

Developer Guide for Reddot

This developer guide provides technical instructions for contributors or maintainers looking to understand the codebase and technical infrastructure of Reddot.

# 1. Web Version (ReactJS + Vite Frontend, Spring Boot + MySQL Backend)

## Frontend (ReactJS + Vite)

### Project Setup

1. Clone the repository:  
  
 git clone https://github.com/your-repo/reddot-web.git  
  
2. Install dependencies:  
  
 npm install  
  
3. Start the development server:  
  
 npm run dev  
  
4. The application should now be running at http://localhost:3000.

### Component Breakdown

* **QuestionFeed**: Displays a list of questions posted by users.
* **UserProfile**: Allows users to view and edit their profile information.
* **VotingMechanism**: Implements upvoting and downvoting functionality for both questions and answers.

### Routing

Reddot uses React Router for client-side navigation. Routes are configured in `src/routes.js`, mapping URLs to different components.  
For example:

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

function App() {

return (

<Router>

<Switch>

<Route path="/" exact component={Home} />

<Route path="/questions" component={QuestionFeed} />

<Route path="/question/:id" component={QuestionDetail} />

<Route path="/profile/:id" component={UserProfile} />

</Switch>

</Router>

);

}

### State Management

State is managed using the Context API. Global state is stored in `AppContext.js`, which provides access to data like user information, questions, and answers across components.

## Backend (Spring Boot + MySQL)

### API Documentation

Reddot follows RESTful principles for its backend API, using the following key endpoints:

* **User API:**
  + POST /auth/signup: Registers a new user.
  + POST /auth/login: Authenticates a user and returns a JWT token.
  + GET /users/{id}: Fetches user profile data.
* **Question API:**
  + GET /questions: Retrieve all questions.
  + POST /questions: Submit a new question.
  + GET /questions/{id}: Retrieve a specific question by ID
* **Answer API:**
  + POST /answers: Submit a new answer to a question.
* **Vote API:**
  + POST /questions/{id}/vote: Upvote or downvote a question.

Authentication is handled via JWT tokens passed in the `Authorization` header.

### Database Schema

Reddot uses MySQL as its database. The key tables include:

- **Users**: Stores user information (e.g., ID, name, email, hashed password).

- **Questions**: Contains question details (e.g., ID, title, description, user\_id).

- **Answers**: Stores answers associated with questions (e.g., ID, content, user\_id, question\_id).

- **Votes**: Contains vote details of a subject (e.g., ID, user\_id, question\_id, comment\_id, vote\_type\_id).

A visual diagram of the schema can be found in the `docs/database-schema.png` file.

### Authentication

Reddot uses JWT (JSON Web Tokens) for authentication. Users authenticate via the `auth/login` endpoint, which returns a JWT token. This token is stored in localStorage on the client-side and passed with each request in the `Authorization` header for protected routes.

OAuth2 can also be implemented for third-party login options (e.g., Google or GitHub).

### Error Handling

Custom exceptions are thrown in the backend for known issues (e.g., `UserNotFoundException`, `ResourceNotFoundException`).

Logging: Use Slf4j for logging errors and key actions. Log error details for debugging and operational monitoring purposes.

Global Exception Handling: A @ControllerAdvice class catches exceptions and provides consistent error responses to the frontend.

# 2. Mobile Version (Flutter)

## Project Setup

1. Clone the Flutter project:

git clone https://github.com/trongtoannguyen/Reddot-React.git  
 cd Reddot-React

2. Set up Android Studio or VSCode for development.  
3. Install dependencies:  
  
 flutter pub get  
  
4. Run the app on a connected device or emulator:  
  
 flutter run

## Navigation

Reddot mobile uses `Navigator 2.0` for handling in-app navigation. Routes are defined in `lib/routes.dart`, mapping URLs to screens like `HomeScreen` and `QuestionScreen`. Use `push()` and `pop()` methods to navigate between pages.

Example

dart

Navigator.push(

context,

MaterialPageRoute(builder: (context) => QuestionDetailScreen(id: questionId)),

);

## Platform-Specific Code

Flutter’s platform channels are used to access native APIs for platform-specific functionality. For example, using `MethodChannel` to invoke camera or location services in Android/iOS:  
dart  
const platform = MethodChannel('com.example.reddot/native');

try {

final result = await platform.invokeMethod('getBatteryLevel');

} on PlatformException catch (e) {

print("Failed to get battery level: '${e.message}'.");

}

## Testing

- Unit Testing: For testing individual functions, such as validating input or calculating votes.

To run unit and widget tests:  
  
 flutter test  
  
For integration tests:  
  
 flutter drive --target=test\_driver/app.dart  
  
Include tests for critical functions like submitting a question, voting, and logging in.