Powerzone Product Management

Major Course Output #1

Project Proposal and Standards

Team Number	AWS CodeBuild
Section	STSWENG – S12
Team Members	Berjamin, Sandra Angela E. Cua, Lander Peter E. Gaba, Jacob Bryan B. Gonzales, Mark Edward M. Lee, Hylene Jules G. Mata, Angeli Dianne F. Ong, Phoebe Clare L. Racoma, Ian Angelo T.

Date Updated November 12, 2021

TABLE OF CONTENTS

PART III: Team Standards

PART IV: Team Workflow

PART I: Proposed Application

1 Description

The Powerzone Product Management application seeks to act as an all-in-one management system for Powerzone, a fuel company based in Catanduanes, Bicol. The application will keep track of three major components handled by Powerzone: inventory, transactions, and deliveries. The inventory management system in the application will log all products that come in and out the inventory of the company. This includes both the purchasing of products from suppliers and the selling of products through transactions.

The transaction management system in the application will handle the digitization of all transactions made by the company with customers. This includes all of the customer's purchases, as well as the delivery details provided by the customer, which in turn will be handled by the delivery management system. Transactions and deliveries have a one-is-to-one relationship, in that every transaction has a corresponding delivery, as the company delivers all purchased products to a client's given address. In summary, the Powerzone Product Management application's goal is to digitize all the necessary affairs that Powerzone must keep track of with regard to its inventories and transactions.

2 Intended Users

The application's intended users are the employees of Powerzone. Because a company's inventory and transactions may contain sensitive information, all users are required to enter their username and password to access the application. Non-registered users must register for a user account, providing their specific role for the application. Before gaining access to the system, as a security measure, all registered accounts must be approved by an administrator account before users are granted access to views authorized for their roles. These are further elaborated in Table I.1 found in **Section 3**.

3 Features & Functionalities

Most of the features of the application deal with the tracking of company affairs through a database; thus, these features are closely related to database create-read-update-delete (CRUD) operations. In this regard, there are three main components of the company that the application must track: stocks, transactions, and deliveries.

3.1 Stock Inventory

The most important feature of the application is to digitize the current state of the company's product stocks. A stock is defined as a purchase of products from a supplier, wherein each purchased product is stored in a separate tank. Thus, products are stored separately per purchase rather than in a single storage tank.

3.1.1 Definition of Terms

The following are terms used throughout the project specifications:

a. **Depleted Stock**: Stock number has reached zero (0)
b. **Critically Low Stock**
   - Diesel: 300,000 liters
   - Gasoline: 500,000 liters
   - Premium 95 Gasoline: 150,000 liters
   - Premium 97 Gasoline: 150,000 liters
   - Kerosene: 300,000 liters
c. **Expiring Stock**: 2 months from purchase date

3.1.2 View, Add, and Edit Stocks

The application must allow authorized personnel to view, add, and edit the stocks database through the application. Each stock will have two possible statuses: “In-Stock” or “Depleted”. If a stock is depleted, it should no longer be displayed in the application, but will not be deleted from the database.

3.1.3 Notifications

The application must notify users of products at a critical level. There are two possible conditions for a product to be deemed critical: (1) the stock of the product is critically low; (2) the stock is at a critical level of expiration such that it could affect product quality. Whenever a product is at a critical condition, the application must inform the user the stock that is critical, as well as the condition(s) of the product.

3.2 Transaction Records

The application must also take note of all transactions that the company makes. A transaction is defined as a process of selling products to a client. Transactions and deliveries are directly related to each other, in that all transactions require clients to provide delivery details as Powerzone also delivers the products to the client’s provided location.

3.2.1 View, Add, and Edit Transaction

All authorized users must be able to use the application to view, add, and edit entries in the transaction database, similar to the requirements of the stocks inventory.

3.2.2 First-In-First-Out (FIFO) Method

Powerzone incorporates a FIFO method when selling an amount of a product that encompasses more than one tank of a product. The FIFO method ensures that older tanks containing a product are depleted first before newer tanks of products are sold, in order to minimize the chances of products expiring or evaporating.

3.2.3 Daily Selling Price

The selling prices of crude-oil distillates such as the products sold by Powerzone change daily based on the dictations of the foreign market, but are regulated by the Department of Energy and Department of Trade and Industry in the Philippines. Thus, the daily selling prices of products can be changed by an authorized user, and any transactions must refer to the price of the purchased products for the specific day that the transaction transpired.

3.2.4 Completed Transactions

A transaction can be deemed "completed" only when its corresponding delivery has also been deemed completed by an authorized user. Furthermore, a transaction's status is "pending" by default, and will only be updated when it is either canceled or completed.

3.3 Stock Inventory

This section covers the records related to the stock inventory.

3.3.1 Delivery Records

As stated above, deliveries are a required aspect of a transaction. Clients cannot purchase a product without providing delivery details as well.

3.3.2 Add Delivery Records

Because deliveries are part of transactions, deliveries are added to the delivery database whenever a transaction is made. To ensure consistency with regards to the data in the transactions and delivery databases, delivery records cannot be added alone, and are only added to the database whenever a transaction is made.

3.3.3 View and Edit Delivery Records

Similar to the other databases, delivery records stored in the delivery database must also be viewable and editable by authorized personnel.

3.3.4 Completed Deliveries

A delivery is deemed completed when all products associated with its corresponding transaction have been fully delivered to the client. An authorized personnel can edit the status of a delivery to "completed", which will also update the transactions database to deem its corresponding transaction completed as well.

3.4 User-Dependent Views and Features

Aside from inventory tracking features, Powerzone Product Management will also implement user-dependent views that will restrict access to some views based on a user's role. Any user can apply for a specific role upon registration, but must be approved by the administrator first before being able to access the application with their declared role. The administrator account is a hard-coded, sole account that cannot be edited or deleted, and can access all features of the application. To better delineate the authorizations of all roles, Table I.1 enumerates all the roles available in the application, as well as their respective authorizations.

Role	Authorizations
Administrator	● Access all views ● Accept applications of new users ● Edit or delete other user's accounts ● Add and edit stocks in the inventory database ● Edit the daily selling price of products ● Add and edit transactions in the transaction database ● Add and edit deliveries in the deliveries database ● View all products, transactions, and deliveries ● View the daily selling prices of all products
Inventory Manager	● Add and edit stocks in the inventory database ● Edit the daily selling price of products ● View all products, transactions, and deliveries ● View the daily selling prices of all products ● Edit own user details (except user role)
Transaction Cashier	● Add and edit transactions in the transaction database ● Add and edit deliveries in the deliveries database ● View all products, transactions, and deliveries ● View the daily selling prices of all products ● Edit own user details (except user role)
Delivery Manager	● Edit deliveries in the deliveries database ● View all products, transactions, and deliveries ● View the daily selling prices of all products ● Edit own user details (except user role)

Table I.1. User Roles and Authorizations

4 Product Backlog

Refer to **Appendix A** for the product backlog, which includes the user stories, scenarios, and acceptance criteria.

PART II: Software Development Methodology

1 Methodology

It has been decided that the Agile methodology will be used for the project since the said framework lends itself to faster development, which is crucial for feature-intensive software solutions such as the current project. Moreover, all members of the group are familiar with the methodology, thus ideally reducing the time taken to getting accustomed to its artifacts and ceremonies. Furthermore, the Agile methodology also enables the team to create prototypes of the expected product within short time cycles which essentially allows for constant testing and revisions over the project period, thereby enhancing the overall quality of the final product.

As part of the Agile methodology, the team has further decided to use the Scrum method. As all team members already have prior experience with the Scrum method, and all members provided positive feedback regarding the method during the team's preliminary meeting, the team chose to implement the same method for this project in order to ideally replicate, or further improve upon, the development velocity achieved during the team members' previous projects, and to remove the need for learning a new framework in addition to the work involved in creating and testing the application. Furthermore, the Scrum method's multiple iterative processes such as the scrum meetings and retrospectives, in conjunction with the creation of a detailed project backlog, allow the team to frequently ensure that the progress made with the application is complete and parallels the ideas agreed upon during the scrum plannings. In the circumstance that inconsistencies within the application and plans arise, the scrum method's incremental nature also assures a level of flexibility that enables the team to quickly and efficiently address these obstacles.

With this, the team will make use of three-week sprints all throughout the term, for a total of three sprints during the development process.

2 Sprint Phases

This section discusses the phases comprising each sprint, with the general flow shown in Figure II.1.

2.1 Sprint Initialization

At the start of a new sprint, the team conducts a synchronous meeting where they choose the user stories to tackle for the sprint (guided by the team's most recent sprint velocity, if available) and lists the tasks to be accomplished for each user story. The team members then choose the tasks they want to handle, overseen by the Scrum master who makes sure all tasks have been properly assigned. Finally, the Scrum master provides initial deadlines for each of the tasks to ensure that the parts of the application are moving along the development pipeline at the desired speed.

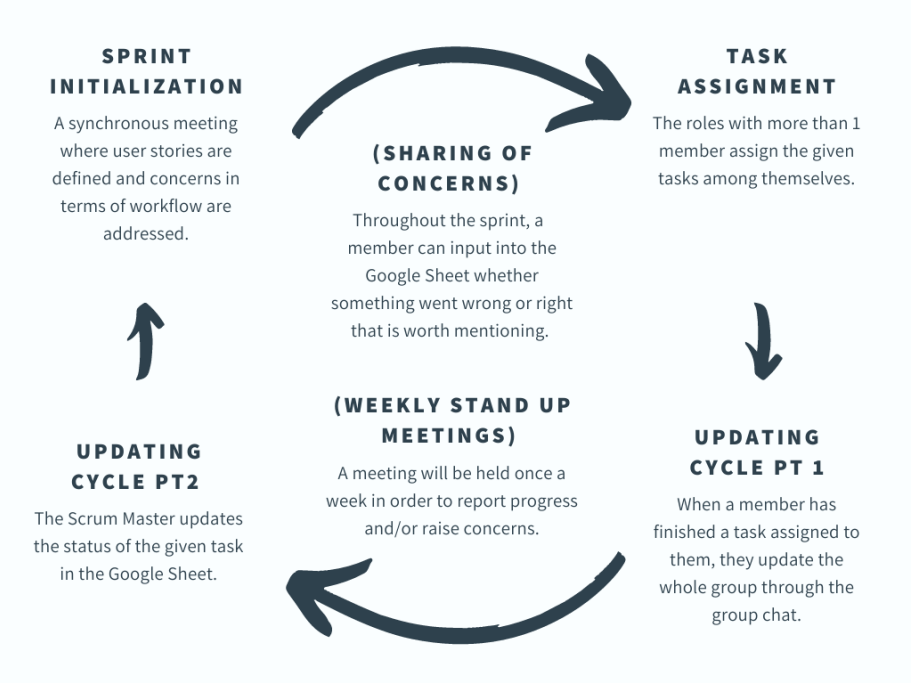

Figure II.1. The general flow of the team.

2.2 Task Assignment

Next, the roles with more than one member, which are the developers, designers, and quality assurance, assign the given tasks among themselves in order for them to properly divide the responsibilities according to each person's abilities, skills and time.

2.3 Updating Cycle

Whenever a certain member finishes a task assigned to them, they will update the whole group through the Messenger group chat. The Scrum master will then update the status of the given task in the Google Sheet, which will serve as the product backlog manager for the whole team. This updating cycle will be used in order to lessen the burden on the other members so that they would no longer have to learn and get accustomed to a specific project tracker software.

2.4 Weekly Stand-Up Meeting

Once a week (tentatively every Friday, though the day and time may be adjusted depending on the availability of the team members), all team members will meet through voice call. Apart from the updating cycle, the stand-up meetings will also be used for the team members to report their progress and raise any issues or clarifications, especially when they would need to confer with members with other roles (e.g., the developers would want to confirm an element of the front-end design with the designers, or the quality assurance members would want to explain the behavior of a detected bug to the developers).

The stand-up meetings may also be used for quick "triaging" sprint retrospectives, wherein aspects of the development methodology that were determined to be causing urgent issues can be discussed and adjusted as soon as possible to avoid similar problems.

2.5 Sharing of Concerns

Throughout the three weeks of the sprint, any member has the option to either input into the Google Sheet (which is constantly being monitored by the Scrum master), message the group chat, or privately inform the Scrum master whether something significant went wrong or right in a specific part of the sprint. This will help the Scrum master quickly gather information about the team workflow, think of possible solutions to resolve certain conflicts and guide the whole team to adapt faster, which is crucial to the development methodology due to the limited time given to complete the application.

2.6 Sprint Retrospective

After each sprint, the team performs a sprint retrospective in order to address the feedback, both positive and negative, compiled in the Google Sheet, as well as any comments or issues raised during the meeting itself. This feedback will be used to improve the team workflow for the succeeding sprints. During the sprint retrospective, the Scrum master also computes the sprint velocity and informs the team about the completion status of the project based on the product backlog. After the sprint retrospective has been completed, the Scrum master transitions to the sprint initialization of the succeeding sprint.

