure 7: Login Functionality

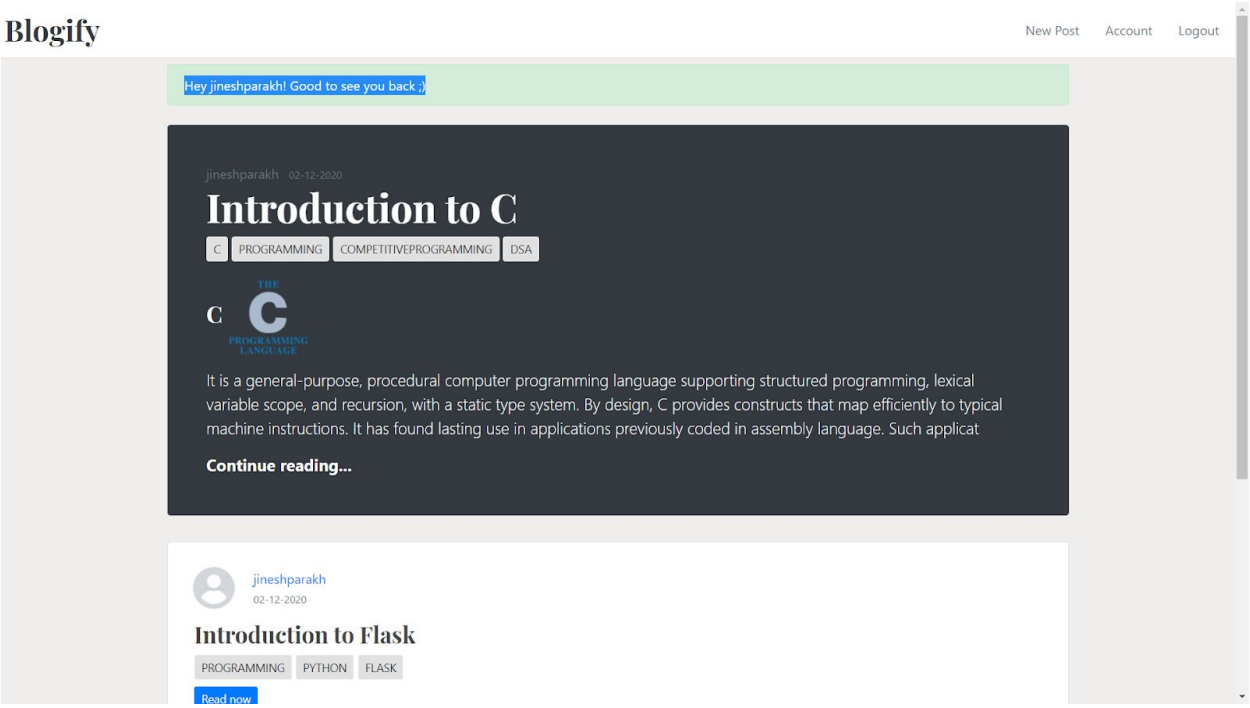

Figure 8: Home Page of Blogify

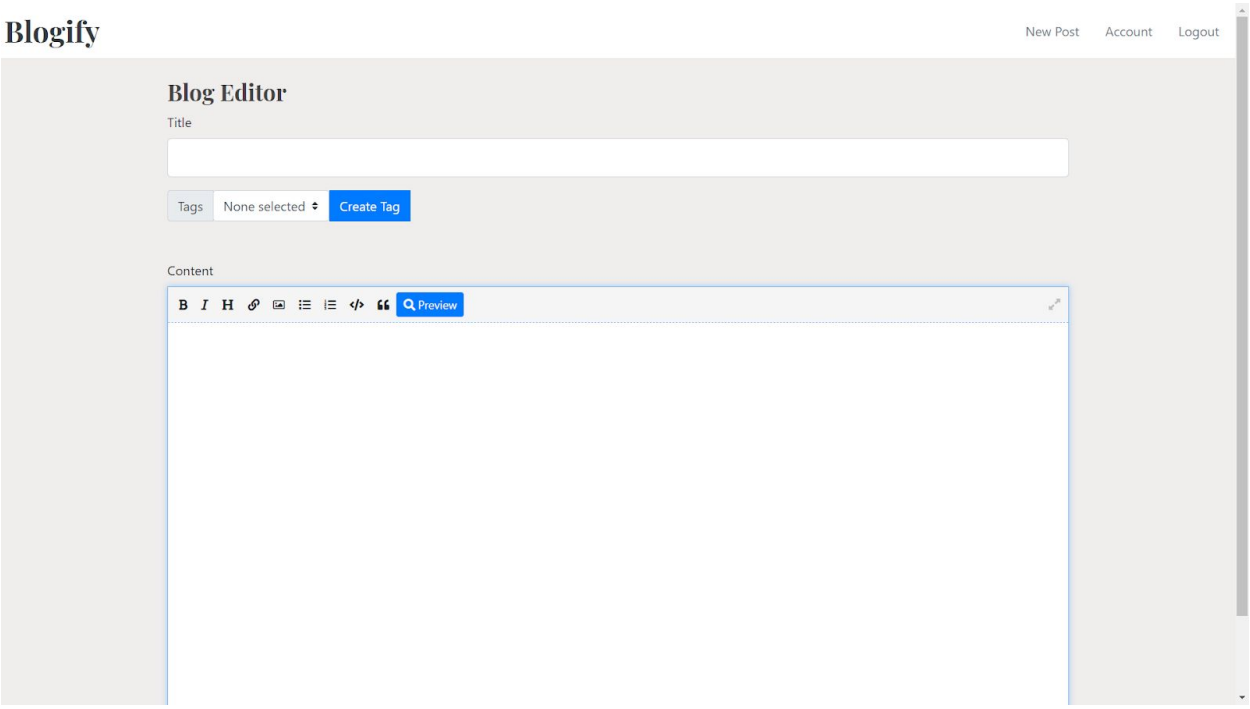

Figure 9: Functionality to Create a New Blog

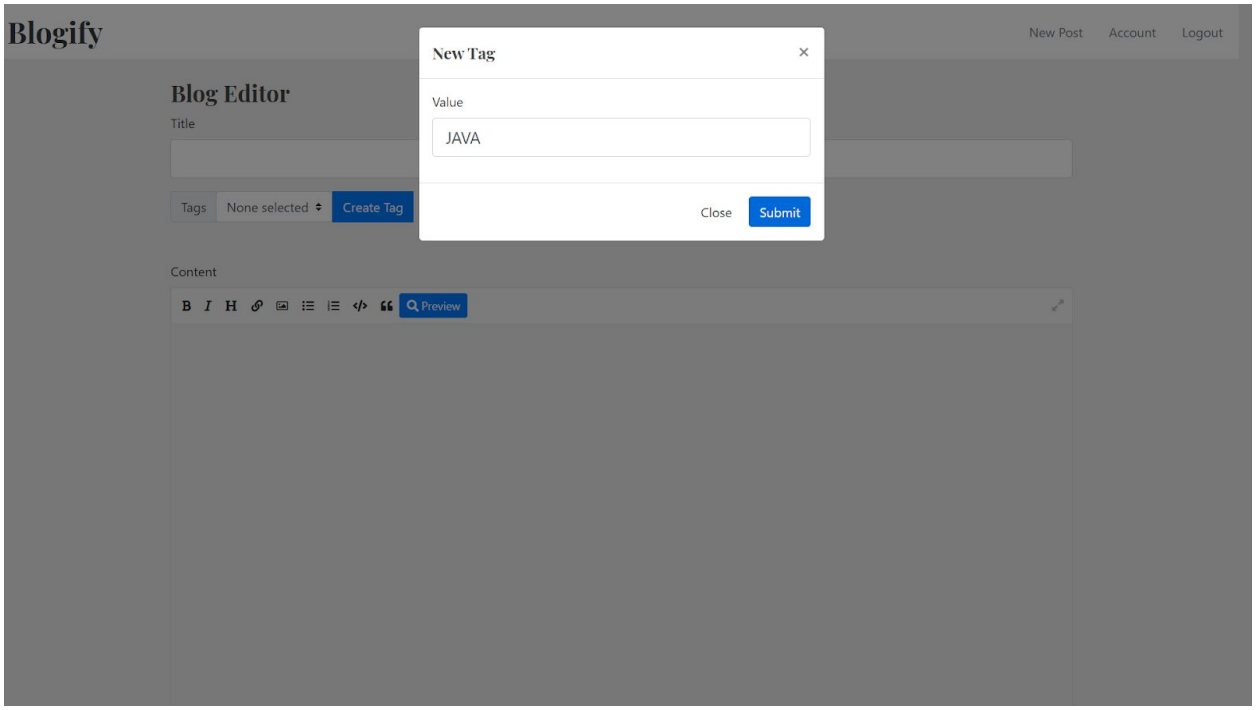

