



KARNATAK UNIVERSITY, DHARWAD

VIDYABHARATI FOUNDATION'S

IBMR College of Arts and Commerce

DEPARTMENT OF COMPUTER APPLICATION (BCA)

HUBBALLI.

PROJECT REPORT ON

“TRAVEL BLOG”

Submitted in fulfillment of the requirement

For the award of the Degree

BACHELOR OF COMPUTER APPLICATION

OF

KARNATAK UNIVERSITY, DHARWAD

Submitted by

SHEETAL SURVE (U02CH22S0093)

During the academic year of 2024-2025



KARNATAK UNIVERSITY, DHARWAD

**VIDYABHARATI FOUNDATION'S
IBMR College of Arts and Commerce**

DEPARTMENT OF COMPUTER APPLICATION (BCA) HUBBALLI.

PROJECT REPORT ON

“TRAVEL BLOG”

Submitted in fulfillment of the requirement

for the award of the Degree

BACHELOR OF COMPUTER APPLICATION

OF

KARNATAK UNIVERSITY, DHARWAD

Submitted by

SHEETAL SURVE (U02CH22S0093)

Under the Guidance of

MRS. SHWETA.W.

**HEAD OF THE DEPT.
PROF. SAUJANYA P**

**PRINCIPAL
DR.ARUN SHETT**

VIDYABHARATI FOUNDATION'S
IBMR College of Arts and Commerce
DEPARTMENT OF COMPUTER APPLICATION (BCA)



CERTIFICATE

This is to certify that the project work entitled

“TRAVEL BLOG”

Submitted in fulfillment of the requirement

for the award of the degree

BACHELOR OF COMPUTER APPLICATIONS

OF

KARNATAK UNIVERSITY, DHARWAD

Is a result of bonafide work carried out

by

SHEETAL SURVE (U02CH22S0093)

During the academic year of 2024-2025

Signature of HOD

Internal Guide

(SHWETA.W.)

Internal Examiner

Signature of Principal

External Guide

External Examiner

DECLARATION

This is to certify that, **SHEETAL SURVE** bearing Register No: **U02CH22S0093** **BCA VI SEMESTER** from IBMR College of Arts and Commerce, Department of BCA, has completed Project Report under **HA EGL** from May to June during the academic year 2024-2025 with the title “**TRAVEL BLOG**” In partial fulfillment of the degree as prescribed by Karnataka University Dharwad.

Place: Hubballi

Date:

ACKNOWLEDGEMENT

It is my proud privilege to express my sincere gratitude to all those who helped me directly or indirectly in completion of project report. I would also like to thank all those persons who have spent their valuable time to contribute the required information and gave me support while preparing this report and I would like to acknowledge and thankful to **My Dean, Principal, IBMR College of Computer Application (BCA) Hubli** for their encouragement and support.

I am greatly indebted to **Mrs. SAUJANYA P , HOD Dept. Computer Application & My Guide (SHWETA.W.) Assistant Professor, Department of Computer Application, IBMR College of BCA** for support guidance and valuable suggestions by which this work has been completed effectively and efficiently.

I would like to express my gratitude to my parents for generating in me a personnel interest to accomplish my Project

I am also thankful to all my friends and non-teaching faculty members for supporting and helping me through the completion of my project work successfully.

Name: SHEETAL SURVE

UUCMS: U02CH22S0093

Ref: HAEGL/June-25/BCA/

INTERNSHIP COMPLETION LETTER

Date : 11/06/2025

This is to certify that **Sheetal Surve**, a student of **IBMR College of BCA Akshay Colony Rd, Vidya Nagar, Hubballi, Karnataka 580021**, has successfully completed a three-month internship at Haegl Technologies Pvt. Ltd., from **01st April 2025** to **10th June 2025**.

During the internship, the student was actively involved in practical learning and contributed to tasks related to **web development using Django**. In addition, the student explored and implemented concepts related to **Convolutional Neural Networks (CNN)** and **Deep Neural Networks (DNN)**. The internship began with a dedicated learning phase and progressed to hands-on application through **real-time projects** in line with Haegl's AI- and IoT-based solutions in Agriculture and Healthcare.

The student exhibited professionalism, technical curiosity and a strong commitment to learning throughout the internship period. The performance and participation during this time were appreciated and aligned well with the objectives of the training program.

We extend our best wishes for future academic and career pursuits.


Gautam Shigaonkar
Founder & CEO
HAEGL Technologies Pvt Ltd



Office 1: #3 SDMCET Incubations SDMCET Dhavalagiri Dharwad Karnataka -580020

Office 2: Ground Floor, Stellar mall, Beside JGCC College, Vidya Nagar, Hubballi, Karnataka 580021

Tel: +91 9590499570

Website: <https://www.haegl.in>

Table of Content

1.Introduction.....	1
2.System Analysis.....	3
2.1 Existing system.....	3
2.2 Proposed System.....	3
2.3 System Architecture.....	4
3.Software Requirement Specification.....	6
3.1 Tools And Technologies.....	6
3.2 Software Requirement.....	6
3.3 Functional Requirement.....	7
3.4 Non-Functional Requirement.....	8
4.Hardware Requirement Specification.....	10
1.1 Hardware Requirement.....	10
5.System Design.....	11
5.1 Data Flow Diagram(DFD).....	11
5.2 E-R Diagram.....	17
5.3 Use Case Diagram.....	21
6. Implementation.....	24
6.1 Introduction to Technologies.....	25
6.2 Source Code.....	26
7.Testing And Results.....	63

8.Screenshoot.....	65
8.1 Home page.....	65
8.2All Blogs page.....	66
8.3 Contact.....	67
8.4 Create Page.....	68
8.5 Admin Login page.....	69
8.6 Admin Home page.....	70
9.Conclusion.....	72
10.Future Enhancement.....	73
11.Bibliography.....	74

1. Introduction

The Travel Blog Application is a full-featured web platform developed to cater to the needs of travel enthusiasts who wish to share their experiences, itineraries, insights, and memories with a broader audience. Unlike traditional blogs that are limited to static content, this application emphasizes user interactivity, dynamic content management, and community building.

Users can register on the platform, create personal profiles, and author blog posts enriched with images and videos to enhance storytelling. The platform fosters engagement through a robust commenting system, like and bookmark and connect with other users.

In addition to content creation, the platform includes powerful features such as search filters for destination or blog type, pagination for browsing large volumes of posts, and notification systems to alert users about activity on their posts or from their favorite bloggers. An intuitive admin dashboard enables content moderation, spam control, and user management, ensuring the platform remains safe and enjoyable for all.

Built using Django and SQLite, the application offers a scalable and secure environment suitable for both casual travelers and professional bloggers. The goal is to create a collaborative travel diary that not only shares adventures but also inspires and informs future travelers.

In the digital age, travel blogs have emerged as powerful platforms that go far beyond being simple diaries of trips. What sets them apart from traditional travel websites or guidebooks is their deeply personal, visually engaging, and experience-driven content. A travel blog uniquely combines storytelling, photography, cultural insights, and practical tips, all from the perspective of an individual traveler.

Each travel blog offers a distinct voice and personality, allowing readers to explore the world through the eyes of someone who has actually lived the experience. This personal touch builds trust and connection with the audience. Unlike commercial travel portals, blogs highlight offbeat paths, hidden gems, local cultures, and authentic experiences, which are often overlooked in mainstream media. Moreover, travel blogs are highly interactive. They often feature comment sections, social media integration, maps, and downloadable resources like travel checklists and itineraries.

Many travel bloggers also engage in content creation as a full-time profession, combining passion, storytelling, and entrepreneurship. Their unique perspective helps promote local tourism, support small businesses, and foster cross-cultural understanding.

A blog website is an online platform where individuals or organizations can publish written content regularly, often in the form of articles or posts. It serves as a space for sharing thoughts, experiences, information, or news on various topics such as travel, food, technology, fashion, education, and more. Originally started as personal online journals, blogs have evolved into powerful tools for communication, marketing, education, and brand building.

A typical blog website features a homepage displaying recent or popular posts, a system to categorize content, and a dedicated space for each article that may include text, images, videos, and comment sections for reader interaction. Blog websites often include search functionality and tagging to help visitors easily find content. With the rise of digital media, many businesses use blog websites to connect with their audience, promote their products or services, and improve their online visibility through SEO. Whether created for personal passion or professional goals, a blog website is a versatile and effective

2. System analysis

2.1 Existing System

The concept of an "existing system" isn't explicitly detailed in the provided workflow, as the document outlines the design for a *new* blog application. However, if we were to consider a hypothetical "existing system" that this project aims to improve upon, it would likely be characterized by:

- **Manual Content Management:** Blog posts and comments might be managed manually, requiring direct database interaction or basic file system operations.
- **Lack of User Interaction Features:** No integrated commenting, liking, or bookmarking systems.
- **Limited Search and Navigation:** Users might struggle to find specific content due to the absence of robust search and pagination.
- **No Role-Based Access Control:** All users might have similar permissions, leading to potential security and content management issues.
- **Absence of Notifications:** Users wouldn't receive automated alerts for new comments or updates.

2.2 Proposed System

The proposed Blog Application aims to be a full-fledged content management system with strong user interaction features. It will provide:

- **User Management:** Secure user registration, authentication, profile management, and role-based access control (User, Admin).
- **Blog Post Management:** Functionality for users to create, edit, publish, and delete their blog posts, along with a blog listing featuring pagination and search.
- **Comment System:** Users can comment on blog posts, and administrators can moderate these comments.
- **Admin Dashboard:** A central panel for administrators to manage users, posts, and comments, and to moderate content.
- **Notifications:** Timely alerts to users regarding comments, likes, and blog updates.

- Search & Pagination: Efficient search functionality for blog posts and pagination for improved user experience.
- Like System: Allows users to express appreciation for blog posts, fostering engagement.
- Bookmark System: Enables users to save blog posts for later reading.

2.3 System Architecture

The system is composed of multiple interconnected components that work together to provide seamless functionality:

1. Frontend Interface (Client Side):

- Built using HTML, CSS, and JavaScript (optionally using frameworks like Bootstrap or Tailwind for styling).
- Offers a responsive UI for users to interact with blog content, post new blogs, and engage through comments, likes, and bookmarks.
- Handles form submissions for login, registration, content creation, and search queries.

2. Backend Framework (Server Side):

- Developed using Django, which follows the MVT (Model-View-Template) architectural pattern.
- Manages user authentication, session handling, form validations, routing, and business logic.
- Handles CRUD operations for blog posts, comments, likes, and user profiles.

3.Database Layer:

- Uses SQLite as the relational database management system.
- Stores structured data including user accounts, blog posts, comments, likes, bookmarks, and metadata such as timestamps and categories.

- Ensures data consistency and enables relational querying via Django's ORM (Object-Relational Mapping).

3. Admin Panel:

- Django's built-in admin interface allows for content and user moderation.
- Admins can approve or delete comments, manage users, categorize content, and monitor activity logs

4. Media Handling:

- Handles image and video uploads for blog posts.
- Media files are stored locally or in cloud storage (if integrated), and URLs are managed via Django's media routing.

6.Notification System:

- Implements in-app notifications for likes, comments, replies, and bookmarks.
- Optionally extendable to email notifications using Django's built-in mailing system.

7.Search & Pagination Module:

- Allows users to filter and search blog posts by destination, keywords, author, or tags.
- Pagination ensures optimal performance and user experience when navigating through large sets of posts.