3 Project Roles

3.1 Scrum Master

He manages the team workflow, ensuring that a good sprint velocity is maintained all throughout the duration of the project. Team meetings and retrospectives are led by him. When the other team members report issues with the team workflow, he is in charge of editing the workflow and any related tools and documents to resolve the issue. In the event that communication among other team members breaks down, he is responsible for fixing the problems, whether technological or personal, that cause these communication difficulties. Lastly, he is in charge of updating the Master Tracker (via Google Sheet). This also includes updating the Sprint Calendar, which involves the action of tagging finished user stories as "DONE."

3.2 Product Owner

She determines the features and user stories for the application, making a realistic and full visualization of the whole project given the various limitations (i.e. time, skills, and knowledge). Additionally, she provides all the details needed by the development teams regarding the application, from the general palette to be used by the designers and the different devices where the application will be deployed to the sample information to be input and the proper error

messages to be displayed. Finally, the product owner is also in charge of checking accomplished features marked as “PASSED” by the quality assurance team; only after the product owner has approved of the feature or fix can the Scrum master mark the pertinent user stories as “DONE.”

3.3 Designers

Initially, they create wireframes or other design prototypes to simulate the full visual experience of a specific page and ask for the product owner’s approval. Once approved, they develop the static front-end pages in order to increase the team velocity and allow the developers to concurrently work on some of the feature components. Essentially, they are in charge of translating the information provided by the product owner into a visually appealing, intuitive and organized design that suits the context of the business at hand.

3.4 Developers

They develop the functionality of the front end and back end of the application (e.g. database access, front-end validation, back-end validation). They manage access to and from the GitHub repository in order to preserve the organization and integrity of the code. They are also in charge of moving code between branches on the repository by merging accepted pull requests, fixing bugs when they are raised by the quality assurance team through GitHub issues, and determining whether a developed feature can be passed to the quality assurance team for testing.

3.5 Quality Assurance

They discover the bugs in the application through the automated process of exhaustive testing of both the front-end and back-end aspects of the application. Testing will be split into four phases, namely, integration testing, which focuses on testing the incorporation of a new unit into previously implemented functionalities; smoke testing, which serves as a preliminary check whether an implemented feature can perform its function; functional testing, which is a more indepth test of a feature to determine whether all its functionalities work as expected; and, finally, system testing, which is an exhaustive testing of the entire application. Additionally, they inform the product owner when a feature or code segment is working properly and can thus be passed to her for inspection.

4 Definition of Done

4.1 Design and development completed

The design and function of the pages pertinent to the user story have been completed following the specifications from the product owner.

4.2 No errors from unit testing

The developers have performed unit testing for the newly built feature, resulting in no errors.

4.3 No errors from exhaustive testing

The quality assurance team has deemed the user story as “PASSED” after exhaustive testing. These tests include integration testing, smoke testing, functional testing, and system testing. All issues raised by the quality assurance team have been properly addressed.

4.4 Functional and non-functional requirements fulfilled

The quality assurance team, in conjunction with the product owner, has confirmed that the added feature conforms to both the functional and non-functional requirements of the software solution.

4.5 All tasks accomplished

The members who were assigned to each task should have properly acknowledged that they have finished the work assigned. This serves as an additional confirmation that all team members involved in the development of the feature have green-lighted its accomplishment.

4.6 Final status update for the user story

The user story is declared as “DONE” by the Scrum master and the Scrum master updates the Master Tracker, which is implemented as a Google Sheet, accordingly.

PART III: Team Standards

1 Rationale

This section discusses the technology stack and the coding standards to be followed by the team for this project. Emphasis is placed on the software design principle, architecture, and development process that justify the selection of the technology stack. Furthermore, the coding standards cover the rules (meta, style, and formatting), allowed language features, and naming conventions for the following languages used in web application development:

a. Hypertext Markup Language (HTML)
b. Cascading Style Sheets (CSS)
c. JavaScript

The rationale behind the rules prescribed in this document is to promote internal consistency within the team and general compliance with established coding practices. For areas not stipulated in this document, the team members are expected to prioritize readability and follow the overarching principles espoused in this guide. If in doubt, it is imperative to consult with the rest of the team.

This document heavily borrows from the HTML and CSS[1], and JavaScript[2] style guides published by Google; however, there are certain points of deviation, such as the non-omission of optional HTML tags. These are intended to make the source code more explicit (albeit, at times, more verbose) or to reduce complexity in light of the scale of this project.

Note that the terms “source code,” “code,” and “file” are used interchangeably in this document unless otherwise dictated by the context.

2 Software Design Principle

The overlap among structure, styling, and behavior in a single file should be kept to a strict minimum in light of the principle of **separation of concerns**:

a. The structure should be defined only in HTML files.
b. The styling should be defined only in CSS files.
c. The behavior should be defined only in JS files.

In this regard, avoid inline and internal styling and scripting; use external style sheets and scripts instead.

[1] The reference style guide can be accessed through this link: https://google.github.io/styleguide/htmlcssguide.html.
[2] The reference style guide can be accessed through this link: https://google.github.io/styleguide/jsguide.html.

3 Software Architecture & Development Process

In relation to the principle of separation of concerns, this project will follow the **Model-View-Controller (MVC) architectural pattern**. The intention is to allow the developers and designers to collaborate on the features of the web application and work on their logic (model) and interface (view) independently before connecting them via the controller. While this approach comes at the cost of additional layers of indirection and complexity, it consequently leads to higher velocity and better organization for debugging, scaling, and adding new features.

Moreover, the MVC architecture provides strong support for **test-driven development (TDD)**, which will be adopted by the developers. In particular, the separation of concerns implemented by the MVC architecture makes the code components more loosely coupled and thus readily adaptable to unit testing, as expounded upon by Majeed and Rauf (2018)[3].

4 Technology Stack

The software design principle (separation of concerns), architecture (MVC), and development process (TDD) served as the guiding criteria in choosing the technology stack. Given the limited development time, it was in the best interest of the team to select tools and frameworks with extensive documentation or with which they already have prior experience or familiarity. They were also preliminarily evaluated based on support for a continuous integration and continuous development (CI/CD) pipeline.

This section gives a brief description of the key components of the technology stack.

4.1 Database

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)[4].

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)[5].

MongoDB is covered by the Server Side Public License[6]. Mongoose is covered by the MIT License[7].

[3] Majeed, A., & Rauf, I. (2018). MVC architecture: A detailed insight to the modern web applications development. *Peer Review of Solar & Photoenergy Systems, 1*(1). https://crimsonpublishers.com/prsp/pdf/PRSP.000505.pdf

[4] MongoDB. (2021). *Advantages of MongoDB*. https://www.mongodb.com/advantages-of-mongodb

[5] Medlock, J. (2017). *An overview of MongoDB & Mongoose.* Medium. https://medium.com/chingu/an-overview-of-mongodb-mongoose-b980858a8994

[6] The license can be accessed through this link: https://www.mongodb.com/licensing/server-side-public-license.

[7] The license can be accessed through this link: https://opensource.org/licenses/MIT.

4.2 Back-end

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)[8].

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)[9]; 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.

Node.js and Express.js are both covered by the MIT License.

4.3 Front-end

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)[10].

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)[11].

Bootstrap and Handlebars are both covered by the MIT License.

4.4 Testing[12]

Mocha and **Chai** will be used as the primary unit and integration testing framework. 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

[8] 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/

[9] 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/

[10] 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/

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

[12] The technology stack for testing is tentative and subject to change as the software development team progresses with the design and implementation of the continuous integration and continuous delivery pipeline.

the additional role of unit testing as a method of documentation (Borys, 2021)[13]. Consequently, 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) and performing integration tests (Chai HTTP).

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)[14].

Mocha and Chai are both covered by the MIT License. Selenium is covered by the Apache 2.0 License[15].

4.5 Deployment

The web application will be deployed on **Heroku**. The team chose to make use of this PaaS due to the range of functionalities available on its free tier and its handling of server management components (Clark, n.d.)[16].

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 BSON document size limit (MongoDB, 2021)[17].

GridFS is covered by the MIT License.

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

[14] 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/

[15] The license can be accessed through this link: https://www.selenium.dev/documentation/about/copyright_and_attributions.

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

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

5 HTML Coding Standards

This section delineates the coding standards for HTML files.

5.1 Meta Rules

5.1.1 File Name

Use hyphens to separate words in file names. All the letters should be in lowercase.

Incorrect:
```
pupperApp.html
```

Correct:
```
pupper-app.html
```

The file extension may change depending on the template engine used. For example, the file extension of a Handlebars template is `.hbs`.

5.1.2 Protocol

Ensure that embedded resources are taken from content delivery networks (CDNs) that use HTTPS (Hypertext Transfer Protocol Secure). In the interest of security, do not embed any resources from CDNs that use only HTTP (i.e., without the "Secure").

Do not omit `https:` when specifying the source.

Incorrect:
```
<script src =
"//maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.
js"></script>
```

Correct:
```
<script src =
"https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstra
p.min.js"></script>
```

5.1.3 Encoding

Use UTF-8, and specify it explicitly in the source code. As some text components of the web application would require symbols not supported by the ASCII format, such as the Philippine peso sign (₱) when reporting transaction data, the UTF encoding is required. Moreover, stating the use of UTF-8 explicitly prevents confusion with other encoding schemes, such as UTF-16, which the team does not intend to use.

```
<meta charset = "utf-8">
```

5.1.4 Comments

Use comments as needed to explain the purpose or the scope of a code segment. Capitalize the first letter of the comment, and place a single space before it. If a comment describes the purpose or behavior of a code segment, use the imperative tone as much as possible. End it with a period if it is a complete sentence.

Incorrect:
```
<!-- map each order to an ID number -->
```

Correct:
```
<!-- Map each order to an ID number. -->
```

5.1.4.1 `TODO`

Use `TODO` to mark action items. Separate `TODO` and the action item with a colon followed by a single space, and capitalize the first letter of the action item.

Incorrect:
```
<!-- TODO Convert to ordered list for clarity -->
```

Correct:
```
<!-- TODO: Convert to ordered list for clarity. -->
```

5.2 Style Rules

5.2.1 Document Type

Place the HTML 5 document type declaration at the start of the source code. While most browsers already use the proper rendering for HTML files, the inclusion of this line also serves as documentation within the team and for others who may access the source code.

```
<!DOCTYPE html>
```

5.2.2 Void Elements

Do not close void elements.

Incorrect:
```
<br/>, <hr/>
```

Correct:
```
<br>, <hr>
```

5.2.3 Multimedia Fallback

To promote accessibility, provide alternative content for multimedia elements.

Incorrect:
```
<img src = "pupper.png">
```

Correct:
```
<img src = "pupper.png" alt = "Image of a wolf pup">
```

5.2.4 Entity References

Use entity references as needed. In the interest of readability, prioritize entity names over entity numbers.

Incorrect:
```
¢
```

Discouraged:
```
&#162;
```

Correct:
```
&cent;
```

If the usage of entity numbers is unavoidable (for browser compatibility), place a comment indicating the name of the character.

```
<!-- &#162; is the symbol for cent. -->
```

5.2.5 Optional Tags

Though defined as optional in HTML 5, do not omit optional tags.

This non-omission is enforced to promote readability.

Incorrect:
```
<!DOCTYPE html>
<title>Dog Webpage</title>
<p>Lorem ipsum dolor sit amet.
```

Correct:
```
<!DOCTYPE html>
<html>
      <head>
            <title>Dog Webpage</title>
      </head>
      <body>
            <p>Lorem ipsum dolor sit amet.</p>
      </body>
</html>
```

5.2.6 `type` Attribute

Omit the `type` attribute for style sheets and scripts.

Incorrect:
```
<link rel = "stylesheet" href = "sample.css" type =
"text/css">
```

Correct:
```
<link rel = "stylesheet" href = "sample.css">
```

5.3 Formatting Rules

5.3.1 Capitalization

Use only lowercase characters for HTML tags.

Incorrect:

```
<P>Lorem ipsum dolor sit amet.</P>
```

Correct:

```
<p>Lorem ipsum dolor sit amet.</p>
```

5.3.2 Indentation

Indent using a single tab. Do not use a combination of tabs and spaces for indentation.

Incorrect:

```
<ul>
  <li>Dogs</li>
  <li>Cats</li>
</ul>
```

Correct:

```
<ul>
	<li>Dogs</li>
	<li>Cats</li>
</ul>
```

5.3.3 Blocks

Place block, list, or table elements on new lines. Indent the child elements of a block, list, or table element.

Incorrect:

```
<table><tr><td>Shiba Inu</td>
<td>Corgi</td>
<td>Dalmatian</td>
</tr></table>
```

Correct:

```
<table>
	<tr>
		<td>Shiba Inu</td>
		<td>Corgi</td>
		<td>Dalmatian</td>
	</tr>
</table>
```

5.3.4 Line Wrapping

Do not break lines of code.