Figure 10: Functionality to add a New Tag Into the Database

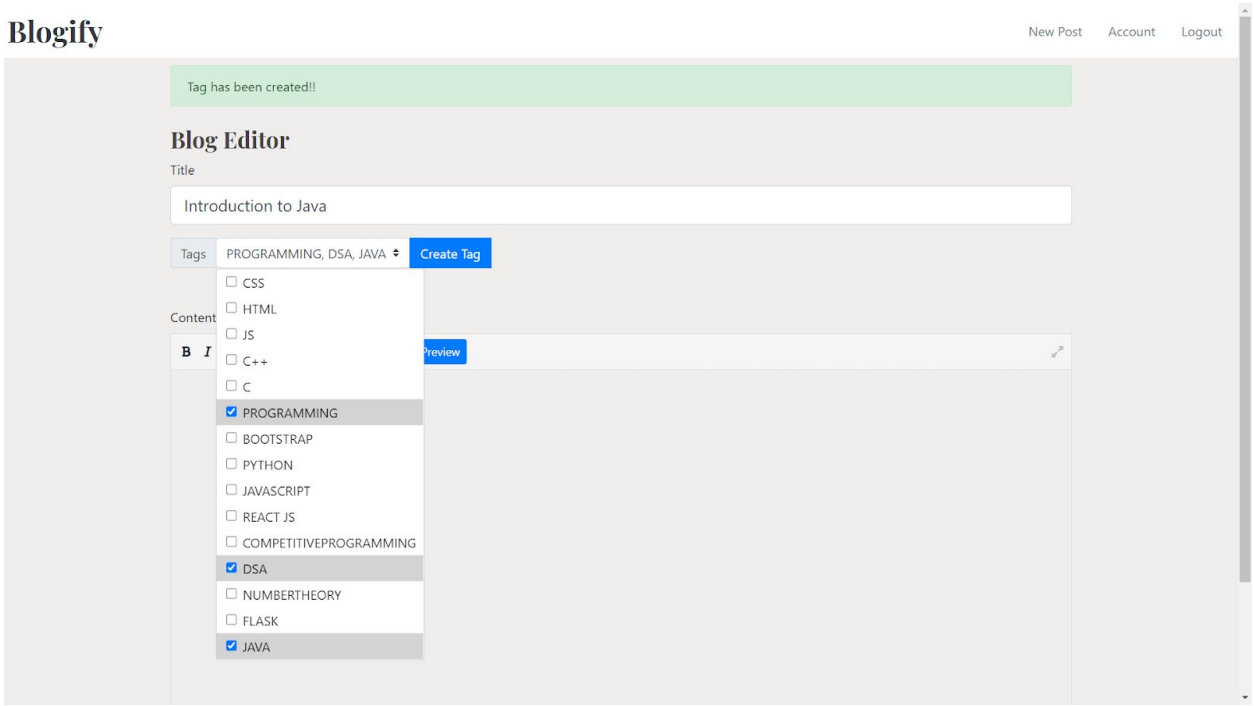

Figure 11: Feature to choose one or many tags for your Blog

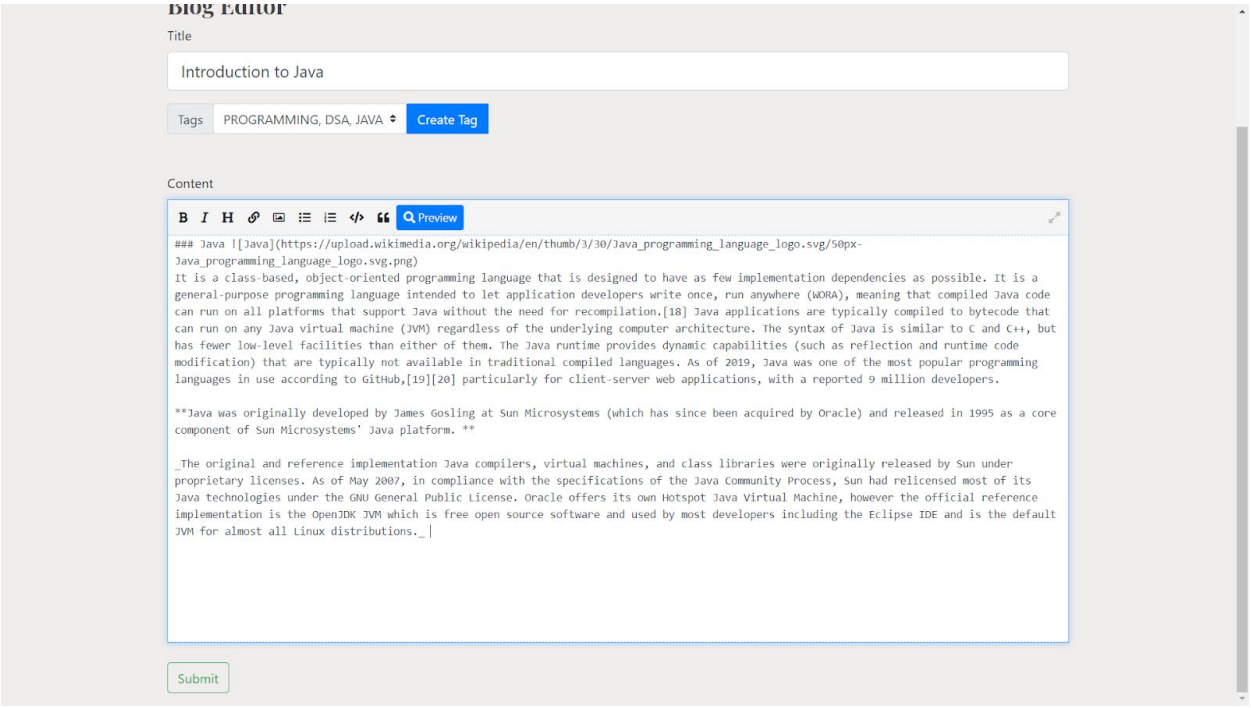

Figure 12: Feature to Write the Blogs in Markdown Feature

