Wesal App

Technical Documentation for Developers

Version 1.0.0

Saleh Bubshait

August 7, 2025

Contents

Acknowledgements

This application in whole was developed completely by Saleh Bubshait. However, we acknowledge that the original idea and concept for the Wesal application is credited to Weed Batarfi and the rest of the Insjiam team. Their vision and initial conceptualization (through the FlutterFlow no-code platform) provided the foundation for this social networking platform. Special thanks goes to Weed Batarfi, who kindly got me up to speed with the application's functional requirements.

This project was developed as part of my Summer Internship at the COD/DPSD/ERP Management Group. Special thanks go to my supervisor, Mohammed Abdulqader, as well as my mentors, Amro Sagga and Mustafa Gafli, whose guidance, support, and instrumental feedback were invaluable throughout the development process.

1 Introduction

1.1 Overview

Wesal is a social networking mobile application designed specifically for connecting colleagues and transforming workplace social interactions. The app name is Arabic for "connection" or "union."

It is a platform that enhances communication and collaboration among division members allowing them to build deeper connections and share experiences beyond formal work-related discussions. There are three main features within the application:

- **Social Feed:** Users can create, view, like, and comment on posts, fostering a sense of community.
- **Invitations:** Users can create and manage event invitations, allowing colleagues to RSVP and participate in social gatherings. Special types of invitations include: Invitation for coffee, lunch, exercise, sports, networking.
- **Puzzles**: A daily challenge is released everyday at 8:00 AM with a new daily puzzle that users can solve. Users can view the Leaderboard to see how they rank against their colleagues. All of this while being timed to add a competitive edge (8:00 AM - 11:00 AM).

Crucially, the application notifies users of new invites, comments on their posts, or people accepting their invites through push notifications, ensuring they stay engaged with the community.

1.2 Current State of Development

The app is currently in beta testing phase with a limited user base of COD/DPSD employees. The original developer is currently on an Out-of-Kingdom assignment, and as such the application maintainence status is unknown. Please feel free to reach out to COD/DPSD for more information.

The Wesal app was completely developed by Saleh Bubshait as a side project during his summer internship at the department which was only 4 weeks long. To allow for rapid development, some steps were skipped such as through testing and documentation. That being said, the app is fully functional and has been tested by a small group of users.

1.3 Technology Stack

Frontend: Flutter 3.8.1+ with Material Design

Backend: Spring Boot 3.5.3 with Java 21

Database: PostgreSQL with JPA/Hibernate

Authentication: JWT tokens with secure storage

Notifications: Firebase Cloud Messaging (FCM)

Deployment: Docker containerization with docker-compose

Documentation: OpenAPI/Swagger for API documentation

2 Resources

Google Drive Link

`https://drive.google.com/drive/folders/1qVhgKtfMFTSJl88WSdFzTnT_Eysldf2h?usp=sharing`

The google drive link contains a number of useful resources including:

- Source code for the Wesal application
- Demo Video
- System Diagram
- Presentation Slides
- Documentation

The code is hosted and available on git.bubshait.me. You can access the repository at: `https://git.bubshait.me/sBubshait/wesal.git`

This file is the technical documentation for the Wesal application. It provides detailed information on installation, setup, development practices, and API documentation.

However, future developers are encouraged to refer to the official documentation of the various technologies used in the application (in addition to the code comments) for more in-depth understanding. These include:

- **Flutter Documentation:** `https://flutter.dev/docs`
- **Spring Boot Documentation:** `https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/`
- **PostgreSQL Documentation:** `https://www.postgresql.org/docs/current/index.html`
- **Firebase Documentation:** `https://firebase.google.com/docs`
- **Docker Documentation:** `https://docs.docker.com/`

2.1 Development Tools

- Android Studio / VS Code with Flutter extensions
- IntelliJ IDEA for Spring Boot development
- Adminer for database testing and management

3 Architecture Overview

3.1 System Architecture

The Wesal app follows a modern microservices-inspired architecture with four main components:

- **Frontend:** The frontend is built using Flutter. Flutter is a framework for building cross-platform applications. It is responsible for the user interface and user interactions. This can be exported into iOS or Android applications, or even a web application.
- **Backend:** The backend is built using Java Spring Boot and is responsible for the business logic and data management. It is responsible for the authentication, data processing, and database CRUD operations.
- **Database:** The database is built using PostgreSQL and is responsible for the persistent storage of the application data.
- **Firebase:** Firebase is used for push notifications.