3.SOFTWARE REQUIREMENT SPECIFIATION

3.1 Tools & Technologies

The document primarily focuses on the functional workflow and database schema, rather than specific implementation technologies. However, based on the typical development practices for such applications, the following tools and technologies would likely be used:

- **Backend Framework:** Python (likely Django or Flask due to the comprehensive nature of the application).
- **Database:** Relational Database Management System (RDBMS) such as PostgreSQL, MySQL, or SQLite, given the detailed table structures. The use of Integer (Primary Key), Varchar, Text, Date Time, Boolean, and Enum data types suggests a relational database.
- **Frontend Technologies:** HTML, CSS, JavaScript (for interactive elements and user interface).
- **Version Control:** Git (for collaborative development and tracking changes).
- **Deployment (Conceptual):** Web server (e.g., Nginx, Apache), application server (e.g., Gunicorn, WSGI), cloud platform (e.g., AWS, Azure, Google Cloud, Heroku).

3.2 Software Requirements:

- Operating System: Linux (e.g., Ubuntu, CentOS), Windows Server, or macOS (for development).
- Programming Language Runtime: Python interpreter.
- Database System: PostgreSQL, MySQL, or SQLite.
- Web Server: Nginx or Apache.
- Application Server: Gunicorn or uWSGI (for Python web frameworks).
- Web Browser: Latest versions of Chrome, Firefox, Edge, Safari (for end-users).
- Code Editor/IDE: VS Code, PyCharm, Sublime Text. Version Control System: Git.

Functional And Non-Functional Requirements

3.3 Functional Requirements:

1. User Management:

- User Registration & Authentication: Users must be able to register new accounts and log in securely.
- Profile Management: Users can view and update their personal profiles.
- Role-Based Access Control: Distinguish between 'user' and 'admin' roles with different permissions.
-

2. Blog Post Management:

- Create Post: Authenticated users can create new blog posts.
- Edit Post: Authors can edit their existing blog posts.
- Publish Post: Users can publish their blog posts to make them public.
- Delete Post: Users can delete their blog posts.
- Blog Listing: Display a list of blog posts.
- Pagination: Implement pagination for Browse blog posts.
- Search Functionality: Allow users to search for blog posts by keywords.

3. Comment System:

- Add Comment: Users can post comments on blog posts.
- Moderate Comments: Administrators can approve, reject, or delete comments.

4. Admin Dashboard:

- User Management: Admins can view, edit, and manage user accounts and roles.
- Content Moderation: Admins can manage blog posts and comments.

5. Notifications:

- Timely Notifications: Notify users about new comments, likes, and blog updates.

- Read Status: Track whether a notification has been read.

6. Like System:

- Like Post: Users can like blog posts.

7. Bookmark System:

- Bookmark Post: Users can save blog posts to read later.

3.4 Non-Functional Requirements:

1. Security:

- **Password Hashing:** Store hashed passwords for user accounts.
- **SQL Injection Prevention:** Protect against malicious database queries.
- **XSS Protection:** Prevent cross-site scripting vulnerabilities.
- **Role-Based Access Control:** Ensure only authorized users (e.g., admins) can perform sensitive actions.

2. Performance:

- **Fast Page Load Times:** Ensure quick loading of blog posts, listings, and other pages.
- **Efficient Search:** Search queries should return results quickly.
- **Scalability:** The system should be able to handle an increasing number of users, posts, and comments without significant performance degradation.

3. Usability:

- **Intuitive UI/UX:** The interface should be easy to navigate and understand for all user types.

- **Clear Feedback:** Provide clear messages for successful operations, errors, or pending actions.
- **Responsive Design:** The application should be accessible and visually appealing across various devices (desktops, tablets, mobile).

4. Reliability:

- **Data Integrity:** Ensure the consistency and accuracy of all data in the database (users, posts, comments, likes, bookmarks, notifications).
- **Error Handling:** Graceful handling of errors and exceptions to prevent system crashes.
- **Backup and Recovery:** Implement a strategy for regular database backups.

5. Maintainability:

- **Modular Codebase:** Well-organized and documented code for easy understanding and modification.
- **Testability:** Code should be structured to allow for easy unit and integration testing.

4.HARDWARE REQUIREMENTS SPECIFICATION

1.1 Hardware Requirements:

- **Server (Production):**

- Processor: Multi-core CPU (e.g., Intel Xeon, AMD EPYC) to handle concurrent user requests.
- RAM: 8GB to 16GB or more, depending on expected traffic and data volume.
- Storage: SSD (Solid State Drive) for faster database operations and content delivery; sufficient space for database and media files.
- Network Interface: Gigabit Ethernet for high-speed data transfer.

Development Machine:

- Processor: Intel Core i5/i7 or equivalent AMD Ryzen.
- RAM: 8GB to 16GB.
- Storage: 256GB SSD or higher.

Internet Connection: Stable broadband connection

5.SYSTEM DESIGN

5.1 Data Flow Diagram

A **Data Flow Diagram (DFD)** for a blogging system helps visualize how information moves between users, processes, and storage components. At a high level, the system consists of three main entities: the **User**, the **Blog System**, and the **Database**.



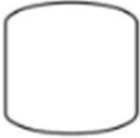



When a user interacts with the blog—such as submitting a new post or leaving a comment—the system processes the input and stores the data in a database. The blog system retrieves and displays content whenever a user requests a blog post or navigates different sections.

In more detail, users submit their credentials to log in, write and edit blog posts, and engage through comments. These actions are processed by the blog management system, which verifies, stores, and retrieves data from the database as needed. The system also incorporates content moderation, ensuring that only approved posts and comments are published.

A Data Flow Diagram (DFD) for a blog website visually represents how data moves through the system and how different components interact with each other. At the context level (Level 0), the blog website is shown as a single central process that interacts with external entities like users and administrators. Users can perform actions such as reading blog posts, registering, logging in, and submitting comments, while admins can add, edit, or delete blog posts. As we move to Level 1, the single process is broken into multiple sub-processes like "User Registration", "User Login", "Create Blog Post", "View Posts", and "Comment on Post". Each of these processes exchanges data with data stores such as the "User Database", "Blog Posts Database", and "Comments Storage".

For example, when a user logs in, their credentials are passed to the login process, which then verifies the data from the user database. Similarly, when a new blog post is created by the admin, the post data flows into the blog post storage, and is later retrieved when users browse the website. The DFD also illustrates how data flows like login information, post content, and comments travel between users, processes, and data storage, giving a clear picture of how the system operates internally. This helps developers understand the functional requirements of the blog site and identify how to structure the backend logic effectively.

DATA FLOW DIAGRAM NOTATION:

	Rectangle. It defines a source or destination of system data.
	Circle. It represents a process that transforms incoming data flow into outgoing data flow.
	Oval Bubble. Same as Circle.
	Pipe. It shows a data store – data at rest or a temporary repository of data.
	Open Rectangle. Same as pipe.
	Arrows. They identify data flow i.e. data in motion.

Data Flow Diagram: DFD Level 0

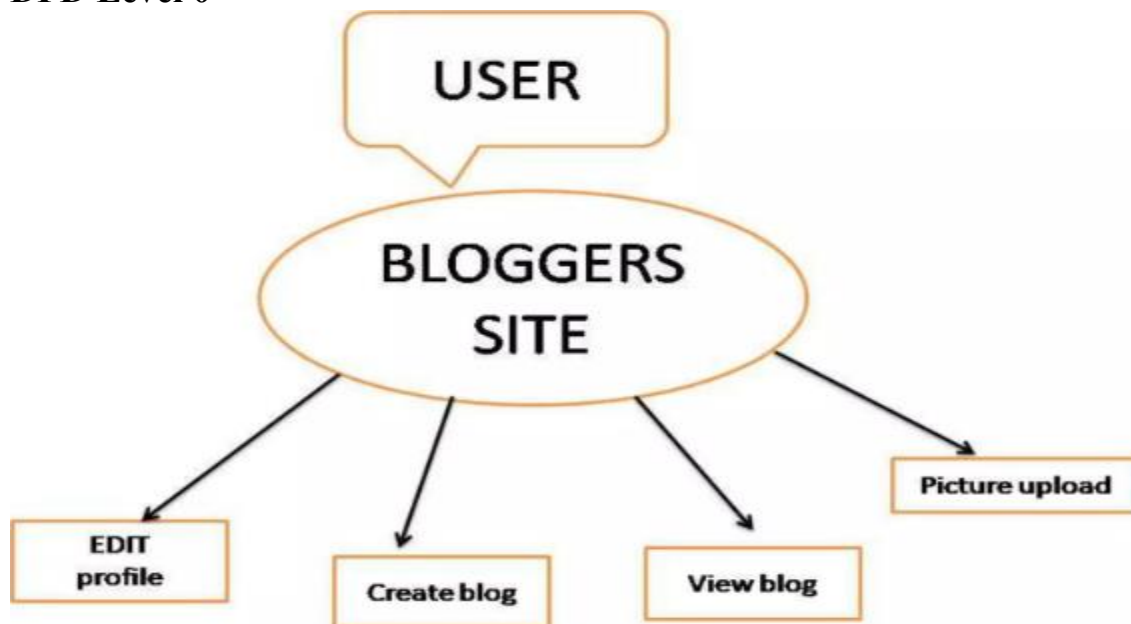


Fig 5.1 DFD Level 0

Data Flow Diagram (DFD) Level 0 , also known as context Diagram, provides a high-level overview of a system. It represents the entire system as a **single process** and shows how it interacts with **external entities** . The focus is on the **flow of data** between the system and these external entities, without detailing the internal workings of the system. The entire system is depicted as **one process**, usually represented by a circle or a rectangle. **Data Flows** Arrows indicate the movement of data between the system and external entities.

