# Spring Boot Reddit Clone - Backend Documentation

## Table of Contents

1. [Overview](#overview)
2. [Technology Stack](#technology-stack)
3. [Project Structure](#project-structure)
4. [Architecture](#architecture)
5. [API Documentation](#api-documentation)
6. [Database Schema](#database-schema)
7. [Security](#security)
8. [Configuration](#configuration)
9. [Testing](#testing)
10. [Deployment](#deployment)
11. [Development Setup](#development-setup)

## Overview

This is a Reddit clone backend built using Spring Boot 3.0.3, providing a RESTful API for a social media platform with features like user authentication, post creation, voting, commenting, and subreddit management. The application follows a layered architecture pattern with clear separation of concerns.

### Key Features

* User registration and authentication with JWT
* Email verification for new accounts
* Post creation and management
* Subreddit creation and management
* Comment system
* Voting system (upvote/downvote)
* Refresh token mechanism
* RESTful API with OpenAPI documentation

## Technology Stack

### Core Framework

* **Spring Boot 3.0.3** - Main application framework
* **Java 17** - Programming language
* **Spring Security 6** - Authentication and authorization
* **Spring Data JPA** - Data access layer
* **Spring Web** - REST API support

### Database

* **PostgreSQL** - Primary database
* **Hibernate** - ORM framework

### Security

* **JWT (JSON Web Tokens)** - Stateless authentication
* **BCrypt** - Password hashing
* **OAuth2 Resource Server** - JWT validation

### Documentation

* **OpenAPI 3** - API documentation (Swagger UI)
* **SpringDoc** - OpenAPI integration

### Additional Libraries

* **Lombok** - Reduces boilerplate code
* **MapStruct** - Object mapping
* **TimeAgo** - Relative time formatting
* **TestContainers** - Integration testing

## Project Structure

backend/  
├── src/main/java/com/programming/techie/springredditclone/  
│ ├── config/ # Configuration classes  
│ │ ├── OpenAPIConfiguration.java  
│ │ ├── SecurityConfig.java  
│ │ └── WebConfig.java  
│ ├── controller/ # REST API controllers  
│ │ ├── AuthController.java  
│ │ ├── CommentsController.java  
│ │ ├── PostController.java  
│ │ ├── SubredditController.java  
│ │ └── VoteController.java  
│ ├── dto/ # Data Transfer Objects  
│ │ ├── AuthenticationResponse.java  
│ │ ├── CommentsDto.java  
│ │ ├── LoginRequest.java  
│ │ ├── PostRequest.java  
│ │ ├── PostResponse.java  
│ │ ├── RegisterRequest.java  
│ │ └── ...  
│ ├── exceptions/ # Custom exceptions  
│ │ ├── PostNotFoundException.java  
│ │ ├── SpringRedditException.java  
│ │ └── SubredditNotFoundException.java  
│ ├── mapper/ # Object mappers  
│ │ ├── CommentMapper.java  
│ │ ├── PostMapper.java  
│ │ └── SubredditMapper.java  
│ ├── model/ # Entity classes  
│ │ ├── Comment.java  
│ │ ├── Post.java  
│ │ ├── Subreddit.java  
│ │ ├── User.java  
│ │ ├── Vote.java  
│ │ └── ...  
│ ├── repository/ # Data access layer  
│ │ ├── CommentRepository.java  
│ │ ├── PostRepository.java  
│ │ ├── UserRepository.java  
│ │ └── ...  
│ ├── security/ # Security components  
│ │ └── JwtProvider.java  
│ ├── service/ # Business logic layer  
│ │ ├── AuthService.java  
│ │ ├── PostService.java  
│ │ ├── CommentService.java  
│ │ ├── MailService.java  
│ │ └── ...  
│ └── SpringRedditCloneApplication.java  
├── src/main/resources/  
│ ├── application.properties  
│ ├── app.pub # JWT public key  
│ ├── app.key # JWT private key  
│ └── templates/  
│ └── mailTemplate.html  
└── src/test/ # Test classes

## Architecture

The application follows a **layered architecture** pattern:

### 1. Controller Layer (REST API)

* Handles HTTP requests and responses
* Validates input data
* Delegates business logic to service layer
* Returns appropriate HTTP status codes