In addition to the two main components, there are two other helpful components:

- **API Documentation:** The API documentation is built using Swagger. It is responsible for the documentation of the API endpoints and their usage. This is available at `http://localhost:8080/docs` when the backend is running locally.
- **Adminer:** Adminer is a database management tool that allows you to manage the PostgreSQL database used by the application in the browser directly without the need for any additional software.

The system diagram below shows the architecture of the Wesal app with the four main components and the two other helpful components.

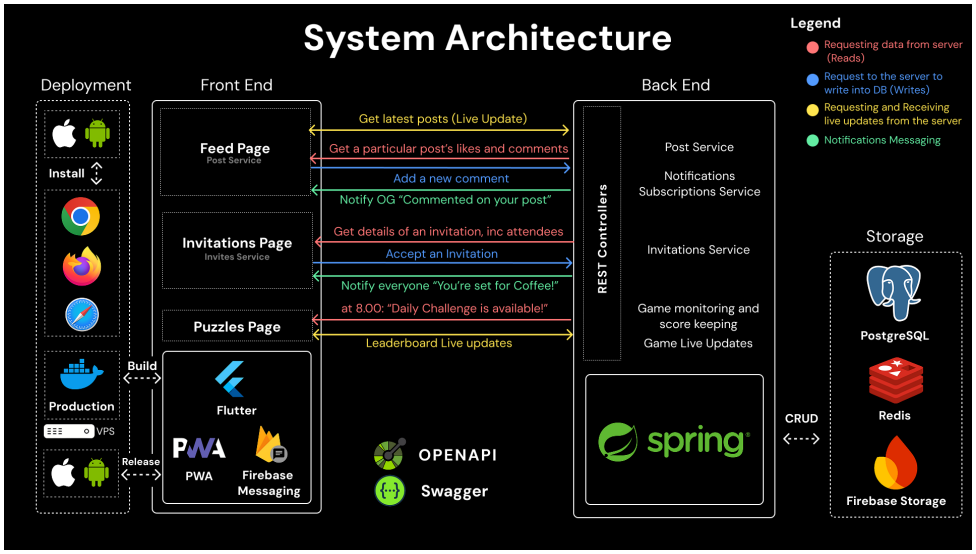

Figure 1: System Architecture Diagram

3.2 Frontend Architecture

The frontend follows a layered architecture with service classes for business logic, HTTP client for API communication, model classes for data representation, and screen-based navigation with reusable widgets. State management is handled locally using setState() with optimistic UI updates.

3.3 Backend Architecture

The backend implements a traditional layered architecture with REST controllers for API endpoints, service layer for business logic, repository layer for data access using Spring Data JPA, entity layer for database mapping, and security layer for JWT authentication and authorization.

3.4 Database Schema

The database schema is shown below.