DFD Level 1

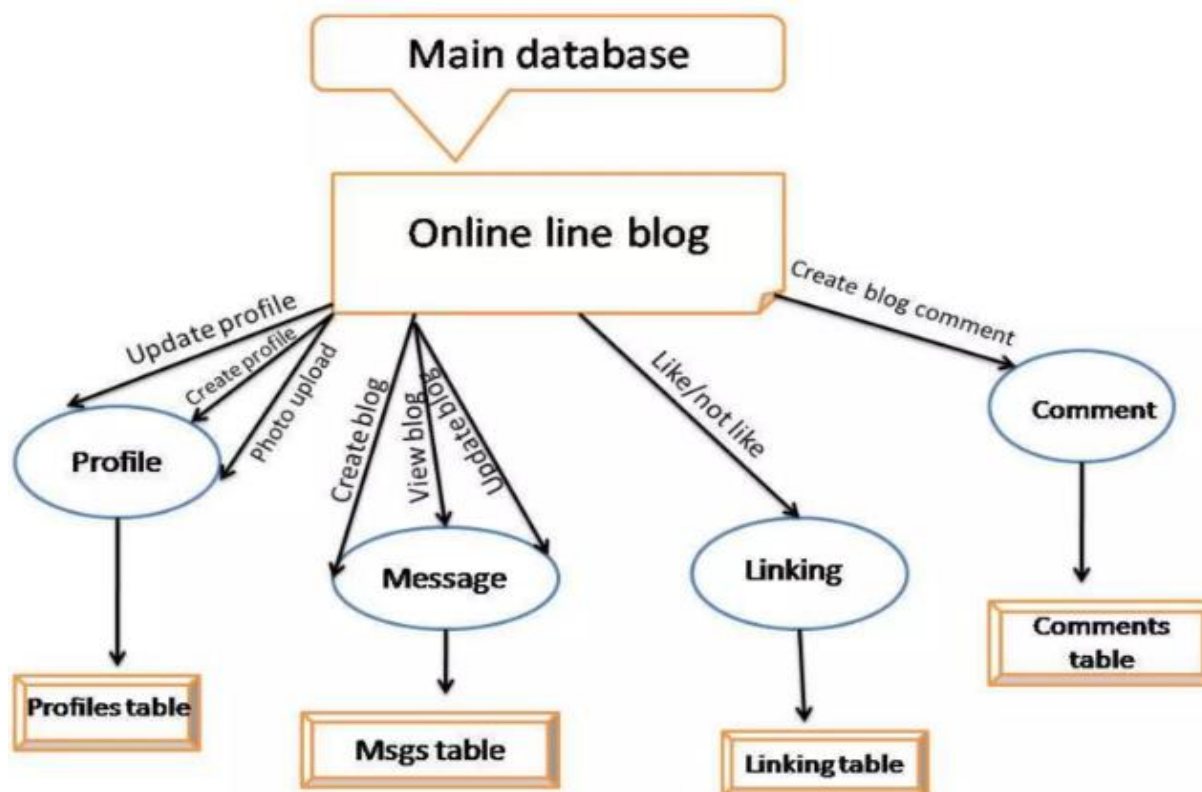
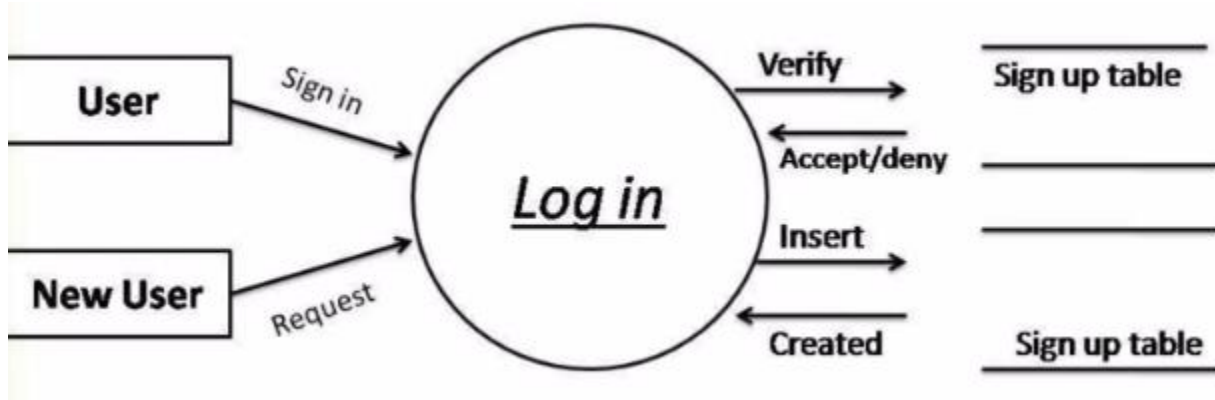


Fig 5.2 DFD Level 1

The Level 1 Data Flow Diagram (DFD) of the blog website breaks down the central system, labeled as "Online Line Blog," into several key subprocesses that show how data flows between users, the system, and various data storage tables. The central blog system is connected to the main database, which stores all information handled by the site. The diagram includes subprocesses such as Profile, Message, Linking, and Comment, each representing an important function within the blog.

The Profile process allows users to create and update their personal information, which is saved in the Profiles Table. The Message process enables users to write new blog posts or view existing ones, and these messages are stored in the Msgs Table. The Linking process represents interactions such as liking or disliking posts, with data stored in the Linking Table. Lastly, the Comment process handles user-generated comments on blog posts, storing them in the Comments Table. All these processes communicate with the main blog system, ensuring smooth operation and data storage. This DFD Level 1 provides a detailed view of how user activities are managed and how data flows through different parts of the blog application, helping in the design and understanding of the internal structure of the system.



DFD Level 1-Login

In this DFD Level 1 for the login process, the system separates two types of users: existing users and new users. Existing users go through the Sign In process by entering their login credentials (such as username and password), which the system then verifies by comparing the input with the data stored in the Sign Up Table. If the credentials are correct, access is accepted, otherwise it's denied.

On the other hand, new users must first request to sign up. Their data is inserted into the Sign Up Table, creating a new account in the system. This data is also used for future login verification. Once registered successfully, the new user can log in like an existing user.

This diagram emphasizes the flow of data between users and the login system, including actions like signing in, verifying credentials, inserting new data, and storing it in a signup database. It clearly outlines how authentication and user registration are handled within the blog website.

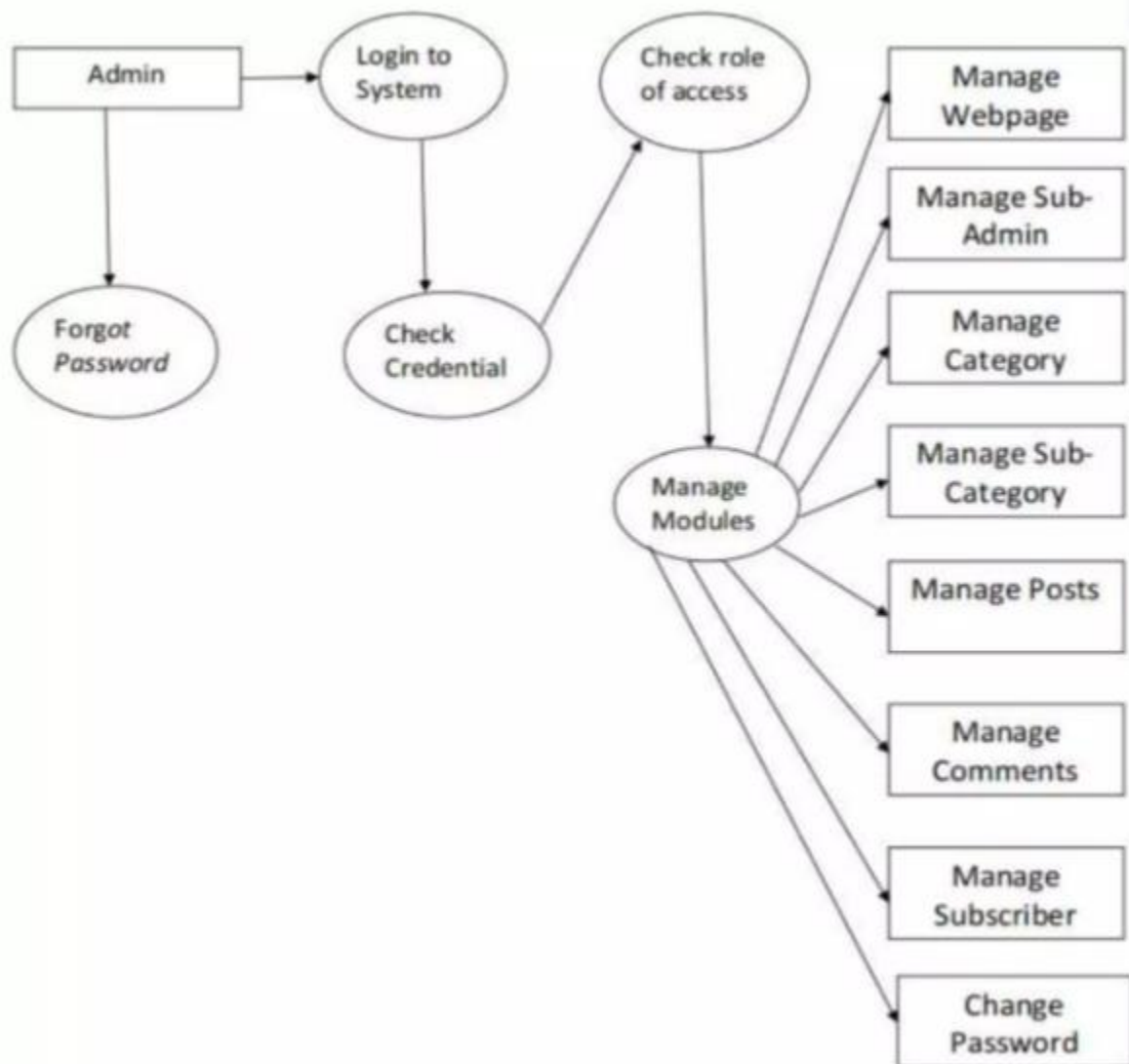


Fig 5.4 DFD LEVEL-2 Admin

In this DFD Level 2 diagram, the Admin is shown as the primary actor who logs into the system by entering credentials. The process begins with the Login to System module, where the admin's identity is verified through a Check Credential process. If credentials are correct, the system proceeds to Check Role of Access, ensuring the user has admin privileges. There's also a provision for Forgot Password if the admin cannot log in. Once authenticated and role is verified, the admin gains access to the Manage Modules section, which allows them to control various parts of the blog platform. These modules include managing the Webpage, Sub-Admins, Categories, Sub-Categories, Posts, Comments, and Subscribers. Additionally, the admin has the ability to Change Password to maintain account security. This level of the DFD clearly defines the responsibilities and control the admin holds within the system.

5.2 ER Diagram




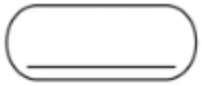
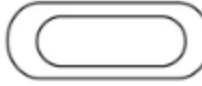

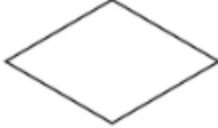
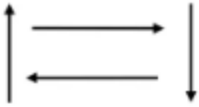
An Entity-Relationship Model (ERM) is an abstract and conceptual representation of data. Entity-relationship modelling is a database modelling method, used to produce a type of conceptual schema or semantic data model of a relational database,

Diagram created using this process are called entity-relationship diagrams,

The Entity Relationship Diagram, or ER Diagram, provides a graphical notation for representing data models that are typically used in first stage of information system design. that is to be stored In the database during the phase of requirement analysis

Today all computers software use Data Base Management System for information storage and retrieval alone with its manipulation in such scenario, ER Diagram symbolizes the design of an information system that is based on databases. The conceptual data model is a later stage of designing mapped to a logical data model such as the relational model. therefore, ER Diagram is a data model or data diagram for high level description of conceptual data models.

Entity Relationship notations

	Rectangle. It represents the entities, the things about which we seek information.
	Weak Entity. It depends on another entity to exist.
	Ellipse. It shows attributes that are properties of the entity.
	Primary Key / Attribute. It is the unique, distinguishing property of the entity.
	Multivalued Attribute. It can have more than one value.
	Derived Attribute. It is based on another attribute to exist.
	Diamond. It shows relationships that provide the structure which draws information from multiple entities.
	Arrows. They identify the flow i.e. movement of information in a flow chart.