### 2. Service Layer (Business Logic)

* Contains core business logic
* Handles transactions
* Orchestrates operations between repositories
* Implements business rules and validations

### 3. Repository Layer (Data Access)

* Extends Spring Data JPA repositories
* Provides data access methods
* Handles database operations
* Uses JPA/Hibernate for ORM

### 4. Model Layer (Entities)

* JPA entities representing database tables
* Defines relationships between entities
* Contains validation annotations

### 5. DTO Layer (Data Transfer Objects)

* Separate objects for API requests/responses
* Prevents entity exposure to external clients
* Handles data transformation

## API Documentation

### Base URL

http://localhost:8080

### Swagger UI

http://localhost:8080/swagger-ui.html

### Authentication Endpoints

#### 1. User Registration

POST /api/auth/signup  
Content-Type: application/json  
  
{  
 "email": "user@example.com",  
 "username": "username",  
 "password": "password"  
}

#### 2. User Login

POST /api/auth/login  
Content-Type: application/json  
  
{  
 "username": "username",  
 "password": "password"  
}

**Response:**

{  
 "authenticationToken": "jwt\_token\_here",  
 "refreshToken": "refresh\_token\_here",  
 "expiresAt": "2023-12-31T23:59:59Z",  
 "username": "username"  
}

#### 3. Account Verification

GET /api/auth/accountVerification/{token}

#### 4. Refresh Token

POST /api/auth/refresh/token  
Content-Type: application/json  
  
{  
 "refreshToken": "refresh\_token\_here",  
 "username": "username"  
}

#### 5. Logout

POST /api/auth/logout  
Content-Type: application/json  
  
{  
 "refreshToken": "refresh\_token\_here"  
}

### Post Endpoints

#### 1. Create Post

POST /api/posts  
Authorization: Bearer {jwt\_token}  
Content-Type: application/json  
  
{  
 "postName": "My First Post",  
 "subredditName": "programming",  
 "url": "https://example.com",  
 "description": "This is my first post description"  
}

#### 2. Get All Posts

GET /api/posts

#### 3. Get Post by ID

GET /api/posts/{id}

#### 4. Get Posts by Subreddit

GET /api/posts?subredditId={subredditId}

#### 5. Get Posts by Username

GET /api/posts?username={username}

### Subreddit Endpoints

#### 1. Create Subreddit

POST /api/subreddit  
Authorization: Bearer {jwt\_token}  
Content-Type: application/json  
  
{  
 "name": "programming",  
 "description": "Programming related discussions"  
}

#### 2. Get All Subreddits

GET /api/subreddit

#### 3. Get Subreddit by ID

GET /api/subreddit/{id}

### Comment Endpoints

#### 1. Create Comment

POST /api/comments  
Authorization: Bearer {jwt\_token}  
Content-Type: application/json  
  
{  
 "text": "Great post!",  
 "postId": 1  
}

#### 2. Get Comments by Post

GET /api/comments/by-post/{postId}

#### 3. Get Comments by User

GET /api/comments/by-user/{userName}

### Vote Endpoints

#### 1. Vote on Post

POST /api/votes  
Authorization: Bearer {jwt\_token}  
Content-Type: application/json  
  
{  
 "voteType": "UPVOTE",  
 "postId": 1  
}

## Database Schema

### Core Entities

#### User

CREATE TABLE user (  
 user\_id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 username VARCHAR(255) UNIQUE NOT NULL,  
 password VARCHAR(255) NOT NULL,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 created TIMESTAMP,  
 enabled BOOLEAN DEFAULT FALSE  
);

#### Subreddit

CREATE TABLE subreddit (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 name VARCHAR(255) UNIQUE NOT NULL,  
 description TEXT,  
 created\_date TIMESTAMP,  
 user\_id BIGINT,  
 FOREIGN KEY (user\_id) REFERENCES user(user\_id)  
);

#### Post

CREATE TABLE post (  
 post\_id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 post\_name VARCHAR(255) NOT NULL,  
 url VARCHAR(255),  
 description TEXT,  
 vote\_count INTEGER DEFAULT 0,  
 user\_id BIGINT,  
 created\_date TIMESTAMP,  
 subreddit\_id BIGINT,  
 FOREIGN KEY (user\_id) REFERENCES user(user\_id),  
 FOREIGN KEY (subreddit\_id) REFERENCES subreddit(id)  
);

