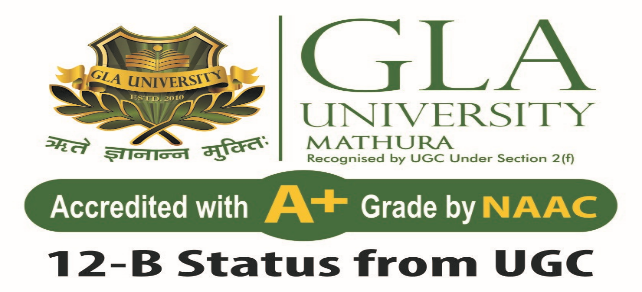
**MINI PROJECT REPORT**

**ON**

**(Journal Scribe – A Web Application)**

**SUBMITTED TO**



**IN PARTIAL FULFILLMENT OF THE REQUIREMENT**

**FOR THE AWARD OF THE DEGREE**

**“MASTERS OF COMPUTER APPLICATION”**

**Batch 2023-24**

**Session 2023-25**

|  |  |
| --- | --- |
| **Submitted by:**  **Shivam Gupta**  **Student Name**  **Roll No.: 2324000398** | **Supervised by:**  **Prof. Aditya Tiwari**  **Assistant Professor**  **CDOE, GLA University** |



### DECLARATION

*I Shivam Gupta, student of Masters of Computer Application (III Semester), Session 2023-2025, Batch 2023-2024 hereby declare that my work entitled “Journal Scribe – A Web Application”, is the outcome of genuine efforts done by me under the able guidance of Prof. Aditya Tiwari and being submitted to CDOE, GLA University, as mini project report in partial fulfillment for the award of the degree.*

Place: Noida

Date: 23/01/2025

**Name: Shivam Gupta**

Course: Masters of Computer Application University Roll No.: 2324000398



### CERTIFICATE

*This is to certify that the project entitled* ***“Journal Scribe – A Web Application”*** *carried out in Mini Project is a bonafide work done by* ***Shivam Gupta (2324000398)*** *is submitted in partial fulfillment of the requirements for the award of the degree Masters of Computer Application.*

**Signature of Supervisor:**

**Name of Supervisor:**

**Date:**

### ACKNOWLEDGEMENT

*It gives us a great sense of pleasure to present the report of the M.C.A. Mini Project undertaken during M.C.A. IInd Year.  This project in itself is an acknowledgement to the inspiration, drive and technical assistance contributed to it by many individuals. This project would never have seen the light of the day without the help and guidance that we have received.*

*We owe special debt of gratitude to* ***,*** *Assistant Professor CDOE for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us. He has showered us with all his extensively experienced ideas and insightful comments at virtually all stages of the project & has also taught us about the latest industry-oriented technologies.*

*We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind guidance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.*

***Name:*** *Shivam Gupta*

***Course:*** *Masters of Computer Application****University Roll No.:*** *2324000398*

# Abstract

**Purpose:** The "Journal Scribe" project aims to provide a simple yet powerful platform for preserving and managing personal memories digitally. In today’s digital era, where countless photos and moments are captured daily, organizing these into an accessible, private, and user-friendly system is often a challenge. Unlike existing platforms, which either lack privacy or are difficult to customize, "Journal Scribe" offers a dedicated solution tailored for memory management. Users can upload images, add detailed descriptions, and revisit memories seamlessly, ensuring a personalized and evolving archive.

**Design/Methodology/Approach:** Developed with Java, Spring Boot for the backend, and MySQL for database management, "Journal Scribe" incorporates modern tools for robust functionality. The frontend, designed with HTML and CSS, ensures a user-friendly experience across devices. REST APIs enable smooth handling of CRUD (Create, Read, Update, Delete) operations, with a modular architecture that supports scalability and maintenance.

**Findings:** The application is a fully functional memory management tool where users can upload, edit, delete, and view memories sorted by username. Its lightweight design ensures quick interactions, while its modularity offers room for enhancements like user authentication and cloud integration. The project is ideal for individuals seeking an organized platform to manage their digital memories efficiently.

**Research Limitations/Implications:** However, certain limitations exist. Features such as multi-user collaboration and extensive scalability testing are yet to be implemented. Despite this, the platform’s potential for personal and professional applications is immense. Event planners and small businesses can adapt it for managing portfolios or inventories.

**Practical Implications:** Socially, "Journal Scribe" promotes emotional well-being by fostering connection to cherished moments and encouraging privacy-first data practices. It exemplifies ethical technology use, aligning with the need for user-controlled platforms in today’s digital landscape.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| Declaration……………………………………………………………………… | ii |
| Certificate………………………………………………………………………. | iii |
| Acknowledgments……………………………………………………………… | iv |
| Abstract………………………………………………………………………… | v |
| Table of Contents………………………………………………………………. | vi |
| 1. **Introduction** | **1** |
| 1.1 Motivation…………………………………………………………….. | 1 |
| 1.2 Overview……………………………………………………………… | 1 |
| 1.3 Objective.……………………………………………………………... | 2 |
| 1. **Software Requirement Analysis** | **3** |
| 2.1 Problem Definition……………………………………………………. | 3 |
| * 1. Modules and their functionalities……………………….…………….. | 3 |
| * 1. Tools and Technologies……………………………………................. | 4 |
| 1. **Software Design** | **5** |
| 3.1 Data Flow Diagram (DFD)………..………………………………….. | 5 |
| 3.2 UML Design……………………………...…………………………... | 9 |
| 3.3 Database Design……………………………………………………… | 14 |
| 1. **Testing** | **18** |
| 4.1 Black Box Testing……………………………………………………. | 18 |
| 4.2 White Box Testing………………………..………………………….. | 20 |
| 4.3 Importance of Test Coverage………………………………………… | 22 |
| 1. **Implementation and User Interface** | **23** |
| 5.1 Input and Output Description ………………………………………… | 24 |
| 5.2 User Interfaces ……………………………………………………….. | 27 |
| 5.3 Implementation Highlights……………………………………………. | 28 |
| **6. References/Bibliography** | **29** |
| 6.1 Development Resources………………………………………………… | 29 |
| 6.2 Academic References…………………………………………………… | 30 |
| 6.3 Online Resources……………………………………………………….. | 30 |
| 6.4 Testing and Placeholder Resources…………………………………….. | 30 |
| **7. Appendices** | **31-35** |

**Chapter 1**

# Introduction

1.1 **Motivation**

In today’s digital landscape, people create countless memories through photos and videos captured on various devices. However, these moments often remain scattered across social media platforms, local storage, or external drives, making it difficult to organize and access them efficiently. Traditional methods like physical photo albums, though sentimental, are impractical in the modern age due to their limited capacity, vulnerability to damage, and lack of dynamic updating options. On the other hand, popular digital platforms often compromise user privacy and lack customization for personal memory management. These challenges highlight the need for a dedicated, user-friendly system that combines accessibility, privacy, and efficient organization. This realization motivated the creation of "Journal Scribe," a tailored solution aimed at bridging this gap. By offering a secure, intuitive, and centralized platform for managing personal memories, "Journal Scribe" addresses a critical need for users seeking to preserve and revisit their cherished moments.

1.2 **Overview**

"Journal Scribe" is a web application that empowers users to create, manage, and revisit their personal memories with ease. Designed with simplicity and user experience in mind, the platform allows individuals to upload images, add detailed descriptions, and assign dates to their memories. These functionalities provide users with a structured way to archive their moments while ensuring they retain complete control over their data. The application’s intuitive design enables seamless navigation and interaction, regardless of the user’s technical proficiency. Beyond basic functionalities, "Journal Scribe" includes options to edit or delete existing memories, ensuring that users can maintain an up-to-date and curated collection. Its responsive interface further enhances usability, making the application accessible across various devices and screen sizes. By focusing on user-centric design and functionality, "Journal Scribe" offers a reliable and efficient tool for digital memory management.

1.3 **Objective**

The overarching objective of "Journal Scribe" is to create a robust and intuitive platform that enhances the experience of digital memory management. To achieve this, the application leverages modern web development technologies, integrating a powerful backend with a visually appealing frontend. Core objectives include:

1. **Simplifying Memory Management**: Provide users with a straightforward process to create, view, edit, and delete memories.
2. **Ensuring Efficiency**: Optimize the application to handle CRUD (Create, Read, Update, Delete) operations seamlessly, minimizing delays and enhancing user satisfaction.
3. **Prioritizing Scalability**: Design the system architecture to support future enhancements, such as user authentication, multi-user collaboration, and cloud storage integration.
4. **Promoting Accessibility**: Develop an intuitive interface that caters to users of varying technical skill levels and ensures compatibility across devices.
5. **Maintaining Data Privacy**: Focus on creating a platform where users retain control over their data without concerns about privacy breaches or exploitation by third-party entities.

