



Hi, Manu 🙌

Let's look into the Product

Let's begin with the product tour and you can setup the authentication quick!

Let's Get Started!

Full-Stack JWT Authentication System

A complete, production-ready authentication solution combining React frontend with Spring Boot backend, featuring secure JWT-based authentication and OTP password recovery

Made with **GAMMA**

Core Security Features



JWT-Based Authentication

Stateless token-based security with encrypted payload validation



OTP Verification

Time-bound, secure password reset flow with email delivery



BCrypt Security

Industry-standard password hashing with salt protection



Protected APIs

Spring Security filters validate JWT tokens on all endpoints

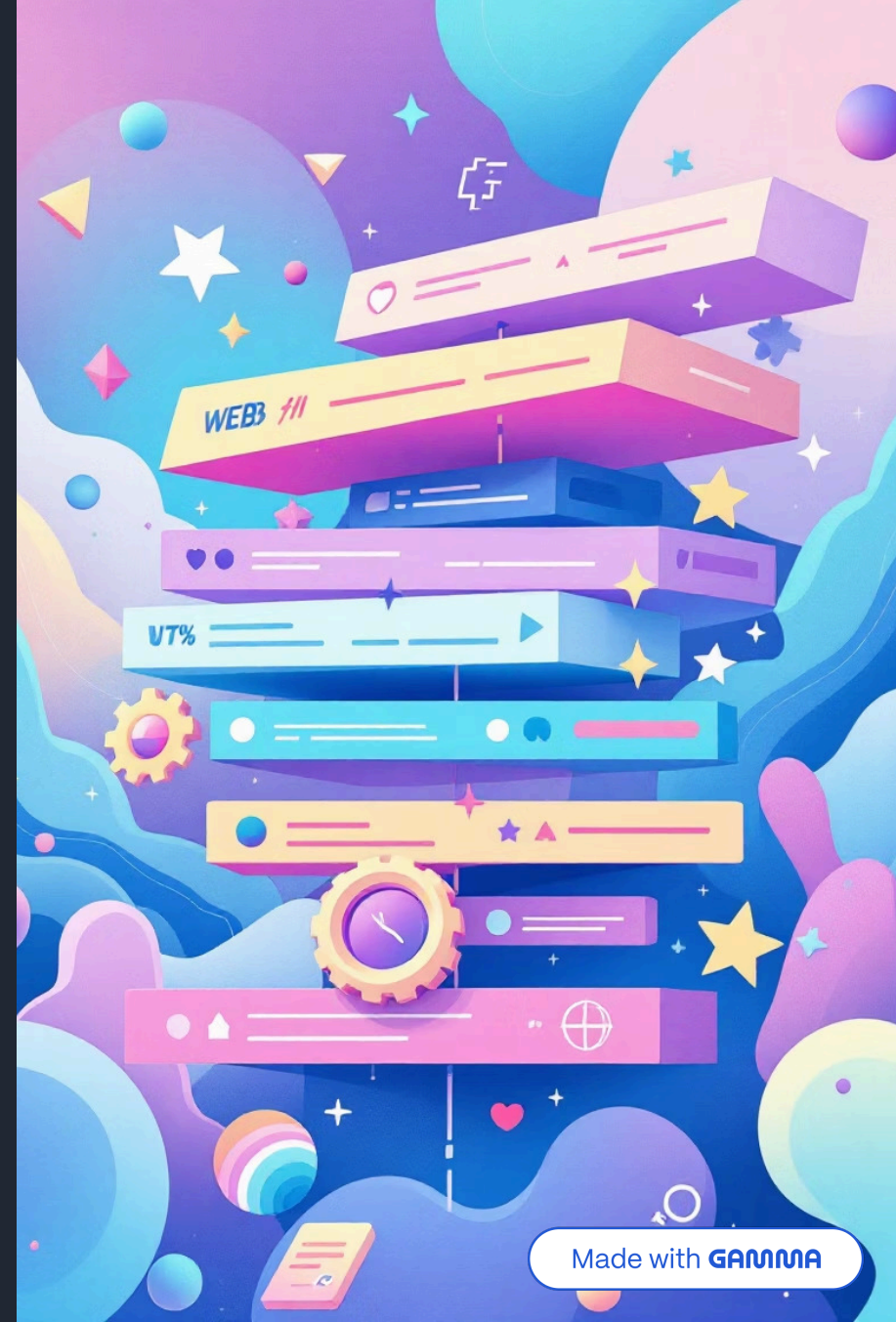
Technology Stack

Frontend

- React (Vite build tool)
- React Router DOM
- Axios for API integration
- React Toastify notifications
- Modern JavaScript (ES6+)

Backend

- Java 21
- Spring Boot 3
- Spring Security
- JWT Authentication
- MySQL with Hibernate/JPA



Public vs. Protected Endpoints

Public APIs (No JWT)

- `/login` – Authenticate credentials
- `/register` – Create new user
- `/send-reset-otp` – Generate password reset OTP
- `/reset-password` – Validate OTP and update password
- `/logout` – Client-side state cleanup

Protected APIs (JWT Required)

All other endpoints require valid Authorization header with Bearer token. Server validates JWT signature, expiration, and claims before granting access.