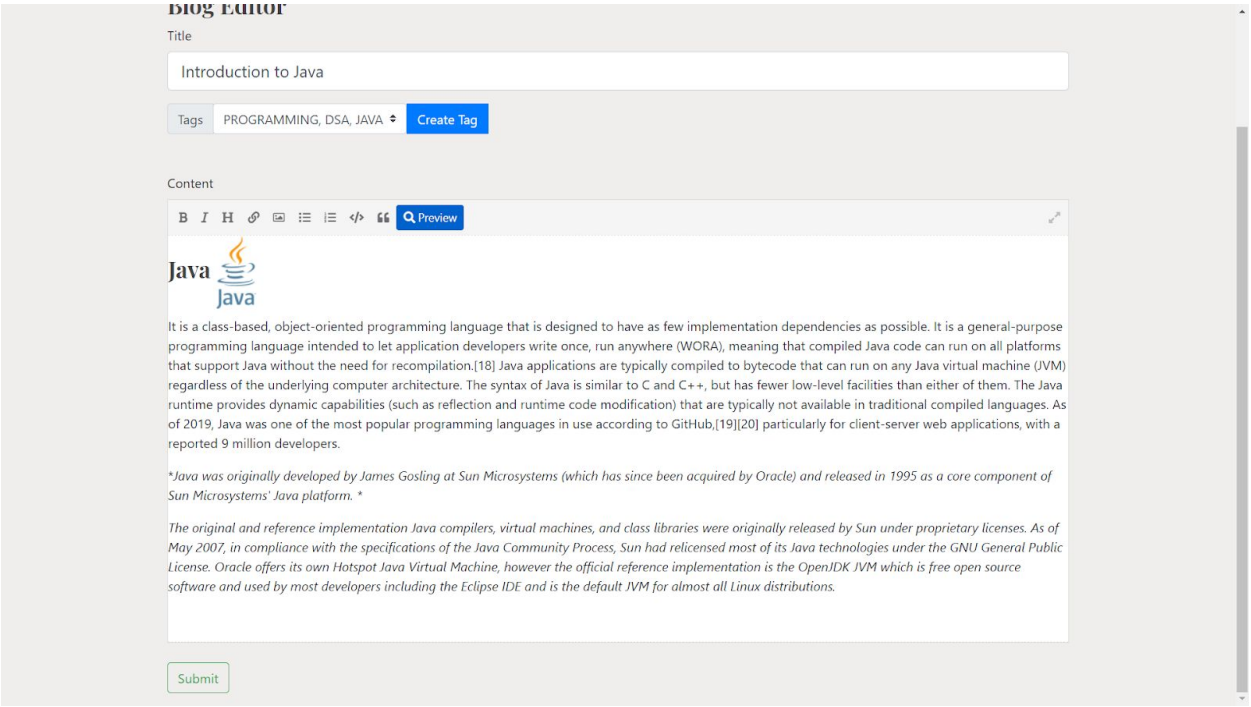

Figure 13: Feature to preview the exact look of the Blog prior to it's Creation

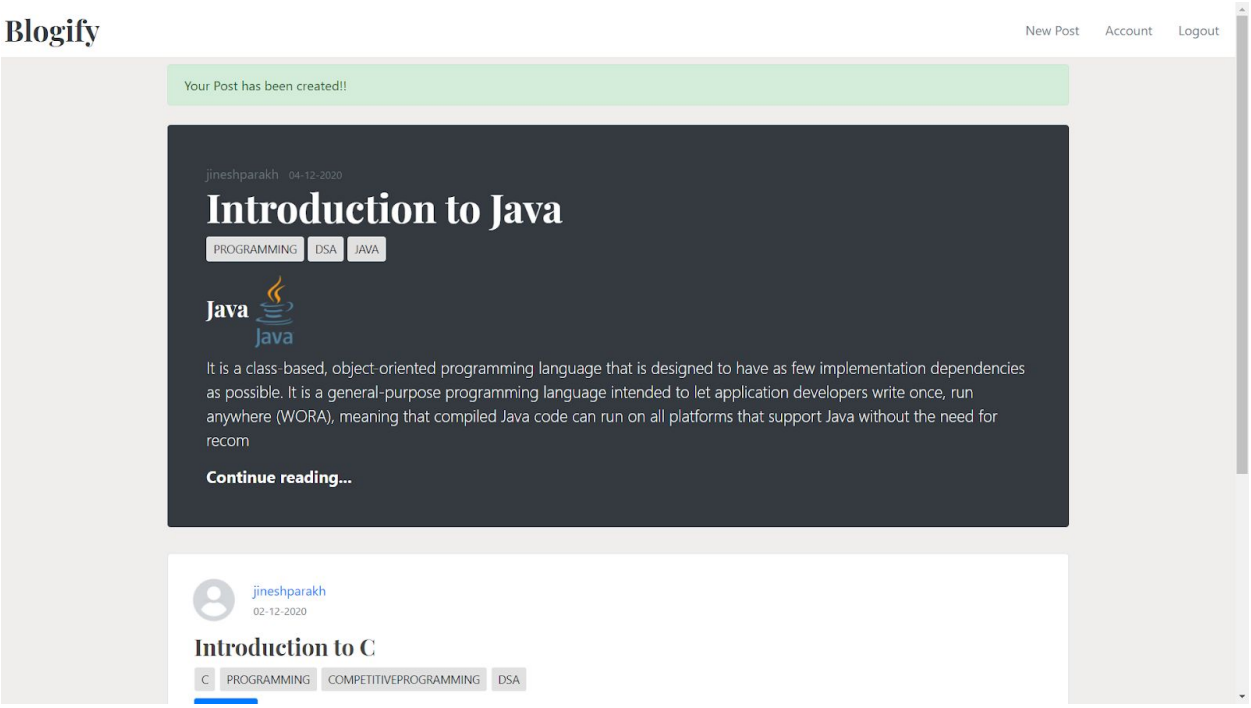

Figure 14: Newly created Post being displayed on the top of the Home Page