By meeting these objectives, "Journal Scribe" aspires to serve as a versatile and reliable solution for personal memory management. The project lays the groundwork for future expansions, such as incorporating advanced features and supporting larger datasets, ensuring its long-term relevance and adaptability to evolving user needs.

**Chapter 2**

# Software Requirements Analysis

**2.1 Problem Definition**

Modern users face significant challenges when it comes to organizing and managing their personal memories. With the proliferation of smartphones and digital cameras, individuals accumulate vast amounts of photos and videos. However, traditional methods of storing these memories, such as photo albums or social media platforms, fall short in several ways. Physical albums are cumbersome, prone to wear and tear, and do not allow for dynamic updates like editing or adding new content. On the other hand, social media platforms, while convenient, often raise concerns about privacy and data ownership. Users lack a secure, private, and user-friendly platform to archive and access their memories efficiently. This problem forms the foundation for the development of "Journal Scribe," which seeks to provide a streamlined and reliable solution.

**2.2 Modules and Functionalities**

"Journal Scribe" is designed with a modular architecture to ensure scalability, maintainability, and ease of use. The application consists of the following core modules:

* **User Module**: This module handles basic user inputs, such as the username and name. While the current version of "Journal Scribe" does not include authentication, the user module ensures that memories are categorized and retrieved based on the username.
* **Memory Management Module**: The heart of the application, this module allows users to create, read, update, and delete memories. Users can upload images from their local storage, add descriptive captions, and specify the date associated with each memory. The module also enables users to edit the content of their memories, including replacing images or modifying descriptions and dates. Additionally, users can delete memories that are no longer relevant.
* **Database Module**: This module ensures the secure storage of user data and metadata. The application uses MySQL to store information such as usernames, image paths, descriptions, and dates. The database schema is designed to maintain data integrity and allow for efficient querying.
* **Frontend Module**: The frontend module is responsible for providing a user-friendly interface. Developed using HTML and CSS, the interface is intuitive and responsive, catering to users with varying levels of technical expertise. The frontend communicates with the backend via REST APIs to perform CRUD operations seamlessly

**2.3 Tools and Technologies**

The development of "Journal Scribe" leverages modern tools and technologies to ensure robustness, scalability, and a pleasant user experience:

* **Backend**: The backend is built using Java and the Spring Boot framework. Spring Boot simplifies the development of REST APIs, allowing for efficient handling of CRUD operations and seamless integration with the database.
* **Frontend**: The frontend is designed using HTML and CSS. These technologies ensure that the application is visually appealing and accessible across different devices and screen sizes.
* **Database**: MySQL is used as the database management system. It stores user data and metadata in a structured format, allowing for quick retrieval and manipulation of information.
* **Project Management**: Maven is used as the build automation tool, managing dependencies and ensuring a smooth development workflow.

**Chapter 3**

# Software Design

In this chapter, we will provide a detailed explanation of the software design for the "Journal Scribe" application. This design will be broken down into several key components, including Data Flow Diagrams (DFD), UML diagrams, database design, and table schemas. The goal of this chapter is to outline the design decisions made to ensure that the application meets its functional and non-functional requirements while remaining scalable, maintainable, and easy to use.

**3.1 Data Flow Diagram (DFD)**

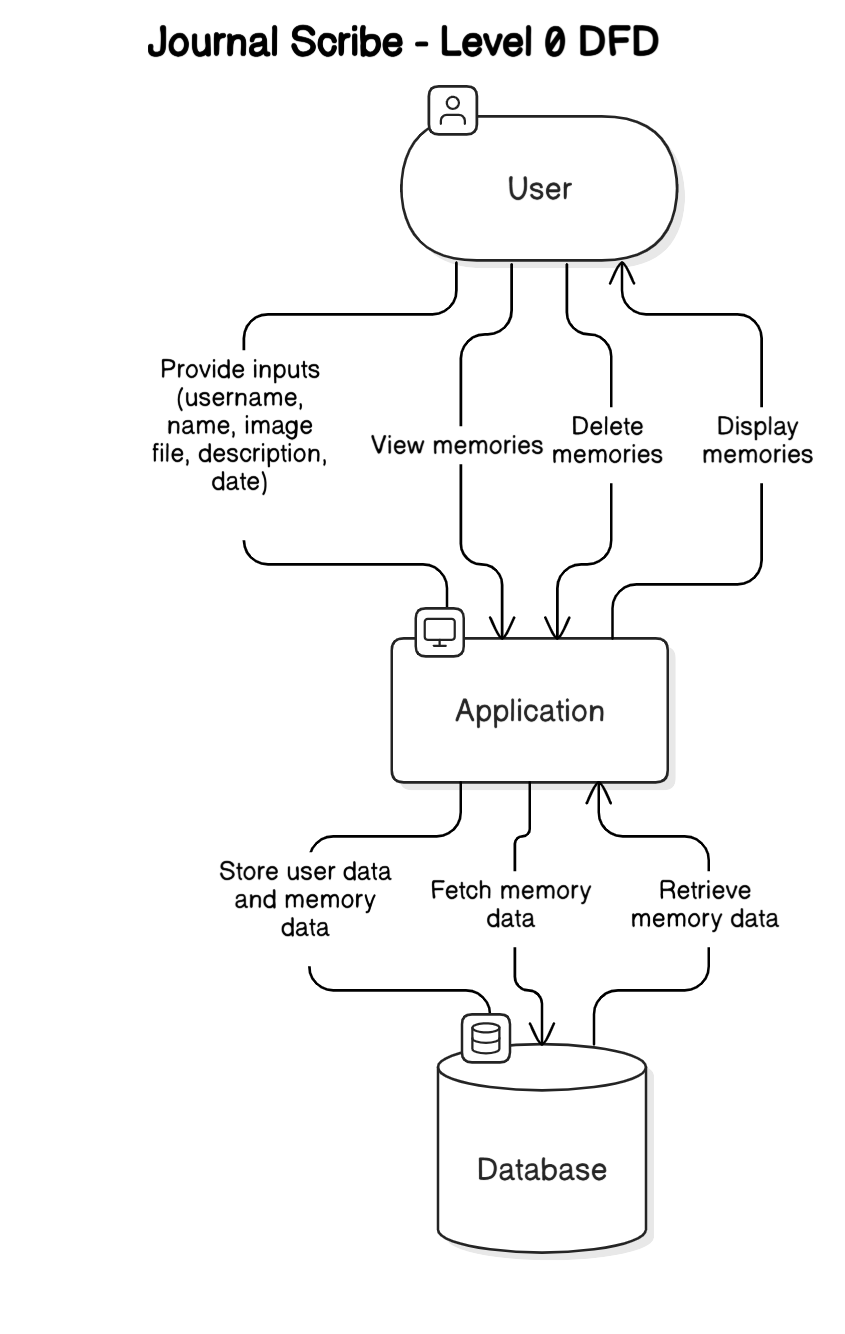
The Data Flow Diagram (DFD) is a visual representation of how data moves within the system. It illustrates the flow of data between external entities, processes, data stores, and outputs. For "Journal Scribe," we created multiple levels of DFDs to provide both high-level and detailed views of the system's functionality.

**3.1.1 Level 0: High-Level DFD**

At this level, we will provide a very high-level view of the system. The diagram shows the interaction between external users (the user who interacts with the system), the application, and the underlying database. The main goal here is to show how the user interacts with the system and what high-level data exchanges occur.

* **User**: The primary external entity. The user interacts with the system by providing inputs such as username, name, image file, description, and date when creating or editing memories. The user is also responsible for viewing and deleting memories.
* **Application**: The system where the functionality of memory management is implemented. The user interacts with the application to create, read, update, and delete memories. The application processes the input provided by the user and sends necessary data to the database for storage.
* **Database**: The storage system, which in this case, is MySQL. The database stores all the data related to users and their memories. It serves as the central repository of information, maintaining details such as usernames, memory images, descriptions, and dates.

The data flow diagram (Level 0) highlights a simple interaction model where the user sends data inputs (user data and memory data) to the application, which in turn stores it in the database. The application fetches the necessary data when the user wants to view their memories.

