College of Computer Studies | Software Technology Department

MCO2. Continuous Integration Plan

Project Profile	
Project Team	AWS CodeBuild
Team Members	Berjamin, Sandra Angela E. - Product Owner Cua, Lander Peter E. - Scrum Master Gaba, Jacob Bryan B. - Quality Assurance (QA) / Tester Gonzales, Mark Edward M. - Developer Lee, Hylene Jules G. - Developer Mata, Angeli Dianne F. - Designer Ong, Phoebe Clare L. - Designer Racoma, Ian Angelo T. - Quality Assurance (QA) / Tester
Application Name	Powerzone Product Management
Git Repository	https://github.com/STSWENG-T1-AY2122-AWS-CodeBuild/powerzone-inventory
Project Technical Stack *What programming languages or JS frameworks are you using?* [1]	
Front-end	**Bootstrap** Bootstrap will be used as the primary CSS framework. Bootstrap was primarily chosen for the variety of browsers and interfaces it supports, its compatibility with JavaScript, and the widely available documentation and support for its components (Gupta, 2017). **Handlebars** Handlebars will be used as the template engine. The team chose to use Handlebars for its logic-less nature, in keeping with the MVC architecture, and its support for compilation rather than interpretation, which may optimize navigation between pages that use similar front-end elements (Handlebars, 2020).
Backend	**Node.js** Node.js will be used as the server environment, primarily due to its use of JavaScript as its programming language and the availability of essential tools such as package managers (Malvi, 2021). **Express.js** Express.js will be used as the back-end framework. Primarily, Express.js was selected for its adaptability with both Node.js and the MVC architecture, as well as its potential for swift processing times due to its asynchronous callback feature (Panchal, 2021); this is crucial for the web application, as the number of pertinent records needed for the application features may increase quickly and become a processing bottleneck.

[1] The descriptions, with the exception of those for the bundler, task runner, and other testing tools (namely SinonJS and Mockingoose) are lifted from the Team Standards.

Unit Testing Framework *(i.e. Jest, Jasmine, Mocha, Chai, etc.)*	**Mocha** As the team decided to use Node.js for its runtime environment, Mocha was chosen for its general compatibility with Node.js, especially its support for asynchronous testing, and its use of logical operators that the team believes would make unit tests more readable, which is particularly advantageous considering the additional role of unit testing as a method of documentation (Borys, 2021). **Chai** Chai was chosen as one of the most popular assertion libraries used alongside Mocha. Moreover, Chai has extensions for testing jQuery-specific assertions (Chai jQuery), which will be useful in writing unit tests for functions that involve the alteration or manipulation of Document Object Model or DOM elements. **<u>Other Testing Tools</u>** **SinonJS** In order to support more comprehensive unit testing and evaluate sections of the code that require connections to networks or databases, SinonJS was chosen as the mechanism for creating test spies, stubs, and mocks (Warugu, 2020). The tool was primarily chosen for its compatibility with other testing frameworks, making it readily incorporable into the current tech stack. **Mockingoose** As the team makes use of Mongoose for creating and maintaining database models, Mockingoose was chosen as the testing tool for mocking the models to be made. Apart from its compatibility with Mongoose, Mockingoose was also chosen for its high quality and security as well as its relatively prevalent support (Kandi, 2021).
Bundler / Task Runner *(i.e. webpack, gulp, grunt, bower)*	**Webpack** Webpack is a module bundler for JavaScript that creates and manages file dependencies to efficiently bundle code into more compact sections while ensuring that code sections are grouped together with the assets they require to execute properly (Srivastava, 2018), such as grouping JavaScript files and their associated HTML pages together. Apart from its compatibility with the Node.js runtime environment, the team also opted for Webpack because of its support for various loaders and plugins, making it possible to bundle non-JavaScript files as well using the same tool. **Github Actions Runner for Running Tasks** Github Actions is a CI/CD environment that Github provides with every repository. It has multiple features that can be assigned to various events that can be performed in Github, such as committing a new file into a branch or creating pull requests from an existing branch to another. Github Actions also has the capacity to help automate testing builds in the repository. This is done by providing contributors to the repository with a runner, particularly a Docker runner, that is capable of creating containers in which the environment for the application can be simulated. This allows for tests to be performed on the selected branch with minimal risk of modifying the existing files of the branch (GitHub Documentation, 2021).
Others *(if you have other tools that you use with your project)*	**MongoDB and Mongoose (Database Management)** MongoDB will be used in storing the data required for the web application's functionalities. The team decided to use MongoDB due to the ease with which its created databases can be accessed and queried using JavaScript as well as the ability to store a wide variety of data types due to its native BSON format (MongoDB, 2021). Mongoose will be used as the object data modeling tool. In addition to MongoDB, the team decided to use Mongoose for the abstraction it offers over the native Node Driver of MongoDB, as well as its implementation of models that lends itself well to the MVC architecture (Medlock, 2017). **Selenium (Automation Testing)** Selenium will be used as the primary tool for automation testing. The team decided to use Selenium for its cross-browser compatibility and its mouse cursor and keyboard simulation features (Chowdhury, 2019). **Heroku (Deployment)** The web application will be deployed on Heroku. The team chose to make use of this platform as a service (PaaS) due to the range of functionalities available on its free tier and its handling of server management components (Clark, n.d.).