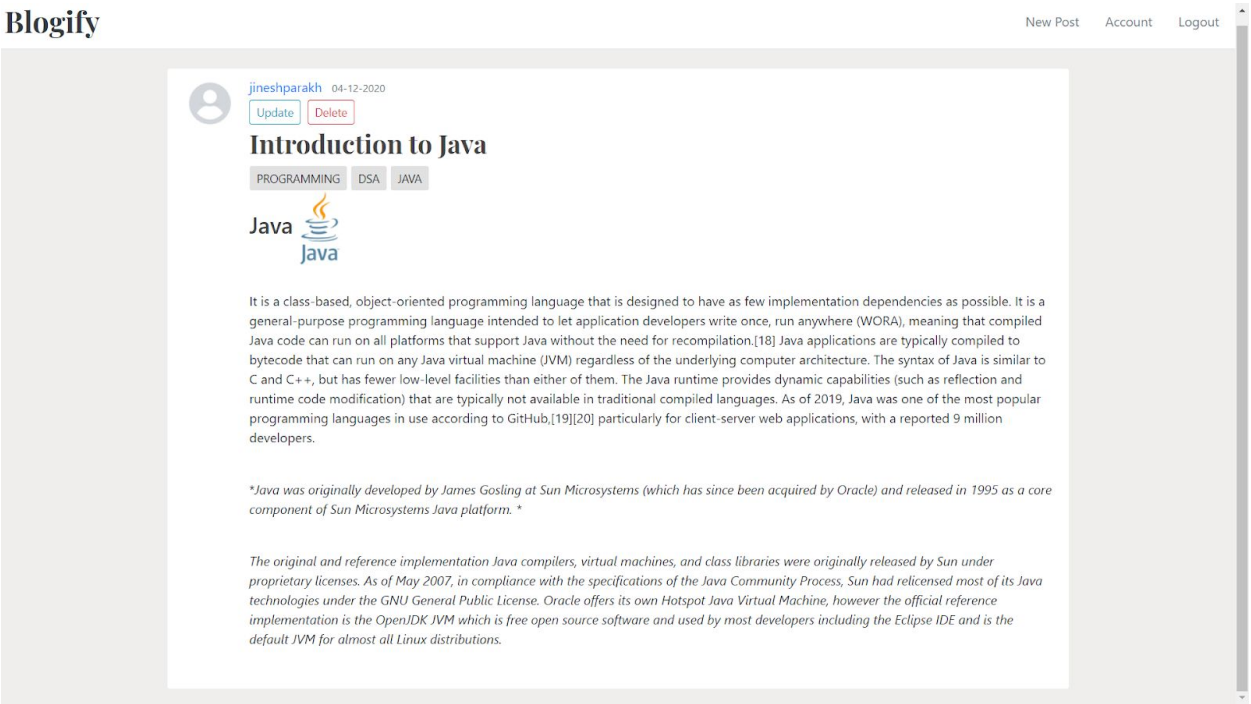

Figure 15: Feature to Update And Delete a Blog for a Registered User

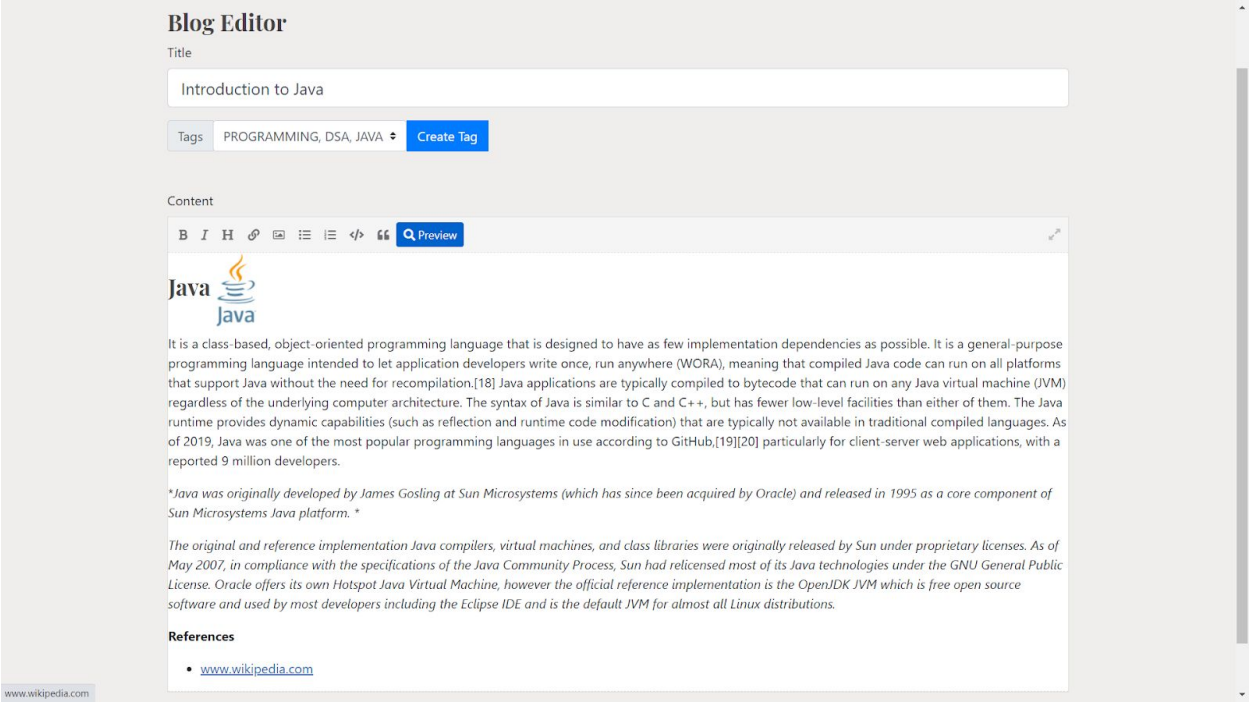

Figure 16: Update the Existing Blog and Add References

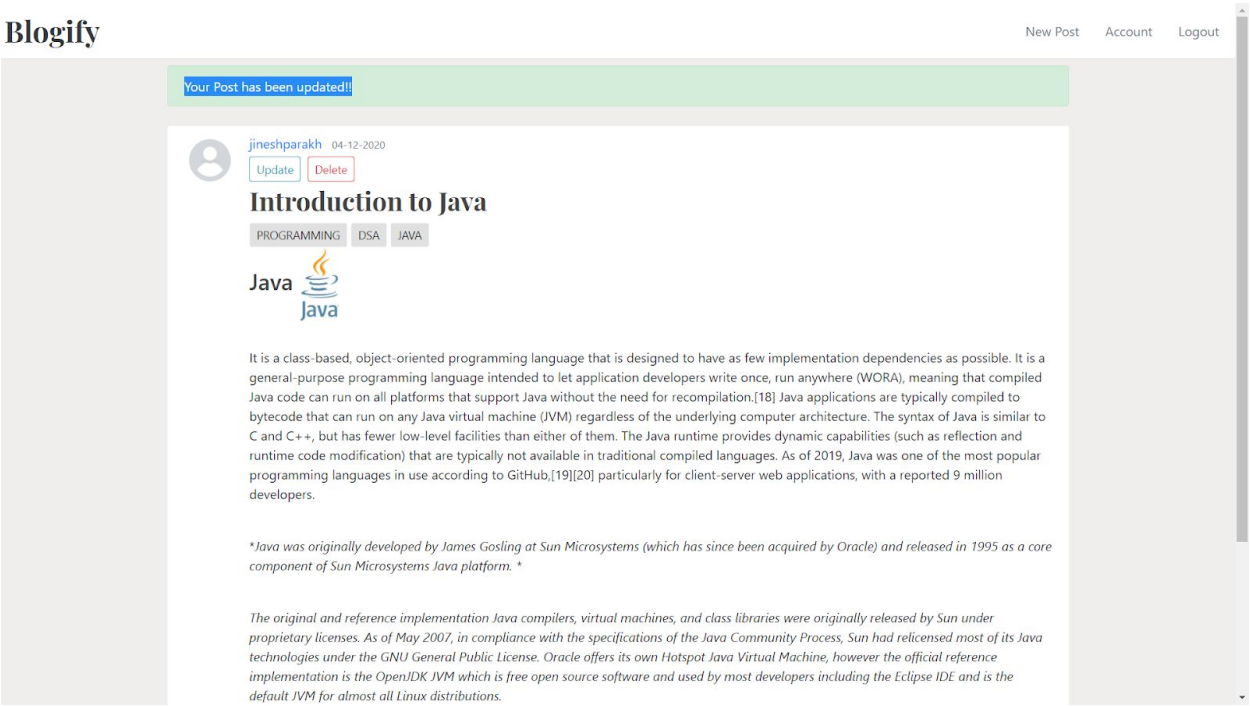

