GitHub Activity Web Application - Project Documentation

1. Project Overview

Project Name: GitHub Activity Web Application
Live Link: https://github-activity-p5g4.onrender.com/
Purpose: - To fetch and display recent activity of any GitHub user in a web interface. - Practice API integration, Spring Boot development, JSON handling, and deployment. - Provide a simple, user-friendly UI to interact with GitHub API.

Key Features: 1. Fetch GitHub user activity (commits, issues, starred repos, etc.) 2. Display activity in a clean, structured web interface 3. Handle errors gracefully (invalid usernames, API failures) 4. Easy deployment using Render

2. Tech Stack Used

Layer	Technology	Purpose
Backend	Java 17, Spring Boot	Core application logic, REST API calls to GitHub, handling requests and responses
HTTP Client	Java HttpClient	Fetch user events from GitHub API
Frontend	Thymeleaf, HTML, CSS	Rendering the user interface and activity list
Build Tool	Maven	Project management and dependency management
Deployment	Render	Hosting the application online

3. Project Structure

```
github-activity/
├── src/
│   ├── main/
│   │   ├── java/com/abhinav/githubactivity/
│   │   │   ├── controller/GithubController.java
│   │   │   ├── service/GithubService.java
│   │   │   └── GithubActivityApplication.java
│   │   └── resources/
│   │       ├── templates/index.html
│   │       └── application.properties
├── pom.xml
├── mvnw
├── mvnw.cmd
├── .mvn/
└── render.yaml
```

Explanation: - `GithubActivityApplication.java`: Main class to run Spring Boot. - `GithubController.java`: Handles web requests, accepts GitHub username input. - `GithubService.java`: Fetches data from GitHub API and processes JSON. - `templates/index.html`: Thymeleaf template for frontend UI. - `application.properties`: Spring Boot configuration (e.g., server port). - `pom.xml`: Maven dependencies and build configurations. - `render.yaml`: Configuration for deployment on Render.

---

4. How the Application Works

1. User opens the web application and enters a GitHub username.
2. `GithubController` receives the username and calls `GithubService`.
3. `GithubService` uses Java HttpClient to call the GitHub API endpoint:

   ```
   https://api.github.com/users/<username>/events
   ```

4. JSON response is parsed, extracting key activities (pushes, issues, starred repos).
5. Data is sent back to the frontend (`index.html`) using Thymeleaf.
6. Frontend renders the activities in a clean list.
7. If the username is invalid or API fails, a friendly error message is displayed.

---

5. Features Implemented

- Fetch and display:
- Commits pushed by the user
- Issues opened
- Repositories starred
- Error handling:
- Invalid usernames
- API rate limits
- High-fidelity frontend with simple design for easy readability

---

6. How to Run the Project Locally

Prerequisites:

- Java 17 installed and JAVA_HOME configured
- Maven installed (or use Maven wrapper `mvnw`)
- Git installed

Steps:

1. Clone the repository:

```
git clone https://github.com/abhinavsinha2002/github-activity.git
```

2. Navigate to the project directory:

```
cd github-activity
```

3. Build the project:

```
./mvnw clean package -DskipTests  # or 'mvn clean package -DskipTests' if no wrapper
```

4. Run the project:

```
java -jar target/github-activity-0.0.1-SNAPSHOT.jar
```

5. Open a browser and visit:

```
http://localhost:8080
```

6. Enter a GitHub username to see recent activity.

---

7. How to Deploy on Render

Using Docker (Recommended)

1. Ensure `render.yaml` exists in the repo root:

```
services:
  - type: web
    name: github-activity
    env: docker
    plan: free
    buildCommand: "./mvnw clean package -DskipTests"
    startCommand: "java -jar target/github-activity-0.0.1-SNAPSHOT.jar"
    envVars:
      - key: PORT
        value: 8080
```

2. Ensure a `Dockerfile` exists (optional if using Render Docker env). Example:

```
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY src ./src
RUN chmod +x mvnw
RUN ./mvnw clean package -DskipTests
EXPOSE 8080
CMD ["java", "-jar", "target/github-activity-0.0.1-SNAPSHOT.jar"]
```

3. Push all changes to GitHub.
4. On Render dashboard:
5. Click **New → Web Service → From GitHub**
6. Select repo → Render detects `render.yaml` → Deploy
7. Render builds the app and provides a live URL.

Live deployed URL: https://github-activity-p5g4.onrender.com/

---

8. Notes & Best Practices

- Free Render apps sleep after 15 minutes of inactivity. First request wakes it automatically.
- Paid plans allow 24/7 uptime.
- Always keep `pom.xml` at repo root for easy detection.
- Use Maven wrapper (`mvnw`) for consistent builds.
- Handle GitHub API rate limits (currently limited for unauthenticated requests).

---

9. Future Enhancements

- Filter activity by type (commits, issues, stars).
- Display activity timestamps.
- Cache API responses for faster loading.
- Add user avatar and profile information.

• Enable authentication to fetch private activities.
• Improve UI/UX for better mobile responsiveness.

Project maintained by: Abhinav Kumar Sinha
Date: October 2025