# Setup Guide

This guide helps you get the FailFeed service up and running on your local machine.

## What You Need

- **Java 25 or higher** - Check with `java -version`
- **Git** (if cloning from repository)
- **VS Code** (recommended) with these extensions:
  - Extension Pack for Java
  - Spring Boot Extension Pack
  - Gradle for Java

## Getting Started

### 1. Get the Code

If you have the repository:

```
git clone <repository-url>
cd failfeed-service
```

Or if you have the files locally, just navigate to the project directory.

### 2. Open in VS Code

Open the project folder in VS Code. The extensions should install automatically if prompted.

### 3. Verify Java Installation

Open a terminal in VS Code and run:

```
java -version
```

You should see Java 25.x.x. If not, install Java 25 from Oracle or Adoptium.

## Building and Running

### Using Gradle Wrapper (Recommended)

The project includes Gradle wrapper, so you don't need to install Gradle separately.

**On Windows:**

```
gradlew.bat build
gradlew.bat bootRun
```

**On Mac/Linux:**

```
./gradlew build
./gradlew bootRun
```

### Using System Gradle (If installed)

```
gradle build
gradle bootRun
```

### Running Tests

```
# Windows
gradlew.bat test

# Mac/Linux
./gradlew test
```

## Starting the Service

After running `bootRun`, you should see:

```
Started ServiceApplication in X.XXX seconds
```

The service will be available at: **http://localhost:8080**

## Testing the Setup

### Create a Test User

```
curl -X POST "http://localhost:8080/users/create?name=Ciarann"
```

You should get a response like:

```
{
  "id": 1,
  "name": "Ciarann",
  "followingIds": []
}
```

### Try More Endpoints

```
# List all users
curl http://localhost:8080/users

# Create another user
curl -X POST "http://localhost:8080/users/create?name=another_user"

# Make first user follow second user
curl -X POST "http://localhost:8080/users/1/follow/2"

# Create a post
curl -X POST "http://localhost:8080/posts/create?userId=2&message=Hello from user 2!"

# Get user 1's feed
curl http://localhost:8080/posts/feed/1
```

## Configuration

### Default Settings

The service uses these defaults (change in `src/main/resources/application.properties`):

```
# Server
server.port=8080

# Database
spring.datasource.url=jdbc:h2:mem:failfeeddb
spring.datasource.username=sa
spring.jpa.hibernate.ddl-auto=update

# H2 Console (for debugging)
spring.h2.console.enabled=true
spring.jpa.show-sql=true
```

### Database Console

When the service is running, you can access the H2 database console at: **http://localhost:8080/h2-console**

Connection details:

- **JDBC URL**: `jdbc:h2:mem:failfeeddb`
- **Username**: `sa`
- **Password**: (leave blank)

## Common Issues

### Java Version Problems

**Error**: `UnsupportedClassVersionError`
**Solution**: Make sure you're using Java 25. Update JAVA_HOME environment variable.

### Port Already in Use

**Error**: `Port 8080 was already in use`
**Solution**:

1. Stop other services using port 8080, or
2. Change the port in `application.properties`: `server.port=8081`

### Gradle Issues

**Error**: Gradle build fails
**Solution**:

1. Delete `.gradle` folder and try again
2. Run `gradlew.bat clean build` (Windows) or `./gradlew clean build` (Mac/Linux)

### H2 Database Issues

**Problem**: Can't connect to H2 console
**Solution**:

- Make sure service is running
- Use correct JDBC URL: `jdbc:h2:mem:failfeeddb`
- Check H2 console is enabled: `spring.h2.console.enabled=true`

### Memory Issues

**Error**: OutOfMemoryError during build
**Solution**: Increase Gradle memory:

```
# Windows
set GRADLE_OPTS=-Xmx2g
gradlew.bat build

# Mac/Linux
export GRADLE_OPTS="-Xmx2g"
./gradlew build
```

## Project Structure

After setup, your project should look like:

```
failfeed-service/
├── build.gradle
├── settings.gradle
├── gradlew
├── gradlew.bat
├── src/
│   └── main/
│       ├── java/com/failfeed/service/
│       │   ├── ServiceApplication.java
│       │   ├── controller/
│       │   ├── service/
│       │   ├── repository/
│       │   ├── model/
│       │   ├── dto/
│       │   └── exception/
│       └── resources/
│           └── application.properties
└── README.md
```

## Next Steps

1. **Explore the API** - Use the test commands above
2. **Check the database** - Use H2 console to see created tables
3. **Read the code documentation** - See `CODE_DOCUMENTATION.md`
4. **Try the endpoints** - See `Test_URLS.md` for example requests

## Development Tips

### Hot Reloading

The service supports hot reloading. When you modify code and save, the service will automatically restart (if using Spring Boot DevTools).

### Database Persistence

The H2 in-memory database resets when the application stops. For persistent storage, switch to a file-based database or PostgreSQL.

### Logging

The service logs SQL queries to console when `spring.jpa.show-sql=true`. This is useful for debugging but should be disabled in production.

## Building for Production

To create a production JAR:

```
./gradlew bootJar
# or
gradlew.bat bootJar
```

The JAR will be created in `build/libs/` directory.