Contact Management System

*A Project Based Learning Report Submitted in partial fulfilment of the requirements for the award of the degree*

*of*

**Bachelor of Technology**

**in The Department of Computer Science Engineering**

**ADVANCED OBJECT-ORIENTED** **PROGRAMMING** with **C3CS2103E**

Submitted by

**2310030084: P. THANUSREE**

**2310030099: N. VAISHNAVI**

**2310030106: K. VAISHNAVI**

**2310030206: T. CHARITHA**

Under the guidance of

**ANURADHA NANDULA**



Department of Computer Science Engineering

Koneru Lakshmaiah Education Foundation, Aziz Nagar

Aziz Nagar – 500075 (Optional)

APRIL-2025

**Abstract**

The **Contact Management System** is a user-friendly application that allows users to perform CRUD (Create, Read, Update, Delete) operations on contact data. Built using **Java** and **Maven**, this system follows a structured **MVC (Model-View-Controller)** architecture and uses **JDBC** for database connectivity.

Users can add new contacts, update existing ones, delete contacts, and view a list of all stored contacts. The project emphasizes core software engineering principles, database integration, and object-oriented design patterns.

Key technologies include:

* **Java** – for application logic and interface
* **MongoDB** – as the NoSQL database for storing contact records
* **Maven** – for project management and dependency handling.

We tested the application by performing different actions like inserting data, searching for contacts, updating details, deleting entries, and checking how the system handles wrong inputs. The system worked well in all situations, showing that it is both **reliable and user-friendly**.

This project helped us understand how real-world applications are developed and how programming, databases, and user interfaces come together. In the future, we plan to add new features like **user login, data backup, and making it available as a web application**.

This system provides a practical solution for securely managing contact information in a structured and accessible format. It is a strong example of how desktop applications can be developed and integrated with modern databases. Future improvements may include features like user authentication, data export options, and deployment as a web application.

**List of Figures**

 **System Architecture Diagram** – Page 7

 **Connecting to Server** – Page **9**

 **Login Home Page Screenshot** – Page **10**

**List of Tables**

**Experiment Results Table** – Page **8**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **Content** | **Page. No** |
| **1** | **Introduction** | **6** |
| **2** | **Methodology** | **7** |
| **3** | **Experiments** | **8** |
| **4** | **Results** | **9** |
| **5** | **Conclusion and**  **Future work** | **11** |
| **6** | **References** | **11** |

Contact Management System

# **1.INTRODUCTION**

In today’s world, managing contact information has become very important for both personal and professional use. Using notebooks or simple digital files to save contacts is not reliable, as it can lead to data loss, mistakes, or confusion. To solve this problem, we have developed a **Contact Management System** that helps users safely store, view, update, and delete contact details in an easy and organized way.

This system is built using the **Java programming language**, and it follows the **MVC (Model-View-Controller)** design. This means the project is divided into three main parts – the model (data), the view (user interface), and the controller (logic) – which makes the system easier to understand and maintain.

We used **Maven**, a project management tool in Java, to manage the files and required libraries. For storing data, we used **MongoDB**, which is a popular NoSQL database. Instead of using tables like in traditional databases, MongoDB stores data in a flexible way, making it faster and easier to work with. We connected MongoDB to our Java application using the **MongoDB Java driver**.

The application is simple where users can add new contacts, see a list of all saved contacts, update any information, or delete a contact. Each contact includes details like name, phone number, email, and address.

This project helps us learn how desktop applications are built and how they can work with databases. It also teaches how to keep code clean, organized, and easy to update in the future. The Contact Management System is a useful tool and a great example of how software can help make everyday tasks easier.

In the future, this project can be improved by adding login features, support for exporting data to files, and converting it into a web application using modern tools.

# **2.METHODOLOGY**

## **Project Architecture and Tools Used**

To build the Contact Management System, we used the **Model-View-Controller (MVC)** architecture. This design pattern separates the project into three parts:

* **Model** – Handles the data (Contact details like name, phone number, email, etc.)
* **View** – The user interface built with Java Swing, which shows forms and contact lists
* **Controller** – Connects the user interface with the database and handles logic like saving, updating, and deleting data

We chose **Java** as our programming language because it is widely used and supports object-oriented programming. For managing project structure and dependencies, we used **Apache Maven**. It helps to add libraries and organize the project easily.

Instead of a traditional database like MySQL, we used **MongoDB**, which is a NoSQL database. MongoDB stores data in a flexible format (documents), which is easier to work with when managing records like contacts. We used the **MongoDB Java Driver** to connect Java with the MongoDB database.