Figure 17: Blog Updated Successfully

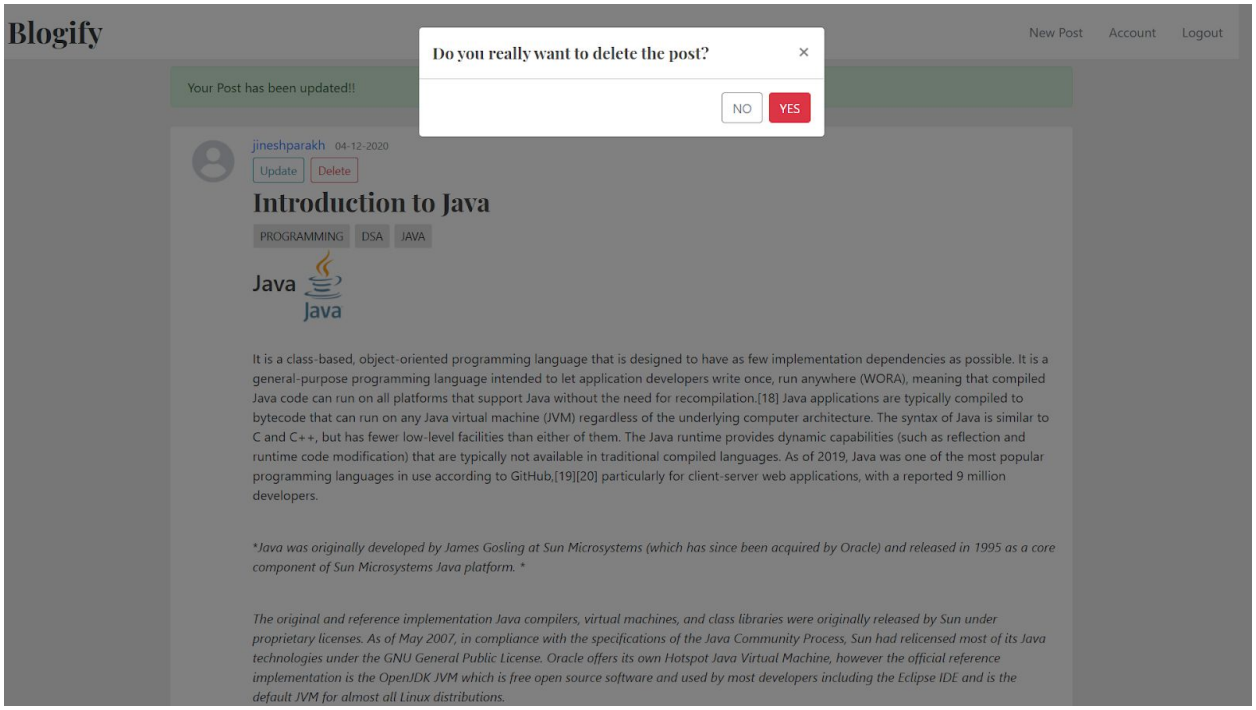

Figure 18: Delete A Post

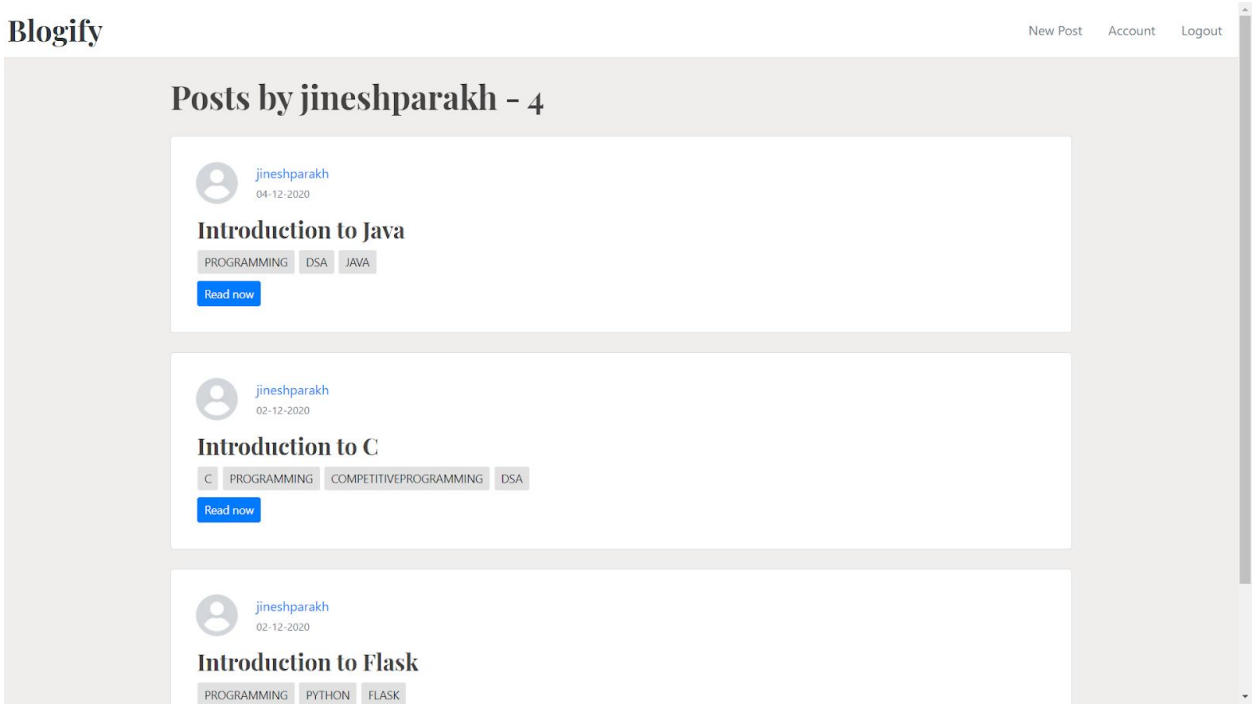

Figure 19: View All the Blogs written by a User(Sorted in descending order of Date of Posting)

