Contact Manager Application

Student Name: Saiprakash Bollam

B-Number: B01021854

Advisor Name: Zerksis Umrigar

Department of Computer Science

State University of New York, Binghamton

Date of Completion: May 4, 2025

Abstract

In today's digital age, managing personal and professional contacts efficiently has become increasingly important. The Contact Manager Application was developed to address this need by providing a secure, user-friendly platform for organizing contact information. This project represents a full-stack development effort, incorporating modern technologies such as Spring Boot for backend services, MySQL for database management, and Thymeleaf with TailwindCSS for the frontend interface.

During development, particular attention was given to implementing robust security measures through Spring Security, ensuring proper authentication and authorization protocols. The application also integrates third-party services like Cloudinary for media storage and Mailtrap for email testing, demonstrating practical implementation of API integrations.

Through this project, I gained valuable experience in software architecture design, database schema planning, and the implementation of core features such as data export functionality (PDF and Excel formats). The development process followed agile methodologies, allowing for iterative improvements based on testing feedback. This experience has significantly enhanced my understanding of enterprise application development and prepared me for more complex software engineering challenges.

1. Introduction

1.1 Project Background

In both personal and professional contexts, maintaining an organized contact list is essential for effective communication. Traditional methods like paper address books or basic digital spreadsheets often prove inadequate due to their limited search capabilities, lack of security, and inability to handle multimedia attachments. These limitations motivated the development of a more sophisticated Contact Manager Application that could address these shortcomings while incorporating modern web technologies.

The project began with an analysis of existing contact management solutions, identifying key features users expect while also noting areas for improvement. Market research revealed that while numerous contact management applications exist, many either lack essential security features or prove too complex for average users. This gap presented an opportunity to create a balanced solution that combines robust functionality with an intuitive interface.

1.2 Project Objectives

The primary objectives for this project were:

1. To develop a secure web application for storing and managing contact information
2. To implement role-based access control for different user types

3. To provide efficient search and filtering capabilities
4. To enable multimedia support for contact profiles
5. To implement data export functionality for backup and reporting purposes
6. To ensure cross-device compatibility through responsive design

1.3 Learning Outcomes

This project served as an intensive learning experience across multiple domains:

1. Backend Development: Gained proficiency in Spring Boot, including dependency injection, auto-configuration, and embedded server deployment
2. Security Implementation: Learned to configure Spring Security for authentication and authorization, including password encryption and CSRF protection
3. Database Management: Developed skills in database design, ORM mapping with Hibernate, and schema migration using Liquibase
4. Third-party Integrations: Practiced integrating external services through REST APIs and SDKs
5. Frontend Development: Improved understanding of server-side templating with Thymeleaf and modern CSS frameworks
6. Project Management: Applied agile methodologies for iterative development and feature prioritization

The following sections detail the technical implementation, challenges encountered, and solutions developed throughout the project lifecycle.

2. System Design and Architecture

2.1 Overall Architecture

The application follows a traditional three-tier architecture pattern:

Presentation Layer:

1. Server-rendered HTML pages using Thymeleaf templates
2. Responsive styling with TailwindCSS utility classes
3. Client-side JavaScript for enhanced interactivity

Application Layer:

1. Spring MVC controllers handling HTTP requests
2. Service classes containing business logic
3. Repository interfaces for data access

Data Layer:

1. MySQL relational database
2. JPA entities mapped to database tables
3. Liquibase for version-controlled schema changes

