UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

"Blogging Website"

Submitted by:

Kaushik J	**PES1UG21CS272**
KNC Vardhan	**PES1UG21CS256**
Karthik Kadewadi	**PES1UG21CS271**
K Sai Vignesh	**PES1UG21CS260**

*6th Semester **E** Section*

Prof. Bhargavi Mokashi
Assistant Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem Statement

Develop a feature-rich blogging platform that enables users to create accounts, publish posts, and interact with content. Users should be able to authenticate securely, manage their profiles, and categorize posts. Additionally, the platform should support functionalities for liking posts and managing likes.

Use Case Model

1) User Authentication:

Actors: User, System

Description: Allows users to register, login, and logout from the system.

Use Cases:

Register: Users can create a new account by providing necessary details. Login: Users can log in to their accounts using valid credentials.
Logout: Users can securely log out from their accounts.

2) Post Management:

Actors: User, System

Description: Enables users to create, edit, delete, and view posts.

Use Cases:

Create Post: Users can compose and publish new blog posts.
View Post: Users can read published blog posts.
Like Post: Users can express their appreciation for posts by liking them.

Class Model

1) User:

Attributes:
id: int
name: String
email: String
password: String
gender: String
dateTime: Timestamp
about: String
profile: String

Methods:
getters and setters for attributes

2) Post:

Attributes:
pid: int
pTitle: String
pContent: String
pCode: String
pPic: String
pDate: Timestamp
catId: int
userId: int

Methods:
getters and setters for attributes

3) Category:

Attributes:
cid: int
name: String
description: String

Methods:
getters for attributes

4) LikeDao:

Methods:
insertLike(int pid, int uid): boolean
countLikeOnPost(int pid): int
isLikedByUser(int pid, int uid): boolean
deleteLike(int pid, int uid): boolean

5) UserDao:

Methods:
saveUser(User user): boolean getUserByEmailAndPassword(String email, String password): User updateUser(User user): boolean

6) PostDao:

Methods:
savePost(Post post): boolean
getAllPosts(): List<Post>
getPostById(int id): Post
getPostsByCategoryId(int catId): List<Post>
deletePost(int id): boolean

7) Message:

Attributes:
content: String
type: String
cssClass: String

Methods:
getters and setters for attributes

8) PostFactory:

Methods:
createPost(int pid, String pTitle, String pContent, String pCode, String pPic, Timestamp pDate, int catId, int userId): Post

9) ConnectionProvider:

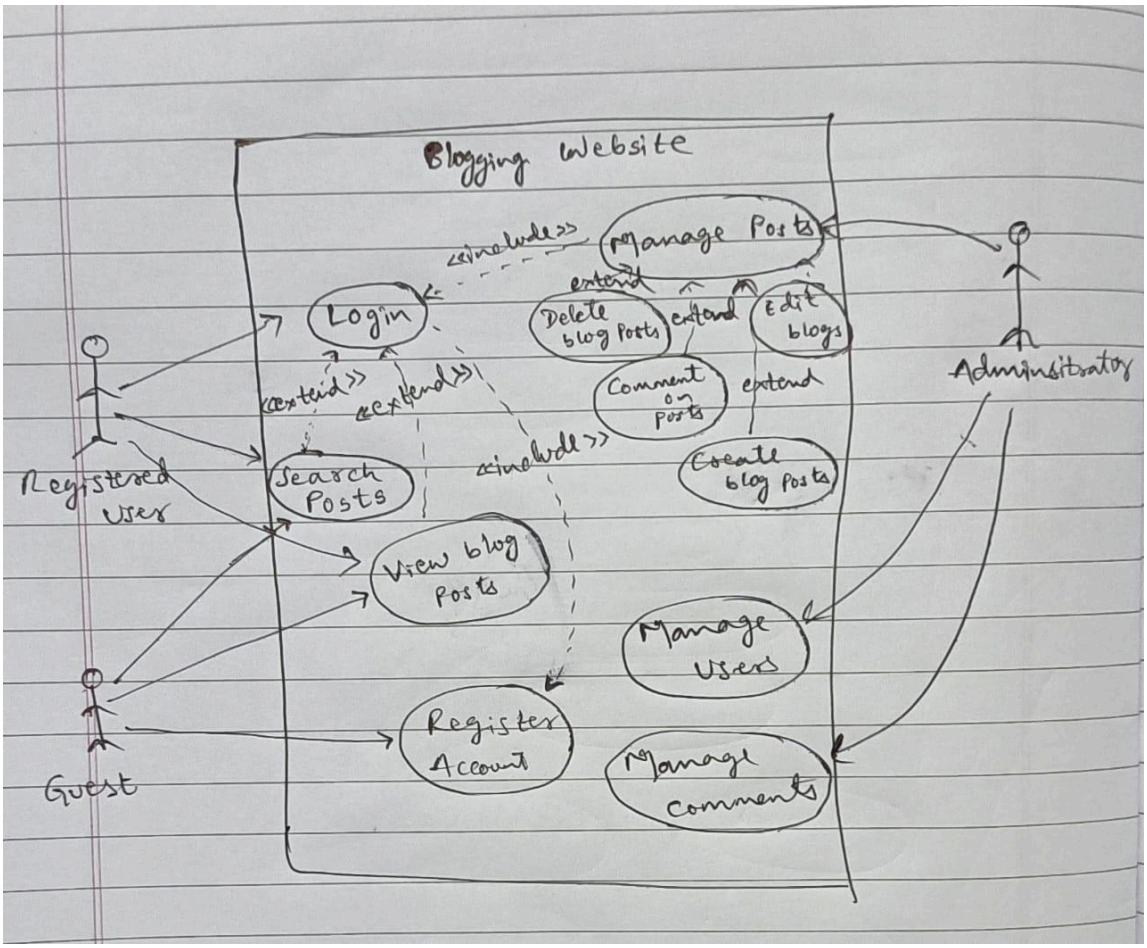
Methods:
getConnection(): Connection

10) Helper:

Methods:

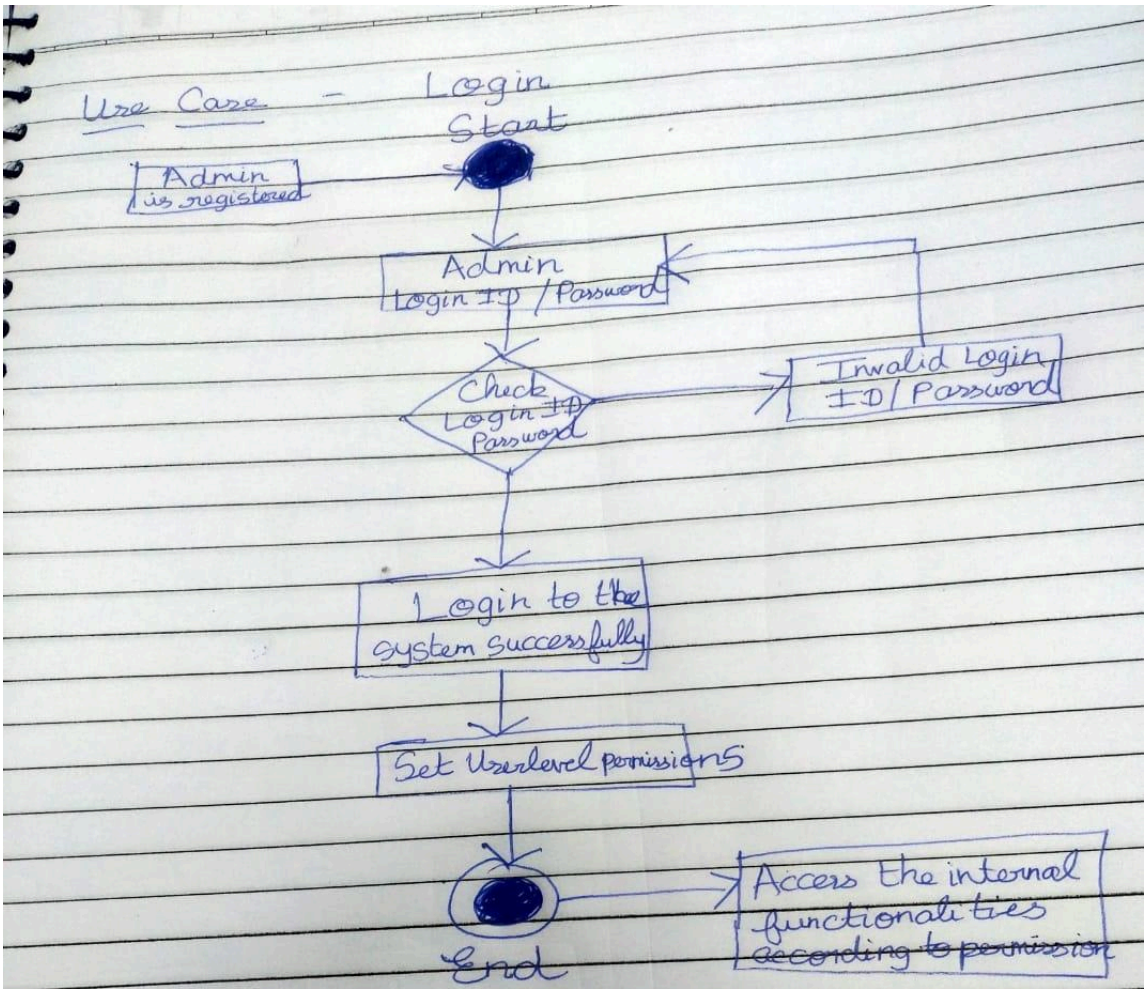
saveFile(InputStream inputStream, String path): boolean deleteFile(String path): boolean