	GridFS (File Storage) Since Heroku has an ephemeral file system, GridFS will be used for the persistent storage of files. Primarily, the team intends to make use of GridFS to store any large files, such as image assets for the application, that exceed the binary JSON (BSON) document size limit (MongoDB, 2021).

Continuous Integration Plan

Current Project Workflow

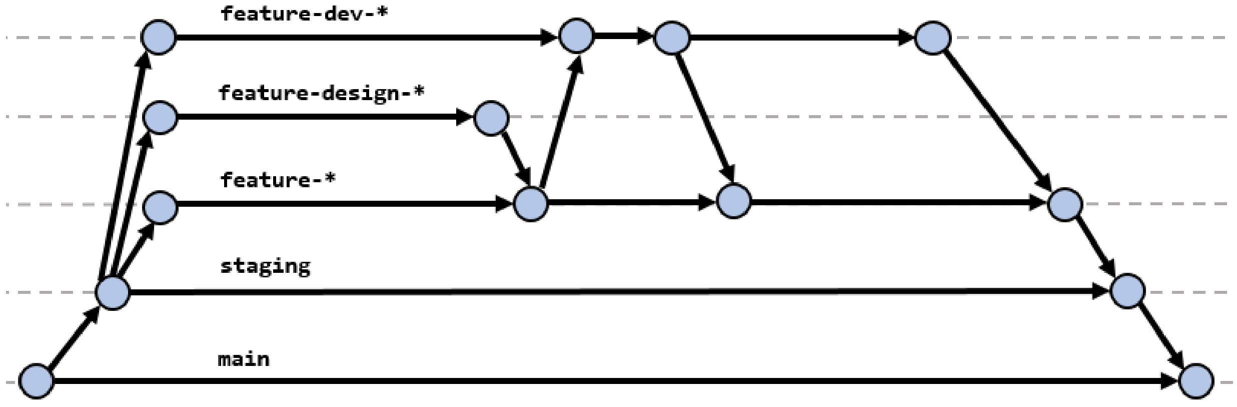

Figure 1. Visualization of the Repository Structure

The designated integration branch in the GitHub repository is the ***`staging` branch.***

At the end of the development of every feature *(that is, upon completion and successful unit testing of both back-end and front-end components), one of the team's developers will create a pull request from the `feature-*` branch into the `staging` branch; this will be reviewed and approved by the other developer. It is imperative — and part of the automated build plan — for the pull request to pass all the unit tests written by the developers before it is merged into the `staging` branch.*

The quality assurance (QA) team will conduct both automated and manual tests in the `staging` branch *in order to test the newly integrated feature and examine its behavior against the rest of the already-implemented features. Issues raised by the QA will be tracked using GitHub Issues; once the designers or developers have fixed the issues, they will submit a pull request to the `feature-*` branch, then into the `staging` branch after undergoing code review and unit testing. Once the QAs have tested the fix alongside the other features and have closed the issue, the feature will be marked as "PASSED."*

At the end of each sprint, all the features in the `staging` branch that have been marked as "APPROVED" by the QA and "DONE" by the Scrum master will be included in the created pull request from the `staging` branch to the `main` branch, which serves as the production branch for the project.