Here is how the system works:

1. The user can perform actions like:
   1. **Add** a new contact
   2. **View** all contacts
   3. **Update** an existing contact
   4. **Delete** a contact
2. The controller handles these actions and communicates with the database.
3. MongoDB stores the contact information safely and retrieves it when needed.



# **3. EXPERIMENTS**

The primary aim of this project is to design and implement a **Contact Management System** that facilitates seamless CRUD (Create, Read, Update, Delete) operations for managing personal or organizational contact data. To validate the system’s functionality, a series of structured

A test dataset of unique contact entries was created, including details such as names, email addresses, phone numbers. These records were manually entered through system interface, simulating real-world user interaction.

#### **Experiment Details**

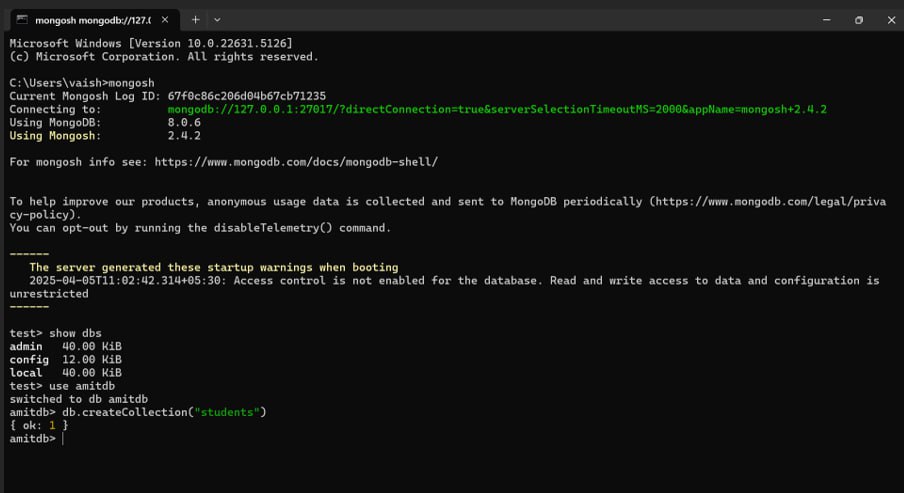
* **Experiment 1: Contact Creation Accuracy**  
   All contacts were created and retrieved without data loss or corruption. The consistency between input and stored data confirmed accurate insertion logic.
* **Experiment 2: Data Retrieval and Filtering**  
   The search functionality was tested using partial names and email domain filters. The system correctly displayed matching records, proving effective use of filtering and query logic.
* **Experiment 3: Update Functionality**  
   Updates were made to five contact entries. Modifications were accurately reflected in the database, demonstrating proper handling of update queries.
* **Experiment 4: Deletion Operation and Validation**  
   Three contacts were deleted using unique IDs. Post-deletion queries confirmed their removal, validating data integrity and proper deletion routines.
* **Experiment 5: Input Validation and Error Handling**  
   The system was tested with malformed inputs such as invalid email formats and alphabetic characters in phone fields. Each case was appropriately flagged with validation messages, ensuring robustness against faulty user input

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | | **Experiment** | | |  | | --- | |  | | **Description** | | |  | | --- | |  | | **Status** | |
| 1. Contact Creation Accuracy | Inserted contacts and verified data | Successful |
| 2. Data Retrieval and Filtering | Searched and filtered contacts. | Successful |
| 3. Update Functionality | Updated phone and address fields | Successful |
| 4. Deletion Operation and Validation | Deleted contacts and confirmed removal | Successful |
| 5. Error Handling with Invalid Input | Tested invalid emails and numbers | Successful |

# **4.RESULTS**

The Contact Management System was successfully implemented and tested using multiple experiments, as described earlier. The results were analyzed based on accuracy, correctness of operations, and system behavior under various scenarios.