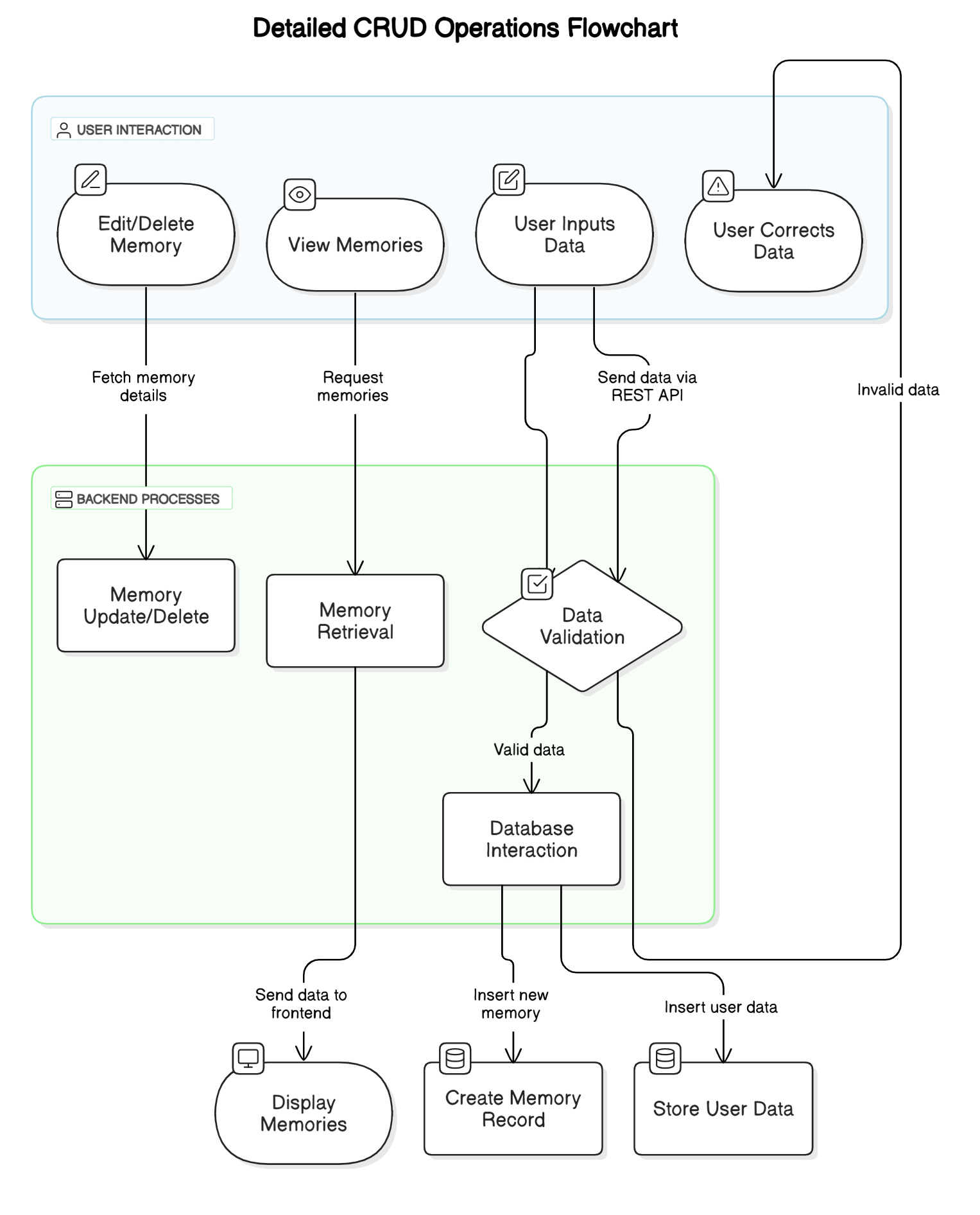
****

**3.1.2 Level 1: Detailed DFD**

The detailed DFD provides more granular insight into how the system handles user requests and data processing. This diagram is broken into several processes, focusing on CRUD (Create, Read, Update, Delete) operations.

* **Process 1: User Inputs Data**: The first process involves the user interacting with the UI. The user provides data (such as the memory image, description, date, and username). This data is sent to the application via REST API calls.
* **Process 2: Data Validation**: Once the data reaches the application, the system first validates the input. This includes ensuring that the memory image is in an acceptable format (e.g., JPEG or PNG), the description is not empty, and the date is valid. If any data is invalid, the system prompts the user to correct it.
* **Process 3: Database Interaction**: After the data is validated, it is sent to the database for storage. For the creation of a memory, a new record is inserted into the "Memory" table. The user’s data is also stored in the "User" table. When the user retrieves their memories, the system queries the database to retrieve relevant records based on the user’s username.
* **Process 4: Memory Retrieval**: When a user wants to view their memories, the system sends a query to the database to retrieve memories that belong to the specified username. The memory details, including the image path, description, and date, are then sent back to the frontend for display.
* **Process 5: Memory Update/Delete**: If the user wishes to edit or delete an existing memory, the system fetches the selected memory’s details, allows the user to modify them (image, description, date), and then stores the updated data back in the database. In the case of deletion, the memory is removed from the database.

These steps are shown in the Level 1 DFD and help ensure the system works efficiently and performs all necessary operations accurately.

****

### 3.2 UML Diagrams

Unified Modeling Language (UML) diagrams are essential tools for representing the structure and behavior of the system. For the "Journal Scribe" project, we use several types of UML diagrams, including the class diagram and use case diagram, to represent the application's architecture and user interactions.

#### 3.2.1 Class Diagram

A class diagram provides a blueprint of the application's structure. It shows the various classes and their relationships. The key classes for the "Journal Scribe" application are:

* **User**: The User class represents each user who interacts with the system. It contains attributes like username (primary key), name, and other potential attributes that could be added in future versions, such as email or password for authentication. The class also has methods to create and retrieve user information.
* **Memory**: The Memory class represents a memory record. It contains attributes like id (primary key), username (foreign key), imagePath, description, and date. The Memory class handles the creation, modification, and deletion of memories and also has methods for retrieving individual memory details based on user input.
* **Database**: The Database class handles all interactions with the MySQL database. It contains methods for inserting, updating, deleting, and retrieving user and memory data. This class uses JDBC (Java Database Connectivity) to communicate with the MySQL database.

In the class diagram, the relationship between these classes is depicted with lines, showing that a User can have multiple memories (one-to-many relationship), and each memory belongs to a specific user.

****

#### 3.2.2 Use Case Diagram

The use case diagram for "Journal Scribe" illustrates how users interact with the system. It provides a high-level view of the primary functionalities and identifies the key actors and their roles. Below are the components and their detailed descriptions:

#### ****Actors:****

1. **User**:  
   The primary actor who interacts with the "Journal Scribe" system. This could be any individual who wants to upload, view, edit, or delete their memories.
2. **System**:  
   Represents the backend of "Journal Scribe," which processes user requests, communicates with the database, and provides the desired output.

#### ****Use Cases:****

1. **Create Memory**:
   * **Description**: Allows the user to add a new memory. This includes uploading a photo, providing a description, and assigning a date to the memory.
   * **Steps**:
     1. User navigates to the "Add Memory" page.
     2. Uploads an image from local storage.
     3. Fills in the description and selects a date.
     4. Submits the form.
   * **System Behavior**: Saves the data in the database and displays it on the memory list page.
2. **View Memory**:
   * **Description**: Enables users to browse through their existing memories. Each memory is displayed with a thumbnail, description, and date.
   * **Steps**:
     1. User navigates to the "View Memories" page.
     2. System retrieves the memories associated with the user from the database.
     3. Displays the memories in a user-friendly format.
3. **Edit Memory**:
   * **Description**: Provides the option to modify an existing memory's details, such as the image, description, or date.
   * **Steps**:
     1. User selects a memory from the list.
     2. Opens the "Edit Memory" page.
     3. Updates the required fields.
     4. Saves the changes.
   * **System Behavior**: Updates the relevant record in the database and refreshes the UI to reflect the changes.
4. **Delete Memory**:
   * **Description**: Allows users to remove a memory permanently.
   * **Steps**:
     1. User selects a memory.
     2. Clicks the "Delete" button.
     3. Confirms the deletion.
   * **System Behavior**: Deletes the record from the database and removes it from the display.
5. **Login/Logout (Future Implementation)**:
   * **Description**: Intended to secure the system by authenticating users before they can interact with memories.
   * **Steps**:
     1. User enters their credentials on the login page.
     2. The system verifies the credentials.
     3. Grants access upon successful authentication.
   * **System Behavior**: Maintains a session for the authenticated user and restricts access for unauthorized users.