Build Automation

The module bundler chosen to help package the application is Webpack. Webpack runs on Node.js and is responsible for bundling the application into a more compact format in order to reduce the overall file size of the application and help in optimizing its runtime during deployment. In addition to this, bundling the application can also ensure that the modules of the application are contained within a single file; this allows the application to have more integrity and become less prone to any accidental modifications or errors related to missing modules or components (Mittal, 2020).

One of the frameworks used to configure the unit testing scripts to help automate the build process is Mocha. This, along with the Chai library, are responsible for executing the unit test scripts written by the developers in order to verify that the functions they have created work as intended. In the `package.json` file, the following script was added to run the unit tests:

```
"test": "mocha --recursive --exit --timeout 5000"
```

The `--recursive` flag ensures a recursive search of all the subdirectories inside the `test` directory. The --exit flag ensures that Mocha will stop all testing-related processes at the completion of all the unit tests, regardless of the results; this helps prevent problems related to errant open sockets and promises (Mocha n.d.), which may cause the system to "hang" and compromise build automation should they be left unattended. Finally, the `--timeout` flag was set to 5000 milliseconds (5 seconds) to provide an allowance for some unit tests that may exceed the 2-second default of Mocha; in this project, this has been documented for one unit test that involves database mocking.

The execution of the scripts will be handled by free runners virtually hosted by Github Actions. Given the nature of GitHub Actions, the execution and automation of jobs become simpler and more convenient for the team since runners no longer have to be hosted in local machines, saving precious time and resources (GitHub Documentation, 2019). The runners are responsible for automating the execution of the testing script whenever a new push or pull request is made in a branch. The runners are specifically configured to build the application in such a way that prevents the accidental modification of the modules folder. This ensures that no files are modified during testing and that the project's dependencies remain untouched when a new automated test is performed.

The following YAML code is the content of the YML file responsible for the workflow used for automation testing:

```
name: Node.js CI

on: pull_request

jobs:

  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Use Node.js 17.1.0
      uses: actions/setup-node@v2
      with:
        node-version: 17.1.0
        cache: 'npm'
    - run: npm ci
    - run: npm run build --if-present

  test:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Use Node.js 17.1.0
      uses: actions/setup-node@v2
      with:
        node-version: 17.1.0
        cache: 'npm'
    - run: npm ci
    - run: npm test
```

The `name` field designates the identifier of the workflow shown on the repository's actions page. The `on` field specifies the conditions for executing the YAML script, which in this case is set to trigger whenever a pull request is made on the branch containing the script. The `jobs` field is where the quality assurance team declares the jobs that the workflow executes whenever it is triggered. The `runs-on` field indicates the virtual environment (i.e., the runner) that will be used to execute the job as well as its version. The `steps` field defines the sequence of actions that the runner needs to perform. The `uses` field is used to introduce a built-in GitHub Action for the runner to execute. Lastly, the `run` field specifies a non-GitHub Action command to be executed by the runner.

At the current stage of the project, the workflow only contains these jobs; however, as the project continues and more features become completed, more tests and scripts will be integrated into the YML file as needed.

Continuous Integration Server

*The continuous integration server provider that will be used for the project is Github (**GitHub Actions**) since it comes equipped with full continuous integration and delivery (CI/CD) capabilities. Moreover, since the repository being used is already stored in GitHub, the repository can easily be accessed by Github Actions without the need for further authorizations or personal tokens. In addition, the tight integration of Github Actions with Github allows for seamless access to the Github API, making it easier and more convenient for the developers (Wickramarachchi, 2020) to use. Workflows and CI/CD pipelines to build, test, and deploy code can also be quickly created and set up within the same GitHub repository using GitHub Actions. Additionally, GitHub also conveniently uses runners from its hosted virtual environments, which removes the need for establishing self-hosted runners to execute jobs specified in the CI/CD workflow; this allows for reduced resource consumption for the team's local machines (GitHub Documentation, 2019).*

Moreover, GitHub Actions also has a marketplace for the convenient addition of third-party JavaScript extensions, such as Send Mail, which is part of the build plan, particularly for notifying the team members of test or build failures. Wickramarachchi (2020) also notes that its support for parallel and asynchronous CI/CD, which is a feature for expediting the processes in the pipeline, is not yet present in some other CI servers like Jenkins.