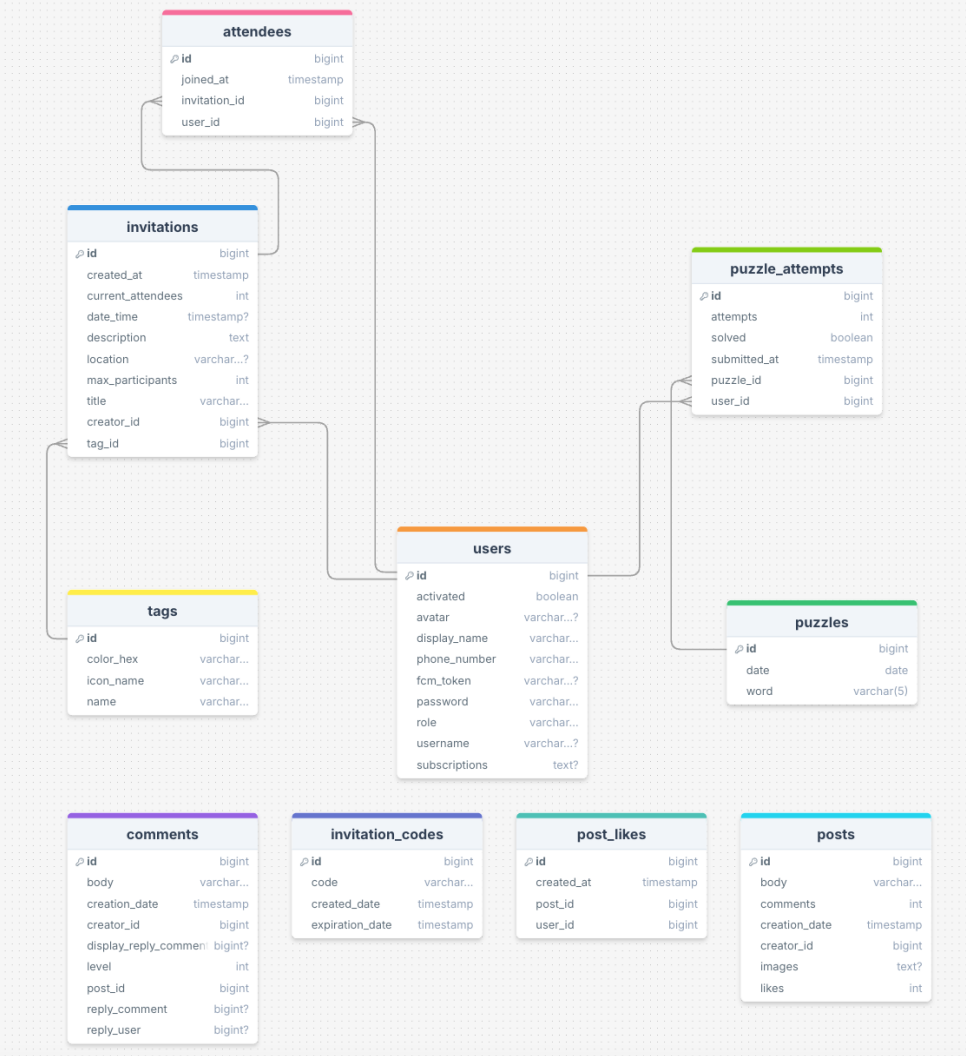

Figure 2: Database Schema

3.5 Landing Page

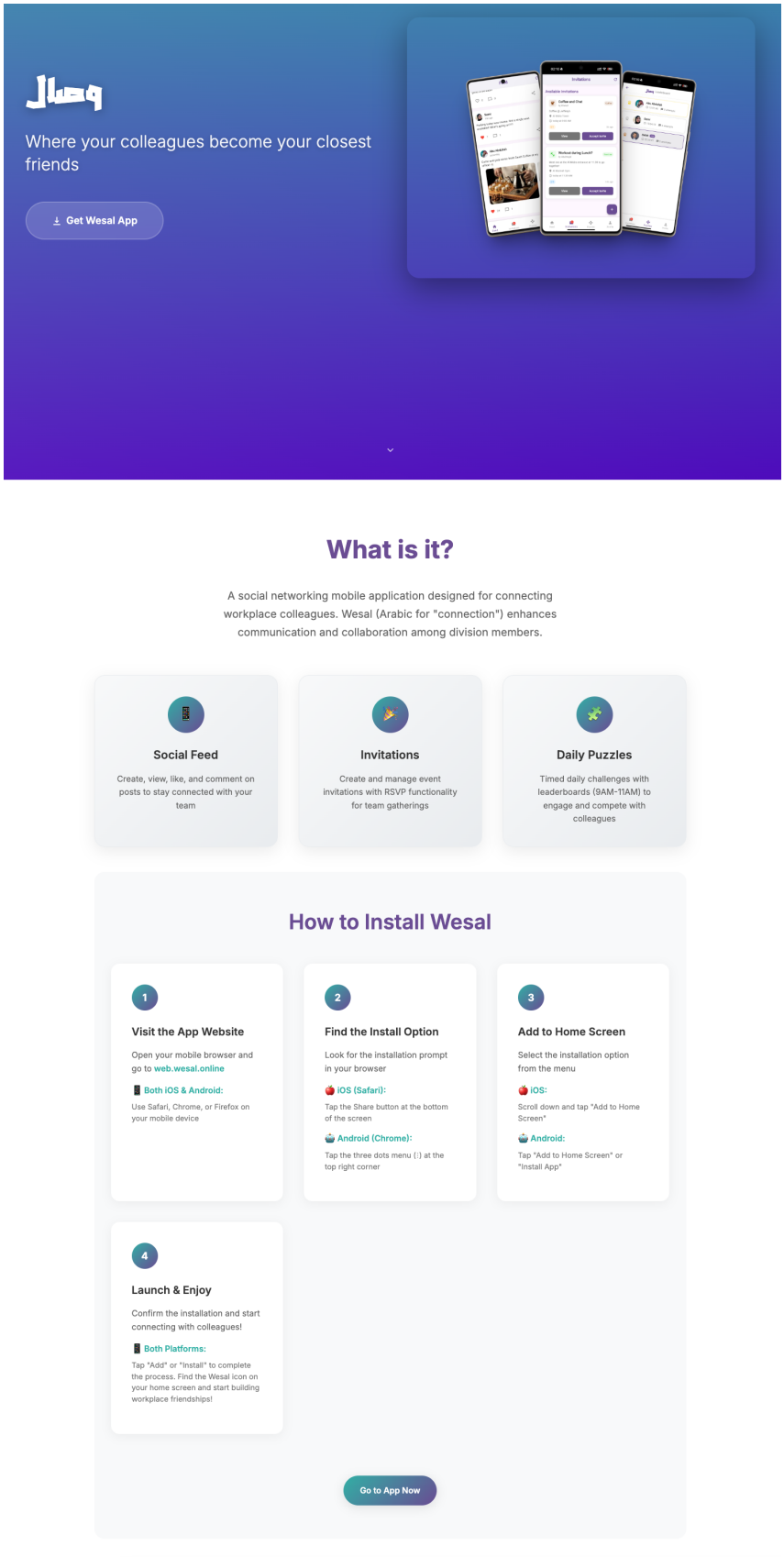

Figure 3: Landing Page

4 Installation and Setup

4.1 Source Code

The source code for the wesal application is hosted on git.bubshait.me. You can clone the repository using the following command:

```
git clone https://git.bubshait.me/sBubshait/wesal.git
```

If this does not work, you can also download the source code in a zip file from the Google Drive Link in the Resources section of this document.

4.2 Secrets and Environment Variables

The application is configured to use environment variables for sensitive information such as database credentials. In addition, firebase service account credentials are stored in a secure JSON file.

To make this easier, a script 'install.py' is provided in the root directory of the project. Regardless of the setup method in below you should start with './install.py'. Ensure you are in the root directory of the project when running this script. You may need to change permissions on the script to make it executable. It is recommended to run this script in a terminal with administrative privileges as it involves creating all required secret files.

If the source code has been cloned to 'wesal' directory from the last step, you can run the script as follows:

```
cd wesal
chmod +x install.py
sudo ./install.py
```

Then follow the prompts to enter the required information.

Info: The process for obtaining Firebase service account credentials was accurate at the time of writing this document, but it may change in the future. If you encounter any issues, please refer to the official Firebase documentation for the most up-to-date instructions.