Incorrect:

```
<button id = "add-user"
      type = "reset"
      class = "btn btn-default btn-success btn-block"
      disabled>
```

Correct:

```
<button id = "add-user" type = "reset" class = "btn
btn-default btn-success btn-block" disabled>
```

5.3.5 Whitespace

5.3.5.1 Vertical Whitespace

Use only a single blank line to create logical groupings of blocks.

Incorrect:

```
<div id = "first-logical-grouping">
      ...
</div>


<div id = "second-logical-grouping">
      ...
</div>
```

Correct:

```
<div id = "first-logical-grouping">
      ...
</div>

<div id = "second-logical-grouping">
      ...
</div>
```

5.3.5.2 Horizontal Whitespace

Put a single whitespace before and after the equals sign that separates the attribute from the attribute value.

Incorrect:

```
<input type="submit">
```

Correct:

```
<input type = "submit">
```

Do not put a whitespace after an opening tag or before a closing tag that appears in the same line as the content being enclosed.

Incorrect:
```
<p> Lorem ipsum dolor sit amet. </p>
```

Correct:
```
<p>Lorem ipsum dolor sit amet.</p>
```

The same rule applies even if there is no content enclosed between the tags.

Incorrect:
```
<script src = "validation.js"> </script>
```

Correct:
```
<script src = "validation.js"></script>
```

5.3.6 Quotation Marks

Use double quotation marks when specifying attribute values.

Incorrect:
```
<a href = 'google.com'>Link</a>
```

Correct:
```
<a href = "google.com">Link</a>
```

6 CSS Coding Standards

This section delineates the coding standards for CSS files.

6.1 Meta Rules

6.1.1 File Name

Use hyphens to separate words in file names. All the letters should be in lowercase.

Incorrect:
```
userStyle.css
```

Correct:
```
user-style.css
```

The file extension should be `.css`.

6.1.2 Encoding

CSS already assumes UTF-8 encoding. Therefore, there is no need to explicitly specify it.

6.1.3 Comments

Use comments as needed to explain the scope of a set of rules or to create logical groupings. Capitalize the first letter of the comment, and place a single space before it. End it with a period if it is a complete sentence.

Incorrect:
```
/*Gallery*/
```

Correct:
```
/* Gallery */
```

6.1.3.1 `TODO`

Use `TODO` to mark action items. Separate `TODO` and the action item with a colon followed by a single space, and capitalize the first letter of the action item.

Incorrect:
```
/* TODO Change background color */
```

Correct:
```
/* TODO: Change background color. */
```

6.2 Style Rules

6.2.1 Declaration Order

Arrange declarations in alphabetical order.

Incorrect:

```
padding-top: 5px;
padding-left: 10px;
padding-bottom: 5px;
padding-right: 10px;
```

Correct:

```
padding-bottom: 5px;
padding-left: 10px;
padding-right: 10px;
padding-top: 5px;
```

6.2.2 Color Specification

At the top of the style sheet, store the hex values of the colors inside variables. Name these variables to reflect the elements being styled, use hyphens to separate multiple words, and append the word `color`. Additionally, place comments to indicate the names of the colors.

```
:root {
      --header-color: #DC143C;          /* Crimson */
      --sidebar-color: #DCDCDC;         /* Gainsboro */
}
```

In the interest of readability, do not use hard-coded color values inside rules; use the created variables instead.

Incorrect:

```
background-color: #DCDCDC;
```

Correct:

```
background-color: var(--sidebar-color);
```

6.2.3 Type Selectors

In the interest of brevity, do not qualify ID or class names with type selectors (selectors that match elements by their node name or tag name).

Incorrect:

```
p.special
```

Correct:

```
.special
```

6.2.4 Shorthand Properties

Shorthand properties for `margin` and `padding` are permitted only if the values are equal. Otherwise, specify them individually to make the code more explicit.

Incorrect:
```
padding: 5px 10px 5px;
```

Correct:
```
padding-bottom: 5px;
padding-left: 10px;
padding-right: 10px;
padding-top: 5px;
```

6.2.5 Units

Specify units for all values, including zeroes.

Incorrect:
```
margin: 0;
```

Correct:
```
margin: 0px;
```

6.2.6 Leading Zeroes

Do not omit leading zeroes in values.

Incorrect:
```
margin: .5px;
```

Correct:
```
margin: 0.5px;
```

6.3 Formatting Rules

6.3.1 Selector Names

Use lowercase characters for ID and class names. Use hyphens to separate multiple words in a name; a numeral is considered a separate word. Note that this rule can be overridden if the selector is derived from a framework. Provide descriptive and unambiguous names.

Incorrect:
```
.IdOne
.header2
```

Correct:
```
.header-logo
.header-2
```

6.3.2 Indentation

Indent using a single tab. Do not use a combination of tabs and spaces for indentation.

Incorrect:

```
.header-logo {
  padding-bottom: 5px;
  padding-left: 10px;
  padding-right: 10px;
  padding-top: 5px;
}
```

Correct:

```
.header-logo {
      padding-bottom: 5px;
      padding-left: 10px;
      padding-right: 10px;
      padding-top: 5px;
}
```

6.3.3 Braces

Use the K&R style (Kernighan and Ritchie style) for brace placement.

Incorrect:

```
.header-logo
{
  padding-bottom: 5px;
  padding-left: 10px;
  padding-right: 10px;
  padding-top: 5px;
}
```

Correct:

```
.header-logo {
      padding-bottom: 5px;
      padding-left: 10px;
      padding-right: 10px;
      padding-top: 5px;
}
```

6.3.4 Blocks

Place block contents on new lines. Indent all block contents.

Incorrect:

```
.header-logo { padding-bottom: 5px;
      padding-left: 10px; padding-right: 10px;
      padding-top: 5px }
```

Correct:

```
.header-logo {
      padding-bottom: 5px;
      padding-left: 10px;
      padding-right: 10px;
      padding-top: 5px;
}
```

6.3.5 Stops

6.3.5.1 Declaration Stops

Use a semicolon after each declaration.

Incorrect:

```
.header-logo {
      padding-bottom: 5px;
      padding-left: 10px;
      padding-right: 10px;
      padding-top: 5px
}
```

Correct:

```
.header-logo {
      padding-bottom: 5px;
      padding-left: 10px;
      padding-right: 10px;
      padding-top: 5px;
}
```

6.3.5.2 Property Name Stops

Use a single space after the colon of each property name.

Incorrect:

```
.header-logo {
      padding-bottom:5px;
      padding-left:10px;
      padding-right:10px;
      padding-top:5px
}
```

Correct:

```
.header-logo {
      padding-bottom: 5px;
      padding-left: 10px;
      padding-right: 10px;
      padding-top: 5px;
}
```

6.3.6 Separation

6.3.6.1 Declaration Block Separation

Use a single space to separate the selector and the opening brace of the declaration block. Place each declaration on a new line.

Incorrect:

```
.header-logo{
      padding-bottom: 5px; padding-left: 10px;
      padding-right: 10px; padding-top: 5px
}
```

Correct:

```
.header-logo {
      padding-bottom: 5px;
      padding-left: 10px;
      padding-right: 10px;
      padding-top: 5px;
}
```

6.3.6.2 Selector Separation

Use a single space to separate multiple selectors. Do not split selectors into multiple lines.

Incorrect:

```
.header-logo,
.login-screen-text,
.credits-text{
      margin-top: 0px;
}
```

Correct:

```
.header-logo, login-screen-text, .credits-text {
      margin-top: 0px;
}
```

6.3.6.3 Rule Separation

Use a single blank line to separate rules.

Incorrect:

```
.header-logo {
      margin-top: 0px;
}
.login-screen-text {
      margin-bottom: 5px;
}
```

Correct:

```
.header-logo {
      margin-top: 0px;
}

.login-screen-text {
      margin-bottom: 5px;
}
```

6.3.7 Quotation Marks

Use single quotation marks when specifying property values.

Incorrect:

```
.header-logo {
      font-family: "Times New Roman";
}
```

Correct:

```
.header-logo {
      font-family: 'Times New Roman';
}
```

7 JavaScript Coding Standards

This section delineates the coding standards for JavaScript files.

7.1 Meta Rules

7.1.1 File Name

Use hyphens to separate words in file names. All the letters should be in lowercase.

Incorrect:
```
createAccount.js
```

Correct:
```
create-account.js
```

The file extension should be `.js`.

7.1.2 Encoding

Use UTF-8 for the encoding of the source code.

7.1.3 Comments

Use comments as needed to explain the purpose or the scope of a code segment. Capitalize the first letter of the comment, and place a single space before it. If a comment describes the purpose or behavior of a code segment, use the imperative tone as much as possible. End it with a period if it is a complete sentence.

Do not use the double slash (`//`) even for single-line comments.

Incorrect:
```
// Check whether the Create Account form is filled
/* check whether the Create Account form is filled */
```

Correct:

```
/* Check whether the Create Account form is filled. */
```

For multiline comments, the first and last lines consist only of the opening slash-asterisk (`/*`) and the closing slash-asterisk (`*/`). Start the body of the comment on the second line, prefix each new line with an asterisk, and align the asterisks across all the lines.

Incorrect:
```
/* Check whether all the entries found in the Create
   Account form are filled and properly validated on the
   client-side */
```

Correct:

```
/*
 * Check whether all the entries found in the Create
 * Account form are filled and properly validated on the
 * client-side.
 */
```

7.1.3.1 Function Comments

Add a comment on top of every function definition, containing a brief description of the function, an optional detailed explanation to note important preconditions or postconditions, its parameters (preceded by `@param`), and, if applicable, its return value (preceded by `@return`).

The format for function comments is patterned after a modified version of Javadoc. Start the brief description of the function with a verb in its singular simple present tense (instead of the usual imperative form for regular comments). End it with a period (even if it is not a sentence).

Start the description of each parameter and of the return value with an uppercase letter (unless the first word is a reserved word or a keyword that starts with a lowercase letter, such as the Boolean values `true` and `false`). End it with a period (even if it is not a sentence).

Note that the first line features two asterisks after the opening slash `(/**)`.

```
/**
 * Brief description of the function.
 *
 * Optional detailed explanation.
 *
 * @param  name Description.
 * @return Description.
 */
```

Correct:

```
/**
 * Checks whether the password input by the user
 * during account creation is valid.
 *
 * A password is considered valid if it contains at
 * least 8 characters and at least one number.
 *
 * @param  password Password input by the user
 *                  during account creation.
 * @return true if the given password is valid;
 *         false, otherwise.
 */
```

7.1.3.2 `TODO`

Use `TODO` to mark action items. Separate `TODO` and the action item with a colon followed by a single space, and capitalize the first letter of the action item.

Incorrect:
```
/* TODO Reject float values for age */
```

Correct:
```
/* TODO: Reject float values for age. */
```

7.2 Style Rules

7.2.1 Special Escape Sequences

In the interest of readability, use special escape sequences rather than numeric escape sequences when applicable. The special escape sequences are as follows[18]:

`\a` (alert), `\b` (backspace), `\f` (form feed), `\n` (newline), `\r` (carriage return), `\t` (horizontal tab), `\v` (vertical tab), `\'` (single quotation mark), `\"` (double quotation mark), `\?` (question mark), and `\\` (backslash)

Incorrect:
```
\u000c
```

Correct:
```
\f
```

Additionally, place a comment indicating the name of the special escape sequence.

```
/* \f is the special escape sequence for form feed. */
```

7.2.2 Non-ASCII Characters

In the interest of cross-browser compatibility, use Unicode escapes for non-ASCII (American Standard Code for Information Interchange) characters.

Incorrect:
```
¥
```

Correct:
```
\u00a5
```

Additionally, place a comment indicating the name of the character.

```
/* \u00a5 is the symbol for Japanese yen. */
```

[18] The list is taken from https://www.ibm.com/docs/en/i/7.3?topic=set-escape-sequences.

7.3 Formatting Rules

7.3.1 Semicolons

Use a semicolon after every statement, except after the closing brace of a function definition or a class declaration.

Incorrect:
```
function isFilled() {
      return validator.trim($('#address').val())
};
```

Correct:
```
function isFilled() {
      return validator.trim($('#address).val());
}
```

7.3.2 Indentation

Indent using a single tab. Do not use a combination of tabs and spaces for indentation.

Incorrect:
```
function isFilled() {
  let username = validator.trim($('#username').val());
  return username;
}
```

Correct:
```
function isFilled() {
      let username = validator.trim($('#username').val());
      return username;
}
```

7.3.3 Braces

Use braces for all control structures, even if the body only contains a single statement. Place the first statement of a non-empty block on a new line.

Incorrect:
```
function isOne(number) {
      if (number == 1)
            return true;
}
```

Correct:
```
function isOne(index) {
      if (number == 1) {
            return true;
      }
}
```

7.3.4 Blocks

7.3.4.1 Empty Blocks

For conciseness, close empty blocks immediately after they are opened.

Incorrect:
```
function empty() {
}
```

Correct:
```
function empty() {}
```

7.3.4.2 Non-empty Blocks

Use the K&R style for brace placement in non-empty blocks.