#### Comment

CREATE TABLE comment (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 text TEXT NOT NULL,  
 post\_id BIGINT,  
 created\_date TIMESTAMP,  
 user\_id BIGINT,  
 FOREIGN KEY (post\_id) REFERENCES post(post\_id),  
 FOREIGN KEY (user\_id) REFERENCES user(user\_id)  
);

#### Vote

CREATE TABLE vote (  
 vote\_id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 vote\_type VARCHAR(10),  
 post\_id BIGINT,  
 user\_id BIGINT,  
 FOREIGN KEY (post\_id) REFERENCES post(post\_id),  
 FOREIGN KEY (user\_id) REFERENCES user(user\_id),  
 UNIQUE KEY unique\_vote (post\_id, user\_id)  
);

#### VerificationToken

CREATE TABLE verification\_token (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 token VARCHAR(255) UNIQUE NOT NULL,  
 user\_id BIGINT,  
 expiry\_date TIMESTAMP,  
 FOREIGN KEY (user\_id) REFERENCES user(user\_id)  
);

#### RefreshToken

CREATE TABLE refresh\_token (  
 id BIGINT PRIMARY KEY AUTO\_INCREMENT,  
 token VARCHAR(255) UNIQUE NOT NULL,  
 created\_date TIMESTAMP  
);

## Security

### Authentication Flow

1. **Registration**: User signs up → Email verification sent → Account activated
2. **Login**: Username/password → JWT token + refresh token issued
3. **API Access**: JWT token in Authorization header
4. **Token Refresh**: Refresh token → New JWT token issued

### Security Configuration

* **CORS**: Enabled for cross-origin requests
* **CSRF**: Disabled (using JWT tokens)
* **Session Management**: Stateless (JWT-based)
* **Password Encoding**: BCrypt with strength 10
* **JWT**: RSA key pair for signing/verification

### Protected Endpoints

* All endpoints except /api/auth/\*\* require authentication
* Public read access to posts and subreddits
* Write operations require authentication

### JWT Token Structure

{  
 "sub": "username",  
 "iat": 1640995200,  
 "exp": 1640998800,  
 "scope": "read write"  
}

## Configuration

### Application Properties

# Database Configuration  
spring.datasource.driver-class-name=org.postgresql.Driver  
spring.datasource.url=jdbc:postgresql://localhost:5432/puppy\_project\_db  
spring.datasource.username=yuqiye  
spring.datasource.password=  
  
# JPA Configuration  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
  
# Mail Configuration  
spring.mail.host=smtp.mailtrap.io  
spring.mail.port=25  
spring.mail.username=8d2a71401e367a  
spring.mail.password=91314f6d3cf186  
spring.mail.protocol=smtp  
  
# JWT Configuration  
jwt.expiration.time=900000  
jwt.public.key=classpath:app.pub  
jwt.private.key=classpath:app.key  
  
# Server Configuration  
server.port=8080

### Environment Variables

For production, consider using environment variables:

export SPRING\_DATASOURCE\_URL=jdbc:postgresql://localhost:5432/prod\_db  
export SPRING\_DATASOURCE\_USERNAME=prod\_user  
export SPRING\_DATASOURCE\_PASSWORD=prod\_password  
export JWT\_PRIVATE\_KEY=path/to/private.key  
export JWT\_PUBLIC\_KEY=path/to/public.key

## Testing

### Test Structure

src/test/java/com/programming/techie/springredditclone/  
├── BaseTest.java  
├── controller/  
│ └── PostControllerTest.java  
├── repository/  
│ ├── PostRepositoryTest.java  
│ └── UserRepositoryTest.java  
└── service/  
 ├── CommentServiceTest.java  
 └── PostServiceTest.java

### Running Tests

# Run all tests  
./mvnw test  
  
# Run specific test class  
./mvnw test -Dtest=PostControllerTest  
  
# Run with coverage  
./mvnw test jacoco:report

### Test Categories

1. **Unit Tests**: Individual service methods
2. **Integration Tests**: Repository layer with TestContainers
3. **Controller Tests**: REST API endpoints
4. **Security Tests**: Authentication and authorization