Build Plan

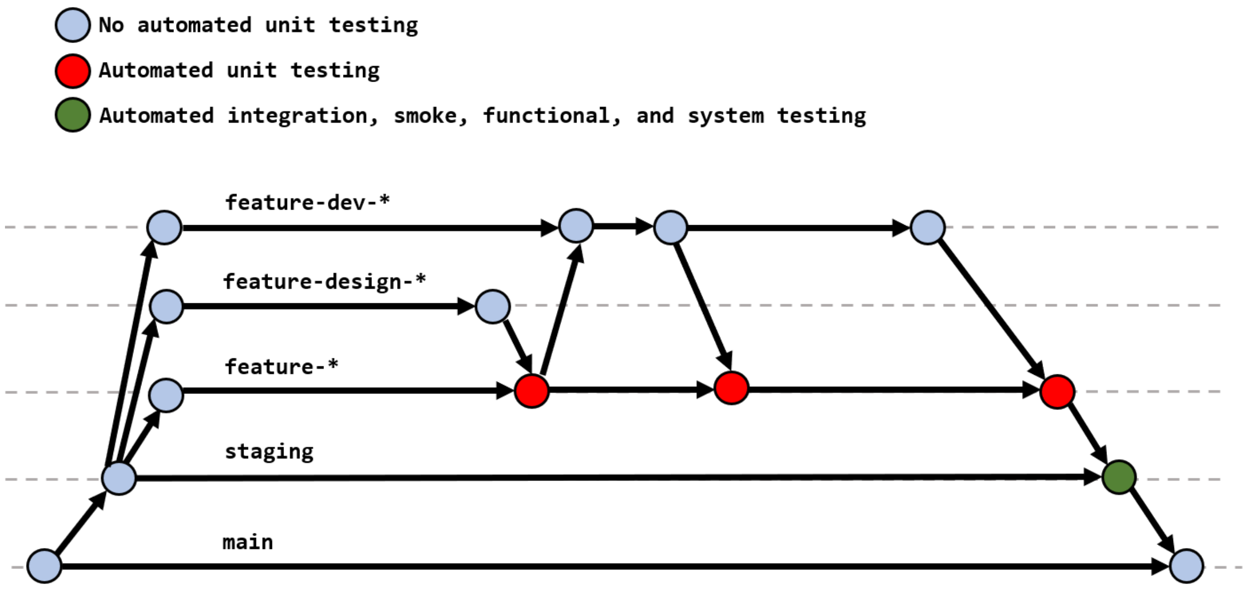
No automated unit testing
Automated unit testing
Automated integration, smoke, functional, and system testing
feature-dev-*
feature-design-*
feature-*
staging
main

Figure 2. Visualization of the Build Plan

The chosen branch for feature integration is `staging`. For every ***pull request*** *targeting the branch, it will trigger the workflow assigned to the `staging` branch to run tests on the application. The workflow is set to trigger for every pull request in order to automatically verify and keep track of when an error occurs on an application. This allows both the developers and the quality assurance team to monitor which commits to the branch cause errors and to immediately determine which version of the repository they should revert to when needed. Upon encountering errors in the build, the Send Mail GitHub Action will be used to notify the developers, designers, and QAs of an issue with the pull request, allowing the team to respond swiftly and address the concern. The Send Mail GitHub Action is a third-party JavaScript action available in GitHub Marketplace (GitHub Marketplace, n.d.).*

Pull requests will also be set as the triggers for the test in order to determine if a merge request is targeting a faulty branch. This allows the team to identify whether the cause of an error would be the current build of the application inside the branch or the changes proposed by the contributor who initiated the pull request. This is determined by checking the history of the workflow's operation. Triggering a pull request initiates two instances of the workflow process, one for the branch being merged and one for the target branch. By checking which test failed upon the initiation of the pull request, the quality assurance team can determine which of the branches was faulty upon the creation of the pull request, and it would be easier to identify which branch needs to be fixed.