ENTITY RELATIONSHIP:

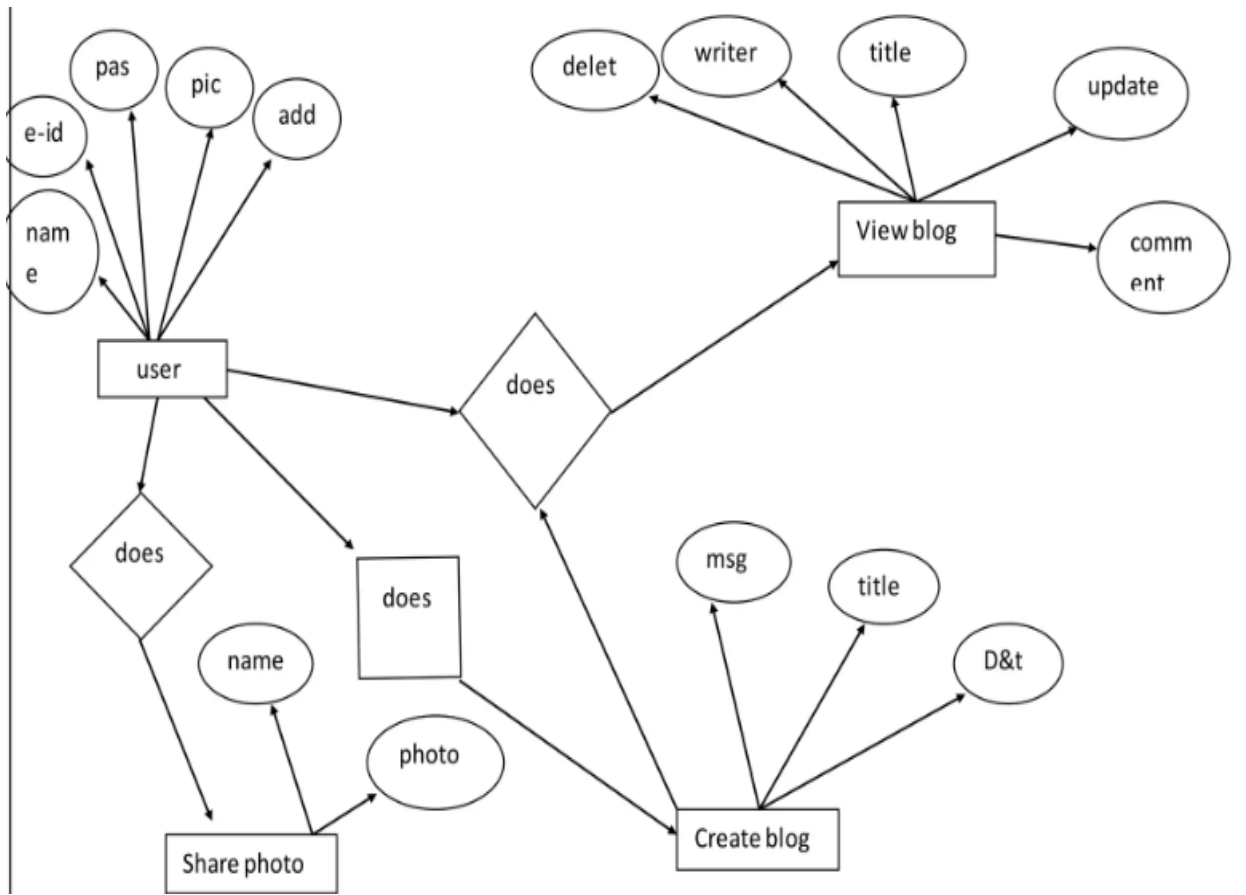


Fig 5.5:ER Diagram

The image shows an Entity relationship (ER) diagram for a blog website, which visually represents how different elements are connected.

1. User: User is an important entity that represents an individual who interacts with the blog system. This entity includes key attributes such as: e-id (Email ID) name pass (Password)(Profile Picture) add (Address) In an ER Diagram: The User entity is shown as a rectangle. Its attributes are shown as ovals connected to the rectangle. The relationship between entities is shown as a diamond with connecting lines.

2. share photo: One-to-One (1:1): This relationship shows that each User can perform a Share Photo action. The Share Photo entity contains attributes like name and photo. This indicates that

each user can share a specific photo having a name and file.

Example: A user uploads one photo with one title or description.

3. create blog: One-to-Many (1:N): The Create Blog action allows a user to create multiple blogs. The Create blog entity contains attributes such as: msg (Message or blog content)title (Title of the blog) D&t (Date and time of blog creation) This shows that one user can create many blog entries, each having unique content and time.

4. view blog: Many-to-Many (M:N): The View Blog entity is used when users read blogs written by themselves or others. This entity has attributes like:writer (Author of the blog)title (Blog title) delete (option to remove the blog)update (option to edit the blog)comment (add feedback)Many users can view many blogs, and each blog can have multiple viewers and comments.

This ERD represents how users interact with blog functionalities like sharing photos, creating blogs, and viewing or managing blog posts, by using clearly defined relationships and attributes

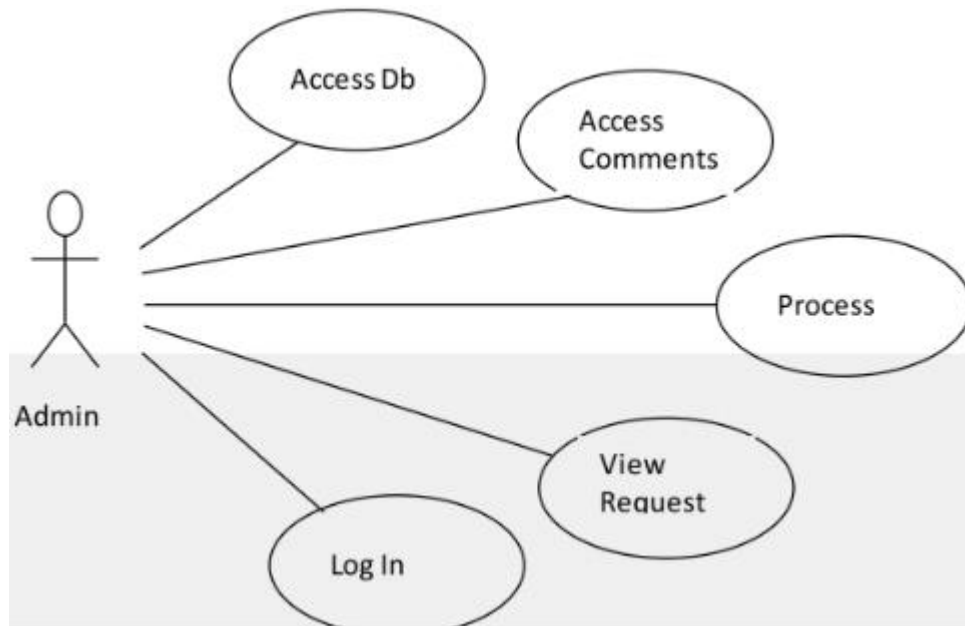
USE CASE DIAGRAM:

Fig 5.6: Admin user case

This image displays an Admin Use Case Diagram, which illustrates the interactions and functionalities available to an administrator within a blog or content management system. In the diagram, the actor labeled "Admin" is shown on the left side, represented by a stick figure. The admin interacts with multiple system functionalities, each represented as an oval on the right side, which are the use cases or actions that the admin can perform.

The main use cases include:

Access Db – The admin can access the database to manage system data or perform administrative queries. Access Comments – The admin has the ability to view, manage, or moderate user comments on blogs or content. Process – This likely refers to handling various background tasks such as content approvals, moderation, or user requests. View Request – The admin can view requests submitted by users, possibly for support or content-related actions. Log in – The admin must log in to the system to gain access to all these administrative features.

This diagram is a high-level representation of how the admin interacts with the system. It is useful for understanding the roles, permissions, and responsibilities assigned to administrators, ensuring that critical functionalities are managed securely and efficiently.

USER CASE DIAGRAM:

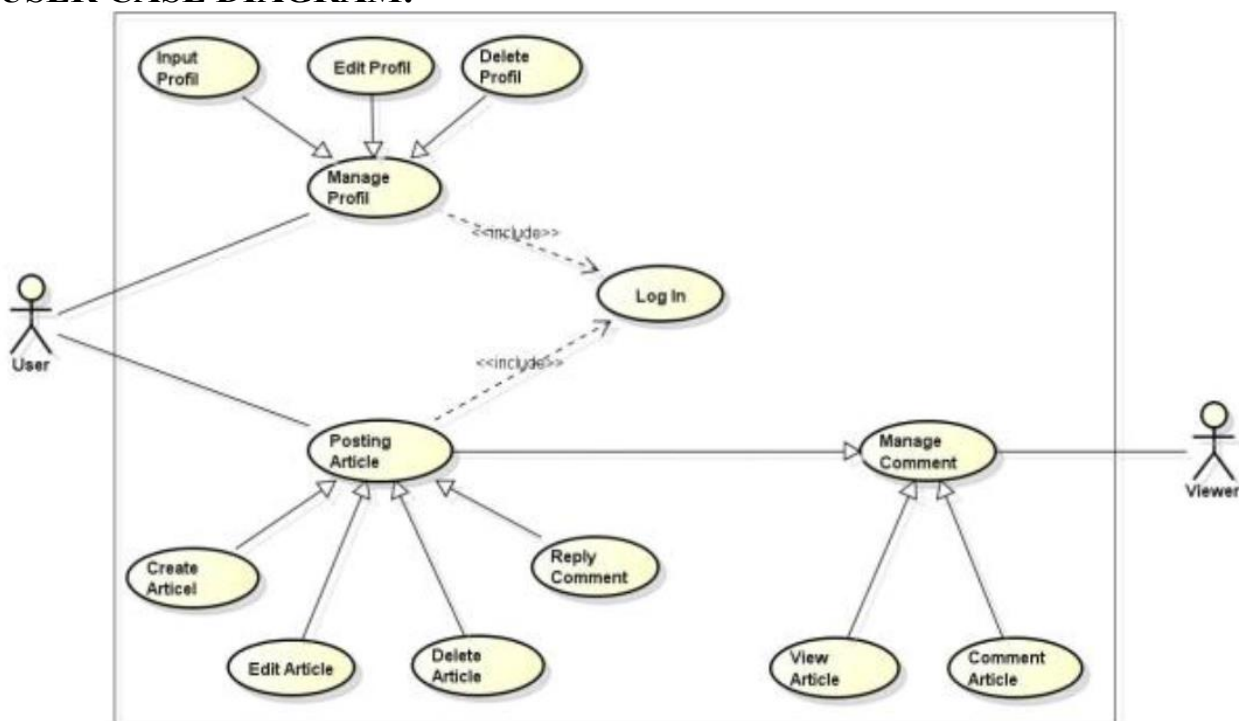


Fig 5.7: User case Diagram

This image shows a User Case Diagram for a blogging system, which visually represents how different types of users interact with various features of the system. In this diagram, there are two main actors: User and Viewer, represented by stick figures. The User can perform a wide range of actions such as managing their profile, posting content, and editing or deleting their articles. The profile management functions are grouped under the use case Manage Profile, which includes sub-actions like Input Profile,

Edit Profile, and Delete Profile. Users can also Log in to the system to access personalized services.

Once logged in, users can perform content-related tasks through the Posting Article use case. This includes Create Article, Edit Article, and Delete Article, reflecting the ability to manage blog content. Users can also Reply to Comments, engaging with the audience.

On the other hand, the Viewer interacts with the system mainly by accessing and responding to content. They can View Article, Comment on Article, and interact through the Manage Comment use case. This highlights the separation of roles in the system: contributors (Users) and consumers (Viewers).

Overall, the diagram effectively illustrates the system's functional requirements by mapping out how different actors perform various actions. It is a useful tool in system design, helping to ensure that both the content creators and viewers have the necessary features for a smooth blogging experience.