Use Case Diagram

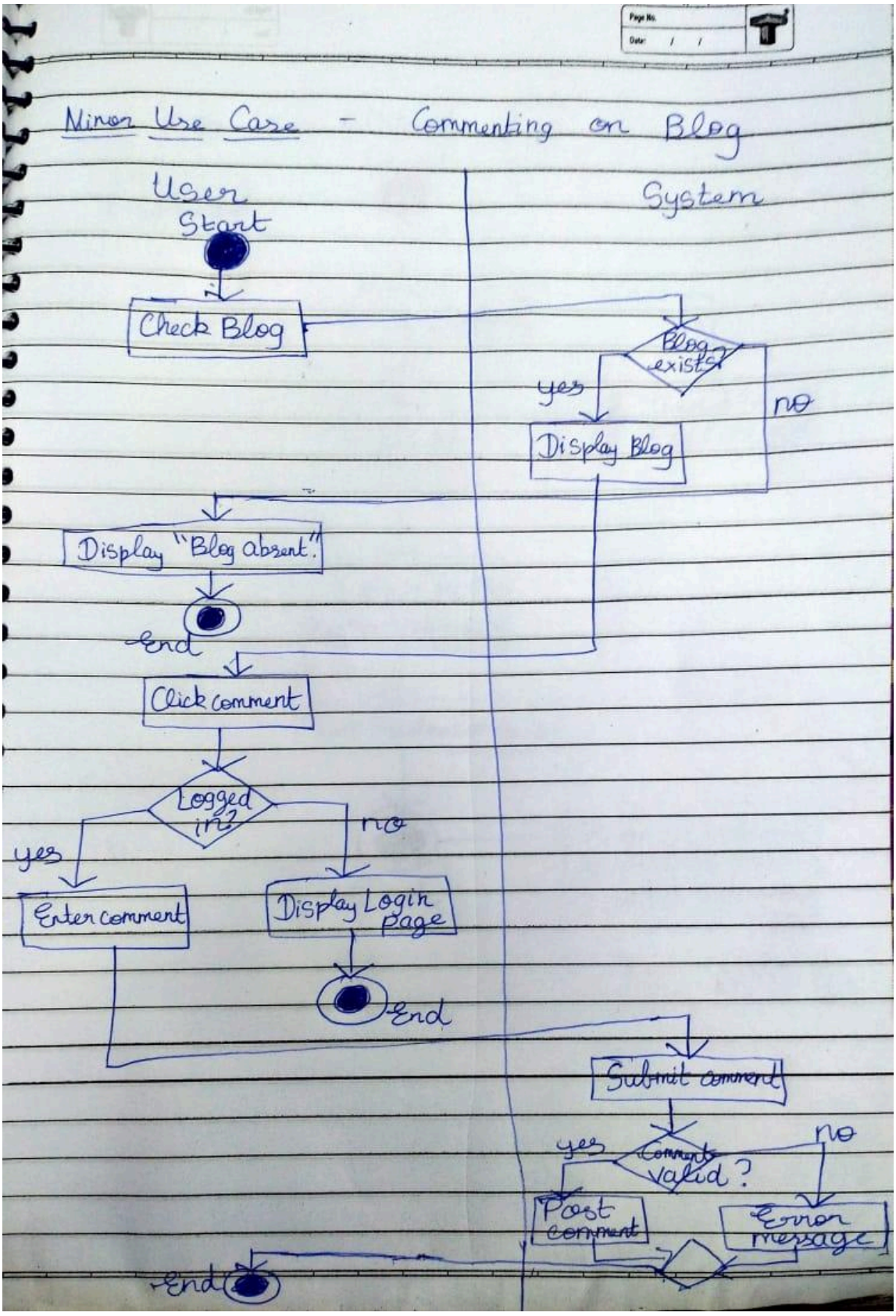


Activity Diagram

Use Case - Login

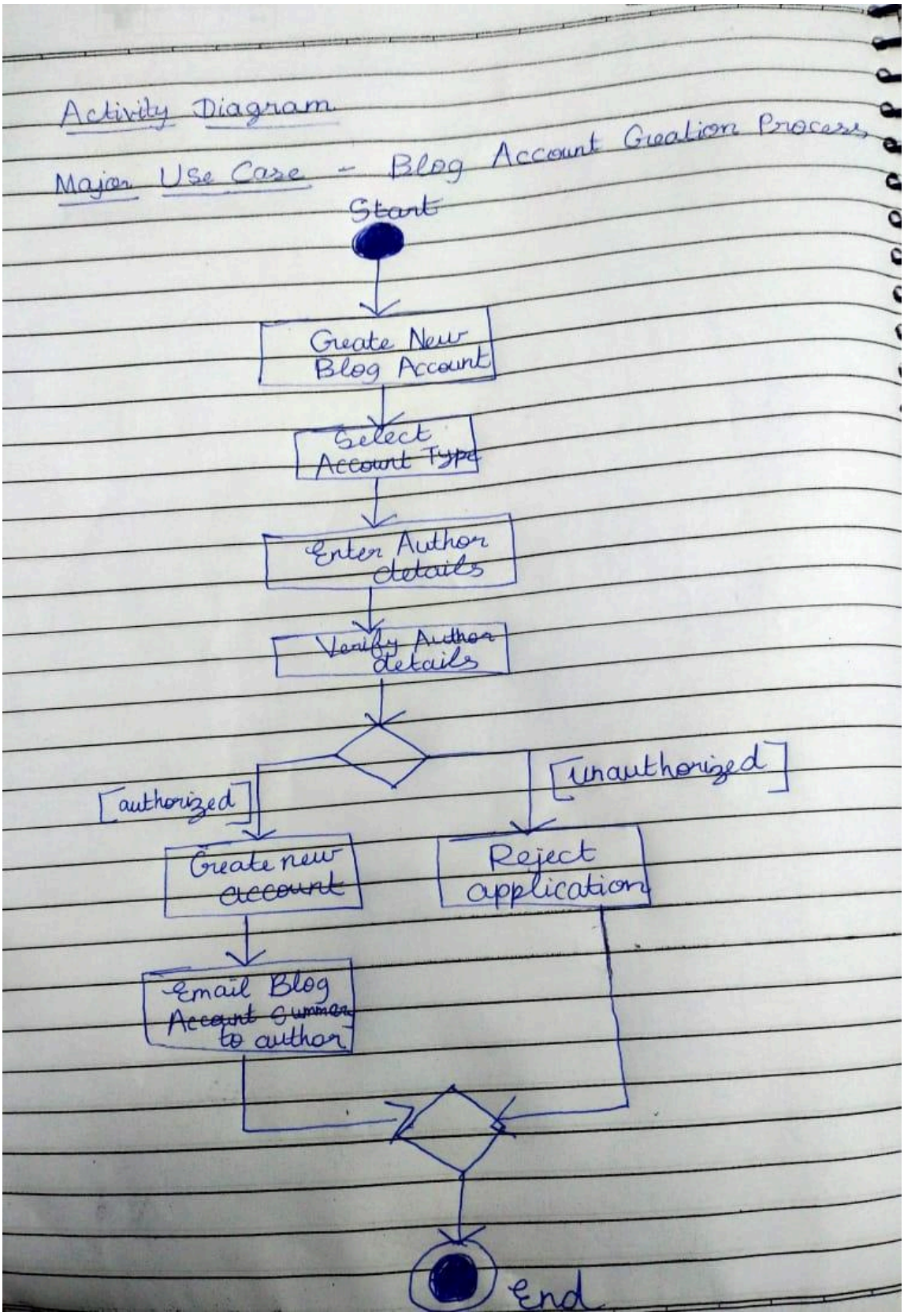


Minor Use Case - Commenting on Blog

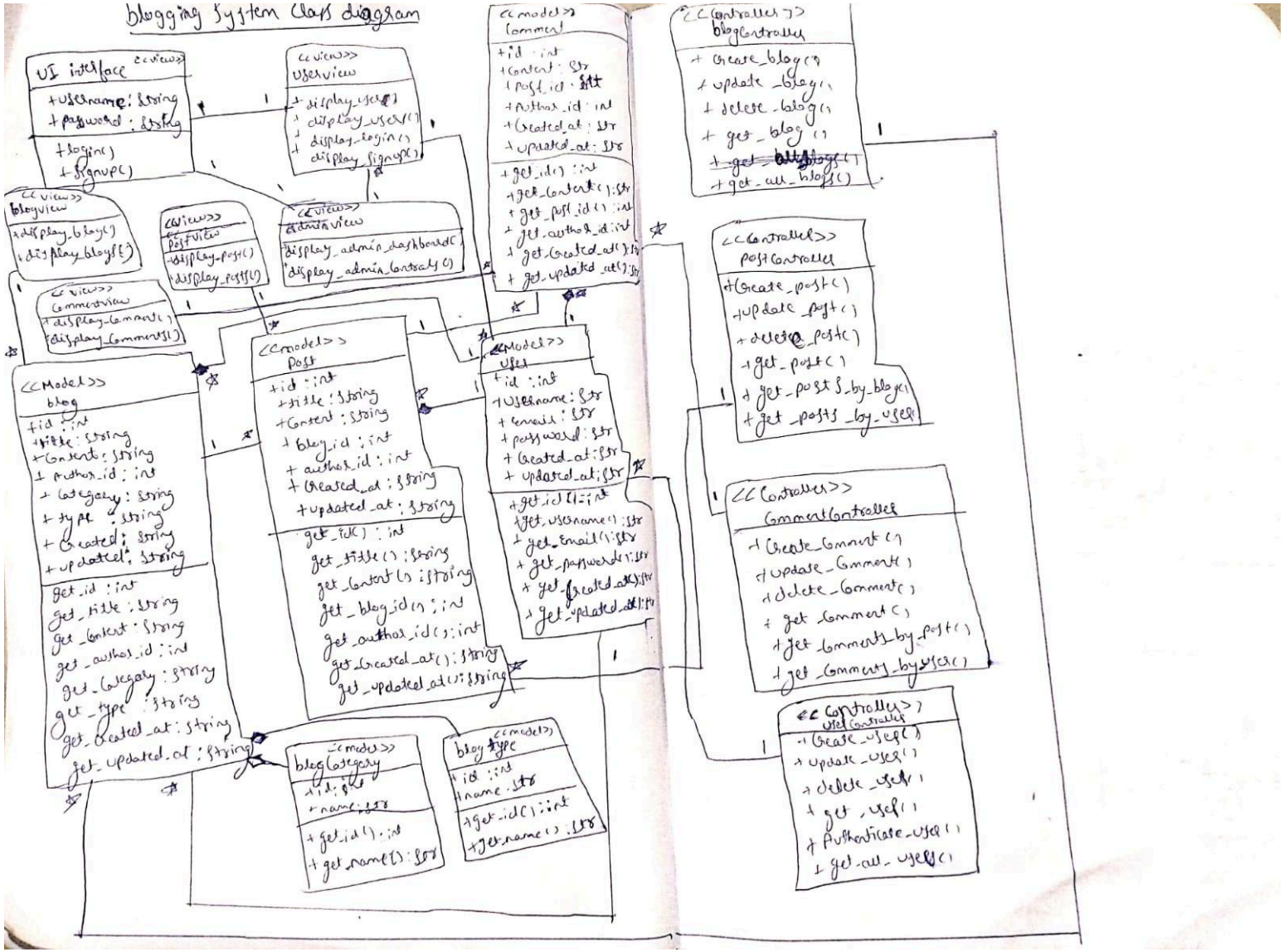


Activity Diagram

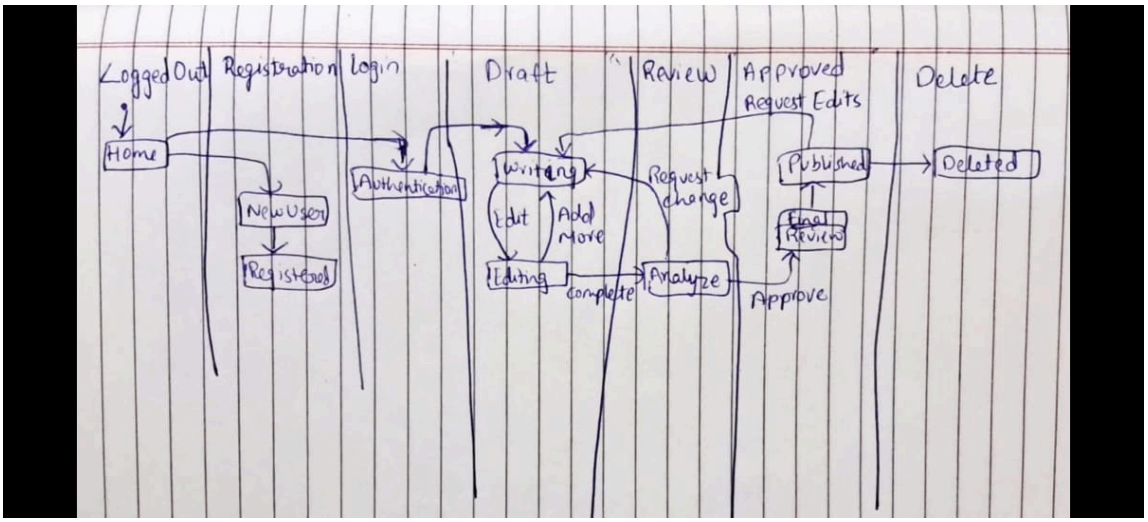
Major Use Case - Blog Account Creation Process