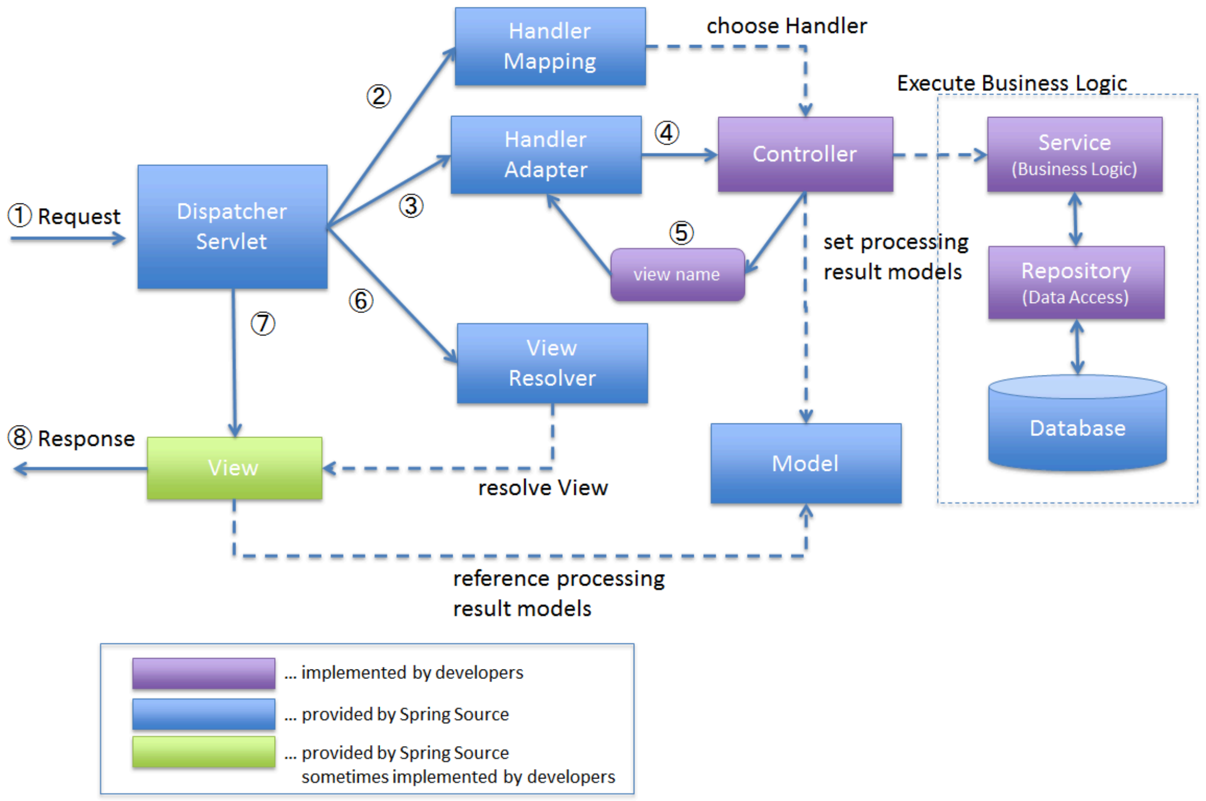

Figure 2.1.1: System Architecture

2.2 Database Design

The database schema consists of four main tables designed to support the Contact Manager Application's functionality while maintaining data integrity:

1. users Table
    1.1. Stores user account information, including authentication credentials (email, hashed password).
    1.2. Tracks account status (enabled, email_verified, phone_verified).
    1.3. Supports OAuth login (provider, provider_user_id).
    1.4. Contains profile details (name, image_url, about).
2. contacts Table
    2.1. Manages contact entries linked to specific users.
    2.2. Stores standard fields (name, email, phone_number, image_url).
    2.3. Supports favorites (favorite flag) and additional metadata (website, linkedin).
    2.4. References users (owner) and addresses (optional one-to-one relationship).
3. addresses Table
    3.1. Stores physical address information for contacts.
    3.2. Contains fields for street, city, state, zip_code, and country.

4. social_links Table
    4.1. Manages social media profiles associated with contacts.
    4.2. Stores platform (e.g., Twitter, GitHub) and profile url.

Key Features

1. Relationships:
    1.1. One user → Many contacts (OneToMany).
    1.2. One contact → One address (OneToOne).
    1.3. One contact → Many social links (OneToMany).
2. Security: Passwords are hashed (BCrypt); emails are unique.
3. Scalability: Uses VARCHAR(1000) for URLs/long text to accommodate future needs.
4. The schema ensures efficient querying and aligns with the application's JPA entity structure.