6.IMPLEMENTATION

is the stage where the theoretical design is turned into a working Implementation system. Once the design is complete, most of the major decisions about the system have been made. The goal of the coding phase is to translate the design of the system into code in a given programming language. For a given design, the aim in this phase is to implement the design in the best possible manner.

The coding phase affects both testing and maintenance profoundly. Since the testing and maintenance costs of software are much higher than the coding cost, the goal of the coding should be to reduce the testing and maintenance effort. Hence, during coding the focus should be on developing the programs that are easy to read and understand, neither simply on developing the programs that are easy to read and understand, nor simply on developing programs that are easy to write.

The implementation phase also focuses on ensuring modular coding, which means dividing the system into manageable modules like user management, blog management, and comment handling. Each module is tested individually to ensure functionality and integrated into the main system smoothly.

Since blogs often deal with dynamic content and frequent updates, the code should be written in a clear, maintainable, and scalable manner. Readability and documentation are essential, especially to support future feature enhancements or bug fixes. Efficient implementation reduces maintenance overhead and increases system reliability.

6.1 Introduction to Technologies

6.1.1 Django

Django is a free and open-source web framework written in **Python** that helps developers build secure, scalable, and maintainable web applications quickly. It follows the **Model-Template-View (MTV)** architecture, which separates the data, user interface, and application logic, making development more organized and efficient. Django comes with many built-in features such as user authentication, form handling, URL routing, and an admin interface that saves developers time and effort. It also includes a powerful ORM (Object-Relational Mapper) for working with databases using Python code instead of SQL. Designed with a “**batteries-included**” philosophy, Django provides everything needed to build full-featured web apps right out of the box. Known for its emphasis on **security**, Django helps protect your applications from common threats like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). It is widely used in the industry by companies like Instagram, Pinterest, and Mozilla, and is supported by a strong, active community and excellent documentation. Django is ideal for both beginners and experienced developers looking to build modern, high-performance web applications.

6.1.2 SQLITE

SQLite is a lightweight, self-contained, and serverless **relational database management system (RDBMS)** that is widely used for applications that require a simple, efficient way to store and retrieve structured data. Unlike traditional databases such as MySQL or PostgreSQL, SQLite does not run as a separate server process. Instead, it reads and writes data directly to ordinary disk files, making it incredibly easy to set up and use—no installation or configuration is needed. Each SQLite database is a single file that can be easily copied, moved, or shared, which is ideal for small to medium-sized applications, embedded systems, desktop software, mobile apps, and even testing environments. Despite its simplicity, SQLite supports most of the SQL standard, including transactions, joins, indexing, and constraints. It is the default database engine used by Django during development because of its simplicity and zero-configuration nature. However, for larger or more complex projects with heavy traffic or concurrency needs, developers often switch to more robust databases like PostgreSQL or MySQL in production environments. SQLite is open-source,

6.1.3 HTML and CSS

HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are the two core technologies used to create and design webpages. HTML is the foundation of every webpage—it provides the structure and content, using elements like headings, paragraphs, images, links, and forms. Think of HTML as the building blocks of a house, defining what appears on the page. On the other hand, CSS is used to control the presentation and layout of the HTML elements. It handles things like colors, fonts, spacing, and positioning, allowing developers to style websites and make them visually appealing. While HTML tells the browser *what* to display, CSS tells it *how* to display it. Together, HTML and CSS are essential for front-end web development, forming the basis for any website you see on the internet. They are also the starting point for learning more advanced technologies like JavaScript, web frameworks, and responsive design.

6.2 Source Code

Base Code:

```
{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <meta name="description" content="">

    <meta name="author" content="TemplateMo">

    <link

href="https://fonts.googleapis.com/css?family=Roboto:100,100i,300,300i,400,400i,500,500i,700,700i,900,900i&display=swap" rel="stylesheet">

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css"
```

```
rel="stylesheet" integrity="sha384-4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDEXuppvnDJzQIu9" crossorigin="anonymous">
```

```
<title>{% block title %}BlogSpot{% endblock %}</title>
```

```
<link href="{% static 'vendor/bootstrap/css/bootstrap.min.css' %}" rel="stylesheet">
```

```
<link rel="stylesheet" href="{% static 'assets/css/fontawesome.css' %}">
```

```
<link rel="stylesheet" href="{% static 'assets/css/templatemo-stand-blog.css' %}">
```

```
<link rel="stylesheet" href="{% static 'assets/css/owl.css' %}">
```

```
<link rel="stylesheet" href="{% static 'assets/css/styles.css' %}">
```

```
<style>
```

```
.notification-dropdown {
```

```
    max-height: 300px;
```

```
    overflow-y: auto;
```

```
    min-width: 300px;
```

```
}
```

```
.notification-item:hover {
```

```
    background-color: #f8f9fa;
```

```
}
```

```
.navbar-brand h2 {
```

```
    margin: 0;
```

```
    font-size: 1.5rem;
```

```
}
```

```
@media (max-width: 600px) {
```

```
.notification-dropdown {  
  
    max-width: 100%;  
  
}  
  
}  
  
</style>  
  
</head>  
  
<body>  
  
    <div id="preloader">  
  
        <div class="jumper">  
  
            <div></div>  
  
            <div></div>  
  
            <div></div>  
  
        </div>  
  
    </div>  
  
    <header>  
  
        <nav class="navbar navbar-expand-lg bg-white">  
  
            <div class="container">  
  
                <a class="navbar-brand" href="{% url 'index' %}">  
  
                    <h2>Traval<span style="color: rgb(241, 110, 70);">Blog</span></h2>  
  
                </a>  
  
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-  
target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-  
label="Toggle navigation">
```

```
<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarResponsive">

  <ul class="navbar-nav ms-auto">

    <li class="nav-item">

      <a class="nav-link" href="{% url 'index' %}">Home <span class="sr-
only">(current)</span></a>

    </li>

    <li class="nav-item">

      <a class="nav-link" href="{% url 'blog' %}">Blogs</a>

    </li>

    <li class="nav-item">

      <a class="nav-link" href="{% url 'contact_us' %}">Contact</a>

    </li>

    {% if user.is_authenticated %}

    <li class="nav-item">

      <a class="nav-link" href="{% url 'create' %}">Create</a>

    </li>

    <li class="nav-item">

      <a class="nav-link" href="{% url 'profile' user.id %}">Profile</a>

    </li>

    <li class="nav-item">
```

```
<a class="nav-link" href="{% url 'notifications' %}" id="notificationIcon">

    <i class="fa fa-bell"></i>

    {% if notifications %}

    <span class="badge bg-danger rounded-pill">{{ notifications.count }}</span>

    {% endif %}

</a>

    <ul class="dropdown-menu dropdown-menu-end notification-dropdown"
id="notificationDropdown" style="display: none;">

        <li class="dropdown-item text-center">Loading...</li>

    </ul>

</li>

<li class="nav-item">

    <a class="nav-link text-primary" href="{% url 'logout' %}">Logout</a>

</li>

    {% else %}

    <li class="nav-item">

        <a class="nav-link" href="{% url 'signin' %}"><i class="fa fa-user"></i></a>

    </li>

    {% endif %}

</ul>

</div>

</div>
```

```
</nav>

</header>

<div class="main-banner header-text">

  <div class="container-fluid">

    <div class="owl-banner owl-carousel"></div>

  </div>

</div>

<main>

  {% if messages %}

  <div class="container mt-3">

    {% for message in messages %}

    <div class="alert alert-{% if message.tags %} {{ message.tags }} {% else %}info{%
endif %} alert-dismissible fade show" role="alert">

      {{ message }}

      <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
label="Close"></button>

    </div>

    {% endfor %}

  </div>

  {% endif %}

  {% block content %}

  {% endblock content %}

</main>
```

<footer>

<div class="container">

<div class="row">

<div class="col-lg">

<div class="p-1">

<p>

© Copyright 2023 | All Rights Reserved

</p>

</div>

</div>

</div>

</div>

</footer>

<script src="{% static 'vendor/jquery/jquery.min.js' %}"></script> <script>
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.min.js"
integrity="sha384-
I7E8VVD/ismYTF4hNIPjVp/Zjvgyol6VFvRkX/vR+Vc4jQkC+hVqc2pM8ODewa9r"
crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.min.js"
integrity="sha384-
Rx+T1VzGupg4BHQYs2gCW9It+akI2MM/mndMCy36UVfodzCJcF0GGLxZIzObiEfa"
crossorigin="anonymous"></script>

<script src="{% static 'assets/js/custom.js' %}"></script>


```
<script src="{% static 'assets/js/owl.js' %}"></script>

<script src="{% static 'assets/js/slick.js' %}"></script>

<script src="{% static 'assets/js/isotope.js' %}"></script>

<script src="{% static 'assets/js/accordions.js' %}"></script>

<script>

$(document).ready(function() {

    $('#notificationIcon').on('click', function(e) {

        e.preventDefault();

        const dropdown = $('#notificationDropdown');

        if (dropdown.is(':visible')) {

            dropdown.hide();

            return;

        }

        $.ajax({

            url: '{% url "notifications" %}',

            method: 'GET',

            success: function(data) {

                dropdown.empty();

                if (data.notifications.length === 0) {

                    dropdown.append('<li class="dropdown-item">No new notifications</li>');

                } else {

                    data.notifications.forEach(function(n) {
```

```

        const link = n.post_id ? `<a class="dropdown-item notification-item"
href="/post/${n.post_id}"/>${n.message}<br><small class="text-
muted">${n.created_at}</small></a>` : `<span class="dropdown-item notification-
item">${n.message}<br><small class="text-muted">${n.created_at}</small></span>`;

        dropdown.append(`<li>${link}</li>`);

    });

}

dropdown.show();

$('#notificationIcon .badge').remove();

},

error: function() {

        dropdown.empty().append(`<li class="dropdown-item">Error loading
notifications</li>`);

        dropdown.show();

    }

});

});

$(document).on('click', function(e) {

    if (!$ (e.target).closest('#notificationIcon, #notificationDropdown').length) {

        $('#notificationDropdown').hide();

    }

});

});