Class Diagram


blogging system class diagram
UI interface
<<view>>
+username: String
+password: String
+login()
+signup()
<<view>>
UserView
+display_user()
+display_user()
+display_login()
display_signup()
<<view>>
blogview
+display_blog()
+display_blogs()
<<view>>
PostView
+display_post()
+display_posts()
<<view>>
AdminView
display_admin_dashboard()
display_admin_controls()
<<view>>
CommentView
+display_comment()
+display_comments()
<<Model>>
blog
+id: int
+title: String
+Content: String
+author_id: int
+category: String
+type: String
+Created: String
+updated: String
get_id: int
get_title: String
get_content: String
get_author_id: int
get_category: String
get_type: String
get_created_at: String
get_updated_at: String
<<model>>
Post
+id: int
+title: String
+Content: String
+blog_id: int
+author_id: int
+Created_at: String
+updated_at: String
get_id(): int
get_title(): String
get_content(): String
get_blog_id(): int
get_author_id(): int
get_created_at(): String
get_updated_at(): String
<<model>>
blog category
+id: int
+name: str
+get_id(): int
+get_name(): str
<<model>>
blog type
+id: int
+name: str
+get_id(): int
+get_name(): str
<<model>>
Comment
+id: int
+Content: str
+post_id: int
+Author_id: int
+Created_at: str
+updated_at: str
+get_id(): int
+get_content(): str
+get_post_id(): int
+get_author_id: int
+get_created_at(): str
+get_updated_at(): str
<<Model>>
User
+id: int
+username: str
+email: str
+password: str
+Created_at: str
+updated_at: str
+get_id(): int
+get_username(): str
+get_email(): str
+get_password: str
+get_created_at(): str
+get_updated_at(): str
<<Controller>>
blogController
+ create_blog()
+ update_blog()
+ delete_blog()
+ get_blog()
+ get_all_blogs()
<<Controller>>
postController
+Create_post()
+Update_post()
+delete_post()
+get_post()
+get_post_$_by_blog()
+get_posts_by_user()
<<Controller>>
CommentController
+Create_Comment()
+Update_Comment()
+delete_Comment()
+get_Comment()
+get_Comments_by_post()
+get_Comments_by_user()
<<Controller>>
UserController
+Create_user()
+update_user()
+delete_user()
+get_user()
+Authenticate_user()
+get_all_users()

State Diagram



Architecture Patterns, Design Patterns and Design Principles

MVC Architecture:

The project follows the Model-View-Controller (MVC) architecture pattern, separating the application into three interconnected components: Model (data), View (user interface), and Controller (logic).

The servlets act as controllers, handling user requests and interactions. Entities like User and Post serve as models, representing data entities. JSP files serve as views, presenting data to users.

Factory Pattern:- Implemented by KNC Vardhan

The PostFactory class implements the factory pattern, providing a centralized way to create Post objects without exposing the instantiation logic.

PostFactory encapsulates the creation logic, allowing flexibility in creating Post objects with different parameters. This promotes code maintainability and scalability.

Builder Pattern:- Implemented by Kaushik J

The Category class uses the builder pattern to construct Category objects with optional parameters.

By providing a fluent interface for constructing Category objects, the builder pattern simplifies object creation and improves readability. It allows creating Category objects with only required parameters while providing flexibility for optional parameters.

Composite Pattern:- Implemented by K Sai Vignesh

The UserDao class demonstrates elements of the composite pattern by encapsulating multiple UserDaoImpl instances.

UserDao acts as a composite, providing a unified interface for interacting with different UserDaoImpl implementations. This promotes code reuse and allows treating individual implementations and compositions uniformly.

Command Pattern:- Implemented by Karthik Kadewadi

The LogoutServlet class implements the command pattern by encapsulating the logout functionality within a Command interface and its concrete implementation, LogoutCommand.

By separating the logout behavior into a command object, the servlet gains flexibility and extensibility. New logout behaviors can be added by implementing additional command classes without modifying the servlet code.

SOLID Principles:-

Single Responsibility Principle (SRP): Each class has a single responsibility, such as handling user data (User class), managing posts (PostDao class), or providing database connections (ConnectionProvider class).

Open/Closed Principle (OCP): Classes are open for extension but closed for modification. For example, new functionality can be added to the application without altering existing code, promoting scalability.

Liskov Substitution Principle (LSP): Subclasses such as UserDaoImpl adhere to the contracts defined by their parent interfaces, ensuring that they can be substituted for their base types without affecting the program's correctness.

Interface Segregation Principle (ISP): Interfaces such as UserDaoInterface are segregated to provide specific sets of functionalities, ensuring that clients are not forced to depend on interfaces they do not use.

Dependency Inversion Principle (DIP): High-level modules such as servlets depend on abstractions (interfaces) rather than concrete implementations (e.g., UserDao), promoting loose coupling and facilitating easier testing and maintenance.

Github Link to the Codebase

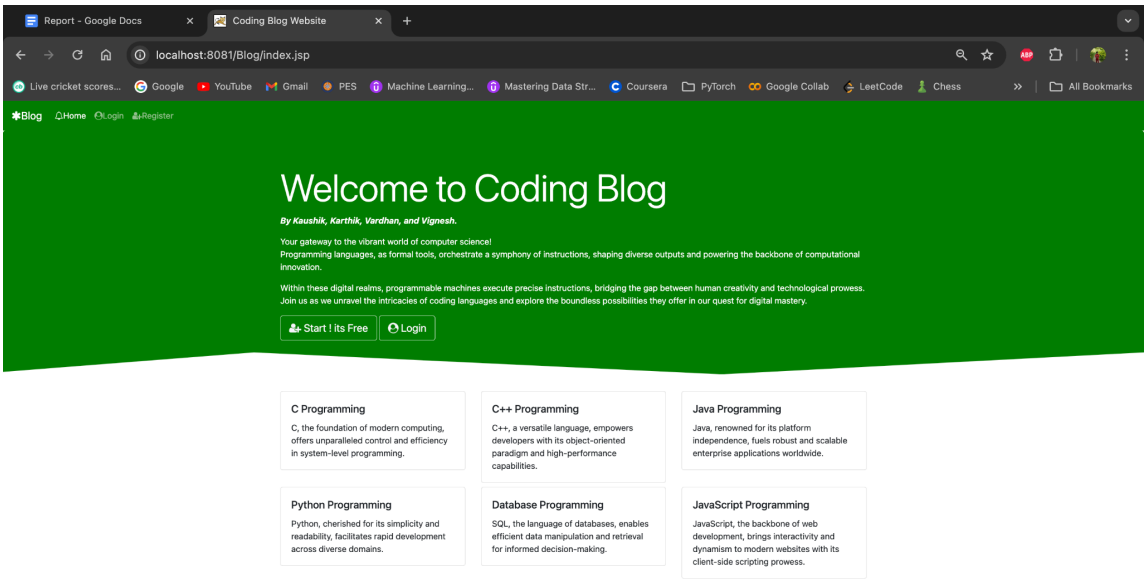
https://github.com/Kaushik-0007/Blogging-Website

Individual contribution of the team members

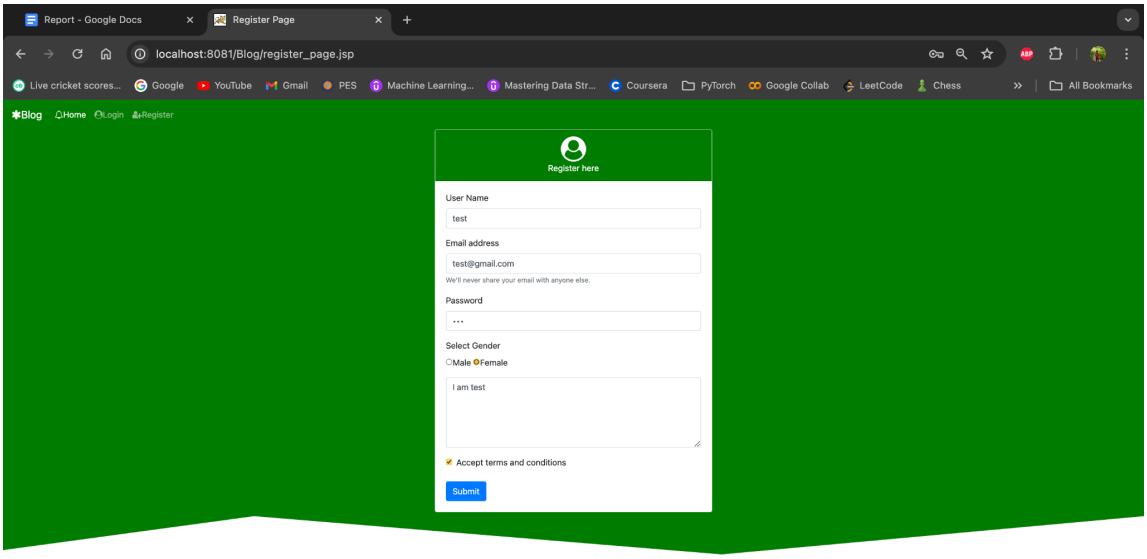
Karthik Kadewadi – Designed the frontend login & register page.
K Sai Vignesh - Designed the frontend profile & posts page.
KNC Vardhan - Managed the backend & database interactions of login and register functionality
Kaushik J - Managed the backend & database interactions of profile and post functionality.