Incorrect:
```
function validateField(empty)
{
      if (empty)
      {
            field.prop('value', '');
            error.text(fieldName + 'should be filled');
      }
      else
      {
            error.text('');
      }
}
```

Correct:
```
function validateField(empty) {
      if (empty) {
            field.prop('value', '');
            error.text(fieldName + 'should be filled');
      } else {
            error.text('');
      }
}
```

7.3.5 Statements

Place each statement on a new line.

Incorrect:
```
function pythagorean(a, b){
      let c = Math.sqrt(a * a + b * b); return c;
}
```

Correct:

```
function pythagorean(a, b){
      let c = Math.sqrt(a * a + b * b);
      return c;
}
```

7.3.5.1 `switch` Statements

Leave a blank line after the last statement to be executed per case. Unless necessary to create logical groupings or promote readability, avoid using vertical whitespaces within a block that corresponds to only a single case; if there are several statements inside a block, consider placing them in functions instead.

Indent each case by one level. Start the statements to be executed on the next line (that is, the line after the `case` keyword) and indent them by another level.

Incorrect:

```
switch (doggoSize) {
case 1:
      smol();
      break;
case 2:
      lorge();
      break;
default:
      pup();
      break;
}
```

Correct:

```
switch (doggoSize) {
      case 1:
            smol();
            break;

      case 2:
            lorge();
            break;

      default:
            pup();
            break;
}
```

7.3.6 Literals

7.3.6.1 Array Literals

Place all array elements on the same line. Add a horizontal whitespace after the comma of each element. Do not use trailing commas.

If placing all the elements on the same line would exceed 80 characters or deter readability, break after the comma of each element and align the first characters of each element. Place the opening bracket on the same line as the first element and the closing bracket on the same line as the last element.

Incorrect:

```
const pups = ['shiba', 'akita', 'husky',];

const quotes = [
                'I am a very funny yellow dog',
                'How do I love thee?',
                'To be not to be, said Shakespeare',
                'Brevity is the soul of the wit',
                'Art is long, life is short',
               ];
```

Correct:

```
const pups = ['shiba', 'akita', 'husky'];

const quotes = ['I am a very funny yellow dog',
                'How do I love thee?',
                'To be not to be, said Shakespeare',
                'Brevity is the soul of the wit',
                'Art is long, life is short'];
```

7.3.6.2 Object Literals

Place each property on a separate line, breaking after the comma of each property. Do not use trailing commas.

Incorrect:

```
const pup = {
      name: 'Shibe', breed: 'Shiba Inu', age: 2,
};
```

Correct:

```
const pup = {
      name: 'Shibe',
      breed: 'Shiba Inu',
      age: 2
};
```

7.3.7 Functions

7.3.7.1 Function Arguments

Place all function arguments on the same line. Add a horizontal whitespace after the comma of every argument. If placing all the arguments on the same line would exceed 80 characters or deter readability, break after the comma of each argument and align the first characters of each argument.

Do not add a horizontal whitespace before the opening parenthesis of the argument list, but add a horizontal whitespace before the opening brace of the definition.

Incorrect:

```
function isOne (number,index) {
      if (number == 1) {
            return true;
      }

      return false;
}

isOne (number,index);
```

Correct:

```
function isOne(number, index) {
      if (number == 1) {
            return true;
      }

      return false;
}

isOne(number, index);
```

7.3.7.2 Anonymous Functions

Do not use the arrow function as a shorthand for anonymous functions, as they have been noted to cause issues when used in constructors and generators (Fogarty, 2017)[19].

Incorrect:

```
let multiply = (a, b) => a * b;
```

Correct:

```
let multiply = function(a, b) {
      return a * b;
}
```

[19] Fogarty, T. (2017, September 28). *Advantages and pitfalls of arrow functions*. Medium. https://medium.com/tfogo/advantages-and-pitfalls-of-arrow-functions-a16f0835799e

7.3.8 Line Wrapping

If a line exceeds 80 characters or if a line is unreadable (even if the 80-character limit has not yet been reached), line wrap. Break at a higher syntactic level and after an operator.

Incorrect:

```
function reverseFactorial() {
      let factorial = getOne() * getTwo() * getThree() *
            getFour() * getFive() * getSix();
}
```

In the preceding example, note that assignment (=) occupies a higher syntactic level compared to multiplication (*). Hence, the break after the assignment operator should be prioritized.

Correct:

```
function reverseFactorial() {
      let factorial =
            getOne() * getTwo() * getThree() * getFour() *
            getFive() * getSix();
}
```

7.3.9 Whitespace

7.3.9.1 Vertical Whitespace

Aside from the rules previously mentioned, place a vertical whitespace between separate methods in a class or object definition, and between logical groupings of statements in the code. Do not place more than one vertical whitespace between any code segments.

Do not add vertical whitespaces after the opening braces of blocks. Unless necessary in the interest of readability (for example, in `if-else` blocks that contain several statements), do not add vertical whitespaces before their closing braces as well.

Incorrect:

```
function getAverage(numOne, numTwo, numThree) {

      let total = numOne;
      total += numTwo;
      total += numThree;


      let average = total / 3;
      return average;

}
```

Correct:

```
function getAverage(numOne, numTwo, numThree) {
      /* Sum up the three values. */
      let total = numOne;
      total += numTwo;
      total += numThree;

      /* Divide the sum by the number of values. */
      let average = total / 3;

      return average;
}
```

7.3.9.2 Horizontal Whitespace

Aside from the rules previously mentioned, place a horizontal whitespace:

a) Between a reserved word and an opening parenthesis or brace on the same line (functions constitute a special case and are covered in **Section 7.3.7**)
b) Between a closing brace and a reserved word
c) After a comma
d) After a colon
e) Before and after a non-unary arithmetic, logical, or relational operator

Incorrect:

```
function getAverage(numOne, numTwo, numThree){
      let total=numOne;
      total+=numTwo;
      total+=numThree;

      let average=total/3;

      return average;
}
```

Correct:

```
function getAverage(numOne, numTwo, numThree) {
      let total = numOne;
      total += numTwo;
      total += numThree;

      let average = total / 3;

      return average;
}
```

7.3.9.3 Horizontal Alignment

Do not perform horizontal alignment.

Incorrect:
```
let singleDigitNumber   = 1;
let multipleDigitNumber = 12345;
```

Correct:
```
let singleDigitNumber = 1;
let multipleDigitNumber = 12345;
```

7.3.10 Quotation Marks

Use single quotation marks for strings and jQuery element references.

Incorrect:
```
function displayPasswordError() {
      if(field.is($("#new-password"))) {
            $("#error").text("Should contain 8 characters");
      }
}
```

Correct:
```
function displayPasswordError() {
      if(field.is($('#new-password'))) {
            $('#error').text('Should contain 8 characters');
      }
}
```

7.4 Language Features

7.4.1 Allowed Features

Use only JavaScript features that are defined and permitted in the 11th edition of ECMA-262[20] (ECMAScript Language Specification) or in the living standard of WHATWG[21] (Web Hypertext Application Technology Working Group).

Additionally, avoid using experimental features, those that may obfuscate code clarity, and those that are known to be potentially dangerous. This rule also extends to deprecated and proprietary features that may affect compatibility with most modern browsers.

[20] The standard can be accessed through this link: https://262.ecma-international.org/11.0/.
[21] The standard can be accessed through this link: https://html.spec.whatwg.org/multipage/.

7.4.2 Local Variable Declaration

7.4.2.1 `let` and `const`

Use `let` for local variable declarations. For variables that have to be protected from unintended reassignment, explicitly mark them by using `const`. To prevent issues related to scoping, do not use the keyword `var`.

Incorrect:
```
var shibe = 'Hachiko';
```

Correct:
```
const shibe = 'Hachiko';
```

7.4.2.2 Declaration and Initialization

Rather than declaring variables at the beginning of a block of code, declare variables when needed (that is, when they are used in the logical flow of the code), and initialize them after declaration as soon as possible.

Incorrect:
```
function addThree (number) {
      const one;
      const two;
      const final;

      one = 1;
      two = 2;

      dummyFunctionOne();
      dummyFunctionTwo();

      final = number + one + two;
      return final;
}
```

Correct:
```
function addThree (number) {
      dummyFunctionOne();
      dummyFunctionTwo();

      const one = 1;
      const two = 2;
      const final = number + one + two;

      return final;
}
```

7.4.3 Arrays

7.4.3.1 Array Creation

Do not use the variadic `Array` constructor; define the array as a literal instead, as some issues have been noted when using array methods for single-element arrays (Aguinaga, 2021)[22].

Incorrect:
```
const pups = new Array('shiba', 'akita', 'husky');
```

Correct:
```
const pups = ['shiba', 'akita', 'husky'];
```

7.4.4 Objects

7.4.4.1 Object Creation

Do not use the `Object` constructor; define the object as a literal instead, as the `new` operator searches for a constructor before creating a new object upon failure.

Incorrect:
```
const pup = new Object();
pup.name = 'Shibe';
pup.breed = 'Shiba Inu';
pup.age = 2;
```

Correct:
```
const pup = {
      name: 'Shibe',
      breed: 'Shiba Inu',
      age: 2
};
```

7.4.4.2 No Mixing of Quoted and Unquoted Keys

Do not mix quoted and unquoted keys when declaring object properties.

Incorrect:
```
const pup = {
      'name': 'Shibe',
      breed: 'Shiba Inu',
};
```

Correct:
```
const pup = {
      name: 'Shibe',
      breed: 'Shiba Inu',
};
```

[22] Aguinaga, J.J.P. (2021, September 29). *Why never use new Array in JavaScript*. Coderwall. https://coderwall.com/p/h4xm0w/why-never-use-new-array-in-javascript

7.4.5 Strings

7.4.5.1 Template Literals

For readability, use template literals in place of complex string concatenations.

Incorrect:

```
let firstName = 'Tofu';
let lastName = 'Chan';
let age = 3;
let faveFood = 'breads';

let intro =
      'Hello, my name is ' + firstName + ' ' +
      lastName + '. I am ' + age + ' years old and ' +
      'myfavorite food is ' + faveFood + '.';
```

Correct:

```
let firstName = 'Tofu';
let lastName = 'Chan';
let age = 3;
let faveFood = 'breads';

let intro =
      `Hello, my name is ${firstName} ${lastName}.
      I am ${age} years old and my favorite food is
      ${faveFood}.`;
```

However, note that template literals are not supported in Internet Explorer.

7.4.5.2 No Line Continuations

Do not use line continuations (\) in string literals. Instead, separate the string into smaller strings and concatenate them with the concatenation (+) operator. This is usually done to fit within the 80-character column limit or to promote readability.

Incorrect:

```
const tofuIntro = 'Hello, my name is Tofu Chan. I am \
      3 years old.';
```

Correct:

```
const tofuIntro = 'Hello, my name is Tofu Chan. I ' +
      'am 3 years old.';
```

If separating the string requires numerous concatenations, consider using template literals, as prescribed in the previous section (**Section 7.4.5.1**).

7.4.6 Control Structures

7.4.6.1 No `for...in` Loops

Do not use `for...in` loops, even when iterating over object elements, as the loop could unintentionally access object properties when iterating over an array of objects. Instead, use `for...of` loops.

Incorrect:

```
const tofu = {
      firstName: 'Tofu';
      lastName: 'Chan';
      age: 3;
      faveFood: 'breads';
};

let tofuDetails = '';

for (const x in tofu) {
      tofuDetails += tofu[x];
}
```

Correct:

```
const tofu = {
      firstName: 'Tofu';
      lastName: 'Chan';
      age: 3;
      faveFood: 'breads';
};

let tofuDetails = '';

for (const x of Object.keys(tofu)) {
      tofuDetails += tofu[x];
}

/*
 * Alternative
 *
 * This version uses Object.entries().
 */
for (const [key, value] of Object.entries(tofu)) {
      tofuDetails += value;
}
```

For iterating over array elements, using C-style `for` loops is permitted, especially if keeping track of the indices is important. However, to promote consistency, using `for...of` loops is generally preferred for straightforward iteration.

7.4.6.2 `switch` Statements

7.4.6.2.1 Fall-through Comment

Mark executions that will fall through to the next case with the following comment: `/* Fall through */`.

Since the usage of the cascading behavior of `switch` statements might be misconstrued as an oversight, the rationale for adding this comment is to make the intention explicit and unambiguous.

Incorrect:

```
switch (doggoSize) {
    case 1:
          smol();

    case 2:
          lorge();

    default:
          pup();
          break;
}
```

Correct:

```
switch (doggoSize) {
    case 1:
          smol();

    /* Fall through */
    case 2:
          lorge();

    /* Fall through */
    default:
          pup();
          break;
}
```

7.4.6.2.2 Mandatory `default`

Always include the `default` as the last case of the `switch` statement.

In order to signal the end of the `switch` control structure explicitly, place a `break` statement to terminate the `default` case, even if there is no behavior specific to it.