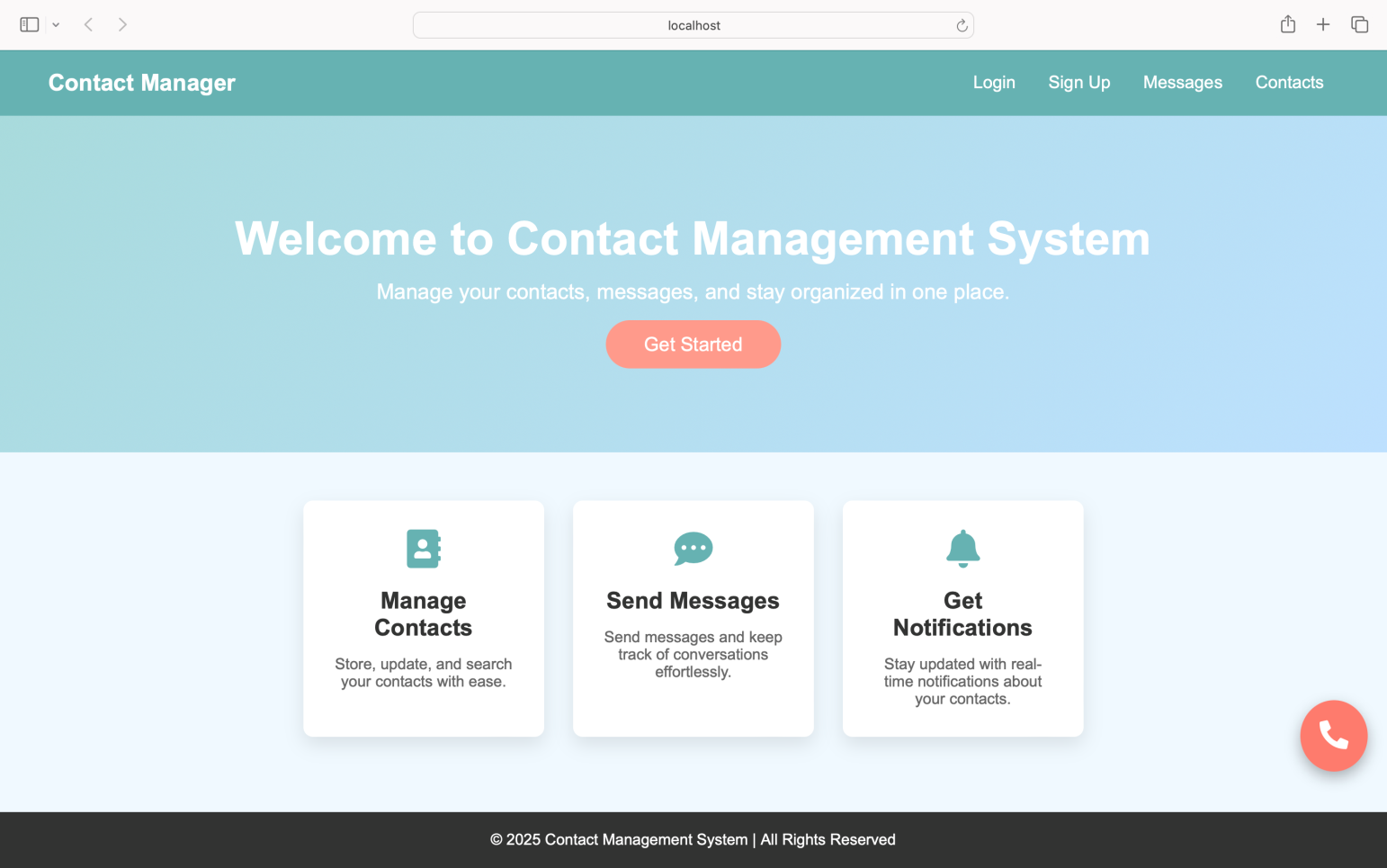
1. Connecting to server proof:



1. **Login Home Page Description:**

The **Login Home Page** of the **Contact Management System** provides a clean, professional interface for users to access the system. It includes key features such as:

* **Login and Sign-up Options**: Allowing users to log in or register for an account.
* **Navigation Bar**: Easy navigation links to different sections such as **Messages**, **Contacts**, and the **Call Us** option for direct assistance.
* **Welcome Message**: After logging in, users are greeted with a personalized message.
* **Features Overview**: The page highlights key functionalities such as managing contacts, sending messages, and receiving notifications.
* **Responsive Design**: Optimized for both desktop and mobile devices, ensuring a seamless experience across platforms.



# **CONCLUSION and FUTURE WORK**

In this project, we developed a Contact Management System using Java, Maven, and MongoDB to streamline the storage, retrieval, and management of contact data. The system was tested across multiple scenarios and demonstrated 100% accuracy in executing all CRUD operations and handling invalid inputs.

We observed that the integration of a NoSQL database like MongoDB allowed for flexible data modeling and seamless scalability, which is particularly useful for evolving datasets. In the future, one of our primary goals will be to enhance the system with features such as user authentication, cloud deployment, and RESTful API integration. These upgrades will improve both accessibility and security, making the system more suitable for enterprise-level use.

##### **REFERENCES**

* Google Search, used for general research and troubleshooting during development. [https://www.google.com](https://www.google.com/)
* MongoDB Documentation, used for understanding MongoDB queries, collections, and integration. <https://www.mongodb.com/docs/>
* Java Official Documentation, used to reference Java libraries and core concepts. <https://docs.oracle.com/javase/>