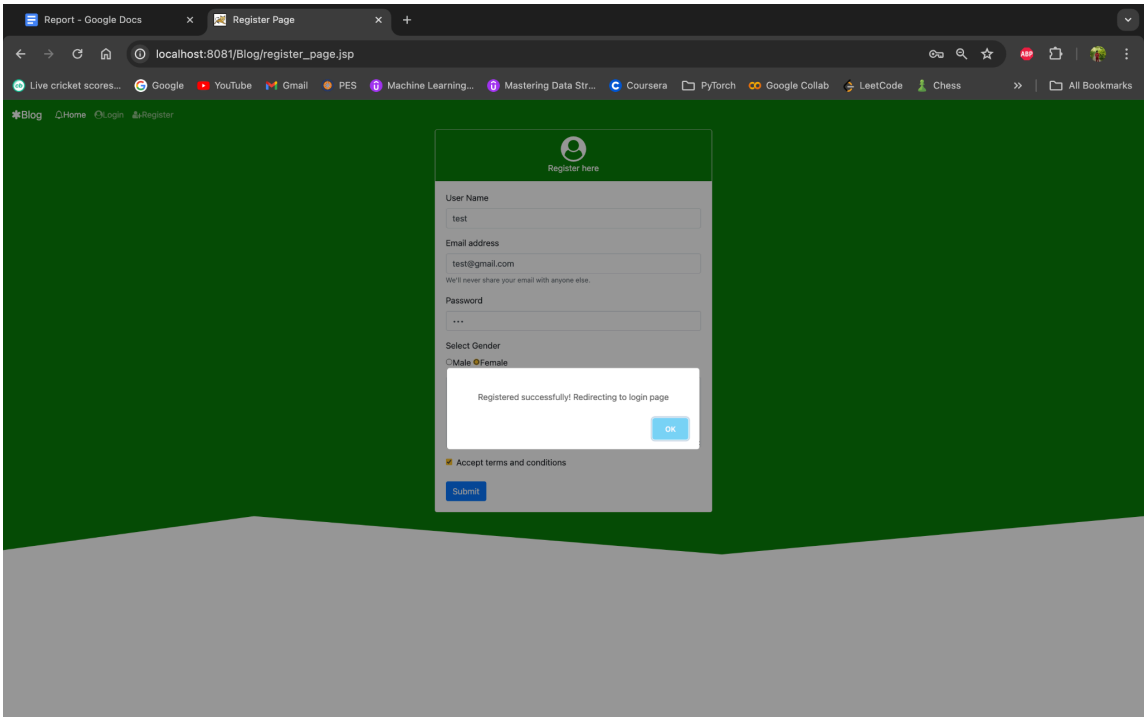
Screenshots with input values populated and output shown