#### ****Connections in the Diagram:****

* **User → Create Memory**: Represents the action of adding a new memory.
* **User → View Memory**: Highlights the ability to browse through stored memories.
* **User → Edit Memory**: Reflects the interaction for modifying memory details.
* **User → Delete Memory**: Indicates the user’s capability to remove a memory.
* **User → Login/Logout**: Demonstrates potential future authentication mechanisms.

### 3.3 Database Design

The database is a critical component of the "Journal Scribe" application. It stores all the data related to users and their memories. The application uses MySQL as the relational database management system, and the database design focuses on simplicity and efficiency.

#### 3.3.1 Entity-Relationship (ER) Diagram

#### This diagram represents the entities in your database and the relationships between them. In this case, the memory table is the only entity, and it has the following attributes:

#### memory

#### id (Primary Key)

#### cover\_image\_loc

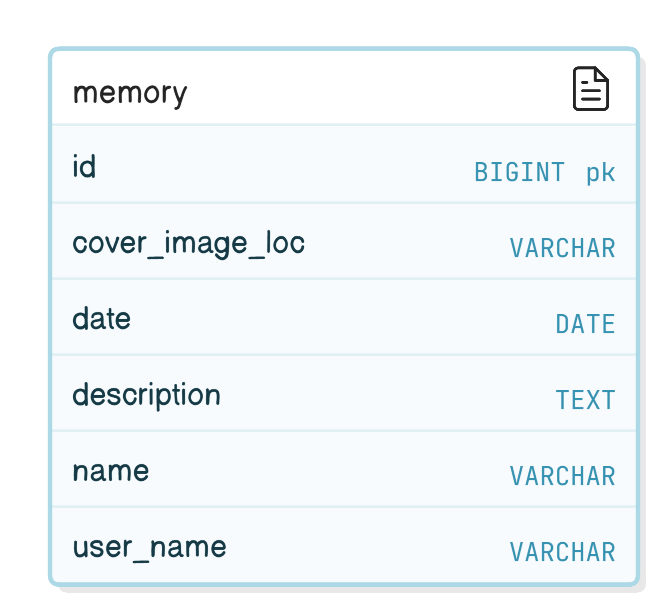
#### date

#### description

#### name

#### user\_name

#### There is no direct relationship between multiple entities (like foreign keys).



#### 3.3.2 Table Schemas

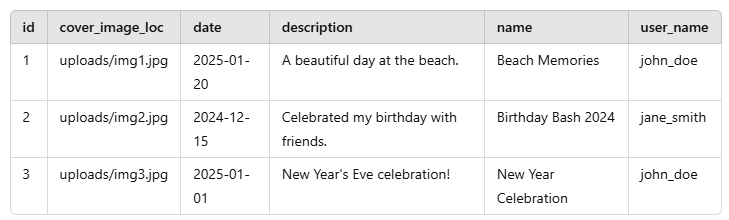
The memory table is the central part of the database for the "Journal Scribe" project. It stores details about each memory post created by users. Here’s a detailed breakdown of each column:

### ****Columns and Their Purpose****

1. **id BIGINT NOT NULL AUTO\_INCREMENT**
   * **Purpose**: This serves as the unique identifier for each memory entry in the table.
   * **Details**:
     + BIGINT: Chosen to accommodate a large number of entries without running out of unique identifiers.
     + AUTO\_INCREMENT: Automatically generates a new unique ID for each memory entry when a new record is inserted.
     + **Example**: The first memory entry might have id = 1, the second id = 2, and so on.
2. **cover\_image\_loc VARCHAR(255)**
   * **Purpose**: Stores the file path or URL of the cover image associated with the memory post.
   * **Details**:
     + VARCHAR(255): Allows storing up to 255 characters, sufficient for most file paths or URLs.
     + This column enables users to visually represent their memory entries with images.
     + **Example**: "uploads/images/memory1.jpg" or "https://example.com/images/memory1.jpg".
3. **date DATE**
   * **Purpose**: Records the date associated with the memory entry, typically when the memory or event occurred.
   * **Details**:
     + DATE: Stores only the date component (year, month, day) without a time.
     + This helps users organize and filter their memories chronologically.
     + **Example**: "2025-01-22".
4. **description TEXT**
   * **Purpose**: Holds the main content or description of the memory entry.
   * **Details**:
     + TEXT: Allows storing large amounts of text, suitable for detailed descriptions.
     + Useful for users to write narratives, reflections, or detailed explanations about their memories.
     + **Example**: "This was the most amazing trip to the mountains. The weather was perfect, and I enjoyed every moment."
5. **name VARCHAR(255)**
   * **Purpose**: Stores the title or name of the memory entry.
   * **Details**:
     + VARCHAR(255): Allows storing up to 255 characters for concise yet descriptive titles.
     + Helps users quickly identify memory entries.
     + **Example**: "Trip to the Himalayas" or "Birthday Celebration 2024".
6. **user\_name VARCHAR(255)**
   * **Purpose**: Identifies the username of the person who created the memory.
   * **Details**:
     + VARCHAR(255): Stores the username, linking the memory to its creator.
     + Useful for associating multiple memories with a single user, especially in multi-user scenarios.
     + **Example**: "john\_doe".
7. **PRIMARY KEY (id)**
   * **Purpose**: Defines the id column as the primary key for the table.
   * **Details**:
     + Ensures that each memory entry has a unique identifier.
     + Helps in efficiently retrieving and managing memory records.

### ****Example Data****

Here’s an example of how data might look in the memory table:



### ****Usage in the Application****

1. **Creating a Memory**: When a user creates a memory, a new row is added to this table with the memory details, image location, and associated date.
2. **Editing a Memory**: Updates can be made to the description, name, date, or cover\_image\_loc.
3. **Viewing Memories**: Memories can be queried and displayed in a chronological order or filtered by user\_name.
4. **Deleting a Memory**: Deleting a row from the table removes the associated memory entry.

### ****Advantages of the Schema****

* **Simplicity**: The schema is straightforward, making it easy to manage memory data.
* **Flexibility**: The description field allows large amounts of text, and the cover\_image\_loc supports dynamic image links.
* **Scalability**: The id column as a BIGINT supports a large number of records, and indexed columns like id and user\_name enable efficient queries.

**Chapter 4**

# Testing

Testing is a crucial part of software development, ensuring that the application functions as intended, meets user expectations, and handles edge cases effectively. The "Journal Scribe" project was tested using both black box testing and white box testing methodologies to validate its functionality and internal code structure.

### ****4.1 Black Box Testing****

Black box testing evaluates the application's functionality without considering the internal code or logic. The focus is on the input-output behavior, ensuring that the system performs as expected when given specific inputs.

#### ****Test Case 1: Creating a New Memory****

* **Objective**: To verify that users can successfully create a new memory in the application.
* **Inputs**:
  + cover\_image\_loc: uploads/img1.jpg
  + date: 2025-01-20
  + description: A beautiful day at the beach.
  + name: Beach Memories
  + user\_name: john\_doe
* **Expected Output**:
  + The memory is stored in the database with the specified details.
  + The system displays the success message: "Memory created successfully."
* **Steps to Execute**:
  + Navigate to the "Create Memory" page in the application.
  + Fill out the form with the inputs mentioned above.
  + Submit the form and verify the confirmation message.
  + Query the database to confirm that the new entry exists.
* **Result**: The memory is created successfully, and all details are stored correctly.

#### ****Test Case 2: Editing an Existing Memory****

* **Objective**: To verify that users can update details of an existing memory.
* **Inputs**:
  + Memory ID: 1
  + Updated description: A relaxing day at the sunny beach.
  + Updated date: 2025-01-21
* **Expected Output**:
  + The memory's details are updated in the database.
  + The success message "Memory updated successfully." is displayed.
* **Steps to Execute**:
  + Locate the memory with ID 1 in the application.
  + Edit the description and date fields with the updated values.
  + Save the changes and observe the confirmation message.
  + Verify the database to confirm the updated details.
* **Result**: The memory is updated successfully, reflecting the new description and date.

#### ****Test Case 3: Viewing Memories by Username****

* **Objective**: To verify that users can filter and view memories associated with their username.
* **Inputs**:
  + Username: john\_doe
* **Expected Output**:
  + A list of memories belonging to john\_doe is displayed.
  + Each memory includes the correct description, date, and image.
* **Steps to Execute**:
  + Navigate to the "View Memories" page.
  + Enter the username john\_doe in the filter input.
  + Submit the filter query and observe the results.
  + Validate that the displayed memories match the data in the database.
* **Result**: The filtered list is displayed correctly, showing all memories for john\_doe.

### ****4.2 White Box Testing****

White box testing involves analyzing the internal structure and logic of the code. It ensures that the application's internal operations, such as algorithms, validations, and error handling, function correctly under various conditions.