4.3 Running the application

4.3.1 Method 1: Using Docker (Recommended)

The application can be easily started using Docker. A 'docker-compose.yml' file is provided in the root directory of the project. This file contains all the necessary configurations to run the application in Docker containers.

Warning

This requires Docker and Docker Compose to be installed on your system. Please refer to the official Docker documentation for installation instructions.
If you are using the Linux Compose Plugin then the replace all instances of 'docker compose' with 'docker-compose' in the commands below.

To start the application using Docker, run the following command in the root directory of the project:

```
docker compose up --build
```

That's it! The application will be started in Docker containers. The application will be accessible through the following ports by default:

- PostgreSQL database: 'localhost:5050'
- Adminer database admin (For database management): 'localhost:8100'
- Backend server: 'localhost:4044'
- Frontend web server: 'localhost:6060'

You can change these ports in the 'docker-compose.yml' file if needed.

4.3.2 Method 2: Manual Setup (Advanced)

For advanced users who prefer to set up the application manually without Docker, the following steps can be followed.

In this method, we will need to set up and run everything separately.

PostgreSQL: First, you need to have PostgreSQL installed and running on your system. You will need to configure the database and create a user for the application. We will skip this step here as it is a generic setup and can be easily found in the official PostgreSQL documentation. Can be downloaded from `https://www.postgresql.org/download/`.

Backend: The backend is built using Spring Boot. You will need to have Java 21. `This is essential.` We will skip the installation steps for Java but you can download it from Oracle `https://www.oracle.com/java/technologies/javase-jdk21-downloads.html`. Other Java distributions such as OpenJDK can also be used.

Once you have Java installed, check the version by running the following command in your terminal:

```
java -version
```

If the version is 21 or higher, you can proceed to the next step.

Next, the source code comes with Maven already packaged within the application. Maven will handle downloading all the required dependencies for the backend. To run the backend, navigate to the 'backend' directory and run the following command:

```
cd backend
mvn clean install
mvn spring-boot:run
```

This will start the backend server on port 8080 by default. You can change the port in the 'application.properties' file if needed.

You can access the backend server at 'http://localhost:8080'. The API documentation is available at 'http://localhost:8080/docs' when the backend is running locally.

Frontend: The frontend is built using Flutter. You will need to have Flutter SDK installed on your system. You can download it from the official Flutter website: `https://flutter.dev/docs/get-started/install`.

Now, navigate to the 'frontend' directory and run the following command to install the dependencies:

```
cd frontend
flutter pub get
```

Once the dependencies are installed, you can run the web application using the following command:

```
flutter run --web-server --web-port 6060
```

This will start the frontend web server on port 6060 by default. You can change the port in the command itself.

5 Using the Application

5.1 Installation

Unfortunately, at the time of writing this document, the Wesal application is not available on the App Store or Google Play Store. This is due to the AppStore fees and the lack of time. However, the website is available as a Progressive Web App (PWA) and can be installed on both iOS and Android devices. This will make it 'feel' like a native application with ability to set it as a home screen icon and receive push notifications.

The installation is dependent on device type and operating system. However, the following should roughly apply to most devices.

iOS: To install the PWA on iOS devices, open Safari and navigate to the web app URL. Tap the Share button and select "Add to Home Screen". Click Add on the top right corner. The steps are shown in the figure below.