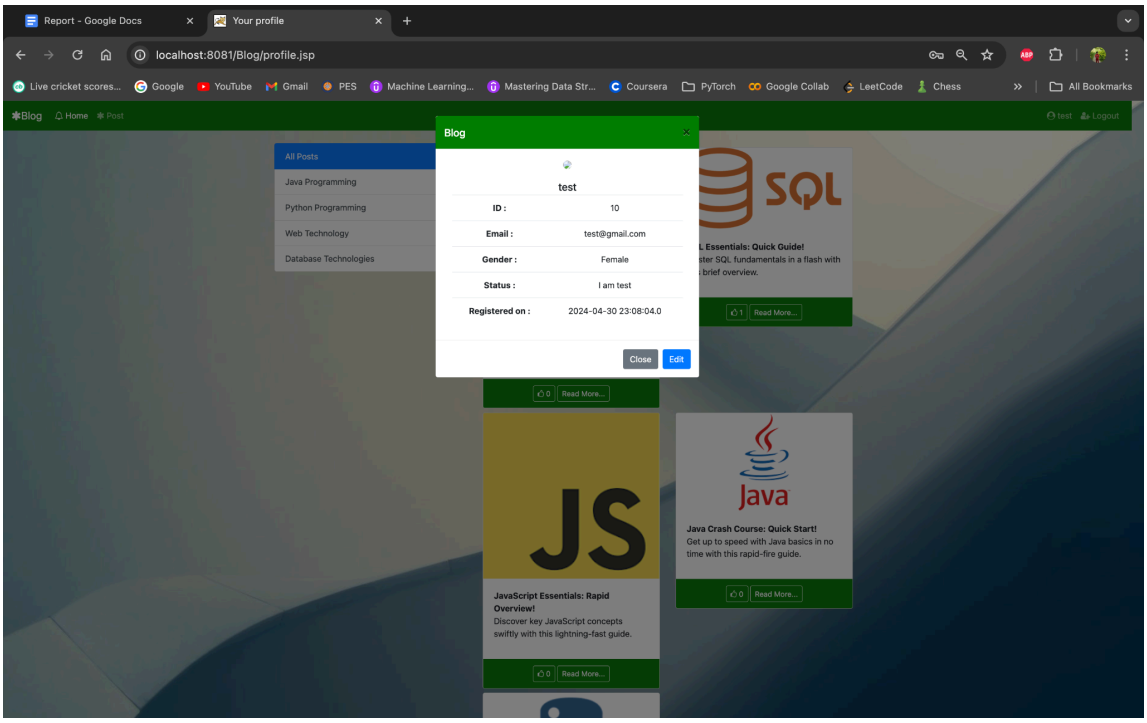
Cover page



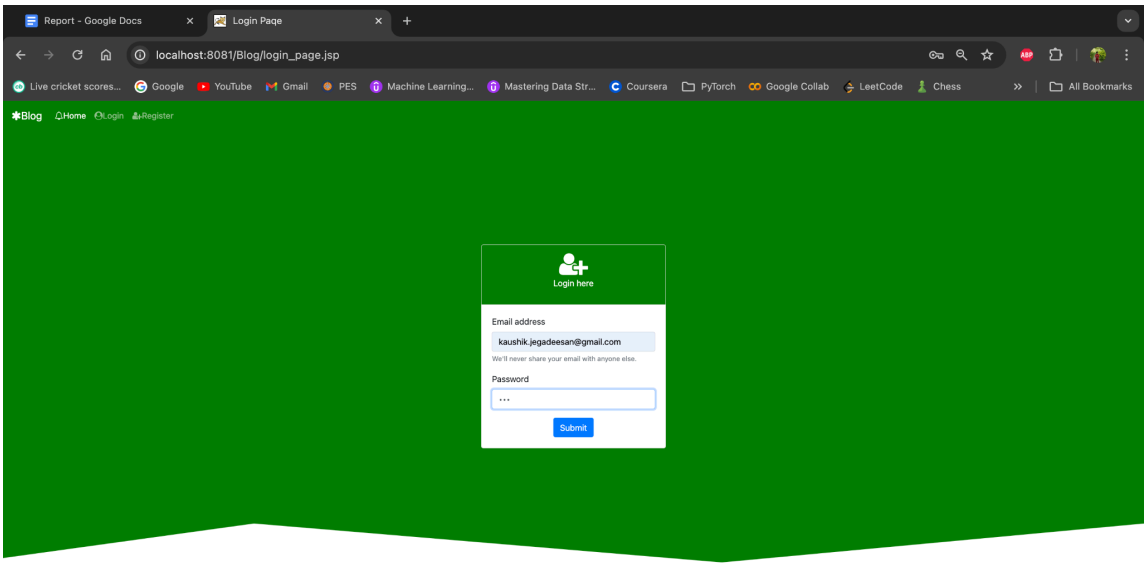
Register an account



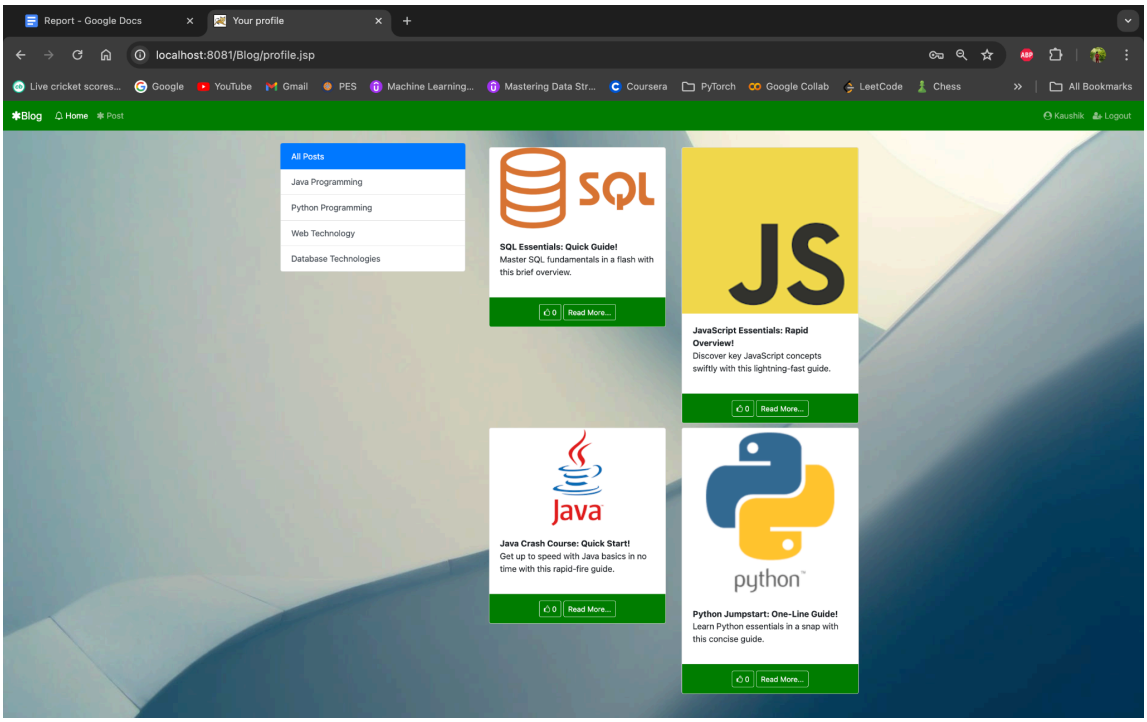




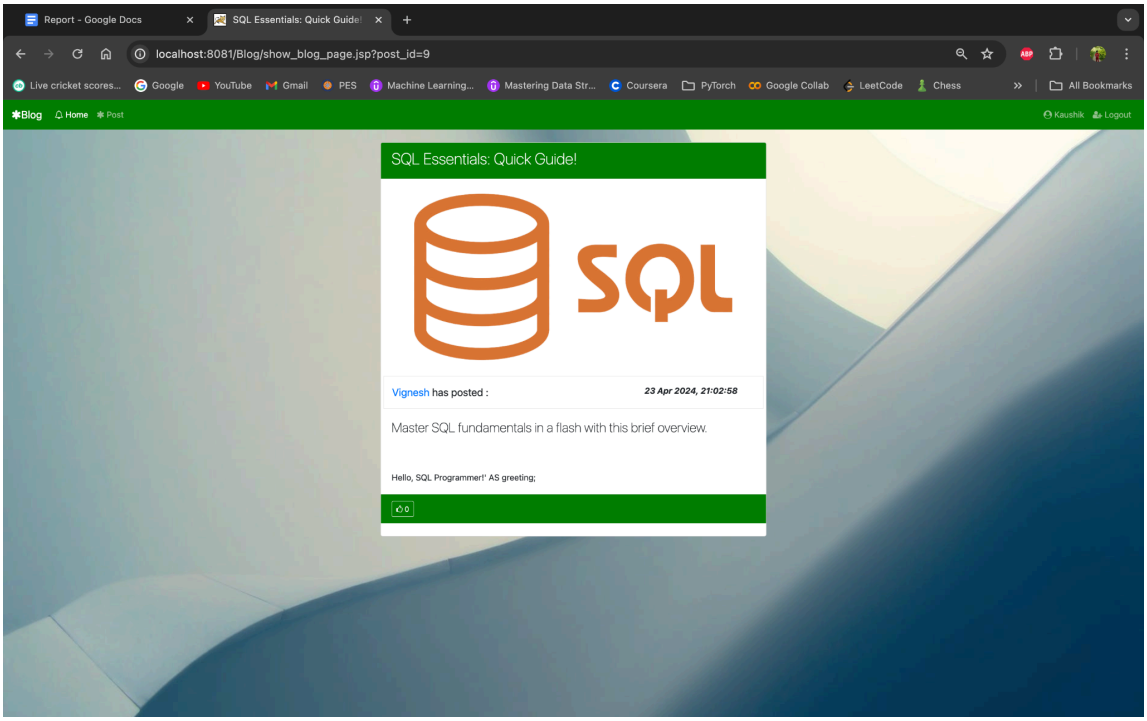
Login into an account



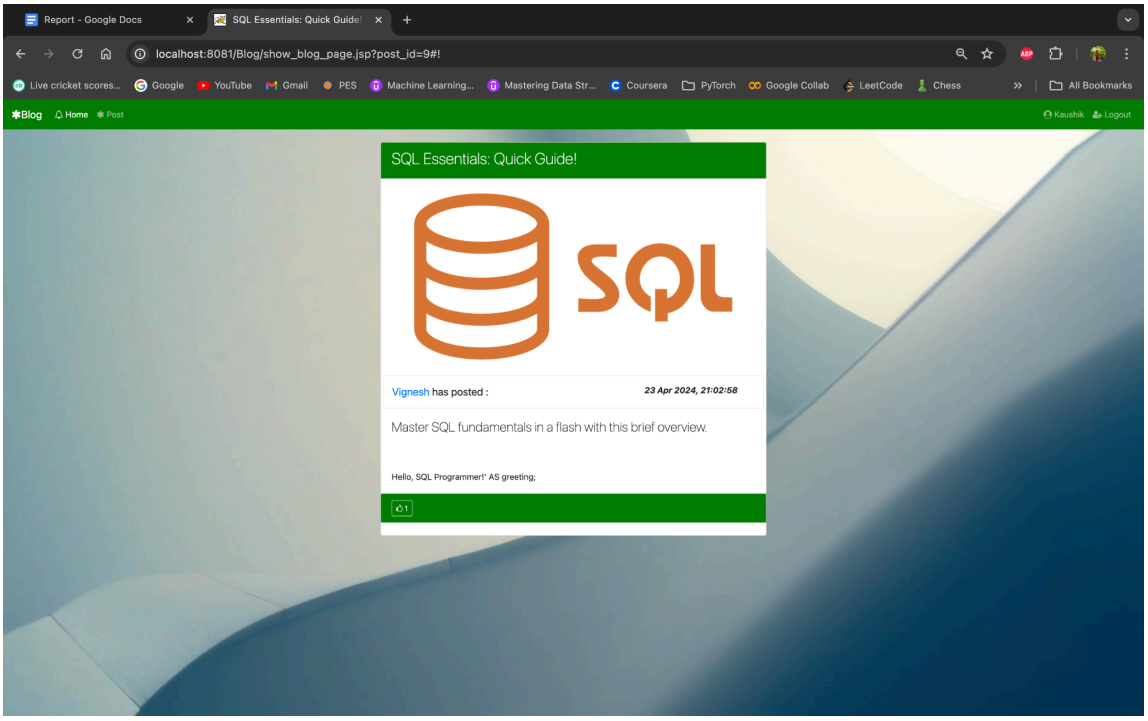
Profile page after logging in



Viewing a post

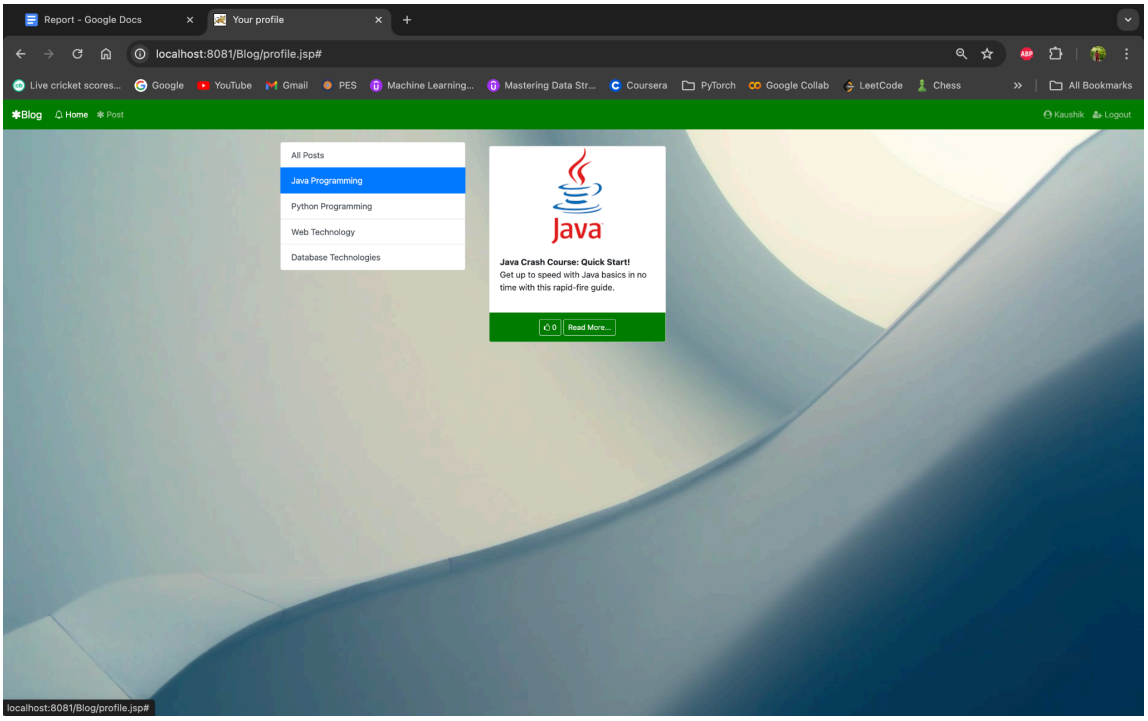


Liking a Post

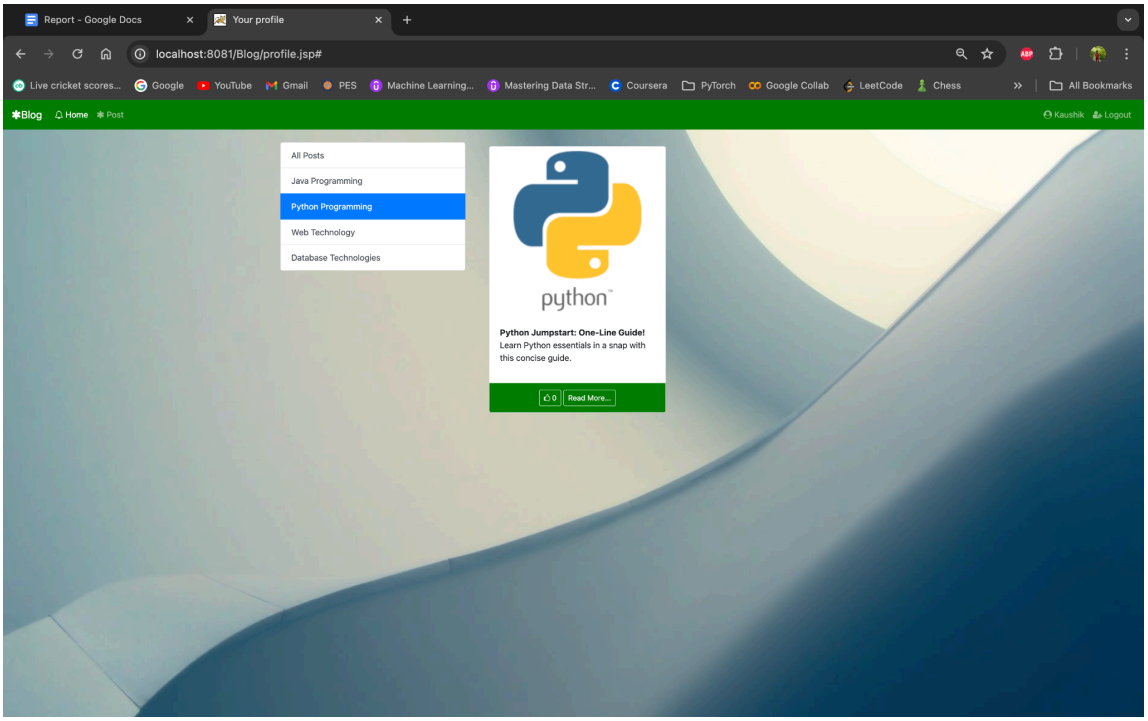


Posts Filtered by Categories

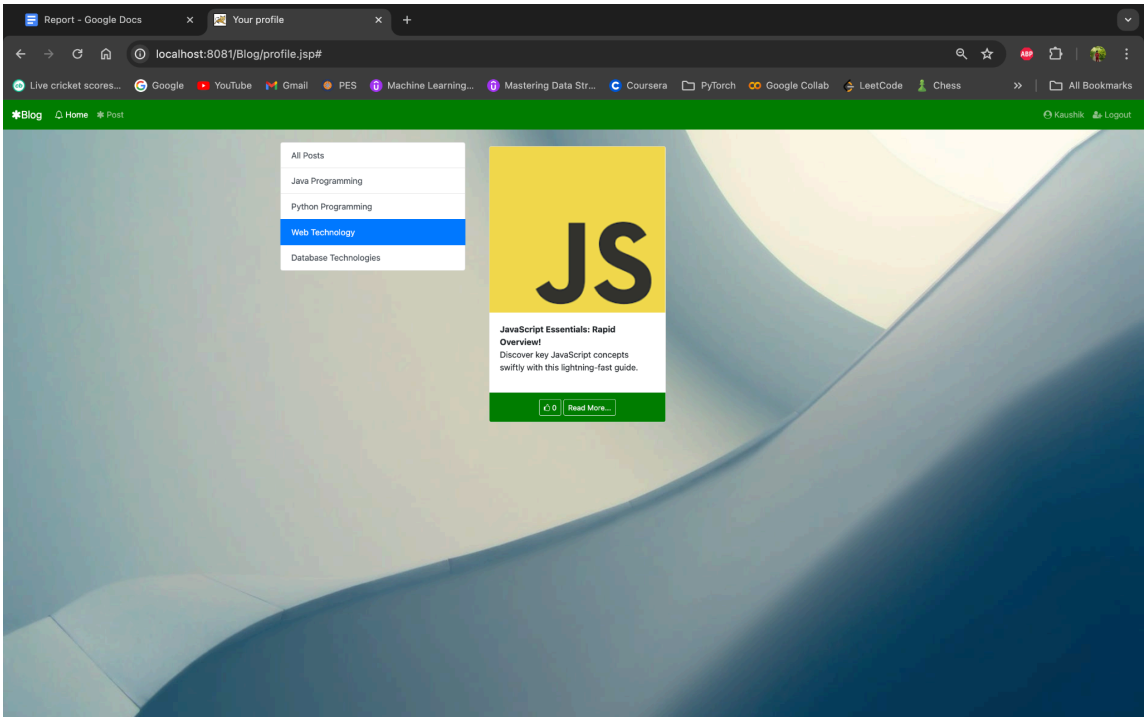
Java Programming



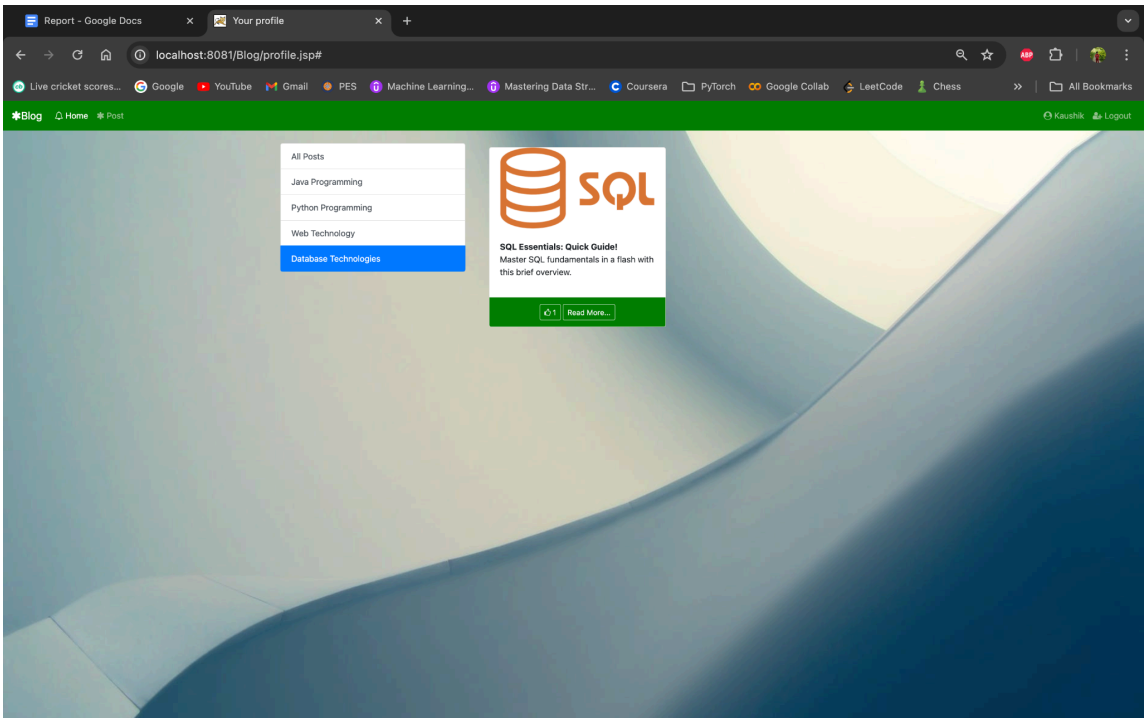
Python Programming



Web Technology

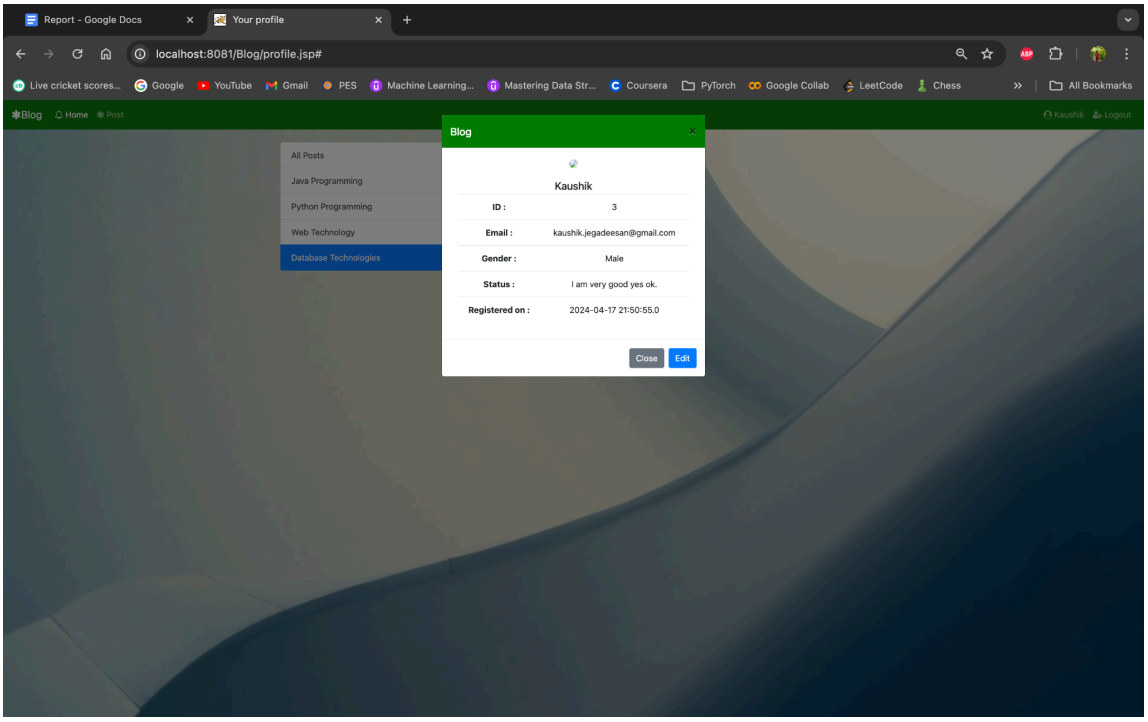


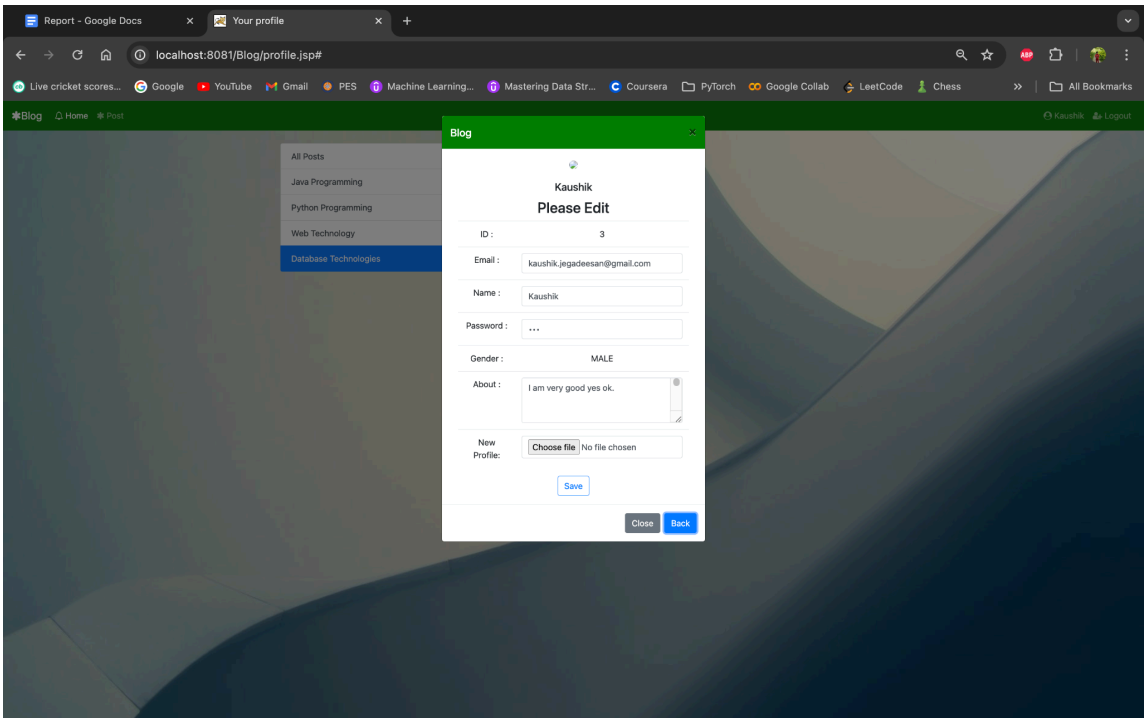
Database Technologies



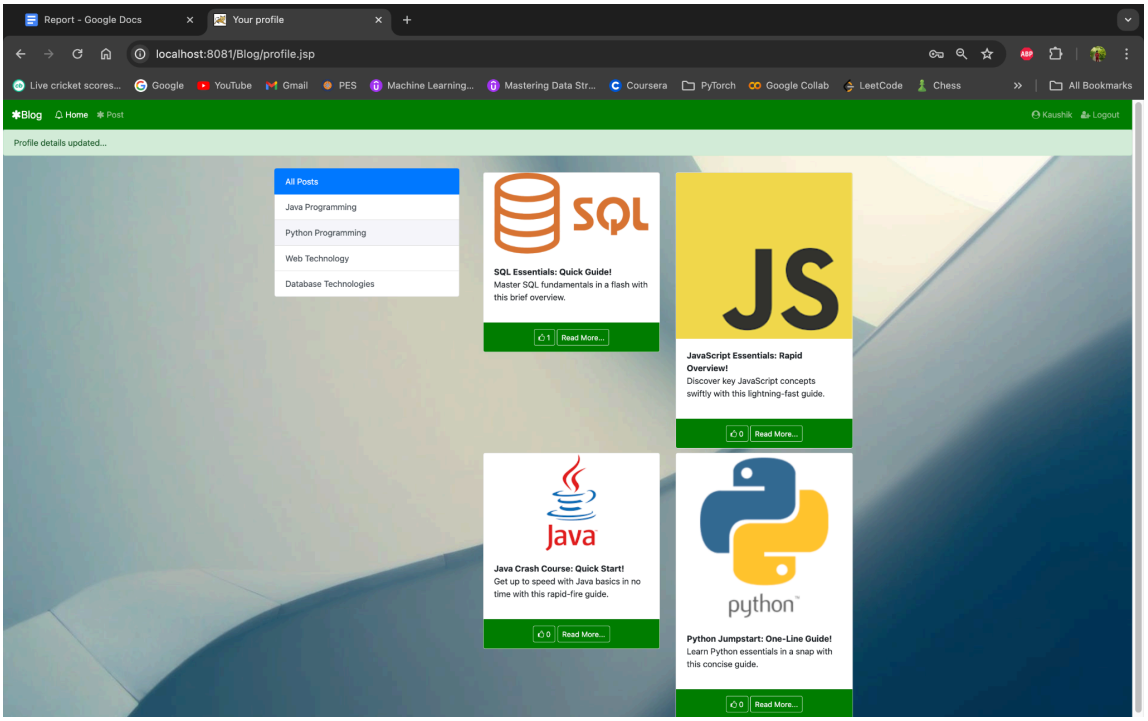
Edit Profile Details

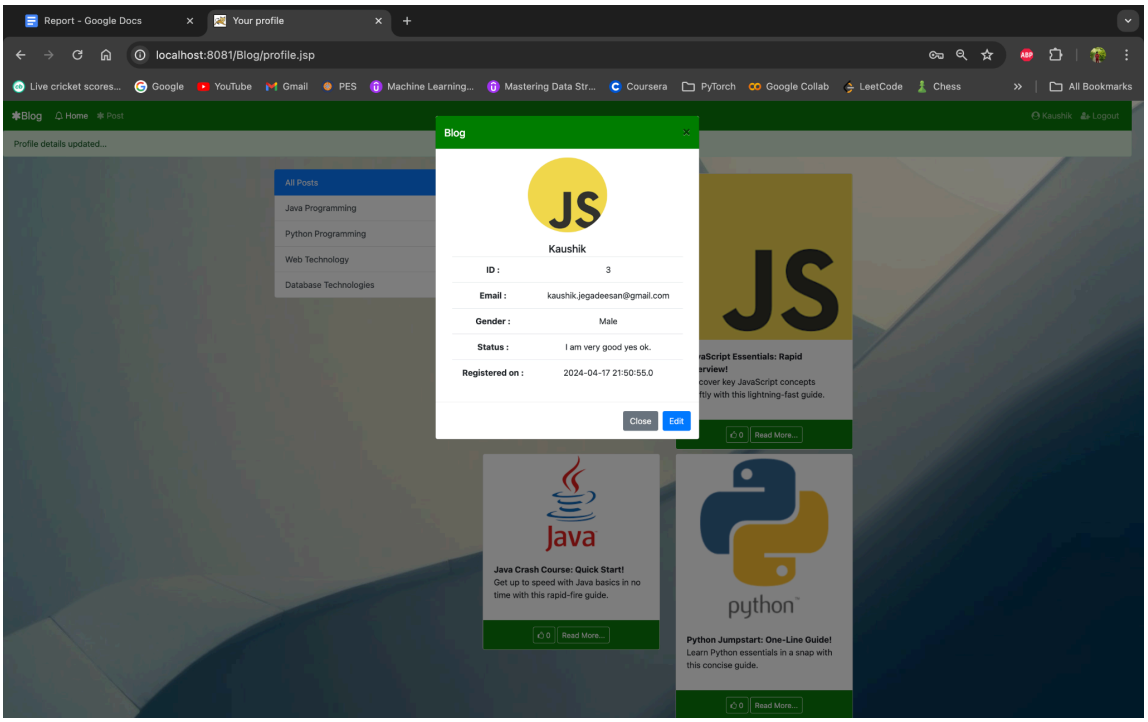
No Profile Picture



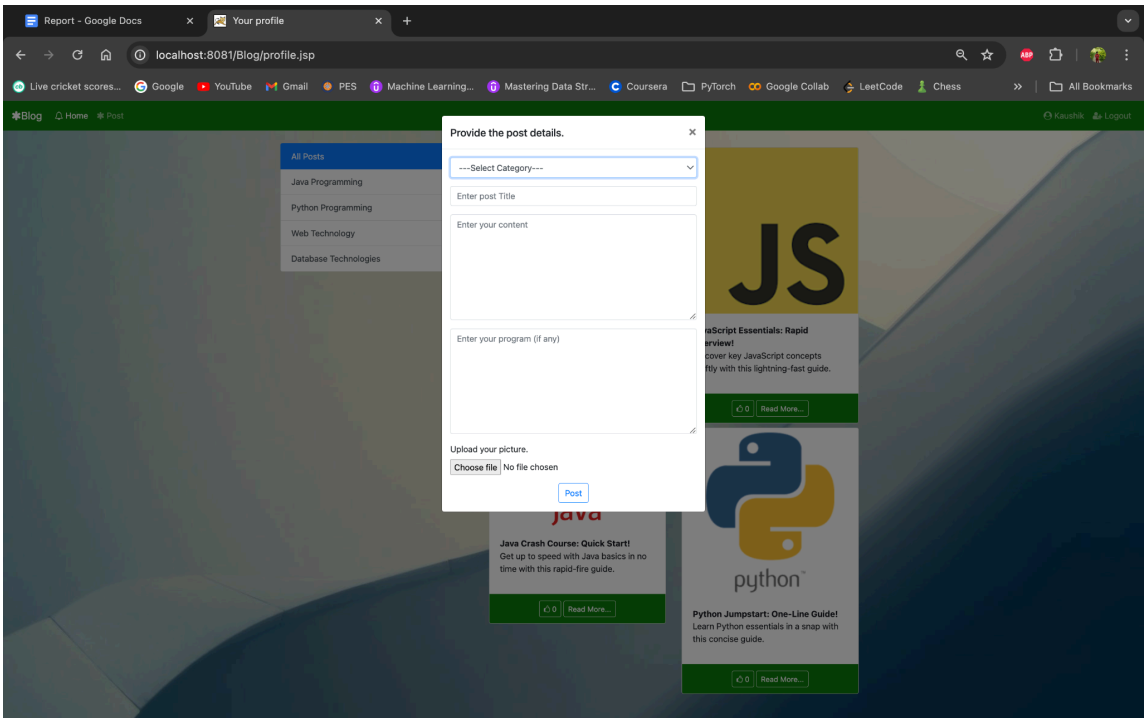