## Deployment

### Prerequisites

* Java 17 or higher
* PostgreSQL database
* Maven 3.6+

### Build Process

# Clean and compile  
./mvnw clean compile  
  
# Run tests  
./mvnw test  
  
# Build JAR file  
./mvnw package  
  
# Run application  
java -jar target/spring-reddit-clone-0.0.1-SNAPSHOT.jar

### Docker Deployment

FROM openjdk:17-jdk-slim  
COPY target/spring-reddit-clone-0.0.1-SNAPSHOT.jar app.jar  
EXPOSE 8080  
ENTRYPOINT ["java", "-jar", "/app.jar"]

### Production Considerations

1. **Database**: Use production PostgreSQL instance
2. **Security**: Generate new JWT keys for production
3. **Logging**: Configure proper logging levels
4. **Monitoring**: Add health checks and metrics
5. **SSL**: Configure HTTPS
6. **Rate Limiting**: Implement API rate limiting

## Development Setup

### Prerequisites

1. **Java 17**

* java -version

1. **PostgreSQL**

* # Install PostgreSQL  
  brew install postgresql # macOS  
  sudo apt-get install postgresql # Ubuntu  
    
  # Start PostgreSQL service  
  brew services start postgresql

1. **Maven**

* # Install Maven  
  brew install maven # macOS  
  sudo apt-get install maven # Ubuntu

### Setup Steps

1. **Clone Repository**

* git clone <repository-url>  
  cd puppy\_project/backend

1. **Database Setup**

* CREATE DATABASE puppy\_project\_db;  
  CREATE USER yuqiye WITH PASSWORD '';  
  GRANT ALL PRIVILEGES ON DATABASE puppy\_project\_db TO yuqiye;

1. **Generate JWT Keys**

* # Generate private key  
  openssl genrsa -out app.key 2048  
    
  # Generate public key  
  openssl rsa -in app.key -pubout -out app.pub  
    
  # Move keys to resources directory  
  mv app.key src/main/resources/  
  mv app.pub src/main/resources/

1. **Configure Application**
   * Update application.properties with your database credentials
   * Configure email settings for verification emails
2. **Run Application**

* ./mvnw spring-boot:run

1. **Verify Setup**
   * Access Swagger UI: http://localhost:8080/swagger-ui.html
   * Test health endpoint: http://localhost:8080/actuator/health

### Development Workflow

1. **Create Feature Branch**

* git checkout -b feature/new-feature

1. **Make Changes**
   * Follow coding standards
   * Add unit tests for new functionality
   * Update documentation
2. **Test Changes**

* ./mvnw test  
  ./mvnw spring-boot:run

1. **Commit and Push**

* git add .  
  git commit -m "Add new feature"  
  git push origin feature/new-feature

### Code Standards

* Use meaningful variable and method names
* Add JavaDoc comments for public methods
* Follow Spring Boot conventions
* Use Lombok annotations to reduce boilerplate
* Implement proper exception handling
* Add validation annotations to DTOs

### Debugging

1. **Enable Debug Logging**

* logging.level.com.programming.techie=DEBUG

1. **Use IDE Debugger**
   * Set breakpoints in IDE
   * Run application in debug mode
2. **Database Debugging**

* spring.jpa.show-sql=true  
  spring.jpa.properties.hibernate.format\_sql=true

## Support and Maintenance

### Common Issues

1. **Database Connection Issues**
   * Verify PostgreSQL is running
   * Check database credentials
   * Ensure database exists
2. **JWT Token Issues**
   * Verify key files exist in resources
   * Check key permissions
   * Regenerate keys if corrupted
3. **Email Issues**
   * Verify SMTP settings
   * Check email credentials
   * Test with different email provider

### Performance Optimization

1. **Database Optimization**
   * Add indexes on frequently queried columns
   * Use pagination for large datasets
   * Optimize JPA queries
2. **Caching**
   * Implement Redis for session storage
   * Cache frequently accessed data
   * Use Spring Cache annotations
3. **Monitoring**
   * Add Spring Boot Actuator
   * Implement health checks
   * Monitor application metrics

*This documentation is maintained as part of the Spring Boot Reddit Clone project. For questions or contributions, please refer to the project repository.*