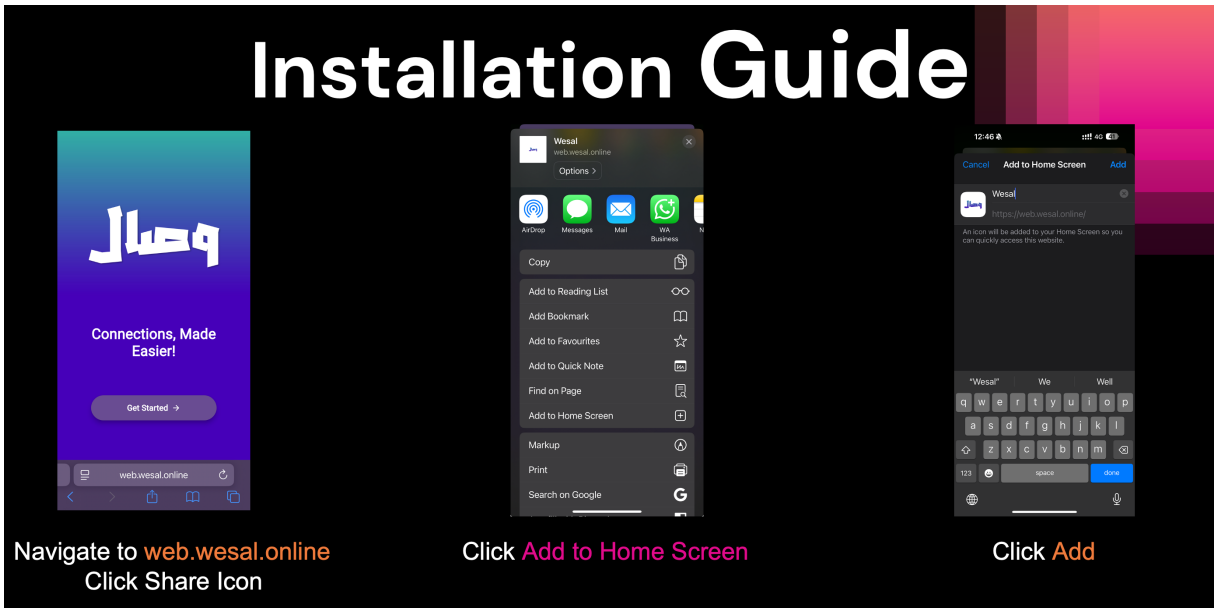

Figure 4: iOS PWA Installation Steps

Android: To install the PWA on Android devices, open Chrome and navigate to the web app URL. Tap the menu (three dots or dashes) and select "Add to Home screen" or "Install App". Follow the installation prompts.

The app remains accessible through the browser, however, due to limitations in Safari and iOS and other operating systems, it will not be possible to receive push notifications if the app is not installed as prescribed above.

Once the app is installed, you can open it from the home screen like any other app. The app will prompt you to log in or register if you are a new user.

5.2 User Registration and Login

Wesal app is intended to be internally used by the division employees. Therefore, user registration is not straightforward. Users will require an invitation to register. Admins (Users with role=ADMIN in the database) can create invitations for new users form the settings page.

Once you have an invitation, you can register normally.

It is worth noting that the app currently only supports phone number based registration. This phone number is not used for any purpose except for user identification and authentication. SMS is not sent. The mere reason for this is to allow users to communicate to each other outside of the app if they are attending a meeting together (eg., through Whatsapp).

After the initial registration, users can log in using their phone number and password. The invitation code is single use.

5.3 Adminstring the application

Unfourtunately, due to the very limited time available to develop the application, the admin panel is very basic. It currently only allows admins to create invitations for new users.

However, technical administrators are able to manage all the information in the application through the database. We have provided a database admin interface using Adminer. This is a lightweight database management tool that allows you to manage the PostgreSQL database used by the application in the browser directly without the need for any additional software. If however you prefer to use a different database management tool, please feel free to do so!

Technical administrators should get comfortable with this for the time being as it is the only way to for example reset user passwords.