OTP-Based Password Reset Flow



Request Reset

User enters email on /email-verify page



Generate OTP

Backend creates time-bound code



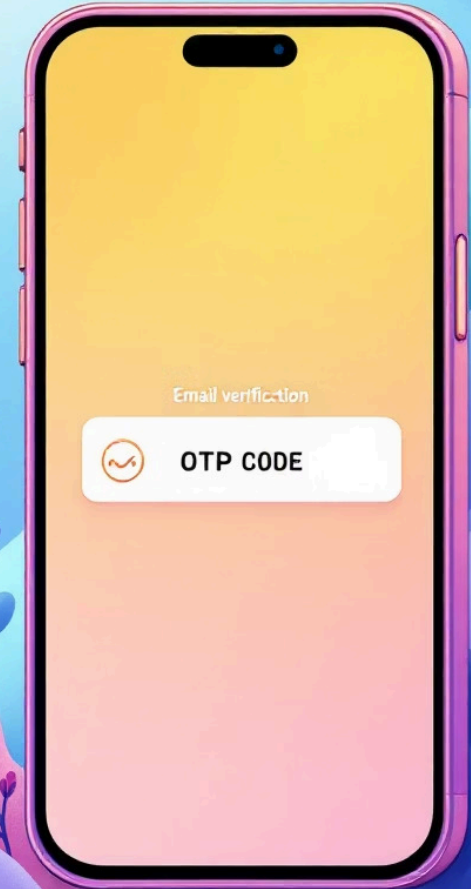
Send Email

OTP delivered via Brevo SMTP



Verify & Update

Validate OTP and reset password



JWT Authentication Flow



```
graph LR; A[User Login] --> B[Validate & Issue JWT]; B --> C[Client Stores Token]; C --> D[Token Validated];
```

User Login

Validate & Issue JWT

Client Stores Token

Token Validated

The system uses stateless security filters to validate JWT signatures, expiration times, and user claims on every protected request. No HTTP sessions are maintained on the server.

Frontend Route Architecture

01

Home Page

Application dashboard and user portal

02

Login / Register

Authentication screens with form validation

03

Email Verify

OTP request page for password recovery

04

Reset Password

Secure password update with OTP verification

📌 **Note:** These are React Router UI routes, not backend APIs. Spring Security only intercepts actual API requests, not page navigation.

Installation & Setup

Backend (Spring Boot)

```
git clone  
https://github.com/Manu5772  
28/FullStack-25-MrAuthy  
cd auth  
mvn clean install  
mvn spring-boot:run  
# Server runs at  
http://localhost:8080
```

Frontend (React)

```
cd client  
npm install  
npm run dev  
# Client runs at  
http://localhost:5173
```



Security Best Practices Implemented

Stateless Architecture

No HTTP sessions stored on server – JWT contains all necessary user information

BCrypt Password Hashing

Adaptive hashing with salt prevents rainbow table attacks and brute force attempts

Time-Bound OTP

One-time passwords expire automatically, limiting attack window for unauthorized access

CORS Protection

Backend only accepts requests from trusted origin (localhost:5173) preventing unauthorized domains



Project Overview & Author

This complete authentication system demonstrates production-style security implementation with clean frontend-backend separation. Features include user signup, secure login, OTP-based password recovery, and protected API endpoints.

Manu Bharadwaj

Full-Stack Developer

YouTube: [code-with-Bharadwaj](#)

GitHub: [Manu577228](#)

Portfolio: <https://manu-bharadwaj-portfolio.vercel.app/>

The Authentic JS/Java/Python CodeBuff

Technologies Used

- JavaScript/React
- Java/Spring Boot
- MySQL/Hibernate
- JWT/BCrypt
- Spring Security



Manu Bharadwaj

Full-Stack Developer

YouTube: [code-with-Bharadwaj](#)

GitHub: [Manu577228](#)

Portfolio: manu-bharadwaj-portfolio.vercel.app