Incorrect:

```
switch (doggoSize) {
    case 1:
          smol();
          break;

    case 2:
          lorge();
          break;
}
```

Correct:

```
switch (doggoSize) {
    case 1:
          smol();
          break;

    case 2:
          lorge();
          break;

    default:
          break;
}
```

7.5 Naming Conventions

7.5.1 Definition of Camel Case

To convert a phrase into camel case, the following rules are applied:

a) Remove diacritics and punctuation marks.
b) Lowercase all the letters, capitalizing only the first letter of each constituent word (with the exception of the first word if lower camel case is used).

Note that rule (b) also applies to acronyms and other similar constructs. Although this does not preserve their capitalization, this scheme is prescribed in the interest of implementing a uniform and deterministic procedure for converting all types of phrases to camel case.

Incorrect:

```
façadeColor, URLString, studentID, newIPAddress
```

Correct (Lower Camel Case):

```
facadeColor, urlString, studentId, newIpAddress
```

7.5.2 Variable Names

Use lower camel case for variable names. Do not use Hungarian notation, and avoid using abbreviations that are not universally recognized.

Incorrect:
```
numComp, nBuyers
```

Correct:
```
numComponents, numBuyers
```

7.5.3 Constant Names

Use constant case (that is, snake case with all the letters capitalized) for names of deeply immutable constants.

Incorrect:
```
DBURL, CONNECTIONSTRING
```

Correct:
```
DB_URL, CONNECTION_STRING
```

Examples of deeply immutable constants are environment variables stored in the `.env` file. Note that not all variables marked with `const` are deeply immutable (most are only marked with the said keyword to prevent unintended reassignment), thus exempting most of them from this naming convention.

7.5.4 Function Names

Use lower camel case for function names. Prefix their names with a descriptive action verb (or a linking verb for most functions that return boolean values). Avoid using abbreviations that are not universally recognized.

Incorrect:
```
numStudents(), validUsername()
```

Correct:
```
getNumStudents(), isValidUsername()
```

PART IV: Team Workflow

1 About the Repository

The source code of the web application is hosted on GitHub, a collaborative version control website for Git repositories. Git is a free and open-source distributed version control system that is covered by the GNU General Public License version 2.0[1].

The link to the team's remote repository (henceforth referred to as the "team repository" or simply as the "repository") is https://github.com/STSWENG-T1-AY2122-AWS-CodeBuild/powerzone-inventory.

Effective its creation on November 5, 2021, until the end of De La Salle University's Term 1, Academic Year 2021–2022, the visibility of the repository is set to public; however, direct access shall be granted only to the team members (and the course instructor, as needed). The decision to set the visibility to public was driven by the need to enforce branch protection rules (explained in **Section 3.2**) — a feature that is available to GitHub Free users only for public repositories. Guidelines are laid out in **Section 6** to maintain data privacy.

To restrict direct access to the repository and streamline the permission levels, a GitHub organization was created, and the team members were assigned to different GitHub teams based on their respective roles in the project. Their complete names, GitHub usernames, project roles, and GitHub team assignments are enumerated in Table IV.1.

Team Member	GitHub Username	Project Role	GitHub Team Assignment
Berjamin, Sandra Angela E.	`SBerj`	Product Owner	Analysts
Cua, Lander Peter E.	`LanderPeterCua`	Scrum Master	Analysts
Gaba, Jacob Bryan B.	`j-gaba`	Quality Assurance	Quality Assurance
Gonzales, Mark Edward M.	`memgonzales`	Developer	Developers
Lee, Hylene Jules G.	`hyleene`	Developer	Developers
Mata, Angeli Dianne F.	`adiannem`	Designer	Designers
Ong, Phoebe Clare L.	`pbong`	Designer	Designers
Racoma, Ian Angelo T.	`ianracoma`	Quality Assurance	Quality Assurance

Table IV.1. Team Members and GitHub Details

[1] The license can be accessed through this link: https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html.

2 Repository Permission Levels

In order to promote accountability among the team members and implement tiers of direct access to the repository, permission levels are set based on their GitHub team assignments (which, in turn, reflect their project roles). These are summarized in Table IV.2. The Github documentation also provides an exhaustive list of repository actions that are allowed for each permission level[2].

GitHub Team Assignment	Permission Level	Brief Description[3]
Developers	Admin	Can read, clone, and push to [the] repository. Can also manage issues, pull requests, and repository settings, including adding collaborators.
Designers	Write	Can read, clone, and push to [the] repository. Can also manage issues and pull requests.
Quality Assurance	Triage	Can read and clone [the] repository. Can also manage issues and pull requests.
Analysts	Read	Can read and clone [the] repository. Can also open and comment on issues and pull requests.

Table IV.2. Team Members and GitHub Details

Note that these permission levels reflect the most restrictive access available on Github Free that will allow the team members to perform their tasks. However, a distinction should be drawn between the actions that are permitted by the Github permission level and those permitted in line with the Project Roles discussed in **Part II.3 (Software Development Methodology)** and the Team Workflow discussed in **Part IV.4** — in this case, the latter has the overriding priority.

For example, although Triage allows the Quality Assurance team members to manage pull requests, this task is outside their purview; their permission level is only set to Triage since it is the most restrictive level that still permits managing issues (which is one of their primary responsibilities).

[2] The official documentation can be accessed through this link: https://docs.github.com/en/organizations/managing-access-to-your-organizations-repositories/repository-roles-for-an-organization.

[3] The brief descriptions are lifted directly from the “Manage access” page, which is accessed via the repository settings.

3 Repository Branches

This section discusses the repository branches and their respective protection rules. As an integral feature of version control, having multiple branches helps in the organization of the team workflow, as team members can see the status of the code based on the branch where it is located, and provides separate environments for each of the smaller teams (i.e., the developers, designers, quality assurance, and analysts) to work on their tasks in isolation, minimizing cases of merge conflicts and the deployment of incorrect versions of the project.

3.1 Repository Structure

The repository structure is patterned after a modified version of the Git workflow. The branches are described in Table IV.3.

Branch	Description
`main`	Master branch that contains the code for the version of the application that is deployed to the end-users
`staging`	Branch where features finished by the developers and designers are integrated with the rest of the code and where the battery of tests by the quality assurance team is conducted
`feature-<feature-name>`	Branch where the front-end and back-end components of a feature are integrated (also serves as the avenue for transferring code to and from the designers and the developers)
`feature-dev-<feature-name>`	Branch where the developers work on the dynamic front-end and back-end components of a feature
`feature-design-<feature-name>`	Branch where the designers work on the design and static front-end components of a feature

Table IV.3. Repository Branches

The connection between these branches is illustrated in Figure IV.1. The specifics of the workflow are detailed in **Section 4 (Team Workflow)**.

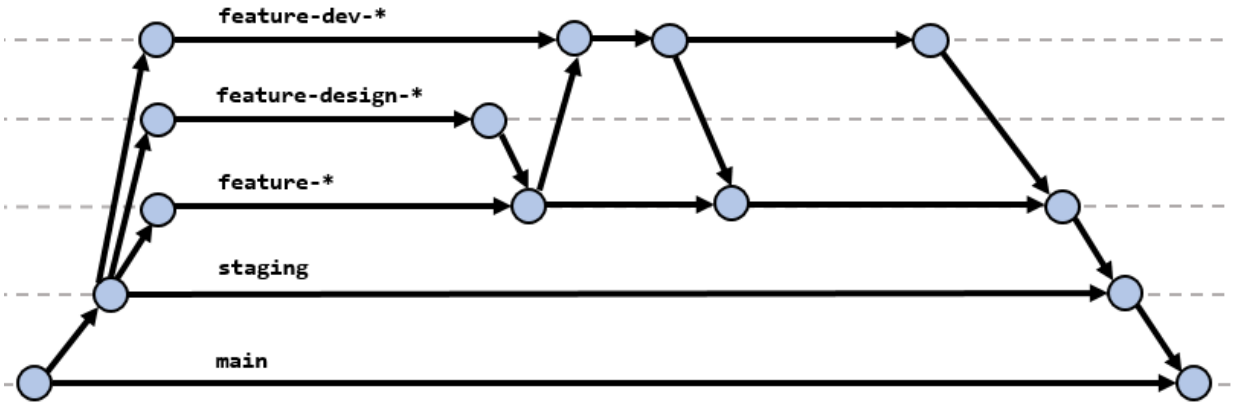

Figure IV.1. Visualization of the Repository Structure

3.2 Branch Protection Rules

The two most important branches in the modified Git workflow followed by the team, namely, `main` and `staging`, are subject to GitHub branch protection rules. Aside from prohibiting branch deletion without the Admin permission level, these rules also:

a. Require a pull request before merging and require an approval before it can be merged; and
b. Restrict who can push to matching branches to only the developers.

On the other hand, the three dedicated branches per feature are actively evolving and are deleted at the end of feature approval; thus, they are not covered by branch protection rules. Nevertheless, the team is expected to follow the specified workflow in properly managing these branches.

> **IMPORTANT:** Directly pushing to any of the branches is not allowed. Every code addition or modification should be coursed via a pull request to promote code review and maintain code quality. Moreover, team members are prohibited from merging their own pull requests.

4 Workflow

This section focuses on the team workflow, which is anchored in the Agile methodology and the Scrum framework explained in **Part II.1 (Software Development Methodology)**. This also expounds on the team's implementation of the modified Git workflow, which was preliminarily presented in the previous section on the repository branches.

4.1 Requirements Specification

Before any development begins, the product owner accomplishes the product backlog. The product owner also groups the stories according to their affinity to one another (i.e., user stories pertaining to a single feature are grouped together) and ranks these groups of stories according to their degree of importance in order to prioritize the most crucial features first.

4.2 Development

4.2.1 Start of New Feature

After the team has agreed on the user story to work on, development of the relevant feature commences. This requires the developers to create three new branches by cloning from the staging branch. These will serve as the dedicated branches for the new feature:

a. `feature-design-<feature-name>`, where the designers can work on the design and front-end development
b. `feature-dev-<feature-name>`, where the developers can work on the dynamic front-end and back-end development
c. `feature-<feature-name>`, where the designers and developers merge their code for that feature

Note that, since the `.env` file is not stored in the remote repository for security reasons (detailed in **Section 6**), it is necessary to include the copy of the `.env` file in the member's local machine inside the local copy of the branch in order for the web application to run.

In the interest of increasing velocity or preventing potential bottlenecks, it is possible for the designers and developers to work on multiple features concurrently. In this case, each feature should have the three dedicated branches.

> **IMPORTANT:** The designers and developers should ensure that they are working and committing to the correct branch. The following command is used to switch branches or restore working tree files in Git:
>
> ```
> git checkout <branch-name>
> ```

4.2.2 Design and Static Front-End Development

The designers shall create the prototype of the design and have it approved by the product owner (or have the consensus of the team, if necessary). Afterwards, they shall code the static front-end elements; commits are to be made in the `feature-designer-*` branch. Directly pushing to this branch is not allowed. Therefore, a designer should submit a pull request to the `feature-design-*` branch, which will be reviewed and merged (or rejected) by the other designer.

Once sufficient front-end elements have been added or the designers deem that dynamic front-end and back-end development can already start, a pull request shall be submitted to the `feature-*` branch. The pull request will be reviewed and merged (or rejected) by any of the developers. Moreover, the `feature-*` branch will also serve as the avenue for transferring code to and from the designers and the developers.

In order to increase the development velocity, it is encouraged that the designers code a bare-bones version of the design first — that is, the only requirement being the presence of all the HTML elements and their IDs — and already submit a pull request to the `feature-*` branch. In this manner, the developers can start working on some of the back-end features concurrent with the refinement of the design.

4.2.3 Dynamic Front-End and Back-End Development

The developers shall implement the dynamic interactions of the application's front-end, code the back-end features, and connect the back-end and the front-end components in light of the MVC architecture. Commits are to be made in the `feature-dev-*` branch, which will be reviewed and merged (or rejected) by the other developer.

Once sufficient dynamic front-end and back-end functionalities have been completed or a significant part of the feature has already been accomplished, a pull request shall be submitted to the `feature-*` branch. The pull request will be reviewed and merged (or rejected) by the other developer. Moreover, the `feature-*` branch will also serve as the avenue for transferring code to and from the designers and the developers.

Following the principles of test-driven development, developers are required to write their own unit tests. These will serve as independent tests to be done whenever a new feature is being developed in order to preliminarily ascertain that the function they are currently writing behaves as expected and does not affect any of the already-implemented methods, before finally integrating it with the rest of the codebase.

4.2.4 End of Feature Development

Once both front-end and back-end components of the feature have been completed, they will again be subjected to a battery of unit tests and manual checks by the developers and designers. These repeated checks are in place to promote code quality and functionality, even prior to testing by the quality assurance team.

After passing these tests, a pull request for that feature will be submitted to the `staging` branch. Note that, even if the feature has already been merged to the `staging` branch, the dedicated feature branches should not be deleted until the associated user story has been marked as "DONE."

4.3 Testing

To ensure the integrity of the web application, a series of tests are to be performed by the quality assurance team. These quality assurance tests are to be performed in the `staging` branch. The following section outlines how issue documentation is performed, along with the processes involved in the testing methodology.