```

```
</script>

</body>

</html>
```

Index code:

```
{% extends 'base.html' %}

{% load static %}

{% block title %}Home - BlogSpot{% endblock %}

{% block content %}

<div class="container mt-5 carousel">

    <div id="carouselExampleCaptions" class="carousel slide">

        <div class="carousel-indicators">

            <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-
to="0" class="active"

                aria-current="true" aria-label="Slide 1"></button>

            <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-
to="1"

                aria-label="Slide 2"></button>

            <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-
to="2"

                aria-label="Slide 3"></button>

            <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-
to="3"

                aria-label="Slide 4"></button>
```

```
<button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-  
to="4"  
  
    aria-label="Slide 5"></button>  
  
</div>  
  
<div class="carousel-inner">  
  
    <div class="carousel-item active">  
  
          
  
        <div class="carousel-caption">  
  
            <h1>NATURE</h1>  
  
        </div>  
  
    </div>  
  
    <div class="carousel-item">  
  
          
  
        <div class="carousel-caption d-none d-md-block">  
  
            <h1>TRAVEL</h1>  
  
        </div>  
  
    </div>  
  
    <div class="carousel-item">  
  
          
  
        <div class="carousel-caption d-none d-md-block">  
  
            <h1>EDUCATION</h1>
```

```
</div>

</div>

<div class="carousel-item">

    <div class="carousel-caption d-none d-md-block">

        <h1>TECHNOLOGY</h1>

    </div>

</div>

<div class="carousel-item">

    <div class="carousel-caption d-none d-md-block">

        <h1>ARTIFICIAL INTELLIGENCE</h1>

    </div>

</div>

</div>

        <button    class="carousel-control-prev"    type="button"    data-bs-
target="#carouselExampleCaptions"

        data-bs-slide="prev">

        <span class="carousel-control-prev-icon" aria-hidden="true"></span>

        <span class="visually-hidden">Previous</span>

    </button>
```

```
        <button class="carousel-control-next" type="button" data-bs-
target="#carouselExampleCaptions"

        data-bs-slide="next">

        <span class="carousel-control-next-icon" aria-hidden="true"></span>

        <span class="visually-hidden">Next</span>

    </button>

</div>

</div>

<!-- Search Form -->

<div class="container mt-5">

    <form method="GET" class="mb-4">

        <div class="input-group">

            <input type="text" name="q" value="{{ query }}" class="form-control"
placeholder="Search posts...">

            <button type="submit" class="btn btn-primary">Search</button>

        </div>

    </form>

</div>

{% if user.is_authenticated %}

<div class="container mt-5">

    <h5>MY POSTS</h5>

    <hr>

    <div class="row row-cols-lg-3 row-cols-md-2 row-cols-1">
```

```
{% for post in posts %}

<div class="col col-lg-4 col-md-6 col-12 mb-2 blog-post">

    {% if post.image %}

    {% else %}

    {% endif %}

<div class="px-3 py-5 shadow">

    <a href="{% url 'post' post.id %}" class="text-decoration-none text-dark">

        <span class="text-white bg-info text-center rounded-3 mt-5" style="padding:
8px;">{{ post.category }}</span>

        <h5 class="mt-4">{{ post.postname }}</h5>

    </a>

    <div class="post-options">

        {% if user.is_authenticated %}

            {% if not post.is_liked %}

                <a href="{% url 'like_post' post.id %}" class="btn btn-outline-success btn-sm">

                    <i class="fa fa-heart text-danger"></i> Like ({{ post.like_set.count }})

                </a>

            {% else %}

                <span class="btn btn-success btn-sm disabled">
```

```
<i class="fa fa-heart text-danger"></i> Liked ({{ post.like_set.count }})

</span>

{% endif %}

{% if not post.is_bookmarked %}

<a href="{% url 'bookmark_post' post.id %}" class="btn btn-outline-info btn-
sm">

<i class="fa fa-bookmark"></i> Bookmark

</a>

{% else %}

<span class="btn btn-info btn-sm disabled">

<i class="fa fa-bookmark"></i> Bookmarked

</span>

{% endif %}

{% endif %}

</div>

<p>{{ post.content|truncatechars:100 }}...</p>

<p class="small text-primary">{{ post.created_at|date:"F d, Y" }}</p>

</div>

</div>

{% empty %}

<p>No posts yet.</p>

{% endfor %}
```



```
</div>

<!-- Pagination for My Posts -->

<nav>

  <ul class="pagination justify-content-center">

    {% if posts.has_previous %}

      <li class="page-item"><a class="page-link" href="?page={{
posts.previous_page_number }}{% if query %}&q={{ query }}{% endif %}">Previous</a></li>

    {% endif %}

    <li class="page-item disabled"><span class="page-link">Page {{ posts.number }}
of {{ posts.paginator.num_pages }}</span></li>

    {% if posts.has_next %}

      <li class="page-item"><a class="page-link" href="?page={{
posts.next_page_number }}{% if query %}&q={{ query }}{% endif %}">Next</a></li>

    {% endif %}

  </ul>

</nav>

  <a class="text-danger text-decoration-none" href="{% url 'profile' user.id %}"
style="cursor:pointer;">View All >></a>

</div>

{% endif %}

<section class="blog-posts">

  <div class="container">

    <h3 class="mb-2" style="color: rgb(227, 73, 73); font-size: 30px; font-weight:
bold;">RECENT POSTS</h3>
```

```
<hr>

<div class="row">

  <div class="col-lg-8">

    <div class="all-blog-posts">

      <div class="row">

        {% for post in page_obj %}

          <div class="col col-lg-6 col-12 pb-2 blog-post">

            {% if post.image %}

              {% else %}

              {% endif %}

            <div class="px-3 py-5 shadow">

              <a href="{% url 'post' post.id %}" class="text-decoration-none text-dark mb-
3">

                <span class="text-white bg-info text-center rounded-3 mt-5" style="padding:
8px;">{{ post.category }}</span>

                <h5 class="mt-4">{{ post.postname }}</h5>

              </a>

              <div class="post-options">

                {% if user.is_authenticated %}
```

```
{% if not post.is_liked %}

<a href="{% url 'like_post' post.id %}" class="btn btn-outline-success btn-
sm">

    <i class="fa fa-heart text-danger"></i> Like ({{ post.like_set.count }})

</a>

{% else %}

<span class="btn btn-success btn-sm disabled">

    <i class="fa fa-heart text-danger"></i> Liked ({{ post.like_set.count }})

</span>

{% endif %}

{% if not post.is_bookmarked %}

<a href="{% url 'bookmark_post' post.id %}" class="btn btn-outline-info
btn-sm">

    <i class="fa fa-bookmark"></i> Bookmark

</a>

{% else %}

<span class="btn btn-info btn-sm disabled">

    <i class="fa fa-bookmark"></i> Bookmarked

</span>

{% endif %}

{% endif %}

</div>

<p class="mt-2">{{ post.content|truncatechars:100 }}...</p>
```

```
<p class="small text-primary">{{ post.created_at|date:"F d, Y" }}</p>

</div>

</div>

{% empty %}

<p>No posts found.</p>

{% endfor %}

</div>

<!-- Pagination for Recent Posts -->

<nav>

<ul class="pagination justify-content-center">

    {% if page_obj.has_previous %}

        <li class="page-item"><a class="page-link" href="?page={{
page_obj.previous_page_number }}{% if query %}&q={{ query }}{% endif
%}">Previous</a></li>

        {% endif %}

        <li class="page-item disabled"><span class="page-link">Page {{
page_obj.number }} of {{ page_obj.paginator.num_pages }}</span></li>

        {% if page_obj.has_next %}

            <li class="page-item"><a class="page-link" href="?page={{
page_obj.next_page_number }}{% if query %}&q={{ query }}{% endif %}">Next</a></li>

            {% endif %}

        </ul>

    </nav>
```

```
</div>

</div>

<div class="col-lg-4">

  <div class="sidebar">

    <div class="row">

      <div class="col-lg-12">

        <div class="sidebar-item recent-posts">

          <div class="sidebar-heading">

            <h2 style="color: rgb(240, 124, 78); font-size: 30px; font-weight:
bold;">Popular Posts</h2>

          </div>

          <div class="content">

            <ul>

              {% for post in recent_posts %}

                <li>

                  {% if post.image %}

                  {% else %}

                  {% endif %}

                  <a href="{% url 'post' post.id %}" class="text-decoration-none text-dark">
```

Post Details code

IBMR College of Arts and Commerce, Hubballi Page 46

```
{% block content %}

<div class="container mt-5">

  <div class="row">

    <div class="col-lg-8">

      <div class="card shadow">

        {% if post.image %}

        {% else %}

        {% endif %}

        <div class="card-body">

          <h3 class="card-title" style="color: rgb(237, 63, 63);">{{ post.postname
}}</h3>

          <p class="card-text text-muted">By {{ post.user.username }} | {{
post.created_at|date:"F d, Y" }} | {{ post.category }}</p>

          <p class="card-text">{{ post.content|linebreaks }}</p>

          <div class="d-flex justify-content-between">

            <div>

              <a href="{% url 'like_post' post.id %}" class="btn btn-sm {% if is_liked
%}}btn-danger{% else %}}btn-outline-danger{% endif %}">

                <i class="fa fa-heart"></i> {{ post.like_set.count }}

              </a>

            </div>

          </div>

        </div>

      </div>

    </div>

  </div>

</div>

{% endblock %}
```

```
<a href="{% url 'bookmark_post' post.id %}" class="btn btn-sm {% if
is_bookmarked %}btn-primary{% else %}btn-outline-primary{% endif %}">
```

```
<i class="fa fa-bookmark"></i> {% if is_bookmarked
%}Bookmarked{% else %}Bookmark{% endif %}
```

```
</a>
```

```
</div>
```

```
<p class="text-muted">{{ total_comments }} Comment{{
total_comments|pluralize }}</p>
```

```
</div>
```

```
</div>
```

```
</div>
```

```
<h4 class="mt-5" style="color: rgb(237, 63, 63);">Comments</h4>
```

```
<hr>
```

```
{% for comment in comments %}
```

```
<div class="card mb-3 shadow">
```

```
<div class="card-body">
```

```
<p class="card-text"><strong>{{ comment.user.username }}</strong> | {{
comment.created_at|date:"F d, Y H:i" }}</p>
```

```
<p class="card-text">{{ comment.content|linebreaks }}</p>
```

```
<div class="d-flex">
```

```
<a href="{% url 'like_comment' comment.id %}" class="btn btn-sm {% if
comment.has_liked %}btn-success{% else %}btn-outline-success{% endif %} me-2">
```

```
<i class="fa fa-thumbs-up"></i> {{ comment.like_count }}
```



```
</a>

<a href="{% url 'dislike_comment' comment.id %}" class="btn btn-sm
{% if comment.has_disliked %}btn-danger{% else %}btn-outline-danger{% endif %} me-2">

    <i class="fa fa-thumbs-down"></i> {{ comment.dislike_count }}

</a>

{% if user == comment.user or user.is_staff %}

<a href="{% url 'deletecomment' comment.id %}" class="btn btn-sm btn-
outline-danger">Delete</a>

{% endif %}

</div>

</div>

</div>

{% empty %}

<p>No comments yet.</p>

{% endfor %}

{% if user.is_authenticated %}

<form action="{% url 'savecomment' post.id %}" method="post" class="mt-4">

    {% csrf_token %}

    <div class="mb-3">

        <label for="message" class="form-label">Add a Comment</label>

        <textarea class="form-control" id="message" name="message" rows="4"
required></textarea>

    </div>
```

```
<button type="submit" class="btn" style="background: rgb(236, 88, 88); color:
#fff; font-size: 20px; padding: 10px;">Submit Comment</button>
```

```
</form>
```

```
{% else %}
```

```
<p class="mt-4">Please <a href="{% url 'signin' %}">log in</a> to add a
comment.</p>
```

```
{% endif %}
```

```
</div>
```

```
<div class="col-lg-4">
```

```
<h4 style="color: rgb(237, 63, 63);">Recent Posts</h4>
```

```
<hr>
```

```
{% for recent_post in recent_posts %}
```

```
<div class="card mb-3 shadow">
```

```
<div class="card-body">
```

```
<h5 class="card-title">{{ recent_post.postname }}</h5>
```

```
<p class="card-text text-muted">By {{ recent_post.user.username }} | {{
recent_post.created_at|date:"F d, Y" }}</p>
```

```
<a href="{% url 'post' recent_post.id %}" class="btn btn-sm btn-outline-
danger">Read More</a>
```

```
</div>
```

```
</div>
```

```
{% endfor %}
```

