Study on Continuous Integration Testing

Continuous Integration (CI) testing is a crucial practice in software development that ensures code quality and reliability by automating the testing process. In this document, we will explore how to implement CI testing using GitHub Actions as our CI tool in the context of our swe group project. Our project is built using the MERN (MongoDB, Express.js, React.js, Node.js) stack, and we used Jest for unit testing.

Step 1- Choose GitHub Actions as CI Tool:

GitHub Actions is a powerful CI/CD service provided by GitHub that integrates seamlessly with our version control system (Git). It allows us to automate build, test, and deployment processes directly from our GitHub repository.

Step 2- Understanding Branches:

In our project, we have multiple branches, including `master`, `main`, `fiha`, `fiha-02`, `munira`, `munira2.0`, `tazim`, `tazim1.0, `palash`, `sakib-01`, and `sakib-02`. Each branch represents a different development environment or feature branch where we work on specific tasks.

Step 3- Setting Up CI Pipeline:

We will create a CI pipeline using GitHub Actions to automate the testing process whenever code changes are pushed to the repository. This pipeline will consist of the following stages:

- **Checkout Repository:** This step ensures that the latest code changes are retrieved from the repository.
- **Install Dependencies:** Install project dependencies using npm or yarn.
- **Run Tests:** Execute unit tests using Jest to ensure code correctness.

Step 4- Configuring CI Workflow:

We will create a YAML configuration file (`.github/workflows/main.yml`) in the root directory of our project repository to define the CI workflow. This file will specify the steps to be executed in the CI pipeline. To create a YAML configuration file in our project repository, we will follow these steps:

1. Navigate to the root directory of our project repository.

2. Create a new file named .github/workflows/main.yml. We can do this using a text editor or through our command line interface.
3. Open the main.yml file in preferred text editor.
4. Write the YAML configuration for our CI workflow in this file. Below is an example of a basic CI workflow configuration:

```yaml
name: Continuous Integration

on:
  push:
    branches:
      - master
      - main
      - fiha
      - fiha-02
      - munira
      - munira2.0
      - tazim
      - tazim1.0
      - palash
      - sakib-01
      - sakib-02
  pull_request:
    branches:
      - master
      - main
      - fiha
      - fiha-02
      - munira
      - munira2.0
      - tazim
      - tazim1.0
      - palash
      - sakib-01
      - sakib-02

jobs:
  build:
    runs-on: windows-latest  # Specify the Windows operating system


    env:

      DATABASE_URL: ${{ secrets.DATABASE_URL }}

      API_KEY: ${{ secrets.API_KEY }}
    steps:
      - name: Checkout Repository
        uses: actions/checkout@v2

      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'  # Specify the Node.js version we're using

      - name: Install Dependencies
        run: npm install
      - name: Run Tests
        run: npm test
```

5. Save the changes to the main.yml file.
6. Commit the main.yml file to your repository's master branch.

```
git add .github/workflows/main.yml  # Replace with the actual path if needed
git commit -m "Add CI workflow configuration"
git push origin <branch-name>  # Replace <branch-name> with the name of your branch
```

This configuration file specifies that the CI pipeline will be triggered on every push to any of the specified branches (`master`, `main`, `fiha`, etc.). It defines three steps: checking out the repository, installing dependencies, and running tests using Jest.

Step 5- Monitoring CI Results:

Once the CI pipeline is set up, we can monitor its execution by navigating to the "Actions" tab in our GitHub repository. We should review the test results and logs generated by GitHub Actions to identify any failures or issues.

Step 6- Integrating CI with Pull Requests:

To ensure that code changes are thoroughly tested before merging into the main branch, we can configure GitHub Actions to run tests automatically on pull requests. This allows us to review the test results and ensure that their changes do not introduce any regressions.

By implementing CI testing using GitHub Actions in our group project, we can ensure code quality, reliability, and consistency throughout the development process. CI testing helps identify and fix issues early, leading to faster release cycles and a more robust application.