4.3.1 Feature Testing

4.3.1.1 Integration Testing

Following unit testing, integration testing will be performed by the quality assurance team to verify that the incorporation of each unit with previously implemented functionalities will proceed seamlessly. In addition, this kind of testing will help minimize the chances that further complications will arise from the unit when future units are integrated into the application (Santacrose, 2021)[4].

4.3.1.2 Smoke Testing

When a feature passes integration testing, smoke testing is performed to conduct broad and shallow tests to quickly look for obvious major defects for each build. Here, the developers provide a build, which is a version of the web application for the quality assurance team to test. Builds are important for monitoring the progress and improvements made for any defects in the development of the application (TestOrigen, 2018)[5]. This testing focuses on verifying whether crucial functionalities of the application work, but not necessarily as intended, to detect any critical bugs sooner (PerformanceLab, 2020)[6].

4.3.1.3 Functional Testing

After smoke testing, more detailed and scrutinous testing will be conducted in the form of functional testing. This form of testing is performed after the more shallow smoke testing in order to ensure that the added feature performs according to its expected functionality (BrowserStack, 2021)[7].

---

[4] Santacrose, F. (2021). *20 Types of tests every developer should know*. Semaphore. https://semaphoreci.com/blog/20-types-of-testing-developers-should-know

[5] TestOrigen. (2018). *Differentiating build, release & deployment in software testing*. https://www.testorigen.com/differentiating-build-release-deployment-in-software-testing/

[6] PerformanceLab. (2020). *The ultimate guide to smoke testing*. https://performancelabus.com/smoke-testing-ultimate-guide/

[7] Bose, S. (2021). *Functional testing: a detailed guide*. https://www.browserstack.com/guide/functional-testing

4.3.1.4 System Testing

In contrast to functional testing which is focused on the performance of a single feature, system testing is performed to evaluate the functionality of a section of the web application or the entire application as a whole. All existing features are expected to undergo functional testing before being subjected to system testing, which will incorporate non-functional testing as well (Eaton, 2016)[8].

4.3.2 Issue Tracking

Two main tools will be used in tracking any issues that arise from testing, the first being GitHub's own issue tracker, and the other being Google Sheets. More detailed issue tracking will be performed in GitHub's Issue tracker by utilizing its collaborator assignment features and labeling functionality. Google Sheets, on the other hand, will be utilized as a general tracking tool showing which features have yet to be implemented and need to be fixed if deemed buggy.

> **IMPORTANT:** Once an issue has been fixed by the developers or designers, the commit and pull request messages should include the issue number. They should also write a comment on the issue itself, including the commit ID (hash) and a brief description of the fix. **However, only the quality assurance members are allowed to close an issue.**
>
> For fixing issues, a pull request may be submitted directly to the `feature-*` branch (without the need to go through the `feature-design-*` or `feature-dev-*` branch) to expedite re-evaluation by the quality assurance team.

4.3.3 Feature Approval

Once a build functions as expected and all tracked issues, if any, have been resolved, the quality assurance team will permit the approval of an implemented feature by marking it as "PASSED."

This signals to the product owner to evaluate the designated user story and the Scrum master to tag it as "DONE," provided that it has met all the functional and non-functional requirements outlined in the acceptance criteria. In the event that a feature is not cleared for approval, concerns and issues will be directed to the developers for modifications and improvements to the web application.

> **IMPORTANT:** After a feature has been successfully tagged as "DONE," the three dedicated branches (`feature-design-*`, `feature-dev-*`, and `feature-*`) should be deleted from the repository to prevent stale branches.

[8] Eaton, M. (2016). *System testing and functional testing*. https://www.agnosticdev.com/blog-entry/testing/system-testing-and-functional-testing

For every new feature, the team will continue cycling through the development and testing phases until the end of the sprint.

4.4 End-of-Sprint Activities

At the end of every sprint, a pull request for the features that have been marked as “DONE” shall be submitted to the `main` branch by one of the developers. It will be merged by the other developer; since this is the master branch containing the code for the version of the application to be deployed to the end users, it must be ensured that the version to be integrated into `main` is exactly the same as the one evaluated as “PASSED” by the quality assurance team and marked as “DONE” by the Scrum master.

The team shall also participate in Scrum ceremonies and review or deliver the necessary Scrum artifacts, as outlined in **Part II.2 (Software Development Methodology)**.

4.5 Deployment

Once the application has been completed, it will be deployed on Heroku for demonstration to the course instructor.

5 Commits & Pull Requests

This section describes the measures that should be taken to maintain the security of the repository.

5.1 Commits

Each commit should be small and focused; while there is no hard rule on when a commit should be made, the rationale is to maximize the use of version control in maintaining a systematic history of changes and promoting ease of reverting to an earlier version should the need arise. Follow these guidelines in writing the commit message:

a. Use an imperative sentence for the subject line (do not use the past tense).
b. Capitalize the first word of the subject line.
c. Do not end the subject line with a period.
d. Keep the subject line concise, and add a body to explain the commit only if necessary.
e. If the commit includes a fix to an issue raised by the quality assurance team, include the issue number in the description for better tracking.

5.2 Pull Requests

5.2.1 Submitting Pull Requests

While a pull request shall cover a collection of commits, it is important for these commits to be related and for the pull request as a whole to be focused on one component or issue fix. The rationale is to promote velocity by reducing the complexity of the code to be reviewed. Follow these guidelines in writing the pull request:

a. Place a brief and descriptive title.
b. Add a summary of the changes or additions included in the pull request.
c. If the pull request includes a fix to an issue raised by the quality assurance team, include the issue number in the description for better tracking.
d. If deemed helpful to quickly navigate through front-end changes, screenshots may be included in the description.

Before submitting a pull request to the `feature` and `staging` branches, the developers are expected to ensure that the code passes all the unit tests at the minimum. Moreover, before submitting a pull request to the `main` branch, it is imperative to double-check that the features have been tagged as “DONE” by the Scrum master.

These restrictions do not apply to submitting pull requests to the `feature-design-*` and `feature-dev-*` branches, as these are rapidly evolving branches meant only for use by either the designers or the developers to allow them to work concurrently. Nevertheless, they are expected to reach their own internal agreement in maintaining code quality.

5.2.2 Merging Pull Requests

The following are the minimum requirements for a pull request to be merged:

a. It should follow the coding standards discussed in **Part III** (Team Standards).
b. It should pass all the unit tests that have been written up to the time when the pull request was submitted.
c. It should undergo manual code review by the designers or developers, following the team workflow discussed in **Section 4**.
d. It should not result in any merge conflict, as determined via GitHub's automatic merge conflict checker.
e. It should not leak any confidential or sensitive information (explained in **Section 6** on the repository security); it should pass GitGuardian's automated security check.

Failing any one of these will result in the pull request being rejected.

6 Repository Security

This section describes the measures that should be taken to maintain the security of the repository.

6.1 Files That Should Not Be Committed

The `.gitignore` file enumerates the files and directories that should not be committed to the repository and are, thus, ignored by Git. The initial version of the `.gitignore` file (henceforth referred to as the "base version") in the team repository is taken from GitHub's recommended template for source codes that use Node.js[9] (which is this project's server environment).

Additional files and directories may be added to `.gitignore` as needed; however, removing files and directories from the base version is prohibited. In other words, all subsequent versions of `.gitignore` should contain all the files and directories specified in GitHub's recommended template for source codes that use Node.js.

Special attention should be given to the `.env` file, which contains the uniform resource locator (URL) for connecting to the remote database. In compliance with the Data Privacy Act of 2012 (Republic Act 10173)[10], this URL is strictly confidential and should be known only to the team members (and the course instructor, as needed); the period of this confidentiality on the part of the team members is perpetual.

Therefore, the `.env` file should be stored only inside the local machines of the team members. Although the `.env` file is included in the `.gitignore` file, care should still be taken to ensure that this file is not to be committed to the repository. The same rule extends to any file that contains confidential or sensitive information, including but not limited to the URL of the remote database, application programming interface (API) keys, secure shell (SSH) keys, and passwords.

This project also uses GitGuardian, a tool for automatically detecting the presence of confidential or sensitive information that has been leaked into the repository. As mentioned in **Section 5.3**, one of the necessary conditions for a pull request to be merged is passing **GitGuardian's** automated security check (alongside a manual security review).

> **IMPORTANT:** Any confidential or sensitive information that has been committed shall already be considered compromised. Therefore, it is imperative to report this immediately to the team so that the necessary actions can be promptly taken[11], the existing repository can be deleted, and a new team repository can be created.

[9] The template can be accessed through this link: https://github.com/github/gitignore/blob/master/Node.gitignore.

[10] The law can be accessed through this link: https://www.privacy.gov.ph/data-privacy-act/.

[11] The steps to be performed in this situation are enumerated and explained on this page from GitHub's documentation: https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository.

6.2 Vulnerable Dependencies

Although secure web development is outside the scope of the course, the team members are ethically responsible to integrate some basic levels of security. To this end, this project also uses Github's **Dependabot** to automatically check for vulnerable dependencies.

It is imperative for the team to update dependencies that are tagged with a "Critical" security alert. However, for alert levels lower than "Critical," the team should first have a meeting to assess the impact of this vulnerability vis-à-vis the magnitude of the ensuing changes to the technology stack or project infrastructure as a result of updating the pertinent package.

Appendix

Appendix A - Product Backlog

See the following page.

Powerzone Product Management

Appendix A

Product Backlog

Team Number AWS CodeBuild

Section STSWENG – S12

Team Members Berjamin, Sandra Angela E.
Cua, Lander Peter E.
Gaba, Jacob Bryan B.
Gonzales, Mark Edward M.
Lee, Hylene Jules G.
Mata, Angeli Dianne F.
Ong, Phoebe Clare L.
Racoma, Ian Angelo T.

Date Updated November 12, 2021

Table of Contents

1 Product Backlog

US #1

As a delivery manager, I want to register as a delivery manager so that I can login to the web application.
Target Sprint: Sprint #1
Pre-condition: The company has given the current user a role of a delivery manager.
Scenario: 1. The delivery manager chooses to register. 2. The system displays all the required information the delivery manager has to input. 3. The delivery manager enters their details respectively and submits. 4. The system validates the information inputted. 5. The system presents a message that their registration is in the queue of being reviewed by the administrator.
Post-condition: The system presents a message that their registration is in the queue of being reviewed by the administrator.
Acceptance Criteria: ● The registrations form for the delivery manager contains the following information: ○ Email ○ Name ○ Username ○ Role ○ Password ● The system uses the following rules to validate the entered information: ○ The email should not have been used by other users in the system.

○ The username has not yet been used by other users in the system. It should also contain at least one character from any of the following groups: lower case, upper case, numbers, and punctuations. ○ The password should at least have 12 characters. It should also contain characters in all of these groups: lower case, upper case, numbers, and punctuations. ● The system shows what is invalid if it finds something wrong with the information inputted.

US #2

As a transaction cashier, I want to register as a transaction cashier so that I can login to the web application.
Target Sprint: Sprint #1
Pre-condition: The company has given the current user a role of a transaction cashier.
Scenario: 1. The transaction cashier chooses to register. 2. The system displays all the required information the transaction cashier has to input. 3. The transaction cashier enters their details respectively and submits. 4. The system validates the information inputted. 5. The system presents a message that their registration is in the queue of being reviewed by the administrator.
Post-condition: The system presents a message that their registration is in the queue of being reviewed by the administrator.
Acceptance Criteria: ● The registrations form for the transaction cashier contains the following information: ○ Email ○ Name ○ Username ○ Role

○ Password ● The system uses the following rules to validate the entered information: ○ The email should not have been used by other users in the system. ○ The username has not yet been used by other users in the system. It should also contain at least one character from any of the following groups: lower case, upper case, numbers, and punctuations. ○ The password should at least have 12 characters. It should also contain characters in all of these groups: lower case, upper case, numbers, and punctuations. ● The system shows what is invalid if it finds something wrong with the information inputted.

US #3

As an inventory manager, I want to register as an inventory manager so that I can login to the web application.
Target Sprint: Sprint #1
Pre-condition: The company has given the current user a role of an inventory manager.
Scenario: 1. The inventory manager chooses to register. 2. The system displays all the required information the inventory manager has to input. 3. The inventory manager enters their details respectively and submits. 4. The system validates the information inputted. 5. The system presents a message that their registration is in the queue of being reviewed by the administrator.
Post-condition: The system presents a message that their registration is in the queue of being reviewed by the administrator.
Acceptance Criteria: ● The registrations form for the inventory manager contains the following information: ○ Email

- Name
- Username
- Role
- Password

- The system uses the following rules to validate the entered information:
  - The email should not have been used by other users in the system.
  - The username has not yet been used by other users in the system. It should also contain at least one character from any of the following groups: lower case, upper case, numbers, and punctuations.
  - The password should at least have 12 characters. It should also contain characters in all of these groups: lower case, upper case, numbers, and punctuations.
- The system shows what is invalid if it finds something wrong with the information inputted.