</div>

</div>

</div>

<style>

.btn:hover {

background: transparent;

color: rgb(237, 63, 63);

border: 1px solid rgb(237, 63, 63);

}

a:hover {

color: rgb(237, 63, 63);

letter-spacing: 1px;

}

@media screen and (max-width: 600px) {

.card-img-top {

max-height: 200px;

}

}

</style>

{% endblock content %}

Contact code:

{% extends 'base.html' %}

```
{% load static %}

{% block content %}

<!-- Page Content -->

<!-- Banner Starts Here -->

<div class="heading-page header-text">

    <section class="page-heading">

        <div class="container">

            <div class="row">

                <div class="col-lg-12">

                    <div class="text-content">

                        <h4>contact us</h4>

                        <h2>let's stay in touch!</h2>

                    </div>

                </div>

            </div>

        </div>

    </section>

</div>

<!-- Banner Ends Here -->

{% if message %}

<script>

    alert('{{ message }}');
```

```
</script>

{% endif %}

<section class="contact-us">

<div class="container">

<div class="row">

<div class="col-lg-12">

<div class="down-contact">

<div class="row">

<div class="col-lg-8">

<div class="sidebar-item contact-form">

<div class="sidebar-heading">

<h2 class="h2">Send us a message</h2>

</div>

<div class="content">

<form id="contact" method="post" action="{% url 'contact_us' %}">

    {% csrf_token %}

    <div class="row">

        <div class="col-md-6 col-sm-12">

            <fieldset>

                <input name="name" type="text" id="name" placeholder="Name"
required>

            </fieldset>
```

```
</div>

<div class="col-md-6 col-sm-12">

  <fieldset>

    <input name="email" type="email" id="email" placeholder="Email"
required>

  </fieldset>

</div>

<div class="col-md-12 col-sm-12">

  <fieldset>

    <input name="subject" type="text" id="subject" placeholder="Subject">

  </fieldset>

</div>

<div class="col-lg-12">

  <fieldset>

    <textarea name="message" rows="6" id="message"
placeholder="Message"

    required=""></textarea>

  </fieldset>

</div>

<div class="col-lg-12">

  <fieldset>

    <center>

      <button type="submit" id="form-submit" class="main-button">Send
```

Message</button>

</center>

</fieldset>

</div>

</div>

</form>

</div>

{% for message in messages %}

<p class="text-danger">{{ message }}</p>

{% endfor %}

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</section>

{% endblock content %}

Signin Code:

{% extends 'base.html' %}

{% load static %}

```
{% block title %}Login - BlogSpot{% endblock %}

{% block content %}

<div class="container">

    <div class="d-flex justify-content-center align-items-center" style="min-height:
100vh;">

        <form action="{% url 'signin' %}" method="post" class="p-5 shadow" style="width:
50%;">

            {% csrf_token %}

            <h3 style="color: rgb(237, 63, 63); font-weight: 500; font-size: 30px; margin-
bottom: 20px;">LOGIN</h3>

            {% for message in messages %}

            <p class="text-danger text-center">{{ message }}</p>

            {% endfor %}

            <div class="mb-3">

                <label for="username" class="form-label">Username</label>

                <div class="input-group">

                    <span class="input-group-text">@</span>

                    <input type="text" class="form-control" id="username" name="username"
required>

                </div>

            </div>

            <div class="mb-3">

                <label for="password" class="form-label">Password</label>
```



```
<div class="input-group">

    <input type="password" class="form-control" id="password"
name="password" required>

</div>

</div>

<button type="submit" class="btn w-100 mt-4" style="background: rgb(236, 88,
88); color: #fff; font-size: 20px; padding: 10px;">Login</button>

<p class="text-center mt-3" style="font-size: 18px; color: #555;">Don't have an
account? <a href="{% url 'signup' %}" style="color: #386cbb; text-decoration: none;">Sign
up</a></p>

</form>

</div>

</div>

<style>

.btn:hover {

    background: transparent;

    color: rgb(237, 63, 63);

    border: 1px solid rgb(237, 63, 63);

}

a:hover {

    color: rgb(237, 63, 63);

    letter-spacing: 1px;
```

```
}

@media screen and (max-width: 600px) {

    form {

        width: 100%;

    }

}

</style>

{% endblock content %}
```

Signup code:

```
{% extends 'base.html' %}

{% load static %}

{% block title %}Signup - BlogSpot{% endblock %}

{% block content %}

<div class="container">

    <div class="d-flex justify-content-center align-items-center" style="min-height:
100vh;">

        <form action="{% url 'signup' %}" method="post" class="p-5 shadow"
style="width: 50%;" onsubmit="return validateForm()">

            {% csrf_token %}

            <h3 style="color: rgb(237, 63, 63); font-weight: 500; font-size: 30px; margin-
bottom: 20px;">SIGNUP</h3>

            {% for message in messages %}

                <p class="text-danger text-center">{{ message }}</p>
```

```
{% endfor %}

<div class="mb-3">

    <label for="username" class="form-label">Username</label>

    <div class="input-group">

        <span class="input-group-text">@</span>

        <input type="text" class="form-control" id="username" name="username"
required>

    </div>

</div>

<div class="mb-3">

    <label for="email" class="form-label">Email</label>

    <input type="email" class="form-control" id="email" name="email" required>

    <div id="emailError" class="text-danger" style="font-size: 14px;"></div>

</div>

<div class="mb-3">

    <label for="password" class="form-label">Password</label>

    <input type="password" class="form-control" id="password"
name="password" required>

    <div id="passwordError" class="text-danger" style="font-size: 14px;"></div>

</div>

<div class="mb-3">

    <label for="password2" class="form-label">Confirm Password</label>

    <input type="password" class="form-control" id="password2"
```

name="password2" required>

<div id="password2Error" class="text-danger" style="font-size: 14px;"></div>

</div>

<button type="submit" class="btn w-100 mt-4" style="background: rgb(236, 88, 88); color: #fff; font-size: 20px; padding: 10px;">Signup</button>

<p class="text-center mt-3" style="font-size: 18px; color: #555;">Have an account? Login</p>

</form>

</div>

</div>

<style>

.btn:hover {

background: transparent;

color: rgb(237, 63, 63);

border: 1px solid rgb(237, 63, 63);

}

a:hover {

color: rgb(237, 63, 63);

letter-spacing: 1px;

}

@media screen and (max-width: 600px) {

form {

```
        width: 100%;

    }

}

</style>


<script>

function validateForm() {

    let isValid = true;

    // Email validation (must be Gmail)

    const email = document.getElementById('email').value;

    const emailError = document.getElementById('emailError');

    if (!email.endsWith('@gmail.com')) {

        emailError.textContent = 'Please enter a valid Gmail address';

        isValid = false;

    } else {

        emailError.textContent = "";

    }

    // Password validation

    const password = document.getElementById('password').value;

    const passwordError = document.getElementById('passwordError');

    const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#$$%^&*])[A-Za-z\d!@#$$%^&*]{8,}$/;
```

```
        if (!passwordRegex.test(password)) {

            passwordError.textContent = 'Password must be at least 8 characters long and include
1 lowercase, 1 uppercase, and 1 special character';

            isValid = false;

        } else {

            passwordError.textContent = "";

        }

// Confirm password validation

const password2 = document.getElementById('password2').value;

const password2Error = document.getElementById('password2Error');

if (password !== password2) {

    password2Error.textContent = 'Passwords do not match';

    isValid = false;

} else {

    password2Error.textContent = "";

}

return isValid;

}

</script>

{% endblock content %}
```

7.TESTING AND RESULTS

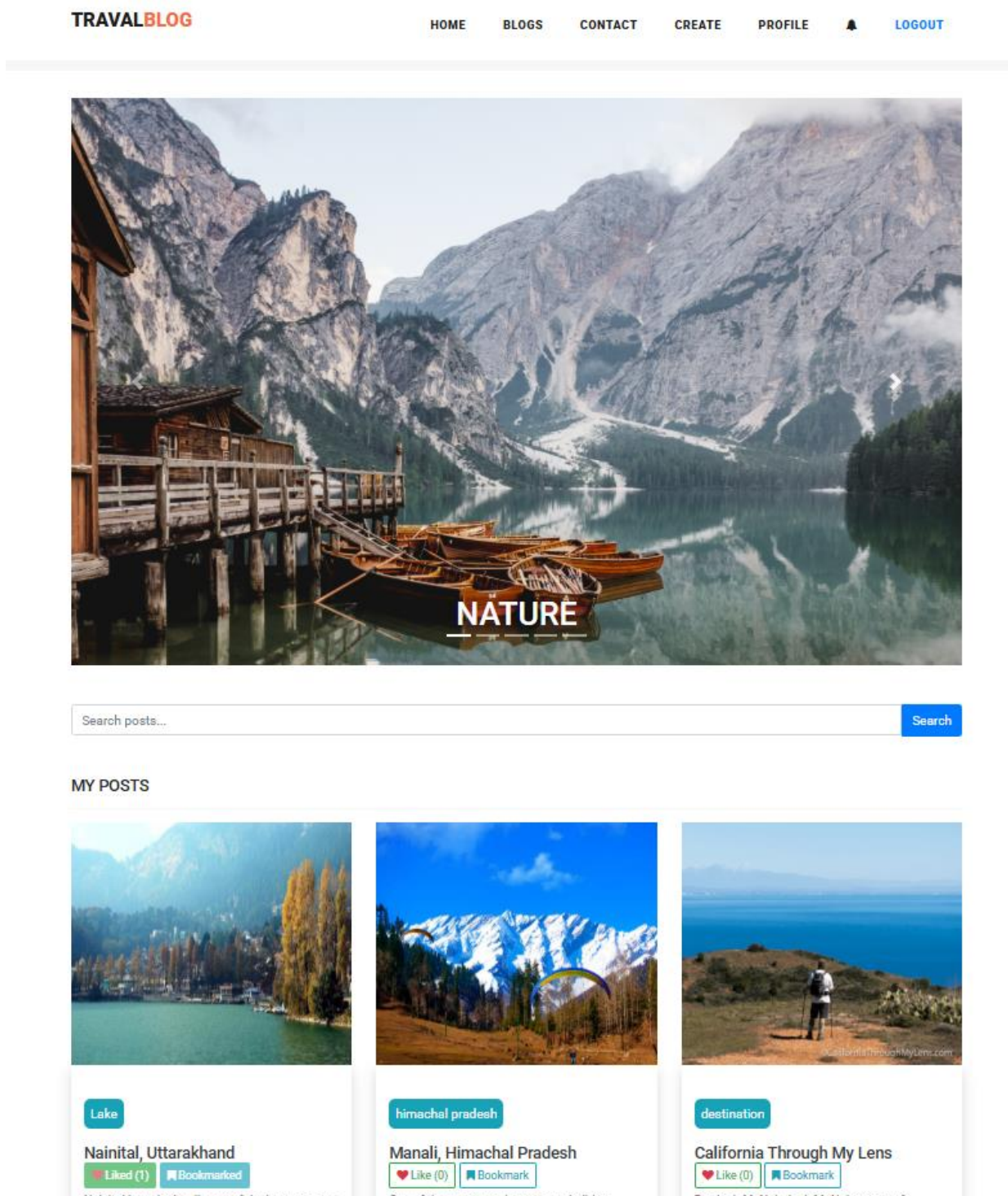
| Test ID | Description Case | Input | Expected Output | Result |
|---------|-------------------|--|---|-----------|
| TC01 | User Registration | Valid user details (username, email, password) | Account created, redirect to login page | Pass/Fail |
| TC02 | User Login | Valid credentials | Redirect to dashboard or home page | Pass/Fail |
| TC03 | User Login | Invalid credentials | Error message displayed | Pass/Fail |
| TC04 | Create Blog Post | Valid title and content by authenticated user | Post created and listed | Pass/Fail |
| TC05 | Edit Blog Post | Edited content by the post owner | Post updated with new content | Pass/Fail |
| TC06 | Delete Blog Post | Post deletion request by post owner | Post removed from list | Pass/Fail |
| TC07 | Post Pagination | Large number of posts | Paginated display (e.g., 10 per page) | Pass/Fail |
| TC08 | Search Posts | Keyword entered in search | Matching posts listed | Pass/Fail |
| TC09 | Add Comment | Comment submitted by logged-in user | Comment visible under the post | Pass/Fail |
| TC10 | Moderate Comment | Admin deletes or approves a comment | Comment updated based on action | Pass/Fail |
| TC11 | Like Post | User clicks like on a post | Like count increments | Pass/Fail |
| TC12 | Bookmark Post | User bookmarks a post | Post appears in bookmarks section | Pass/Fail |
| TC13 | Notification on | User receives comment on | Notification appears | Pass/Fail |