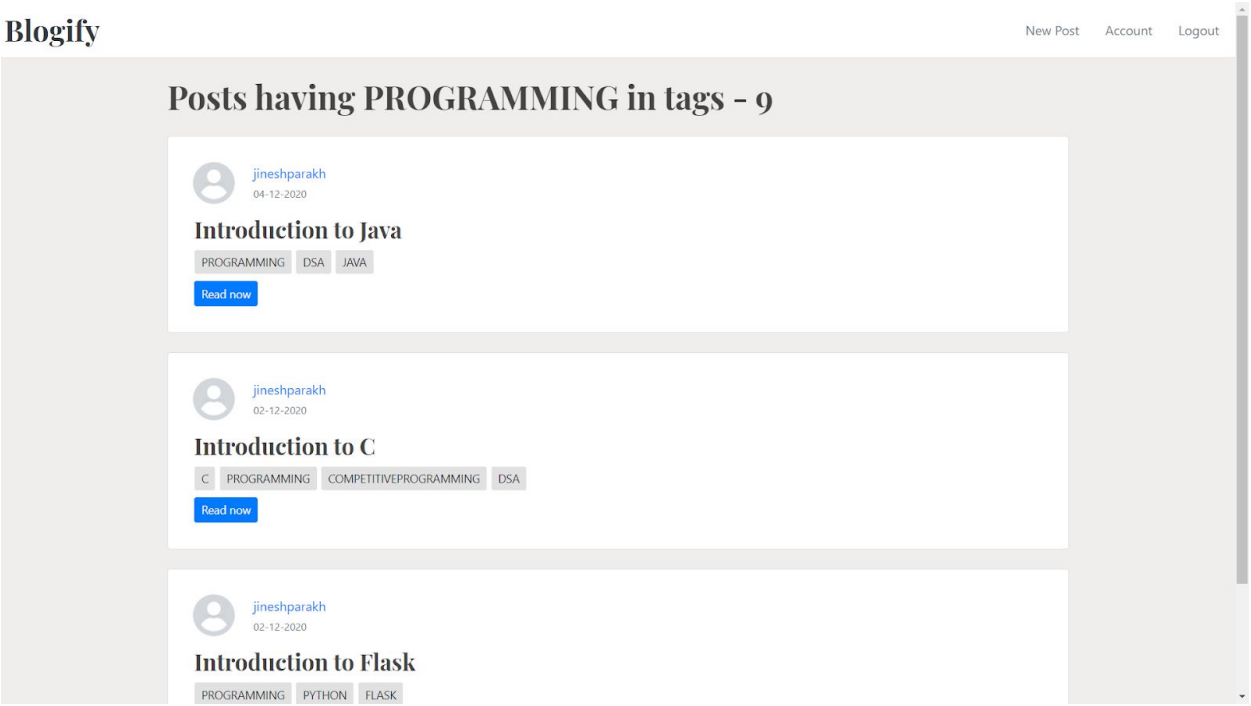

Figure 20: View All the Blogs Having a Specific Tag(Sorted in descending order of Date of Addition)