US #4

As a delivery manager, I want to login so that I can so that I can access information necessary for my role.
Target Sprint: Sprint #1
Pre-condition: The administrator has reviewed and passed the delivery manager's registration.
Scenario: 1. The delivery manager chooses to login. 2. The system displays all the required information the delivery manager has to input. 3. The delivery manager enters their details respectively and submits. 4. The system validates the information inputted. 5. The system displays the menu of the features available to the delivery manager.
Post-condition: The system displays the menu of the features available to the delivery manager.
Acceptance Criteria:

- The login form contains the following information:
    - Username
    - Password
- The system uses the following rules to validate the entered information:
    - The username and password should match an entry in the database of users, who have validated access to the web application.
- The menu of the features for the delivery manager contains the following:
    - View the transactions
    - View, add and edit the deliveries
    - View the inventory
    - View the selling price of products for the day
- The system shows what is invalid if it finds something wrong with the information inputted.

US #5

As a transaction cashier, I want to login so that I can so that I can access information necessary for my role.
Target Sprint: Sprint #1
Pre-condition: The administrator has reviewed and passed the transaction cashier's registration.
Scenario: 1. The transaction cashier chooses to login. 2. The system displays all the required information the transaction cashier has to input. 3. The transaction cashier enters their details respectively and submits. 4. The system validates the information inputted. 5. The system displays the menu of the features available to the transaction cashier.
Post-condition: The system displays the menu of the features available to the transaction cashier.

Acceptance Criteria:

- The login form contains the following information:
  - Username
  - Password
- The system uses the following rules to validate the entered information:
  - The username and password should match an entry in the database of users, who have validated access to the web application.
- The menu of the features for the transaction cashier contains the following:
  - View, add, edit the transactions
  - View, add, edit the deliveries
  - View the inventory
  - View the selling price of products for the day
- The system shows what is invalid if it finds something wrong with the information inputted.

US #6

As an inventory manager, I want to login so that I can so that I can access information necessary for my role.
Target Sprint: Sprint #1
Pre-condition: The administrator has reviewed and passed the inventory manager's registration.
Scenario: 1. The inventory manager chooses to login. 2. The system displays all the required information the inventory manager has to input. 3. The inventory manager enters their details respectively and submits. 4. The system validates the information inputted. 5. The system displays the menu of the features available to the inventory manager.

Post-condition: The system displays the menu of the features available to the inventory manager.
Acceptance Criteria: ● The login form contains the following information: ○ Username ○ Password ● The system uses the following rules to validate the entered information: ○ The username and password should match an entry in the database of users, who have validated access to the web application. ● The menu of the features for the inventory manager contains the following: ○ View the transactions ○ View the deliveries ○ View, add, edit the inventory ○ View, edit the selling price of products for the day ● The system shows what is invalid if it finds something wrong with the information inputted.

US #7

As an administrator, I want to login so that I can so that I can access information necessary for my role.
Target Sprint: Sprint #1
Pre-condition: --
Scenario: 1. The administrator chooses to login. 2. The system displays all the required information the administrator has to input. 3. The administrator enters their details respectively and submits. 4. The system validates the information inputted. 5. The system displays the menu of the features available to the administrator.

Post-condition: The system displays the menu of the features available to the administrator.
Acceptance Criteria: ● The login form contains the following information: ○ Username ○ Password ● The system uses the following rules to validate the entered information: ○ The username and password should match an entry in the database of users, who have validated access to the web application. ● The menu of the features for the administrator contains the following: ○ View, add and edit the transactions ○ View, add and edit the deliveries ○ View, add and edit the inventory ○ View, edit the selling price of products for the day ○ View and edit roles of users ● The system shows what is invalid if it finds something wrong with the information inputted.

US #8

As a delivery manager, I want to view and filter the transaction database so that I can see significant details I need to know about.
Target Sprint: Sprint #3
Pre-condition: The delivery manager has selected the viewing of transactions from the menu of features.

Scenario:

1. The system displays the whole database of transactions.
2. The delivery manager has selected a certain filter to view the transactions.
3. The system sorts the transactions according to the selected filter.

Post-condition: The system displays and sorts the transactions according to the selected filter.

Acceptance Criteria:

- The database contains the following information for each transaction:
    - Transaction ID
    - Product/s Purchased
        - Product Name
        - Unit Price
        - Quantity
        - Product Total (i.e. Quantity * Unit Price)
    - Customer Name
    - Customer Phone number
    - Transaction Date (i.e. date the product/s was/were bought from the company)
    - Total Transaction Price (i.e. total of all Product Totals for this transaction)
    - Status (i.e. Completed, Cancelled, Pending)
- The transactions of products should follow a FIFO method. For multiple purchases of a certain product, the older purchase of that product should be sold first.
- The status of a transaction is considered to be “Completed” once the status of the delivery that is connected to it has already been noted as “Completed.”
- The transactions with a status of “Completed” should no longer be displayed in the database but should not be completely deleted from the database.
- The database can be filtered according to the following attributes:

- Transaction Date
- Customer name
- Total Transaction Price

US #9

As a transaction cashier, I want to view and filter the transaction database so that I can see significant details I need to know about.
Target Sprint: Sprint #3
Pre-condition: The transaction manager has selected the viewing of transactions from the menu of features.
Scenario: 1. The system displays the whole database of transactions. 2. The transaction manager has selected a certain filter to view the transactions. 3. The system sorts the transactions according to the selected filter.
Post-condition: The system displays and sorts the transactions according to the selected filter.
Acceptance Criteria: ● The database contains the following information for each transaction: ○ Transaction ID ○ Product/s Purchased ■ Product Name ■ Unit Price ■ Quantity ■ Product Total (i.e. Quantity * Unit Price) ○ Customer Name

- Customer Phone number
- Transaction Date (i.e. date the product/s was/were bought from the company)
- Total Transaction Price (i.e. total of all Product Totals for this transaction)
- Status (i.e. Completed, Cancelled, Pending)

- The transactions of products should follow a FIFO method. For multiple purchases of a certain product, the older purchase of that product should be sold first.
- The status of a transaction is considered to be “Completed” once the status of the delivery that is connected to it has already been noted as “Completed.”
- The transactions with a status of “Completed” should no longer be displayed in the database but should not be completely deleted from the database.
- The database can be filtered according to the following attributes:
  - Transaction Date
  - Customer name
  - Total Transaction Price

US #10

As an inventory manager, I want to view and filter the transaction database so that I can see significant details I need to know about.
Target Sprint: Sprint #3
Pre-condition: The inventory manager has selected the viewing of transactions from the menu of features.
Scenario: 1. The system displays the whole database of transactions. 2. The inventory manager has selected a certain filter to view the transactions. 3. The system sorts the transactions according to the selected filter.

Post-condition: The system displays and sorts the transactions according to the selected filter.

Acceptance Criteria:

- The database contains the following information for each transaction:
    - Transaction ID
    - Product/s Purchased
        - Product Name
        - Unit Price
        - Quantity
        - Product Total (i.e. Quantity * Unit Price)
    - Customer Name
    - Customer Phone number
    - Transaction Date (i.e. date the product/s was/were bought from the company)
    - Total Transaction Price (i.e. total of all Product Totals for this transaction)
    - Status (i.e. Completed, Cancelled, Pending)
- The transactions of products should follow a FIFO method. For multiple purchases of a certain product, the older purchase of that product should be sold first.
- The status of a transaction is considered to be “Completed” once the status of the delivery that is connected to it has already been noted as “Completed.”
- The transactions with a status of “Completed” should no longer be displayed in the database but should not be completely deleted from the database.
- The database can be filtered according to the following attributes:
    - Transaction Date
    - Customer name
    - Total Transaction Price

US #11

As an administrator, I want to view and filter the transaction database so that I can see significant details I need to know about.
Target Sprint: Sprint #3
Pre-condition: The administrator has selected the viewing of transactions from the menu of features.
Scenario: 1. The system displays the whole database of transactions. 2. The administrator has selected a certain filter to view the transactions. 3. The system sorts the transactions according to the selected filter.
Post-condition: The system displays and sorts the transactions according to the selected filter.
Acceptance Criteria: ● The database contains the following information for each transaction: ○ Transaction ID ○ Product/s Purchased ■ Product Name ■ Unit Price ■ Quantity ■ Product Total (i.e. Quantity * Unit Price) ○ Customer Name ○ Customer Phone number ○ Transaction Date (i.e. date the product/s was/were bought from the company) ○ Total Transaction Price (i.e. total of all Product Totals for this transaction) ○ Status (i.e. Completed, Cancelled, Pending)

- The transactions of products should follow a FIFO method. For multiple purchases of a certain product, the older purchase of that product should be sold first.
- The status of a transaction is considered to be “Completed” once the status of the delivery that is connected to it has already been noted as “Completed.”
- The transactions with a status of “Completed” should no longer be displayed in the database but should not be completely deleted from the database.
- The database can be filtered according to the following attributes:
    - Transaction Date
    - Customer name
    - Total Transaction Price

US #12

As a transaction manager, I want to add transactions so that I can update the database for new transactions.
Target Sprint: Sprint #3
Pre-condition: The transaction manager has selected to add a new transaction.
Scenario: 1. The system displays all the required information the transaction manager has to input. 2. The transaction manager enters the details respectively and submits. 3. The system validates the information inputted. 4. The system adds the newly made transaction to the database. 5. The system displays the newly updated transaction database.
Post-condition: The system displays the newly updated transaction database.
Acceptance Criteria: ● The information to be filled up for each transaction consists of the following:

- Transaction ID
- Product/s Purchased
  - Product Name
  - Unit Price
  - Quantity
  - Product Total (i.e. Quantity * Unit Price)
- Customer Name
- Customer Phone number
- Transaction Date (i.e. date the product/s was/were bought from the company)
- Total Transaction Price (i.e. total of all Product Totals for this transaction)

- The default status of the transaction is “Pending.”
- The system uses the following rules to validate the entered information:
  - All prices should have only up to 2 decimal points.
  - The phone number has to consist of 10 integers.
- The system shows what is invalid if it finds something wrong with the information inputted.

US #13

As an administrator, I want to add transactions so that I can update the database for new transactions.
Target Sprint: Sprint #3
Pre-condition: The administrator has selected to add a new transaction.

Scenario:

1. The system displays all the required information the administrator has to input.
2. The administrator enters the details respectively and submits.
3. The system validates the information inputted.
4. The system adds the newly made transaction to the database.
5. The system displays the newly updated transaction database.

Post-condition: The system displays the newly updated transaction database.

Acceptance Criteria:

- The information to be filled up for each transaction consists of the following:
    - Transaction ID
    - Product/s Purchased
        - Product Name
        - Unit Price
        - Quantity
        - Product Total (i.e. Quantity * Unit Price)
    - Customer Name
    - Customer Phone number
    - Transaction Date (i.e. date the product/s was/were bought from the company)
    - Total Transaction Price (i.e. total of all Product Totals for this transaction)
- The default status of the transaction is “Pending.”
- The system uses the following rules to validate the entered information:
    - All prices should have only up to 2 decimal points.
    - The phone number has to consist of 10 integers.
- The system shows what is invalid if it finds something wrong with the information inputted.

US #14

As a transaction cashier, I want to edit transactions so that I can update the database accordingly.
Target Sprint: Sprint #3
Pre-condition: The transaction cashier has selected to edit a transaction.
Scenario: 1. The system displays the transaction in edit mode. 2. The transaction cashier edits all the attributes of the transaction and updates the status to “Cancelled.” 3. The system validates the information inputted. 4. The system updates the transactions database with the newly made changes. 5. The system displays the newly updated transactions database.
Post-condition: The system displays the newly updated transactions database.
Acceptance Criteria: ● The system uses the following rules to validate the entered information: ○ All prices should have only up to 2 decimal points. ○ The phone number has to consist of 10 integers. ● The system shows what is invalid if it finds something wrong with the information inputted.

US #15

As an administrator, I want to edit transactions so that I can update the database accordingly.
Target Sprint: Sprint #3
Pre-condition: The administrator has selected to edit a transaction.

Scenario:

1. The system displays the transaction in edit mode.
2. The administrator edits all the attributes of the transaction and updates the status to “Cancelled.”
3. The system validates the information inputted.
4. The system updates the transactions database with the newly made changes.
5. The system displays the newly updated transactions database.

Post-condition: The system displays the newly updated transactions database.

Acceptance Criteria:

- The system uses the following rules to validate the entered information:
    - All prices should have only up to 2 decimal points.
    - The phone number has to consist of 10 integers.
- The system shows what is invalid if it finds something wrong with the information inputted.

US #16

As a delivery manager, I want to view and filter the delivery database so that I can see significant details I need to know about.

Target Sprint: Sprint #3

Pre-condition: The delivery manager has selected the viewing of deliveries from the menu of features.

Scenario:

1. The system displays the whole database of deliveries.
2. The delivery manager has selected a certain filter to view the deliveries.
3. The system sorts the deliveries according to the selected filter.

Post-condition: The system displays and sorts the deliveries according to the selected filter.