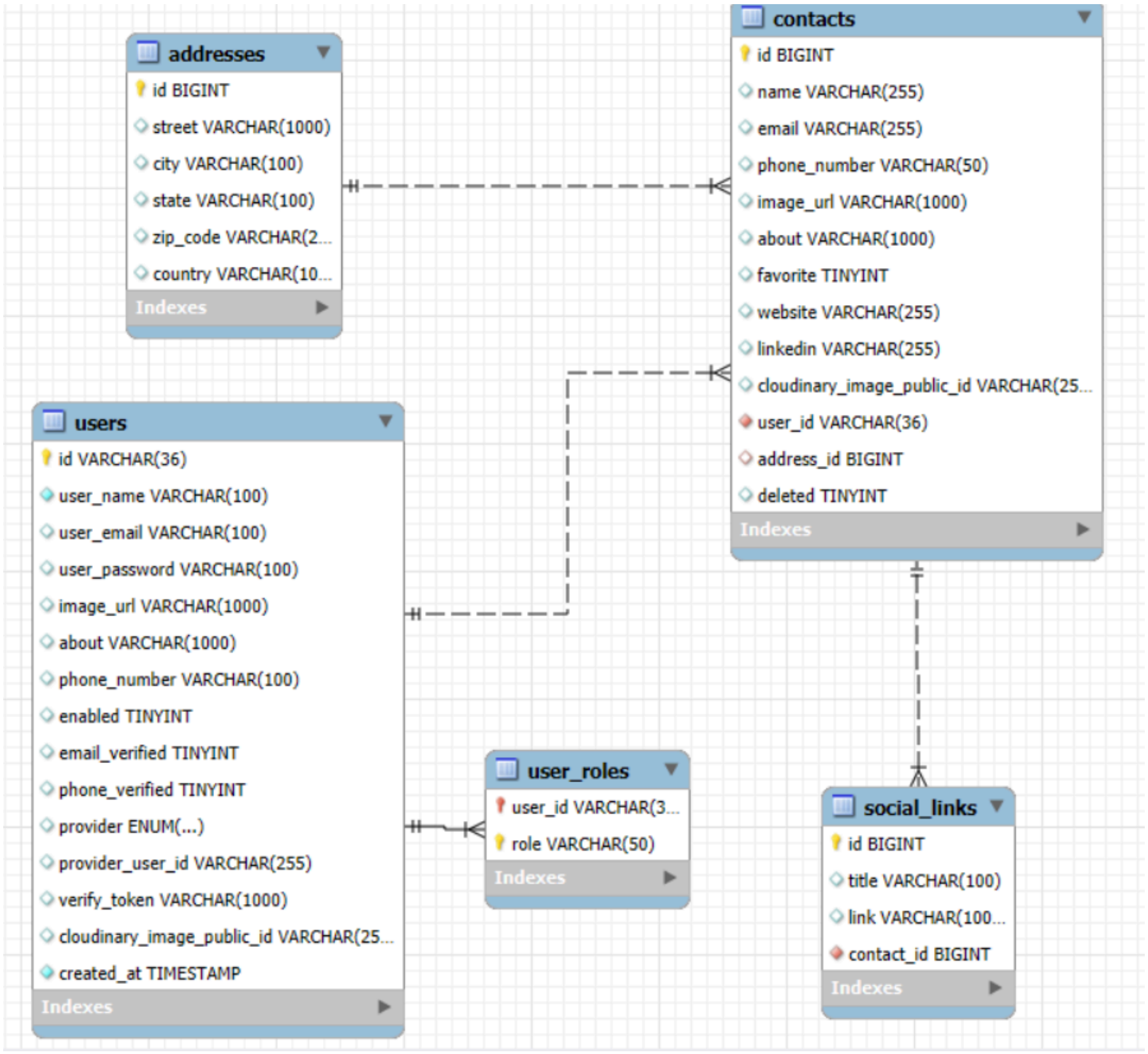

Figure 2.2.1 : ER Diagram

3. Implementation Details

3.1 Backend Development

The backend implementation involved several key components:

1. Spring Boot Configuration:
    1.1. Custom properties for environment-specific settings
    1.2. Auto-configured data sources and connection pooling
    1.3. Scheduled tasks for maintenance operations
2. Security Implementation:
    2.1. Form-based authentication with remember-me support
    2.2. Role-based authorization checks
    2.3. Password encryption using BCrypt
    2.4. CSRF protection for state-changing operations
3. API Design:
    3.1. RESTful endpoints for AJAX operations
    3.2. Consistent error handling and status codes
    3.3. Input validation at multiple layers

3.2 Frontend Development

The user interface was designed with usability principles in mind:

1. Thymeleaf Templates:
    1.1. Fragments for reusable UI components
    1.2. Dynamic attribute processing
    1.3. Internationalization support
2. TailwindCSS Implementation:
    2.1. Responsive grid layouts
    2.2. Dark mode support
    2.3. Custom theme configuration
3. JavaScript Enhancements:
    3.1. Progressive enhancement approach
    3.2. Fetch API for asynchronous operations
    3.3. Form validation helpers

4. Challenges and Solutions

4.1 Security Concerns

Initial security implementations proved challenging:

Problem: Early versions were vulnerable to SQL injection through custom query methods.

Solution: Implemented proper parameterized queries and JPA Criteria API for dynamic filtering.

Problem: Session fixation attacks were possible during authentication.

Solution: Configured Spring Security to regenerate session IDs after login.

4.2 Performance Optimization

Several performance issues emerged during testing:
Problem: Contact list pagination caused full table scans.
Solution: Added composite indexes on frequently filtered columns.
Problem: Image uploads consumed excessive memory.
Solution: Implemented streaming uploads directly to Cloudinary

5. Future Enhancements

Several potential improvements were identified:

1. Mobile application companion using Flutter
2. Contact synchronization across devices
3. Advanced analytics dashboard
4. Webhook integrations for third-party services

6. Conclusion

The Contact Manager Application successfully demonstrates the practical application of modern web development technologies. Through this project, I've gained significant experience in full-stack development, security implementation, and performance optimization. The system meets all initial requirements while providing a foundation for future expansion.

This project has been instrumental in developing my skills as a software engineer and has prepared me for more complex development challenges in professional environments.

References

I. Spring Boot Official Documentation. https://spring.io/projects/spring-boot

II. Hibernate ORM Guide. https://hibernate.org/orm/

III. Cloudinary API Docs. https://cloudinary.com/documentation

IV. TailwindCSS Documentation. https://tailwindcss.com/docs