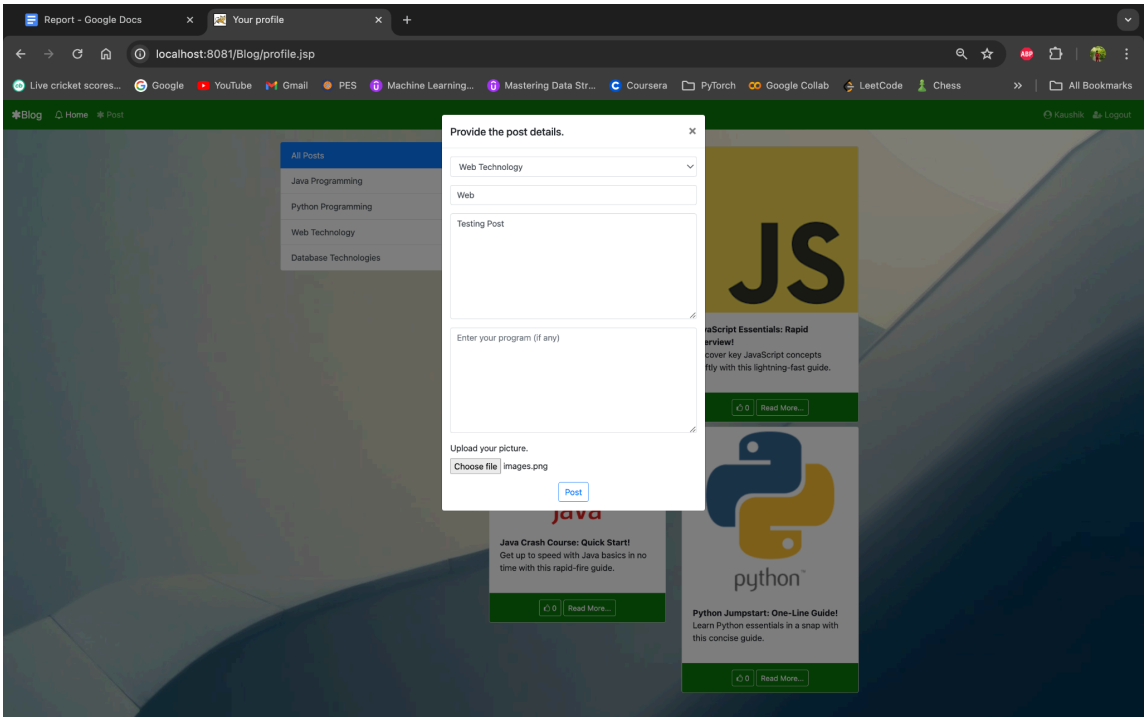
Updated with Profile Picture

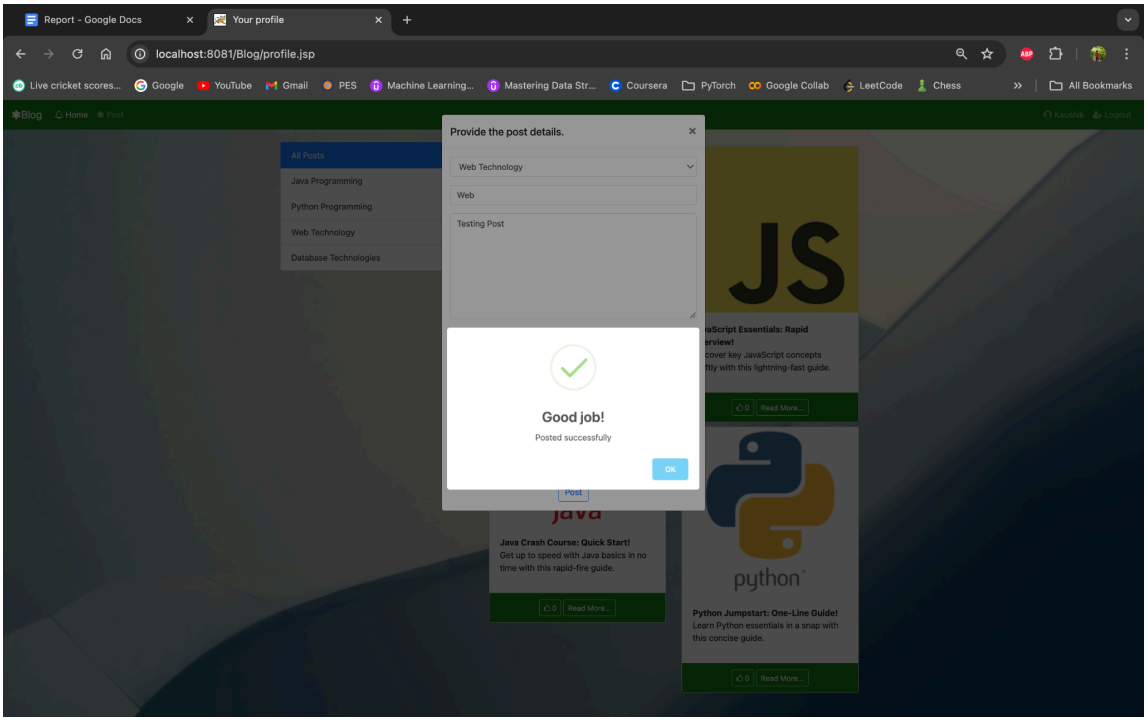




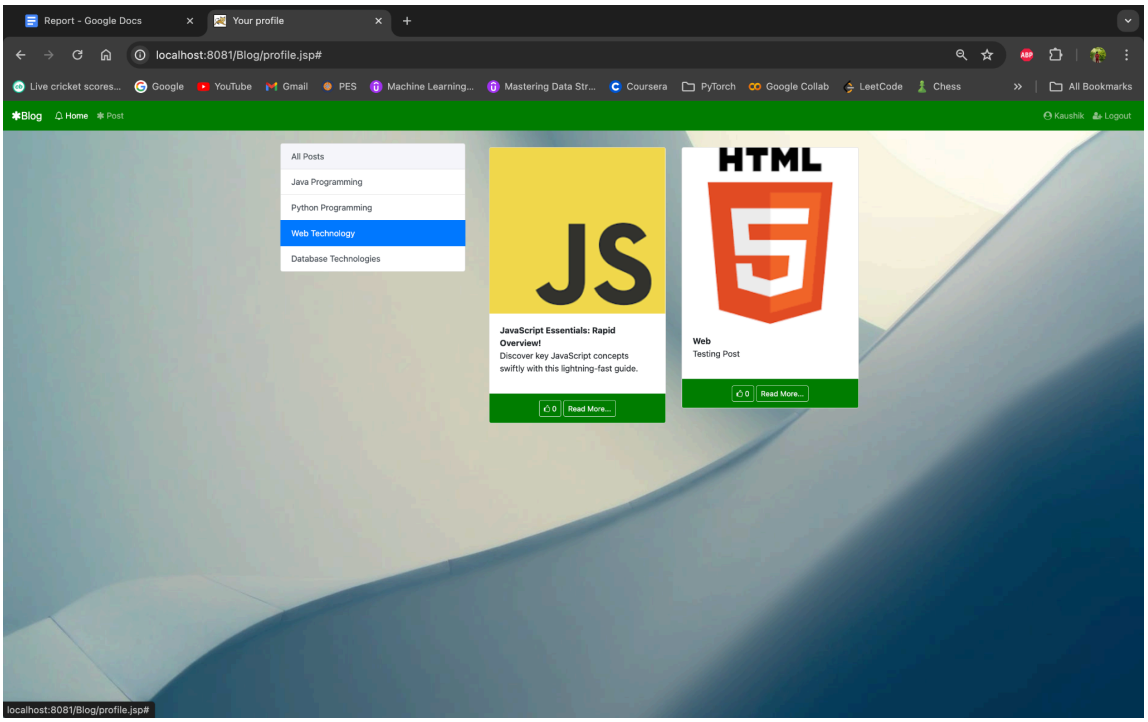
Post Functionality

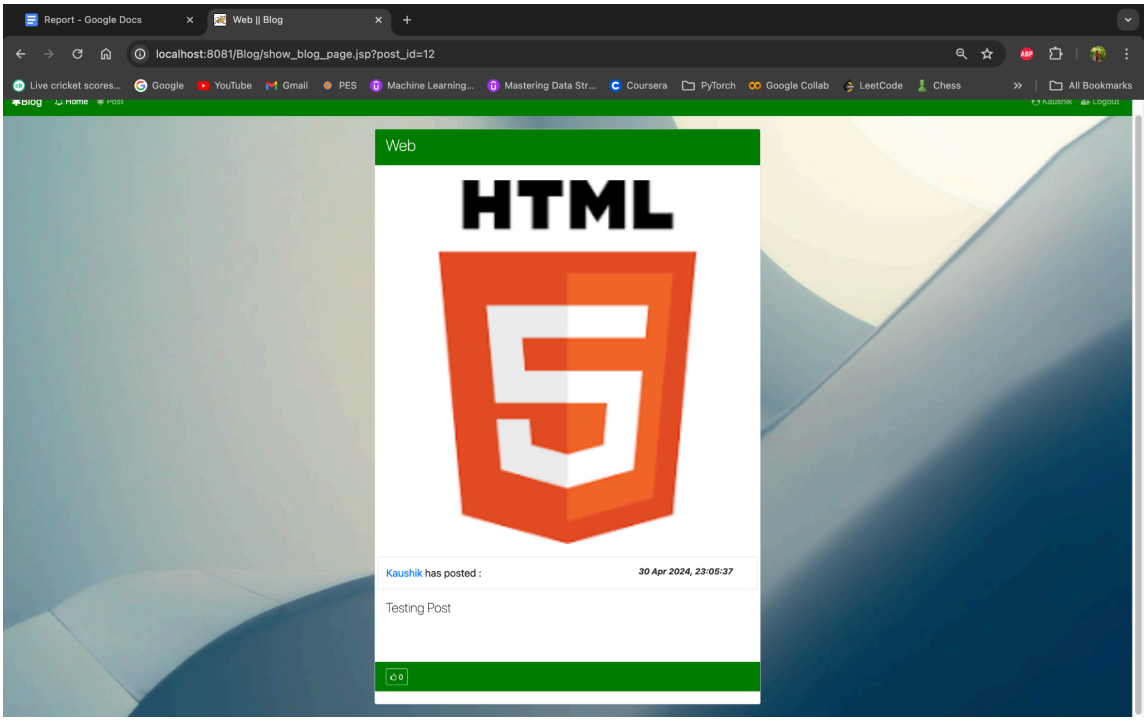



Report - Google Docs
Your profile
localhost:8081/Blog/profile.jsp
Live cricket scores...
Google
YouTube
Gmail
PES
Machine Learning...
Mastering Data Str...
Coursera
PyTorch
Google Collab
LeetCode
Chess
All Bookmarks
Blog
Home
Post
Kaushik
Logout
Provide the post details.
Web Technology
Web
Testing Post
Enter your program (if any)
Upload your picture.
Choose file
images.png
Post
All Posts
Java Programming
Python Programming
Web Technology
Database Technologies
JS
Read More...
Java Crash Course: Quick Start!
Get up to speed with Java basics in no time with this rapid-fire guide.
python
Python Jumpstart: One-Line Guide!