The workflow for the `staging` branch consists of jobs for building and testing. Build jobs help guarantee that the application files can be compiled and built into a functioning application, while the test jobs are dedicated to automating integration, smoke, functional, and system testing. System testing, however, will only be performed once the application has reached its final stages since this type of testing focuses on testing the stability and interactivity of the different features of the system as a whole. Tests for each individual feature will be done on more focused tests such as integration and smoke testing.

As seen in Figure 2, the `feature-*` branches will also be subjected to automated testing; more specifically, these branches will make use of the unit tests prepared by the team's developers. The build and test jobs specified in the workflow of the `feature-*` branches work similarly to those of `staging` branches; however, the main difference between the workflows of these two types of branches is that the test jobs of the `feature-*` branches make use of only the unit testing scripts to verify the build, whereas the test jobs of the `staging` branch, as previously mentioned, utilize testing scripts for integration, smoke, functional, and system testing. The more relaxed rules for the `feature-*` branches reflect the fact that the purpose of these branches is to serve as an avenue for developers and designers to pass code between one another; nevertheless, this code still has to be unit-tested to prevent the addition or alteration of functionalities that may break the existing code base.

Note that the `feature-dev-*` and `feature-design-*` branches are excluded from automated testing since they are intended to be used by the developers and designers, respectively, only for quick commits or functionality experimentations.

On a final note, in order to maintain consistency and integrity in documenting version releases, the application minor version will automatically be incremented in the `package.json` file of the application upon deployment to Heroku. Since the project uses Node.js, this can be controlled by adding `npm version` as a `run` command on the YML file.

References

Borys. (2021, April 28). *Mocha vs. Jest: Comparison of two testing tools for Node.js.* Merixstudio. https://www.merixstudio.com/blog/mocha-vs-jest/

Chowdhury, A.R. (2019, January 24). *Why Selenium WebDriver should be your first choice for automation testing.* LambdaTest. https://www.lambdatest.com/blog/13-reasons-why-selenium-webdriver-should-be-your-first-choice-for-automation-testing/

Clark, M. (n.d.). *What is Heroku?* Back4App. https://blog.back4app.com/what-is-heroku

GitHub Documentation. (2019, November 6). *About GitHub-hosted runners*. GitHub. https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners

GitHub Documentation. (2021, December 1). *Understanding GitHub actions*. GitHub. https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

GitHub Marketplace. (n.d.). *Github Action Send email.* https://github.com/marketplace/actions/send-email

Gupta, A. (2017, July 25). *Pros and cons of Bootstrap & Foundation - Know what you need!* https://www.uplers.com/blog/what-are-the-pros-cons-of-foundation-and-bootstrap/

Handlebars. (2020, April 16). *When (not) to use Handlebars?* https://handlebarsjs.com/installation/when-to-use-handlebars.html

Kandi. (2021, June). *mockingoose.* https://kandi.openweaver.com/javascript/alonronin/mockingoose

Malvi, K. (2021, June 18). *The positive and negative aspects of Node.js web app development*. MindInventory. https://www.mindinventory.com/blog/pros-and-cons-of-node-js-web-app-development/

Mittal, P. (2020, August 4). *Introduction to webpack — why do we need it?* Minds Verse. https://medium.com/habilelabs/introduction-to-webpack-why-do-we-need-it-f9b8d003884d

MongoDB. (2021). *GridFS*. https://docs.mongodb.com/manual/core/gridfs/

Panchal, J. (2021, May 14). *What are the benefits of using Express.js for developing enterprise applications?* Rlogical.

https://www.rlogical.com/blog/what-are-the-benefits-of-using-express-js-for-developing-enterprise-applications/

Srivastava, U. (2018, April 10). *Webpack - why and what*. Imaginea. https://medium.com/js-imaginea/webpack-why-and-what-4948433cc2d3

Warugu, J. (2020, September 15). *How to test Node.js apps using Mocha, Chai, and SinonJS.* DigitalOcean. https://www.digitalocean.com/community/tutorials/how-to-test-nodejs-apps-using-mocha-chai-and-sinonjs

Wickramarachchi, V. (2020, December 26). *Github Actions or Jenkins? Making the Right Choice for You*. Medium. https://blog.bitsrc.io/github-actions-or-jenkins-making-the-right-choice-for-you-9ac774684c8