#### ****Test Case 1: Validation of Input Fields****

* **Objective**: To ensure that all input fields are validated correctly before processing.
* **Scenario**:
  + Input:
    - cover\_image\_loc: Empty
    - description: Empty
    - date: Invalid format (20-01-2025)
  + Code Execution Path: Validation methods in the controller class.
* **Expected Output**:
  + The system should return validation errors:
    - "Image location cannot be empty."
    - "Description cannot be empty."
    - "Date must be in YYYY-MM-DD format."
* **Steps to Execute**:
  + Submit a form with the invalid inputs mentioned above.
  + Observe the error messages displayed by the system.
  + Check that no entry is created in the database.
* **Result**: The system successfully identifies and reports invalid inputs, ensuring data integrity.

#### ****Test Case 2: Database Query for Viewing Memories****

* **Objective**: To ensure that the SQL query retrieves data accurately and efficiently.
* **Scenario**:
  + Code Execution Path: Query execution in the service layer for SELECT \* FROM memory WHERE user\_name = ?.
  + Input: user\_name = 'john\_doe'
* **Expected Output**:
  + The query returns all rows from the memory table where the user\_name matches john\_doe.
  + Performance is optimal, even with a large number of records.
* **Steps to Execute**:
  + Run the application and request memories for john\_doe.
  + Log the query execution time and ensure it completes without errors.
  + Cross-check the output with the database.
* **Result**: The query executes efficiently, and the results match the expected data.

#### ****Test Case 3: Error Handling for Nonexistent Memory Deletion****

* **Objective**: To verify that the application handles errors gracefully when attempting to delete a nonexistent memory.
* **Scenario**:
  + Input: Memory ID = 9999 (nonexistent ID)
  + Code Execution Path: Deletion logic in the backend service.
* **Expected Output**:
  + The service returns an error response: "Memory not found."
  + No unintended side effects occur in the database.
* **Steps to Execute**:
  + Attempt to delete a memory with ID 9999.
  + Observe the system's error response.
  + Verify that no changes are made to other database entries.
* **Result**: The application responds with an appropriate error message, maintaining system stability.

### ****4.3 Importance of Test Coverage****

The combination of black box and white box testing ensures that the "Journal Scribe" application is reliable, user-friendly, and free from major defects. Black box testing validates the application from an end-user perspective, ensuring that all features work as intended. White box testing delves deeper into the code structure, identifying potential logic errors, inefficiencies, and vulnerabilities.

By incorporating these tests, the development team ensures:

* **Functionality**: The **functionality** of the "Journal Scribe" project ensures all features meet user requirements. Users can create, edit, delete, and retrieve memories, with thorough testing confirming that data is stored, updated, and displayed accurately without errors.
* **Robustness**: The system’s **robustness** ensures it handles invalid inputs and errors gracefully. Scenarios like missing data, unsupported file uploads, or invalid dates are managed with user-friendly error messages, preventing crashes or data corruption.
* **Efficiency**: For **efficiency,** database queries and algorithms were optimized to handle operations swiftly, even with growing data. Efficient indexing and query design minimize response times for retrieving or updating records.
* **Scalability**: The application demonstrates **scalability,** designed to manage larger datasets and increased user activity without performance degradation. The architecture supports horizontal and vertical scaling, ensuring that the system remains responsive as the number of users and stored memories grows. Together, these elements ensure the application is reliable and future-proof.

The rigorous testing process demonstrates a commitment to quality and reliability, contributing to the overall success of the project.

**Chapter 5**

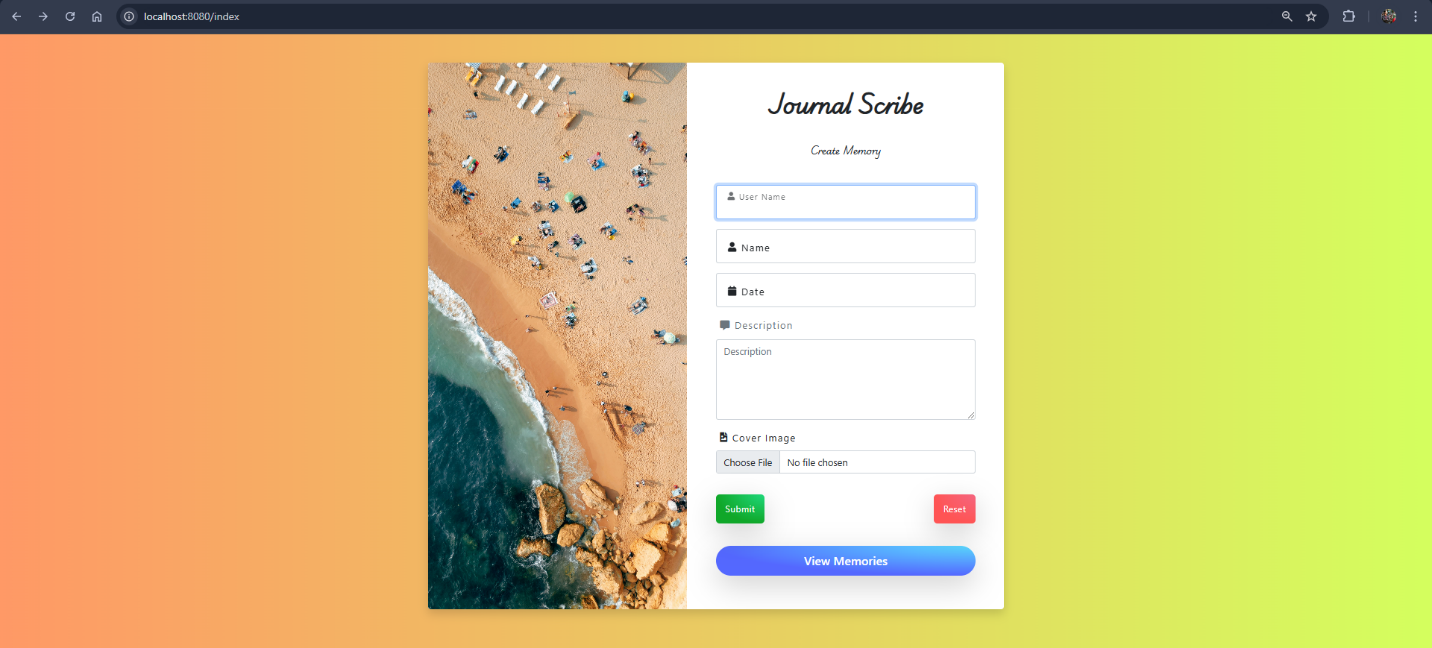
# Implementation and User Interface

The "Journal Scribe" application was implemented by seamlessly integrating robust backend functionalities with an intuitive user interface to deliver an efficient and user-friendly system. This chapter delves into the specifics of the features implemented, the architectural logic, and the detailed design of the user interfaces, explaining the input and output mechanisms in depth.

Backend development utilized Java and Spring Boot frameworks to establish a reliable and scalable server-side architecture. RESTful APIs were designed for smooth and secure interactions between the frontend and the server. Data persistence was handled through a MySQL database, structured to effectively manage user memories by storing crucial details like usernames, image paths, descriptions, and timestamps. For the frontend, HTML and CSS were employed, prioritizing simplicity and accessibility to ensure a positive user experience.