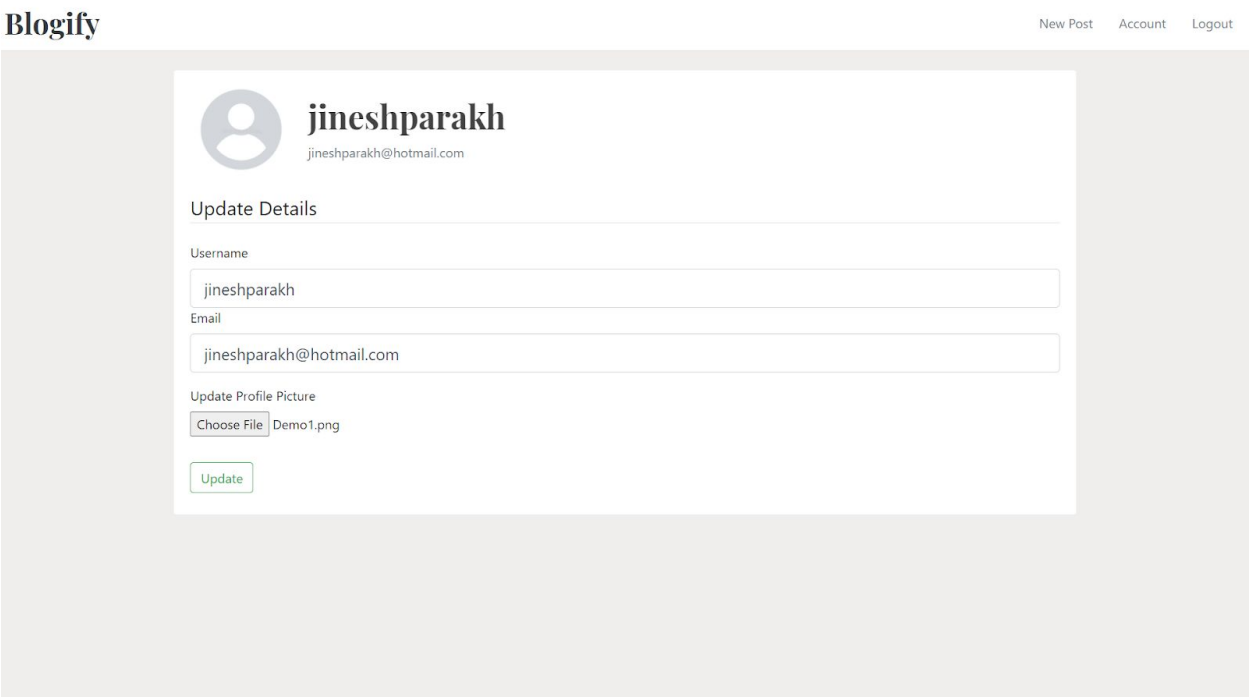

Figure 21: Account Information of A User

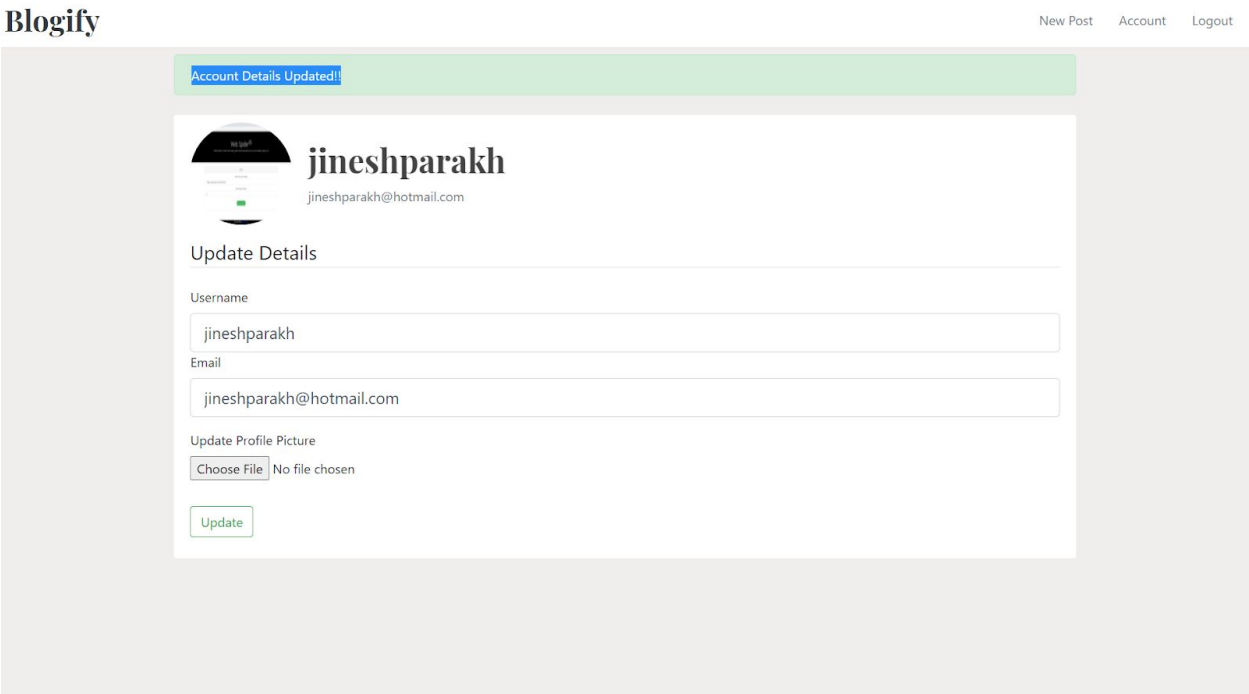

Figure 22: Updated Profile Picture

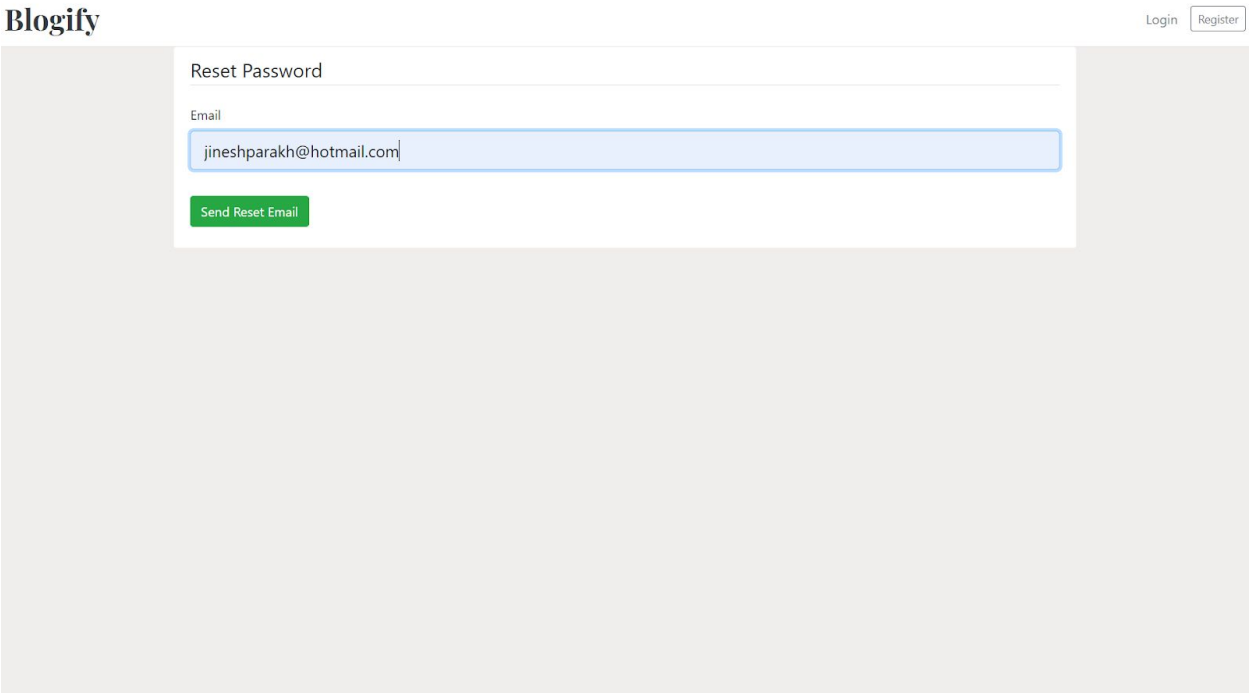

Figure 23: Password Reset Functionality

CHAPTER 9

SOURCE CODE

9.1 Register A User

```
@users.route('/register', methods=['POST','GET'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
    form=RegistrationForm()
    if form.validate_on_submit():

hashed_password=bcrypt.generate_password_hash(form.password.data).decode('
utf-8')
            user= User(username=form.username.data,  email=form.email.data,
password=hashed_password)
        try:
            db.session.add(user)
            db.session.commit()
            flash(f'Account Created for {form.username.data}!, you can now
login :)', category='success')
            return redirect(url_for('users.login'))
        except:
                flash(f'The user is already registered!! Try to login!!',
category='danger')


    return render_template('register.html', title='Register', form=form)
```

9.2 Login A User

```
@users.route('/login', methods=['POST','GET'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
    form=LoginForm()
    if form.validate_on_submit():
```

```
        user=User.query.filter_by(email=form.email.data).first()
                  if  user  and  bcrypt.check_password_hash(user.password,
form.password.data):
            login_user(user, remember=form.remember.data)
            next_page=request.args.get('next')
                  flash(f'Hey {user.username}! Good to see you back ;)',
category='success')
            if next_page:
                return redirect(next_page)
            else:
                return redirect(url_for('main.home'))
        else:
                flash(f'Login Unsuccessful. Please check the email and/or
Password', 'danger')

    return render_template('login.html', title='Login', form=form)
```

9.3 Account Info for A Registered User

```
@users.route('/account', methods=['POST','GET'])
@login_required
def account():
    form=UpdateAccountForm()
    if form.validate_on_submit():
        if form.picture.data:
            picture_file=save_picture(form.picture.data)
            current_user.image_file=picture_file

        current_user.username=form.username.data
        current_user.email=form.email.data
        db.session.commit()
        flash(f'Account Details Updated!!', category='success')
    elif request.method=='GET':
        form.username.data=current_user.username
        form.email.data=current_user.email
    image_file=url_for('static',
filename='profile_pics/'+current_user.image_file)
    return render_template('account.html',title='Account',
image_file=image_file, form=form)
```

