Project Report

1. Project Title

Digital Visiting Card Generator

2. Abstract

This project is a full-stack web application that allows users to create, customize, and share digital visiting cards (e-cards) via a public link and QR code. It targets students and professionals who want a quick, modern way to share contact information without printed cards. The system includes user authentication, card CRUD operations, a live preview, optional image upload, and basic security measures.

3. Problem Statement

Traditional paper visiting cards are easily lost, difficult to update, and wasteful. Individuals need a lightweight digital solution to create and share up-to-date contact cards that are accessible from any device.

4. Objectives

- Build a responsive React front end for creating and previewing visiting cards.
- Implement a Spring Boot backend with secure user authentication and RESTful APIs.
- Store data in MongoDB and optionally host images on Cloudinary.
- Generate shareable public URLs and QR codes for each card.
- Demonstrate basic security best practices (password hashing, JWT/CORS, input validation).
- Deploy the application to public hosting and document the process.

5. Scope

Includes:

- User sign-up, login, profile management.
- Create / Read / Update / Delete visiting cards.
- Public preview pages for cards accessible via a slug.
- QR code generation for sharing.

- Basic analytics (optional: view count) and image uploads.

6. Target Users

- Students, freelancers, job seekers, startup founders.
- Professionals who prefer digital sharing over paper cards.

7. Functional Requirements

1. User Registration and Authentication (email + password).
2. CRUD operations for visiting cards.
3. Live card preview while editing.
4. Publicly accessible card via site.com/card/:slug.
5. QR code generation for each public card link.
6. Image upload for profile photo (validated and stored externally).

8. Non-functional Requirements

- Performance: API responses < 500ms under light load.
- Security: Password hashing, JWT tokens, CORS restrictions.
- Availability: Deployable to cloud with HTTPS.
- Usability: Intuitive UI with minimal steps to share a card.

9. Technology Stack

- Frontend: React (Vite), React Router, Axios, Tailwind CSS (or Bootstrap), qrcode.react
- Backend: Spring Boot, Spring Data MongoDB, Spring Security (with JWT support)
- Database: MongoDB Atlas (free tier)
- Image Storage (optional): Cloudinary (free tier)
- Deployment: Vercel (frontend), Railway/Render/Heroku (backend)

10. System Architecture

A standard 3-tier architecture:

- Client (React) —> REST API (Spring Boot) —> Database (MongoDB)
- Optional: Cloudinary for images

High-level flow:

1. User interacts with React UI.
2. Client sends authenticated requests to Spring Boot REST API.
3. Spring Boot performs validation, persistence to MongoDB, and returns responses.
4. Public preview endpoints provide card JSON for rendering without authentication.

11. UI Pages and Components

Pages:

- Landing / Home
- Register / Login
- Dashboard (list cards)
- Card Editor (form + live preview)
- Public Card Preview
- Profile / Settings

Key components:

- CardForm — form inputs, validation, image upload
- CardPreview — renders card as it will appear publicly
- QRCodeModal — shows QR for quick sharing
- AuthGuard — protects private routes

12. Security Considerations

- Passwords hashed with BCrypt (work factor 10+).
- JWT tokens with expiration; store token securely (httpOnly cookie recommended) to mitigate XSS.
- CORS configured to allow only the deployed frontend origin.
- Server-side input validation and escaping for any user-supplied HTML/text.
- Limit upload size for images; validate MIME type.
- Use HTTPS in production.
- Rate limiting on auth endpoints.

13. Testing Plan

- Unit tests: backend service layer (Spring Boot + JUnit), frontend utility functions.
- Integration tests: API endpoints (use test DB), Authentication flows.
- Manual UI tests: create/edit/delete cards, public preview, QR scanning on mobile.
- Security tests: ensure JWT protected endpoints, XSS checks on public preview.

14. Deployment Strategy

- Configure environment variables for MONGO_URI, JWT_SECRET, CLOUDINARY_*, FRONTEND_URL.
- Backend: push to Railway/Render/Heroku — ensure MongoDB Atlas whitelists server IP / uses SRV connection string.
- Frontend: push to Vercel/Netlify and set REACT_APP_API_URL to backend URL.
- Use HTTPS endpoints provided by hosting platforms.

15. Deliverables

- Source code (Frontend & Backend) on GitHub.
- Deployed demo URLs (frontend + backend).
- Project report (this document).

Prepared by:
Ishu Ranjan
23BCS14216
3A