6 Development Guide

6.1 Project Structure

6.1.1 Frontend Structure

```
frontend/
          lib/
                constants/            # API endpoints
                models/              # Data models with JSON conv
                      *_models.dart
                screens/             # Main screens and pages
                      pages/           # Tab-based pages
                      *.dart
                services/            # logic & API comm.
                      auth_service.dart
                      http_service.dart
                      notification_service.dart
                      *.dart
                utils/               # Helper functions
                widgets/             # Reusable UI components
                main.dart            # App entry point
          assets/                  # Static assets
          android/                # Android-specific configuration
          ios/                    # iOS-specific configuration
          web/                    # Web-specific configuration
          pubspec.yaml            # Dependencies and configuration
```

6.1.2 Backend Structure

```
backend/
          src/main/java/online/wesal/wesal/
                config/             # Configuration classes
                      FirebaseConfig.java
                      JwtUtil.java
                      SecurityConfig.java
                      *.java
                controller/         # REST API controllers
                      AuthController.java
                      PostController.java
                dto/                # Data Transfer Objects
                entity/             # JPA entities
                repository/         # Spring Data repositories
                service/            # Business logic services
                WesalApplication.java
          src/main/resources/
                application.properties
                application.yml
                firebase-service-account.json
          pom.xml                # Maven dependencies
```

6.2 Continuous Integration / Continuous Deployment

A simple GitHub Action CI/CD script is used to continuously deploy the application to a production VPS via SSH. It uses the recommended Method (1) of Running. So after each push to main in the repository, the server, API, and front end are all updated in a matter of seconds. The Workflow is defined in the '.github/workflows/deploy.yml' file.

6.3 Key Development Commands

6.3.1 Flutter Commands

```
# Development
flutter run                    # Run in development mode
flutter run -d chrome          # Run in web browser
flutter hot-reload             # Hot reload changes
flutter hot-restart            # Hot restart application

# Building
flutter build web --release    # Build for web deployment
flutter build apk --release    # Build Android APK
flutter build ios --release    # Build iOS app

# Testing and Quality
flutter test                   # Run unit tests
flutter analyze                # Static analysis
flutter format .               # Format code

# Dependencies
flutter pub get                # Install dependencies
flutter pub upgrade            # Update dependencies
flutter clean                  # Clean build artifacts
```

6.3.2 Spring Boot Commands

```
# Development
./mvnw spring-boot:run           # Run application
./mvnw clean compile             # Compile source code
./mvnw test                      # Run tests

# Building
./mvnw clean package             # Build JAR file
./mvnw clean install             # Install to local repository

# Database
./mvnw flyway:migrate            # Run database migrations (if
    configured)
./mvnw jpa:generate              # Generate JPA metamodel
```

7 Testing

7.1 Frontend Testing

To test any changes to the front-end locally, you can utilise Chrome Debug service. This works particularly well with Visual Studio Code. To do this, you can run the following command in the terminal:

```
flutter run -d chrome
```

This will open a new Chrome window with the app running. You can then make changes to the code and see the changes in the browser.

7.2 Backend Testing

To test any changes to the backend locally, you can, after implementing the changes and re-running the application, head to the API documentation at `http://localhost:8080/docs` to see the changes and "try them" using the try it feature. You need to provide the authorization token at the top in the first time.

8 Common Issues and Troubleshooting

8.1 Docker Issues

8.1.1 Port Conflicts

Warning

Problem: Port already in use errors
Solution: This is by far the most likely issue to encounter if you are running the system on a used machine. This is because the ports used by the application are already in use by other services. You have two options to resolve this issue if you wish not to use a VM or abstract further:

1. Stop conflicting services. You can see which service is using a specific port by running `lsof -i :<port-number>` on the host machine.
2. Change port mappings in `docker-compose.yml`. The error message will indicate which port is conflicting. Simply change that port in compose file.

8.1.2 Container Build Failures

Warning

Problem: Docker containers fail to build, or changes are not being reflected
Solution: This can and does happen because of the heavy caching Docker does. Most of the time, this can be solved easily by forcing rebuilding without a cache. In the very unlikely event it doesn't work, see the logs and debug from there.

```
# Clean Docker cache
docker system prune -a

# Rebuild without cache
docker compose build --no-cache

# Check Docker logs
docker compose logs [service-name]
```

You really shouldn't be facing any other issues with Docker. If you do, it is likely Docker is not installed correctly on your machine or you have not copied the files correctly.