9.4 Reset Password and Token Generation

```
@users.route('/reset_password',methods=['GET', 'POST'])
def reset_request():
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
    form=RequestResetForm()
    if form.validate_on_submit():
        user=User.query.filter_by(email=form.email.data).first()
        send_reset_email(user)
        flash(f'Check Your Registered Email for further instructions',
'info')
        return redirect(url_for('users.login'))
    return render_template('reset_request.html',title='Reset Password',
form=form)



@users.route('/reset_password/<token>',methods=['GET', 'POST'])
def reset_token(token):
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
    user=User.verify_reset_token(token)
    if user is None:
        flash(f'That is an invalid or expired token', 'warning')
        return redirect(url_for('users.reset_request'))
    form=ResetPasswordForm()
    if form.validate_on_submit():

hashed_password=bcrypt.generate_password_hash(form.password.data).decode('
utf-8')
        user.password=hashed_password
        db.session.commit()
        flash(f'Password Updated! You can now login :)',
category='success')
        return redirect(url_for('users.login'))
    return render_template('reset_token.html',title='Reset
Password',form=form)
```

9.5 Create A New Blog

```
@posts.route('/post/new', methods=['GET', 'POST'])
@login_required
def new_post():
    form=PostForm()
    tagForm = TagForm()
    tags=Tag.query.all()
    form.tags.choices = [(tag.id, tag.value) for tag in tags]
    if form.validate_on_submit():
        post=Post(title=form.title.data, content=repr(form.content.data),
author=current_user,
tags=Tag.query.filter(Tag.id.in_(form.tags.data)).all())
        db.session.add(post)
        db.session.commit()
        flash(f'Your Post has been created!!', category='success')
        return redirect(url_for('main.home'))

    return render_template('create_post.html', title='New
Post',legend='New Post', form=form, tags=tags, tagForm=tagForm)
```

9.6 Update A Previously Created Blog

```
@posts.route('/post/<int:post_id>/update', methods=['GET', 'POST'])
@login_required
def update_post(post_id):
    post=Post.query.get_or_404(post_id)
    if post.author!=current_user:
        abort(403)
    form=PostForm()
    tagForm = TagForm()
    tags=Tag.query.all()
    form.tags.choices = [(tag.id, tag.value) for tag in tags]
    if form.validate_on_submit():
        post.title=form.title.data
        post.content=repr(form.content.data)
        post.tags = Tag.query.filter(Tag.id.in_(form.tags.data)).all()
```

```
        db.session.commit()
        flash(f'Your Post has been updated!!', category='success')
        return redirect(url_for('posts.post', post_id=post.id))
    elif request.method=='GET':
        form.title.data=post.title
        form.content.data=post.content
        selectedTags=post.tags
        return render_template('create_post.html', title='Update Post',
legend='Update Post', form=form, tags=tags, tagForm=tagForm,
selectedTags=selectedTags)
    return render_template('create_post.html', title='Update Post',
legend='Update Post', form=form, tags=tags, tagForm=tagForm)
```

9.7 Deleting A Blog

```
@posts.route('/post/<int:post_id>/delete', methods=['POST'])
@login_required
def delete_post(post_id):
    post=Post.query.get_or_404(post_id)
    if post.author!=current_user:
        abort(403)
    db.session.delete(post)
    db.session.commit()
    flash(f'Your Post has been deleted!!', category='success')
    return redirect(url_for('main.home'))
```

9.8 Viewing a Single Blog

```
@posts.route('/post/<int:post_id>')
def post(post_id):
    post=Post.query.get_or_404(post_id)
    return render_template('post.html', title=post.title, post=post)
```

9.9 Viewing All Blogs by A User

```
@users.route('/user/<string:username>')
def user_posts(username):
    page=request.args.get('page', default=1, type=int)
    user=User.query.filter_by(username=username).first_or_404()
```

```
posts=Post.query.filter_by(author=user).order_by(Post.created_at.desc()).p
aginate(page=page, per_page=3)
    return render_template('user_posts.html', posts=posts,user=user)
```

9.10 Creating A New Tag For a Blog

```
@tags.route('/tags/new', methods=['POST'])
@login_required
def new_tag():
    tagForm=TagForm()
    if tagForm.validate_on_submit():
        tag=Tag(value=tagForm.value.data.upper())
        try:
            db.session.add(tag)
            db.session.commit()
            flash(f'Tag has been created!!', category='success')
            return redirect(url_for('posts.new_post'))
        except:
            flash(f'Tag Already Exists', category='warning')
            return redirect(url_for('posts.new_post'))
    flash(f'Could not create tag. Something went wrong.',
category='error')
    return redirect(url_for('posts.new_post'))
```

9.11 View All Blogs Having A Specific Tag

```
@tags.route('/tags/<string:selectedTag>')
@login_required
def searchPostsViaTag(selectedTag):
    tagId=Tag.query.filter_by(value=selectedTag).first_or_404().id
    print(tagId)
    allPostsWithGivenTag=PostTag.query.filter_by(tag_id=tagId).all()
    postIds=[post.post_id for post in allPostsWithGivenTag]
    print(postIds)
    page=request.args.get('page', default=1, type=int)

posts=Post.query.filter(Post.id.in_(postIds)).order_by(Post.created_at.des
c()).paginate(page=page, per_page=3)
    print(posts)
```

```
    print(allPostsWithGivenTag)
    return render_template('tagged_posts.html', posts=posts,
tag=selectedTag)
```

9.12 Logout Feature

```
@users.route('/logout')
def logout():
    logout_user()
    return redirect(url_for('main.home'))
```

CHAPTER 10

TESTING

Test Case ID	Test Scenario	Test Case	Pre-Condition	Test Steps	Test Data	Expected Result	Post Condition	Actual Result	Bugs Encountered	Status (PASS/FAIL)
TC01	Login	Login A User using Email Id and Password	User should be already registered	Enter details in Login Page As Required	Email and Password	User Logged In And Redirected To Home Page	-	User Logged In And Redirected To Home Page	-	PASS
TC02	Create A New Blog	Creation of A New Blog By a Registered User	User should Have An Account	Enter the Required Details in the Blog Editor	Blog Title, Tags, Content	A new Blog is created and displayed at the top of the Home Page	Blog Stored in Database	A new Blog is created and displayed at the top of the Home Page	-	PASS
TC03	Update An Old Blog	Updation of a Previously created Blog	User should Have An Account with an Old Blog	Enter data to Update Blog	Updated Content	The Old Blog Post is Updated	Update Blog stored in the Database	The Old Blog Post is Updated	-	PASS

TC04	Delete An Old Blog	Deletion of a Previously created Blog	User should Have An Account with an Old Blog	Click on Delete Blog For that Blog and confirm	-	The Blog is deleted.	Blog entry is removed from the database	The Blog is deleted.	-	PASS
TC05	Create A New Tag	Creation of A new tag for a Blog	User Should be Logged In	Click on Create Tag Button in the New Blog Section	A New Tag	A New tag is created.	New Tag is stored in the Database	A New tag is created.	-	PASS

Table 3. Testing

CHAPTER 11

CONCLUSION

The blogging website has been successfully implemented. After careful consideration and thought we fixed the technologies and database to be used so that they suit the requirements appropriately. With the help of these selected technologies and resources, we successfully created a basic prototype for a Blogging Website. This website successfully incorporates numerous features and functionalities that could provide the users a hassle free and smooth experience of blogging.