Learn Python essentials in a snap with this concise guide.

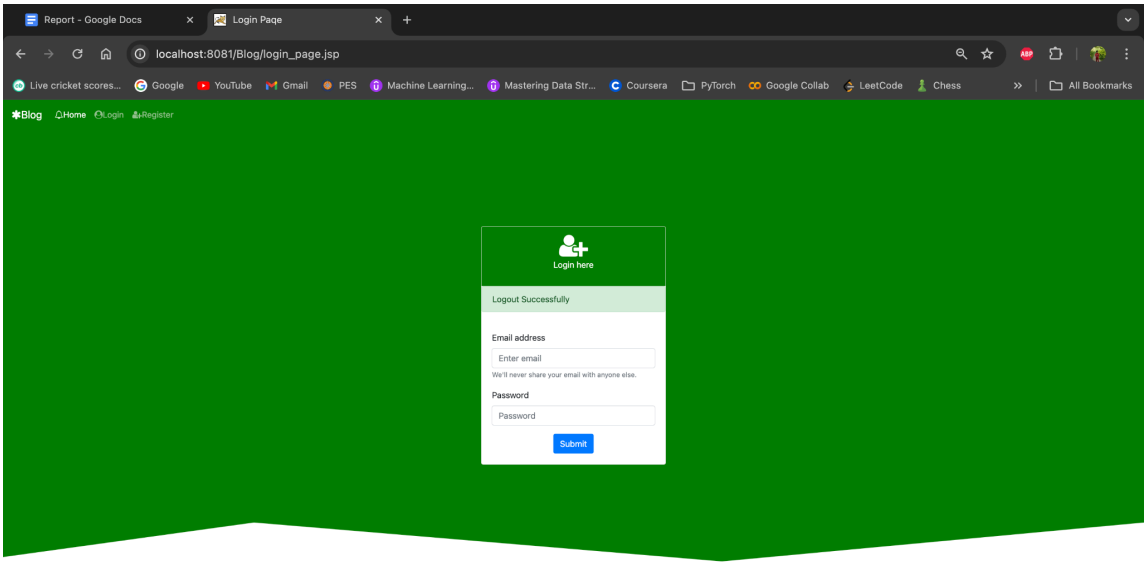

Report - Google Docs
Your profile
localhost:8081/Blog/profile.jsp
Live cricket scores...
Google
YouTube
Gmail
PES
Machine Learning...
Mastering Data Str...
Coursera
PyTorch
Google Collab
LeetCode
Chess
All Bookmarks
Blog
Home
Post
Kaushik
Logout
Provide the post details.
Web Technology
Web
Testing Post
Good job!
Posted successfully
OK
Post
All Posts
Java Programming
Python Programming
Web Technology
Database Technologies
JS
Read More...
Java Crash Course: Quick Start!
Get up to speed with Java basics in no time with this rapid-fire guide.
python
Python Jumpstart: One-Line Guide!
Learn Python essentials in a snap with this concise guide.

Post Updated in Profile Page


Report - Google Docs
Your profile
localhost:8081/Blog/profile.jsp#
Blog
Home
Post
Kaushik
Logout
All Posts
Java Programming
Python Programming
Web Technology
Database Technologies
JS
JavaScript Essentials: Rapid Overview!
Discover key JavaScript concepts swiftly with this lightning-fast guide.
0
Read More...
HTML
Web
Testing Post
0
Read More...
localhost:8081/Blog/profile.jsp#


Report - Google Docs
Web || Blog
localhost:8081/Blog/show_blog_page.jsp?post_id=12
Web
HTML
Kaushik has posted :
30 Apr 2024, 23:05:37
Testing Post
0

Logout


Report - Google Docs
Login Page
localhost:8081/Blog/login_page.jsp
Live cricket scores...
Google
YouTube
Gmail
PES
Machine Learning...
Mastering Data Str...
Coursera
PyTorch
Google Collab
LeetCode
Chess
All Bookmarks
Blog
Home
Login
Register
Login here
Logout Successfully
Email address
Enter email
We'll never share your email with anyone else.
Password
Password
Submit