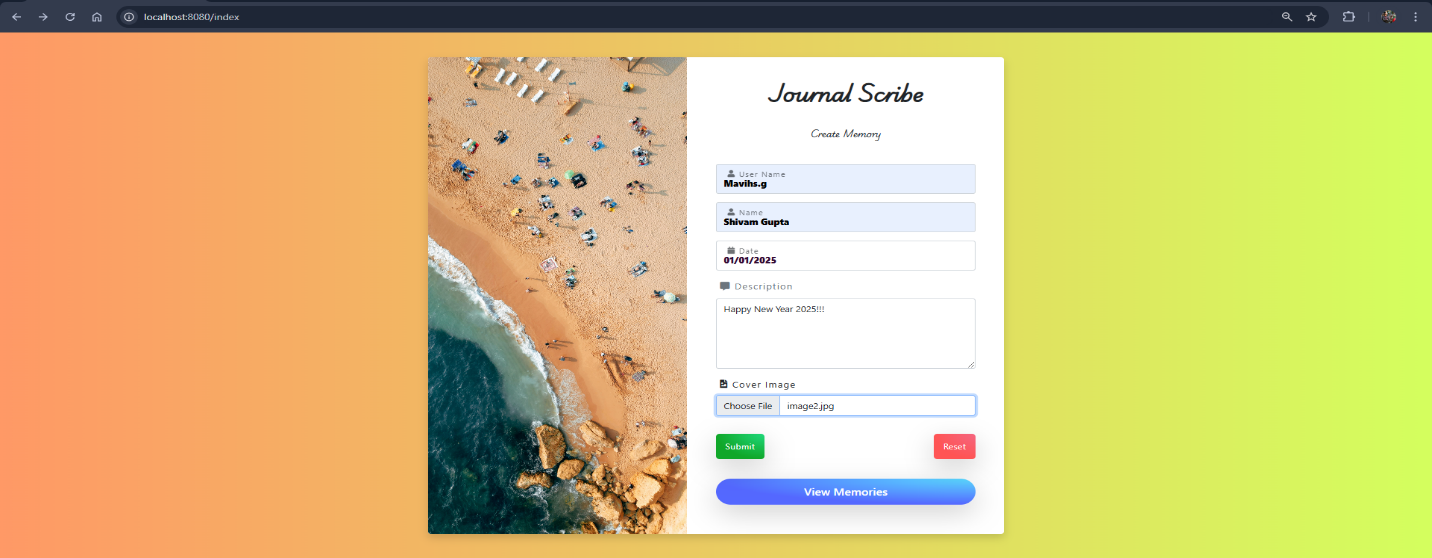
The implementation highlights the collaboration between backend logic and frontend usability, forming the backbone of the "Journal Scribe" application. This cohesive system provides a solid foundation for user interaction, making it both practical and enjoyable to use.

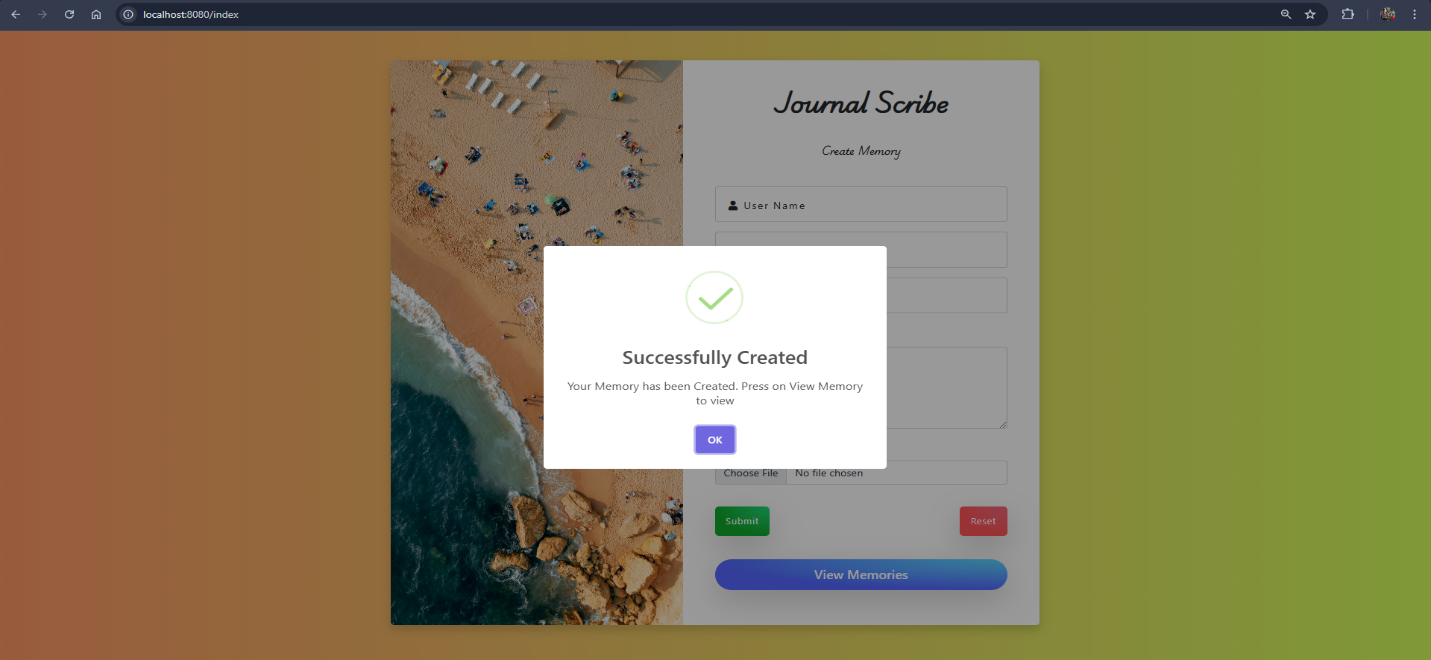
Below are the Output Screens of Interface:



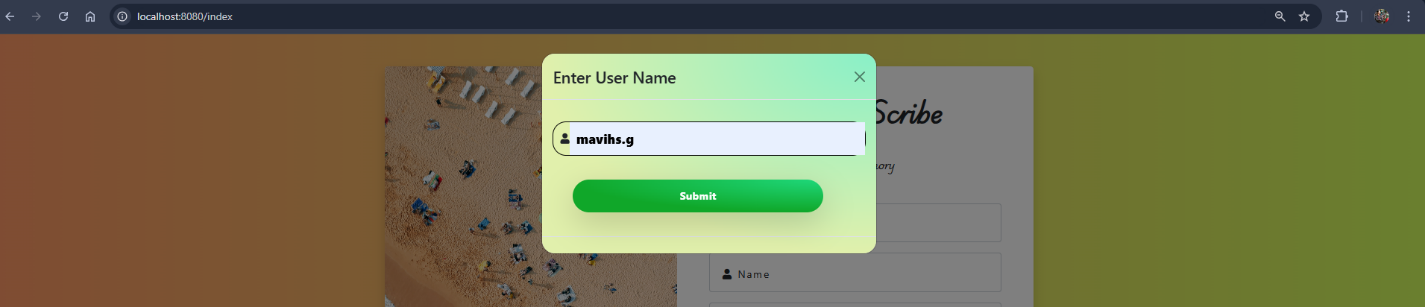
#### 5.1 Input and Output Description

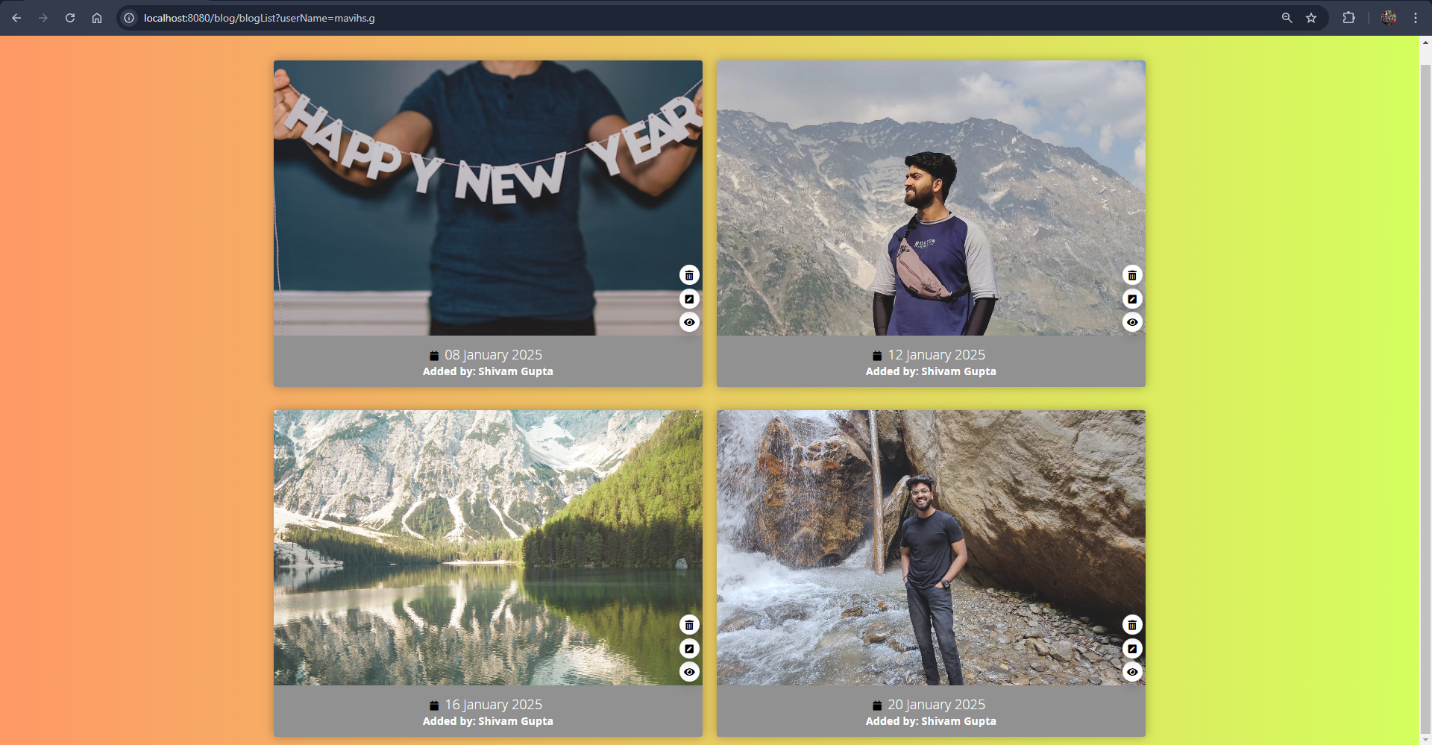
1. **Add New Memory**
   * **Input:** Users are presented with a form to input details such as their username, name of the memory, description, date, and an image file.
   * **Processing:** Upon submission, the data is validated in the backend. If valid, the memory is stored in the database, and the image file is uploaded to the server.
   * **Output:** A confirmation message is displayed, and the new memory appears in the user’s list of memories.