| Test ID | Description Case | Input | Expected Output | Result |
|---------|-------------------------------|-------------------------------------|--|-----------|
| | Comment | their post | in dashboard | |
| TC14 | Admin Access Control | Non-admin user accesses admin panel | Access denied or redirected | Pass/Fail |
| TC15 | Profile Update | User updates email or bio | Profile updated successfully | Pass/Fail |
| TC16 | SQL Injection Attempt | Malicious input in search field | Application handles securely, no injection | Pass/Fail |
| TC17 | XSS Injection Attempt | Script tag in comment | Script sanitized, not executed | Pass/Fail |
| TC18 | Logout Functionality | User clicks logout | Session ends, redirect to login page | Pass/Fail |
| TC19 | Unauthenticated Post Creation | Guest tries to create post | Redirected to login | Pass/Fail |
| TC20 | Mobile Responsiveness | Application opened on mobile | Responsive layout renders correctly | Pass/Fail |

8.SCREENSHOOT

This below presented pictures are the screenshots of the project that were captured during the running of the project

8.1 Home Page



8.2 All Blogs

TRAVALBLOG

HOME

BLOGS

CONTACT

CREATE

PROFILE



LOGOUT

ALL BLOGS

Search posts...

Search



Nature

Pesticide-Free Seeds Could Be Critical to Helping Native Bees

♥ Unlike (2)

🔖 Unbookmark

It's not easy being a bee. Modern honey bee colonies face the threat of Colony Collapse Disorder, a.....

May 23, 2025



Mountain

Nomadic Matt

♥ Like (0)

🔖 Unbookmark

I have to be honest with you — I love everything about Nomadic Matt. Everything from his websi.....

May 27, 2025

POPULAR POSTS



Nainital, Uttarakhand

May 28, 2025



8.3 Contact

TRAVALBLOG

[HOME](#) [BLOGS](#) [CONTACT](#) [CREATE](#) [PROFILE](#) [🔔](#) [LOGOUT](#)

CONTACT US

LET'S STAY IN TOUCH!

SEND US A MESSAGE

NAME

EMAIL

SUBJECT

MESSAGE

SEND MESSAGE

© COPYRIGHT 2023 | ALL RIGHTS RESERVED

8.4 Create Page

TRAVALBLOG

HOMEBLOGSCONTACTCREATEPROFILE🔔LOGOUT

Create Blog

Title

Content

Tag

Upload Image

Choose File

No file chosen

Create Blog

8.5 Admin login page

BLOGSPOT | ADMIN PANEL

You are authenticated as User, but are not authorized to access this page. Would you like to login to a different account?

Username:

Password:

Log in

8.6Admin Home Page

BLOGSPOT | ADMIN PANEL

WELCOME, ADMIN. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Authentication and Authorization > Users

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

MYAPP

Comments + Add

Contacts + Add

Posts + Add

Select user to change

Q

Search

Action:

Go

0 of 4 selected

| <input type="checkbox"/> | USERNAME | EMAIL ADDRESS | FIRST NAME | LAST NAME | STAFF STATUS |
|--------------------------|----------|---------------------------|------------|-----------|--------------|
| <input type="checkbox"/> | User | user@gmail.com | - | - | <div></div> |
| <input type="checkbox"/> | admin | admin@gmail.com | - | - | <div></div> |
| <input type="checkbox"/> | dee | d@gmail.com | Deepak | G | <div></div> |
| <input type="checkbox"/> | sheetal | sheetalsurve231@gmail.com | - | - | <div></div> |

4 users

ADD USER

FILTER

Show counts

↓ By staff status

All

Yes

No

↓ By superuser status

All

Yes

No

↓ By active

All

Yes

No

BLOGSPOT | ADMIN PANEL

WELCOME, ADMIN. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Myapp > Contacts > Contact object (1)

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#)

Users [+ Add](#)

MYAPP

Comments [+ Add](#)

Contacts [+ Add](#)

Posts [+ Add](#)

«

Change contact

CONTACT OBJECT (1) [HISTORY](#)

Name:

Email:

Subject:

Message:

hello welcome to my travel blog

SAVE

Save and add another

Save and continue editing

Delete

9.Conclusion

The Blog Application, as detailed in the workflow document, presents a well-structured and comprehensive design for a modern blogging platform. By focusing on essential features such as robust user management, versatile content creation, interactive commenting, and administrative control, it lays the groundwork for a highly engaging user experience. The inclusion of search, pagination, likes, and bookmarks further enhances usability and content discoverability. The clear definition of database schemas for each module indicates a solid foundation for development, ensuring data integrity and efficient management. Overall, the project aims to deliver a functional, scalable, and user-friendly blog application.

The Travel Blog developed using Django and Bootstrap is an effective and feature-rich web application designed to cater to travel enthusiasts who wish to share their experiences and gather information about various travel destinations. Leveraging Django's robust backend capabilities, the system efficiently manages dynamic content, including blog posts, user profiles, comments, and media files, ensuring data integrity and security. The built-in authentication and authorization system allows users to register, log in, and interact with the content in a secure environment.

On the frontend, Bootstrap's responsive framework guarantees that the blog is accessible and visually appealing across a wide range of devices, from desktops to smartphones, providing an excellent user experience. The use of Bootstrap components, such as navigation bars, cards, modals, and forms, enhances the overall interface design and usability.

Throughout the development process, emphasis was placed on creating a scalable and maintainable codebase, making future enhancements—like adding social sharing, search functionality, or multimedia support—straightforward. The Travel Blog project not only showcases the seamless integration of Django and Bootstrap but also highlights the importance of combining strong backend functionality with attractive, user-friendly design.

In conclusion, this Travel Blog platform provides a comprehensive solution for creating, managing, and sharing travel content online. It successfully meets the objectives of offering a reliable, secure, and engaging environment for users, and serves as a solid foundation for further expansion and customization based on evolving user need

10.Future Enhancement

Future Enhancements

1. Rich Text Editor: Implement a more advanced WYSIWYG editor for blog post content creation (e.g., Tiny MCE, Quill.js) to support rich formatting, image embedding, and media.
2. Categories and Tags: Allow users to categorize and tag blog posts for better organization and discoverability.
3. User Following/Followers: Enable users to follow other bloggers to receive updates on their new posts.
4. Drafting and Scheduling Posts: Allow users to save posts as drafts and schedule them for future publication.
5. Analytics Dashboard for Users: Provide individual authors with insights into their post performance (views, likes, comments).
6. SEO Optimization: Implement features like sitemaps, meta descriptions, and clean URLs to improve search engine visibility.
7. Social Media Integration: Allow users to easily share blog posts on various social media platforms.
8. Commenting Enhancements: Introduce nested comments (replies), comment editing, and reporting inappropriate comments.
9. Email Subscriptions: Allow users to subscribe to receive email notifications for new posts from specific authors or categories.
10. Monetization Features: (If applicable) Consider implementing features like paid subscriptions or advertising options.
11. Image/Media Management: A dedicated section for users to upload and manage their media files.
12. Two-Factor Authentication (2FA): Enhance security for user logins.

11.BIBLIOGRAPHY

Bibliography

- Blog Application - Project Workflow: "1. User Management (User Table)"
- Blog Application - Project Workflow: Table for User Management, "Field Name", "Data Type", "Description", "Example Data" rows.
- Blog Application - Project Workflow: Output section for "1. User Management".
- Blog Application - Project Workflow: "2. Blog Post Management (Blog Table)", "Tasks Performed" bullet point.
- Blog Application - Project Workflow: "2. Blog Post Management (Blog Table)", second "Tasks Performed" bullet point.
- Blog Application - Project Workflow: Table for Blog Post Management, "Field Name", "Data Type", "Description", "Example Data" row starting with "id".
- Blog Application - Project Workflow: Table for Blog Post Management, rows starting with "author_id", "title", "content", "created_at", "updated_at".
- Blog Application - Project Workflow: Output section for "2. Blog Post Management".
- Blog Application - Project Workflow: "3. Comment System (Comment Table)", "Tasks Performed" bullet points.
- Blog Application - Project Workflow: Table for Comment System, "Field Name", "Data Type", "Description", "Example Data" rows.
- Blog Application - Project Workflow: Output section for "3. Comment System", first bullet point.
- Blog Application - Project Workflow: Output section for "3. Comment System", second bullet point; "4. Admin Dashboard (Admin Panel)", "Tasks Performed" first bullet point.
- Blog Application - Project Workflow: "4. Admin Dashboard (Admin Panel)", "Tasks Performed" second bullet point.
- Blog Application - Project Workflow: Table for Admin Dashboard, "Field Name", "Data Type", "Description", "Example Data" rows.

- Blog Application - Project Workflow: Output section for "4. Admin Dashboard"; "5. Notifications (Notification Table)", "Tasks Performed" bullet point.
- Blog Application - Project Workflow: "5. Notifications (Notification Table)", "Table" heading.
- Blog Application - Project Workflow: Table for Notifications, "Field Name", "Data Type", "Description", "Example Data" rows.
- Blog Application - Project Workflow: Output section for "5. Notifications".
- Blog Application - Project Workflow: "6. Search & Pagination", "Tasks Performed" bullet points.
- Blog Application - Project Workflow: Output section for "6. Search & Pagination"; "7. Like System (Like Table)", "Tasks Performed" bullet point.
- Blog Application - Project Workflow: "7. Like System (Like Table)", "Table" heading.
- Blog Application - Project Workflow: Table for Like System, "Field Name", "Data Type", "Description", "Example Data" rows.
- Blog Application - Project Workflow: Output section for "7. Like System"; "8. Bookmark System (Bookmark Table)", "Tasks Performed" bullet point.
- Blog Application - Project Workflow: "8. Bookmark System (Bookmark Table)", "Table" heading.
- Blog Application - Project Workflow: Table for Bookmark System, "Field Name", "Data Type", "Description", "Example Data" rows for "id", "user_id".
- Blog Application - Project Workflow: Table for Bookmark System, rows for "blog_id", "created_at".
- Blog Application - Project Workflow: Output section for "8. Bookmark System"