AS01: Setup musicFinder 🎸

Version Control, CI/CD Pipeline, Dockerisation, and Basic Application Setup

Introduction:

This assignment will introduce you to the essential components of modern software engineering: version control, continuous integration/continuous delivery (CI/CD), Dockerization, and basic application setup. The goal is to familiarize you with tools and practices such as **Git**, **GitHub**, **GitHub Actions**, **Docker**, and **Spring Boot**.

By the end of this assignment, you will have:

- Set up and used **Git** and **GitHub** for version control and collaboration.
- Implemented a basic **CI/CD pipeline** to automate the build process.
- Containerised a Spring Boot application using **Docker**.
- Successfully run a basic **Spring Boot** application locally.

▼ Task 1.1: Set Up Git and GitHub Repository

Goal:

The objective of this task is to introduce you to **version control** and collaborative development using **Git** and **GitHub**. You will create a repository, manage branches, commits, and pull requests, and understand how teams collaborate on software projects.

Instructions:

1. **Fork and Clone the Repository**:
    - Fork the **musicFinder** repository from GitHub.
    - Clone it to your local machine:

```
git clone https://<the clone url after you accept
the invite>
```

```
cd musicFinder
```

2. **Set Up Git Locally**:
   - Ensure Git is installed and configured with your username and email:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@exampl
e.com"
```

3. **Branching and Commit**:
   - Create a new branch for your changes:

```
git checkout -b feature/setup
```

   - Add some changes (like updating the README or adding a new file).
   - Stage, commit, and push your changes:

```
git add .
git commit -m "Added initial setup"
git push origin feature/setup
```

4. **Collaboration**:
   - Open a pull request (PR) on GitHub to merge your branch into the main branch.
   - After review, merge the PR.

💡 **Submission Instructions:**

- Ensure you've created a **GitHub repository** for the project.
- Make sure the repository has:
  - At least one branch other than `main`.
  - A clear commit history showing multiple meaningful commits.
  - At least one pull request (PR) demonstrating collaborative development.

What the GitHub Action will check:

- The presence of branches.
- The number of commits.
- The existence of a merged pull request.

What you need to do:

- Push your work to GitHub and ensure the action runs correctly.

Tips:

- Use branches for isolated development. This allows you to make changes without affecting the main codebase.
- Always pull the latest changes from the main branch before creating new branches.

Helpful Link:

- Git Basics: Branching and Merging

▼ Task 1.2: Create a Basic CI/CD Pipeline Using GitHub Actions

Goal:

The goal of this task is to introduce you to **Continuous Integration (CI)** using **GitHub Actions**. You'll create a pipeline that automatically builds your

project whenever you push code to the repository.

Instructions:

1. **Create a Workflow Directory**:
   - Inside your repository, create a directory for GitHub Actions workflows:

     ```
     mkdir -p .github/workflows
     ```

2. **Define the CI/CD Workflow**:
   - Create a workflow file that automates the build process whenever you push changes or open a pull request.
3. **Trigger the Pipeline**:
   - Ensure the pipeline is triggered on every push to the repository.

💡

Submission Instructions:

- Make sure you have created a **CI/CD pipeline** using **GitHub Actions**.
- The pipeline should:
  - Build the application on push or pull requests.

What the GitHub Action will check:

- The existence of a valid `.github/workflows/ci.yml` file.
- Whether the CI/CD pipeline runs and completes successfully on push events.

What you need to do:

- Ensure that the pipeline runs when you push your work to the repository.

Tips:

- Your pipeline should focus on building the project using **Maven** to verify that the build is successful.

- Check the **Actions** tab in your GitHub repository to see the results of your CI pipeline.

Helpful Link:

- GitHub Actions: Getting Started with CI

▼ Task 1.3: Docker Containerisation

Goal:

This task introduces you to **Docker**, a tool that helps package applications into containers. You will write a Dockerfile to package the **musicFinder** application and run it inside a Docker container.

Instructions:

1. **Write a Dockerfile**:
    - Write a Dockerfile to define the environment and how the **musicFinder** application should run.
2. **Build the Docker Image**:
    - Build the Docker image based on your Dockerfile.
3. **Run the Docker Container**:
    - Run the application inside the Docker container and ensure that it is accessible through `localhost`.

💡 **Submission Instructions:**

- Ensure that a **Dockerfile** is present in your repository.
- The Dockerfile should:
  - Successfully build a Docker image for the project.
  - Allow the application to run in a container and be accessible on port 8080.

What the GitHub Action will check:

- That the Dockerfile builds successfully.
- That the Docker container runs and is accessible on the required port.

What you need to do:

- Ensure your Dockerfile is complete and functional, and that the action runs when you push.

Tips:

- Pay attention to **port mapping** when running the Docker container (`p` flag).
- Test your Dockerfile by manually running the app locally before building the container.

Helpful Link:

- Dockerfile Best Practices
- Beginner's Guide to Docker

▼ Task 1.4: Basic Application Setup

Goal:

The goal is to ensure that the **musicFinder** application is working correctly by running it locally using **Spring Boot** and testing its API functionality.

Instructions:

1. **Run the Application**:
   - Use Maven to run the **musicFinder** application locally:

     ```
     mvn spring-boot:run
     ```

2. **Test the API**:
   - Access the API to fetch song lyrics using the provided endpoint:

     ```
     http://localhost:8080/song/{artist}/{name}
     ```

3. **Verify the Functionality**:
   - Input a real artist and song title to test if the application fetches the correct lyrics.

💡 **Submission Instructions:**

- The **musicFinder** application must run locally, and the API must respond correctly when queried.
- You should:
  - Ensure that the application builds and runs using Maven.
  - Test the `/song/{artist}/{name}` API endpoint to ensure it returns valid data.

What the GitHub Action will check:

- That the application builds successfully using Maven.
- That the API endpoint responds with valid data when queried.

What you need to do:

- Ensure the application runs and responds correctly when the GitHub Action is triggered by your push.

Tips:

- If the application fails to run, check the **logs** for detailed error messages.

- Make sure the application is using the correct version of **Java** and **Maven**.

Helpful Link:

- Spring Boot Documentation

Disclaimer:

Please note that this assignment will be evaluated using **GitHub Actions**, an automated tool that runs scripts to validate your submissions. The following points outline how the evaluation will work:

1. **Automated Testing**:
   - Each time you push your code to GitHub, **GitHub Actions** will automatically validate your work based on the requirements for each task (e.g., CI/CD pipeline setup, Docker containerization, application functionality).
   - It is your responsibility to ensure that the automated tests pass before the assignment deadline.
2. **No Manual Submission**:
   - You do not need to manually submit screenshots, videos, or reports. The automated pipeline will check all the required criteria.
   - You must regularly push your work to GitHub to trigger the validation process.
3. **Monitoring Your Progress**:
   - You can view the status of your submission by visiting the **Actions** tab in your GitHub repository. The pipeline will show whether each task passed or failed the checks.
   - If any checks fail, you should review the error logs, fix the issues, and push the changes to re-trigger the tests.
4. **Final Evaluation**:
   - Your grade will be based on the successful completion of the tasks as verified by the **GitHub Actions** workflows.
   - Be aware that the automated pipeline will strictly follow the assignment criteria. Ensure your repository, files, and configurations meet the

requirements.

5. **Deadline and Pushes**:
    - Ensure that all your work is committed and pushed before the submission deadline. The last successful build and test before the deadline will be considered for grading.

⚠ Important Note:

> 💡 Failure to push your work or fix failed checks before the deadline may result in an incomplete submission and impact your grade. Regularly check the status of your pipeline and address any issues promptly.