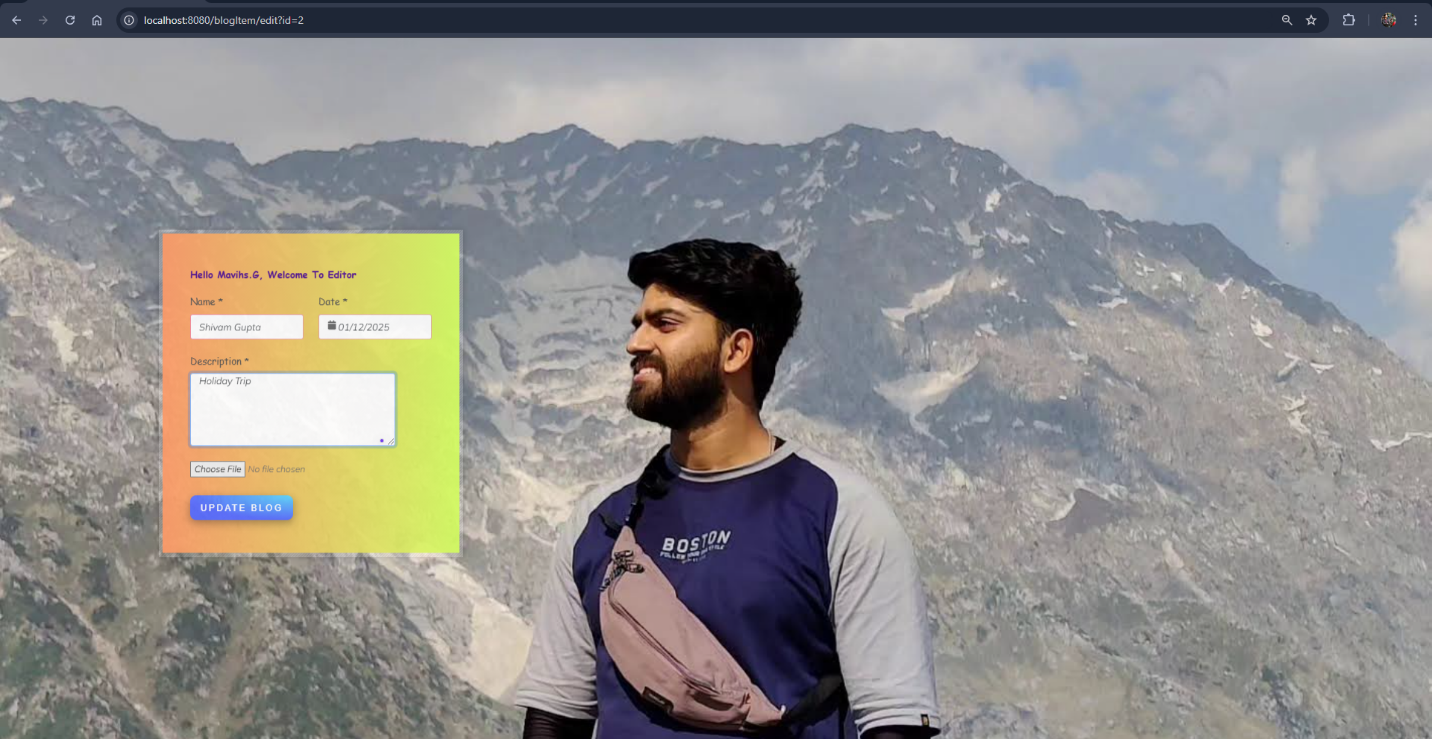


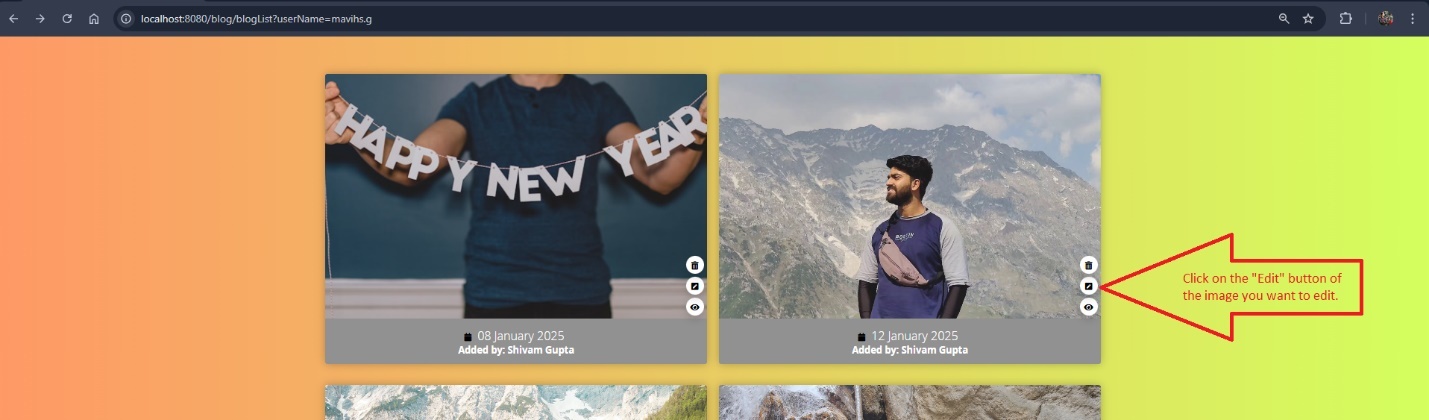
1. **View Memories by Username**
   * **Input:** Users enter their username or select it from a dropdown list.
   * **Processing:** The system retrieves all memories associated with the username from the database.
   * **Output:** A gallery-style interface displays the memories, including their images, descriptions, and dates.



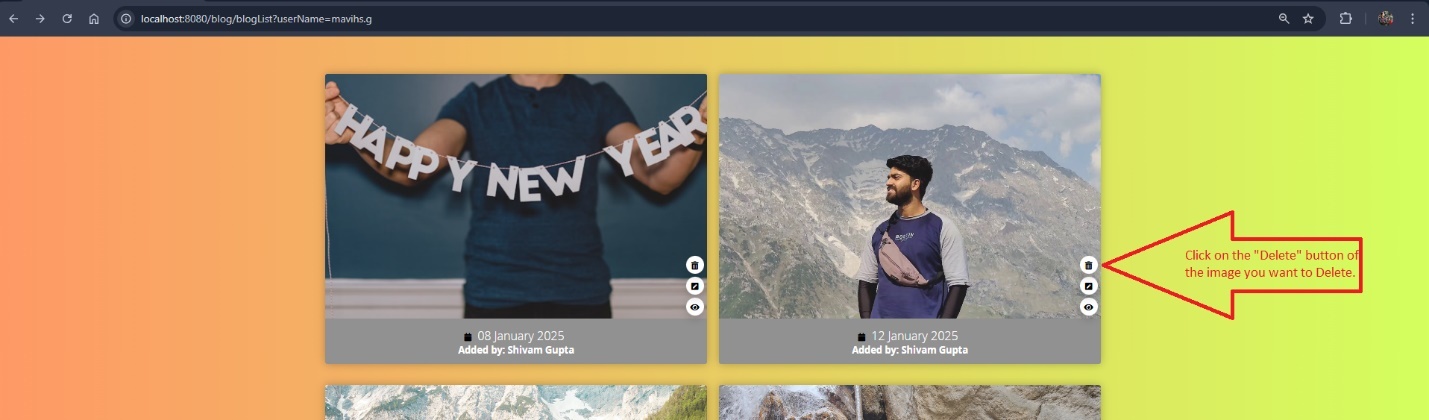


1. **Edit Memory**
   * **Input:** Users select a memory to edit, modify the description, date, or replace the image.
   * **Processing:** The changes are validated and updated in the database. The old image file is replaced if necessary.
   * **Output:** A success message is shown, and the updated memory is reflected immediately.