8.2 Flutter Issues

8.2.1 Build Issues

Warning

Problem: Flutter build fails with dependency conflicts
Solution:

```
flutter clean
flutter pub get
flutter pub deps
```

8.2.2 General Front End Errors

Warning

Problem: Flutter build fails with various errors OR there are logical errors (builds succesfully)
Solution: In this case 'flutter analyze' is your best friend. It will tell you what is wrong with the code.
If it is logical error, see the testing section within the document which will help you debug the code. When it opens in Chrome, you can use the Console and the rest of the Inspection Tools to debug your issue.

8.2.3 Web Build Issues

Warning

Problem: Web build fails or shows CORS errors
Solution: Ensure the backend CORS configuration allows the frontend domain.

8.3 Backend Issues

8.3.1 Database Connection Issues

Warning

Problem: Cannot connect to PostgreSQL (any DB issues)
Solution: This almost always happens for two reasons: (1) The DB is actually down, (2) The installation has not been done properly (.env files and .env.properties and firebase service accounts are all required).

1. Check database is running: `docker compose ps` within the directory.
2. Ensure you have ran 'install.py' script. Review the secret files. If you have an issue with this step inspect the install.py script and do the steps manually.
3. Ensure the server can actually see the db: `docker exec -it <db-container-name> ping <db-host>` eg Firewall

That said, if you did change some entity in the codebase, the server will automatically try to migrate the DB to the version you edited. However, this is not always possible (conflicting change). In this case, simply remove the table from the database and it will be recreated to the latest code (Don't forget to keep your data!)

8.4 Website Issues

8.4.1 JWT Authentication Issues

Warning

Problem: Authentication / Requests fail with 403 errors
Solution: This most of the times happen when the token has expired but Flutter for some reason did not sign you out. The code is currently configured to automatically sign out when the token expires. That said, the solution is very simple just invalidate all the caches and storage.

8.4.2 Firebase Notification Issues

Warning

Problem: Not receiving notifications
Solution: Make sure you have followed the installation steps in this guide as both the front end and backend require a Service Account from Google Firebase to work. If was working but stopped working that is because (1) the user disabled the notifications, (2) the user skipped the notifications accepting in creation of the account.

8.4.3 Seeing an old version of the app

Warning

Problem: User sees an old version of the app or changes are not reflected
Solution: Unfourtunately, this is one of the limitations of PWAs. Because this is not actually installed as a real app (for the time being), it is treated as a normal website. Browsers cache (save a version of) the app to enhance performance and reduce loading times. This is especially true for Safari on iOS devices. Safari caches aggressively and does not always update the app when changes are made.
One can confirm the issue by opening the app in an incognito window. Long term solution for this issue includes either releasing the app to the App Store or using Versioning and Updating Service Worker on a newer version in JS.
The work around currently however is to simply clear the browser cache. As this problem is notable on iOS devices, the steps are shown below for iOS:
iOS: Delete the current application. Open Settings ¿ Safari ¿ Advanced ¿ Website Data ¿ Search for "wesal.online" or the hostname of the app ¿ Swipe left and tap Delete. Reinstall the application.

8.5 Backend Optimization

- Use database indexing for frequently queried fields
- Implement connection pooling for database connections
- Use caching for frequently accessed data
- Optimize JPA queries and avoid N+1 problems
- Implement pagination for large result sets

9 Security Considerations

9.1 Authentication Security

- The project uses strong JWT secrets
- sensitive tokens and information are stored securely in the device using `flutter_secure_storage` to avoid other applications / user from accessing it (encrypted within the device)

- Both front end and the backend validates all user input. Double checking happens in each and every endpoint.

9.2 API Security

- HTTPS is highly suggested for all API communication
- You are also suggested to enforce stricter CORS restrictions appropriately
- The server error handling without information disclosure

9.3 Database Security

- Use environment variables and secret files (ignored) for database credentials
- Database connection encryption
- Passwords are not stored in plaintext but are hashed using Bycrypt.
- Proper backup is recommended for production databases