1. **Delete Memory**
   * **Input:** Users select a memory to delete.
   * **Processing:** A confirmation dialog ensures the user’s intent before removing the record from the database and deleting the associated image file.
   * **Output:** A notification confirms the successful deletion, and the memory is removed from the list.



#### 5.2 User Interfaces

* **Add Memory Form:** A user-friendly form with fields for username, name, description, date, and an image upload option.
* **Memory Gallery:** Displays all user memories in a card-like format, with images, descriptions, and edit/delete buttons.
* **Memory View Page:** Displays s specific memory in full resolution with all details like date, description and user name.
* **Edit Memory Page:** A pre-filled form that allows users to update specific details.

#### 5.3 Implementation Highlights

* **File Upload Handling:** The backend efficiently processes image uploads, stores them in a designated directory, and updates the database with the file path.
* **Data Validation:** Input fields are validated for constraints such as non-empty values, correct date formats, and image file types to prevent errors.
* **API Integration:** RESTful APIs ensure smooth communication between the frontend and backend, with JSON responses for actions.

The implementation of "Journal Scribe" focused on creating a robust and user-friendly experience, ensuring that both technical and non-technical users could interact with the system effortlessly. The described user interfaces and their functionalities form the foundation of this application’s usability.

**Chapter 6**

# References/Bibliography

This chapter acknowledges all the tools, resources, and references that were integral to the development and completion of the "Journal Scribe" project. By crediting these resources, this report ensures transparency and provides future developers with a pathway to explore these materials further.

#### 6.1 Development Resources

1. **Programming Languages and Frameworks**
   * Java Development Kit (JDK) 17: Essential for developing backend functionalities.
   * Spring Boot 3.x Framework: Facilitated rapid application development and REST API implementation (<https://spring.io/>).
   * MySQL Database: Used for creating and managing the database to store user information and memories (<https://www.mysql.com/>).
   * HTML and CSS Standards: Utilized for creating the user interface, ensuring a visually appealing and user-friendly frontend (<https://developer.mozilla.org/>).
2. **Development Tools and Libraries**
   * IntelliJ IDEA: Used as the primary integrated development environment (IDE) (<https://www.jetbrains.com/idea/>).
   * Apache Commons FileUpload: Handled file uploads efficiently (<https://commons.apache.org/proper/commons-fileupload/>).
   * Maven: Managed dependencies and project builds (<https://maven.apache.org/>).
3. **Testing Tools**
   * JUnit: Used for unit testing of backend functionalities (<https://junit.org/>).
   * Mockito: Simplified mocking in unit tests (<https://site.mockito.org/>).

#### 6.2 Academic References

1. Spring in Action by Craig Walls – A valuable resource for understanding Spring Boot and its applications.
2. Head First Java by Kathy Sierra and Bert Bates – Provided foundational knowledge of Java programming concepts.

#### 6.3 Online Resources

1. TutorialsPoint: Provided tutorials on Java, Spring Boot, and database integration (<https://www.tutorialspoint.com/>).
2. Stack Overflow: Helped resolve specific implementation challenges (<https://stackoverflow.com/>).
3. Baeldung: Detailed guides on Spring Boot and REST API development (<https://www.baeldung.com/>).

#### 6.4 Testing and Placeholder Resources

1. Test Images: Placeholder images for testing the application were sourced from Unsplash (<https://unsplash.com/>).

The comprehensive use of these resources ensured that the project adhered to industry standards, streamlined development, and maintained high-quality results. By citing these tools and references, the project report demonstrates both academic and technical rigor.

**Chapter 7**

# Appendices

This chapter provides an overview of the coding structure and templates used in the "Journal Scribe" application. Each significant class, method, and file is documented to explain its functionality, input parameters, and output.

#### 1. ****Main Application****

**File:** JournalScribeApplication.java

* **Description:** The entry point for the Spring Boot application.
* **Key Method:**
  + main(String[] args)
    - **Functionality:** Launches the Spring Boot application.
    - **Input:** Command-line arguments (if any).
    - **Output:** Initializes and runs the application context.

#### 2. ****Controllers****

**2.1 MemoryController**

* **Description:** Handles user requests related to memory entries.
* **Methods:**
  + getMemorys()
    - **Functionality:** Fetches all memory entries.
    - **Input:** None.
    - **Output:** List of memories in JSON format.
  + addMemory(MemoryForm memoryForm)
    - **Functionality:** Accepts memory details and stores a new entry.
    - **Input:** MemoryForm object.
    - **Output:** HTTP response indicating success or failure.

**2.2 MemoryItemController**

* **Description:** Manages specific memory items.
* **Methods:**
  + getMemoryById(Long id)
    - **Functionality:** Retrieves a memory entry by its ID.
    - **Input:** Memory ID (Long).
    - **Output:** Memory entry in JSON format.

**2.3 MainController**

* **Description:** Serves the main pages of the application.
* **Methods:**
  + home()
    - **Functionality:** Returns the home page.
    - **Input:** None.
    - **Output:** Home page view.

#### 3. ****Models****

**Memory**

* **Description:** Represents the memory entity in the database.
* **Attributes:**
  + id: Memory ID (Primary Key).
  + coverImageLoc: Path to the cover image.
  + date: Memory date.
  + description: Description of the memory.
  + name: Name of the memory.
  + userName: Username associated with the memory.
* **Methods:** Getters and setters for each attribute.

#### 4. ****Repositories****

**MemoryRepository**

* **Description:** Interface for interacting with the memory database.
* **Methods:**
  + findAll()
    - **Functionality:** Retrieves all memory entries.
    - **Input:** None.
    - **Output:** List of Memory objects.
  + findByUserName(String userName)
    - **Functionality:** Fetches memorys by username.
    - **Input:** String userName.
    - **Output:** List of Memory objects.

#### 5. ****Services****

**5.1 MemoryService**

* **Description:** Defines business logic for memory operations.
* **Key Methods:**
  + getAllMemories()
    - **Functionality:** Fetches all memories.
    - **Input:** None.
    - **Output:** List of memories.
  + saveMemory(MemoryForm memoryForm)
    - **Functionality:** Saves a new memory.
    - **Input:** MemoryForm object.
    - **Output:** None.

**5.2 MemoryServiceImpl**

* **Description:** Implementation of MemoryService.
* **Key Methods:** Implements methods from MemoryService with detailed logic for database operations.

**5.3 ImageStorageService**

* **Description:** Manages image storage and retrieval.
* **Key Methods:**
  + saveImage(MultipartFile file)
    - **Functionality:** Stores an uploaded image file.
    - **Input:** MultipartFile object.
    - **Output:** Path to the saved file.
  + loadImage(String filename)
    - **Functionality:** Retrieves an image file by its name.
    - **Input:** String filename.
    - **Output:** MultipartFile object.
  + deleteImage(String filename)
    - **Functionality:** Deletes a specified image file.
    - **Input:** String filename.
    - **Output:** None.

#### 6. ****Resources****

**6.1 Properties**

* **File:** application.properties
* **Description:** Configures database and application settings.
  + Example:
    - spring.datasource.url=jdbc:mysql://localhost:3306/journal\_scribe
    - spring.datasource.username=root
    - spring.datasource.password=\*\*\*\*

**6.2 Templates**

* **HTML Files:**
  + Home Page: Displays navigation links.
  + Add Memory Page: Form for adding memories.
  + Gallery Page: Displays uploaded memories with options to edit or delete.

#### 7. ****Unit Tests****

**ImageStorageServiceTests**

* **Description:** Verifies image storage functionalities.
* **Key Tests:**
  + testSaveImage: Ensures images are correctly stored.
  + testLoadImage: Validates image retrieval.
  + testDeleteImage: Confirms image deletion.

**JournalScribeApplicationTests**

* **Description:** Ensures the application context loads successfully.
* **Test:** contextLoads().
  + **Functionality:** Verifies Spring Boot context initialization.
  + **Output:** Passes if the application context starts without errors.