Acceptance Criteria:

- The database contains the following information for each delivery:
    - Transaction ID
    - Delivery Date
    - Customer Name
    - Customer Phone number
    - Warehouse Location
    - Drop-off Location
    - Signatories
        - Delivery Manager
        - Customer
        - Driver
    - Status (i.e. Completed, Cancelled, Pending)
- The deliveries with a status of “Completed” should no longer be displayed in the database but should not be completely deleted from the database.
- The database can be filtered according to the following attributes:
    - Delivery Date
    - Customer Name

US #17

As a transaction cashier, I want to view and filter the delivery database so that I can see significant details I need to know about.
Target Sprint: Sprint #3
Pre-condition: The transaction cashier has selected the viewing of deliveries from the menu of features.

Scenario:

1. The system displays the whole database of deliveries.
2. The transaction cashier has selected a certain filter to view the deliveries.
3. The system sorts the deliveries according to the selected filter.

Post-condition: The system displays and sorts the deliveries according to the selected filter.

Acceptance Criteria:

- The database contains the following information for each delivery:
    - Transaction ID
    - Delivery Date
    - Customer Name
    - Customer Phone number
    - Warehouse Location
    - Drop-off Location
    - Signatories
        - Delivery Manager
        - Customer
        - Driver
    - Status (i.e. Completed, Cancelled, Pending)
- The deliveries with a status of “Completed” should no longer be displayed in the database but should not be completely deleted from the database.
- The database can be filtered according to the following attributes:
    - Delivery Date
    - Customer Name

US #18

As an inventory manager, I want to view and filter the delivery database so that I can see significant details I need to know about.
Target Sprint: Sprint #3
Pre-condition: The inventory manager has selected the viewing of deliveries from the menu of features.
Scenario: 1. The system displays the whole database of deliveries. 2. The inventory manager has selected a certain filter to view the deliveries. 3. The system sorts the deliveries according to the selected filter.
Post-condition: The system displays and sorts the deliveries according to the selected filter.
Acceptance Criteria: ● The database contains the following information for each delivery: ○ Transaction ID ○ Delivery Date ○ Customer Name ○ Customer Phone number ○ Warehouse Location ○ Drop-off Location ○ Signatories ■ Delivery Manager ■ Customer ■ Driver ○ Status (i.e. Completed, Cancelled, Pending)

- The deliveries with a status of “Completed” should no longer be displayed in the database but should not be completely deleted from the database.
- The database can be filtered according to the following attributes:
    - Delivery Date
    - Customer Name

US #19

As an administrator, I want to view and filter the delivery database so that I can see significant details I need to know about.
Target Sprint: Sprint #3
Pre-condition: The administrator has selected the viewing of deliveries from the menu of features.
Scenario: 1. The system displays the whole database of deliveries. 2. The administrator has selected a certain filter to view the deliveries. 3. The system sorts the deliveries according to the selected filter.
Post-condition: The system displays and sorts the deliveries according to the selected filter.
Acceptance Criteria: ● The database contains the following information for each delivery: ○ Transaction ID ○ Delivery Date ○ Customer Name ○ Customer Phone number ○ Warehouse Location

- Drop-off Location
- Signatories
  - Delivery Manager
  - Customer
  - Driver
- Status (i.e. Completed, Cancelled, Pending)

- The deliveries with a status of “Completed” should no longer be displayed in the database but should not be completely deleted from the database.
- The database can be filtered according to the following attributes:
  - Delivery Date
  - Customer Name

US #20

As a delivery manager, I want to add deliveries so that I can update the database for new deliveries.
Target Sprint: Sprint #3
Pre-condition: The delivery manager has selected to add a new delivery.
Scenario: 1. The system displays all the required information the delivery manager has to input. 2. The delivery manager enters the details respectively and submits. 3. The system validates the information inputted. 4. The system adds the newly made delivery to the database. 5. The system displays the newly updated delivery database.
Post-condition: The system displays the newly updated delivery database.
Acceptance Criteria:

- The information to be filled up for each delivery consists of the following:
    - Transaction ID
    - Delivery Date
    - Customer Name
    - Customer Phone number
    - Warehouse Location
    - Drop-off Location
    - Signatories
        - Delivery Manager
        - Customer
        - Driver
- The default status of the delivery is “Pending.”
- The system uses the following rules to validate the entered information:
    - The Transaction ID is a valid and existing Transaction ID.
    - The phone number has to consist of 10 integers.
- The system shows what is invalid if it finds something wrong with the information inputted.

US #21

As a transaction cashier, I want to add deliveries so that I can update the database for new deliveries.
Target Sprint: Sprint #3
Pre-condition: The transaction cashier has selected to add a new delivery.

Scenario:

1. The system displays all the required information the transaction cashier has to input.
2. The transaction cashier enters the details respectively and submits.
3. The system validates the information inputted.
4. The system adds the newly made delivery to the database.
5. The system displays the newly updated delivery database.

Post-condition: The system displays the newly updated delivery database.

Acceptance Criteria:

- The information to be filled up for each delivery consists of the following:
    - Transaction ID
    - Delivery Date
    - Customer Name
    - Customer Phone number
    - Warehouse Location
    - Drop-off Location
    - Signatories
        - Delivery Manager
        - Customer
        - Driver
- The default status of the delivery is “Pending.”
- The system uses the following rules to validate the entered information:
    - The Transaction ID is a valid and existing Transaction ID.
    - The phone number has to consist of 10 integers.
- The system shows what is invalid if it finds something wrong with the information inputted.

US #22

As an administrator, I want to add deliveries so that I can update the database for new deliveries.
Target Sprint: Sprint #3
Pre-condition: The administrator has selected to add a new delivery.
Scenario: 1. The system displays all the required information the administrator has to input. 2. The administrator enters the details respectively and submits. 3. The system validates the information inputted. 4. The system adds the newly made delivery to the database. 5. The system displays the newly updated delivery database.
Post-condition: The system displays the newly updated delivery database.
Acceptance Criteria: ● The information to be filled up for each delivery consists of the following: ○ Transaction ID ○ Delivery Date ○ Customer Name ○ Customer Phone number ○ Warehouse Location ○ Drop-off Location ○ Signatories ■ Delivery Manager ■ Customer ■ Driver ● The default status of the delivery is “Pending.”

- The system uses the following rules to validate the entered information:
    - The Transaction ID is a valid and existing Transaction ID.
    - The phone number has to consist of 10 integers.
- The system shows what is invalid if it finds something wrong with the information inputted.

US #23

As a delivery manager, I want to edit deliveries so that I can update the database accordingly.
Target Sprint: Sprint #3
Pre-condition: The delivery manager has selected to edit a delivery.
Scenario: 1. The system displays the delivery in edit mode. 2. The delivery manager edits all the attributes of the delivery and updates the status to “Cancelled.” 3. The system validates the information inputted. 4. The system updates the deliveries database with the newly made changes. 5. The system displays the newly updated deliveries database.
Post-condition: The system displays the newly updated deliveries database.
Acceptance Criteria: ● The system uses the following rules to validate the entered information: ○ The Transaction ID is a valid and existing Transaction ID. ○ The phone number has to consist of 10 integers. ● The system shows what is invalid if it finds something wrong with the information inputted.

US #24

As a transaction cashier, I want to edit deliveries so that I can update the database accordingly.
Target Sprint: Sprint #3
Pre-condition: The transaction cashier has selected to edit a delivery.
Scenario: 1. The system displays the delivery in edit mode. 2. The transaction cashier edits all the attributes of the delivery and updates the status to “Cancelled.” 3. The system validates the information inputted. 4. The system updates the deliveries database with the newly made changes. 5. The system displays the newly updated deliveries database.
Post-condition: The system displays the newly updated deliveries database.
Acceptance Criteria: ● The system uses the following rules to validate the entered information: ○ The Transaction ID is a valid and existing Transaction ID. ○ The phone number has to consist of 10 integers. ● The system shows what is invalid if it finds something wrong with the information inputted.

US #25

As an administrator, I want to edit deliveries so that I can update the database accordingly.
Target Sprint: Sprint #3
Pre-condition: The administrator has selected to edit a delivery.

Scenario: 1. The system displays the delivery in edit mode. 2. The administrator edits all the attributes of the delivery and updates the status to “Cancelled.” 3. The system validates the information inputted. 4. The system updates the deliveries database with the newly made changes. 5. The system displays the newly updated deliveries database.
Post-condition: The system displays the newly updated deliveries database.
Acceptance Criteria: ● The system uses the following rules to validate the entered information: ○ The Transaction ID is a valid and existing Transaction ID. ○ The phone number has to consist of 10 integers. ● The system shows what is invalid if it finds something wrong with the information inputted.

US #26

As a delivery manager, I want to view and filter the inventory database so that I can see significant details I need to know about.
Target Sprint: Sprint #2
Pre-condition: The delivery manager has selected the viewing of the inventory from the menu of features.
Scenario: 1. The system displays the whole database of the inventory. 2. The delivery manager has selected a certain filter to view the inventory. 3. The system sorts the inventory according to the selected filter.
Post-condition: The system displays and sorts the inventory according to the selected filter.

Acceptance Criteria:

- The database contains the following information for each stock:
    - Name
    - Quantity
    - Supplier
    - Storage Location
    - Price Purchased
    - Date Purchased
    - Status (i.e. Depleted, Expiring, In Stock, Critically Low)
- The stock is defined to be “Expiring” when it reaches 2 months from Date Purchased.
- The stock is defined to be “Depleted” when it reaches a Quantity of 0.
- The stock with a status of “Depleted” should no longer be displayed in the database but should not be completely deleted from the database.
- The stock is defined to be “Critically Low” if the Quantity reaches the following values:
    - Diesel: 300,000 liters
    - Gasoline: 500,000 liters
    - Premium 95 Gasoline: 100,000 liters
    - Premium 97 Gasoline: 100,000 liters
    - Kerosene: 300,000 liters
- The database can be filtered according to the following attributes:
    - Name
    - Date Purchased
    - Price Purchased

US #27

As a transaction cashier, I want to view and filter the inventory database so that I can see significant details I need to know about.
Target Sprint: Sprint #2
Pre-condition: The transaction cashier has selected the viewing of the inventory from the menu of features.
Scenario: 1. The system displays the whole database of the inventory. 2. The transaction cashier has selected a certain filter to view the inventory. 3. The system sorts the inventory according to the selected filter.
Post-condition: The system displays and sorts the inventory according to the selected filter.
Acceptance Criteria: ● The database contains the following information for each stock: ○ Name ○ Quantity ○ Supplier ○ Storage Location ○ Price Purchased ○ Date Purchased ■ Status (i.e. Depleted, Expiring, In Stock, Critically Low) ○ The stock is defined to be “Expiring” when it reaches 2 months from Date Purchased. ○ The stock is defined to be “Depleted” when it reaches a Quantity of 0. ○ The stock with a status of “Depleted” should no longer be displayed in the database but should not be completely deleted from the database. ○ The stock is defined to be “Critically Low” if the Quantity reaches the following values:

        - Diesel: 300,000 liters
        - Gasoline: 500,000 liters
        - Premium 95 Gasoline: 100,000 liters
        - Premium 97 Gasoline: 100,000 liters
        - Kerosene: 300,000 liters
- The database can be filtered according to the following attributes:
    - Name
    - Date Purchased
    - Price Purchased

US #28

As an inventory manager, I want to view and filter the inventory database so that I can see significant details I need to know about.
Target Sprint: Sprint #2
Pre-condition: The inventory manager has selected the viewing of the inventory from the menu of features.
Scenario: 1. The system displays the whole database of the inventory. 2. The inventory manager has selected a certain filter to view the inventory. 3. The system sorts the inventory according to the selected filter.
Post-condition: The system displays and sorts the inventory according to the selected filter.
Acceptance Criteria: ● The database contains the following information for each stock: ○ Name ○ Quantity

- Supplier
- Storage Location
- Price Purchased
- Date Purchased
- Status (i.e. Depleted, Expiring, In Stock, Critically Low)

- The stock is defined to be “Expiring” when it reaches 2 months from Date Purchased.
- The stock is defined to be “Depleted” when it reaches a Quantity of 0.
- The stock with a status of “Depleted” should no longer be displayed in the database but should not be completely deleted from the database.
- The stock is defined to be “Critically Low” if the Quantity reaches the following values:
  - Diesel: 300,000 liters
  - Gasoline: 500,000 liters
  - Premium 95 Gasoline: 100,000 liters
  - Premium 97 Gasoline: 100,000 liters
  - Kerosene: 300,000 liters
- The database can be filtered according to the following attributes:
  - Name
  - Date Purchased
  - Price Purchased

US #29

As an administrator, I want to view and filter the inventory database so that I can see significant details I need to know about.
Target Sprint: Sprint #2
Pre-condition: The administrator has selected the viewing of the inventory from the menu of features.

Scenario:

1. The system displays the whole database of the inventory.
2. The administrator has selected a certain filter to view the inventory.
3. The system sorts the inventory according to the selected filter.

Post-condition: The system displays and sorts the inventory according to the selected filter.

Acceptance Criteria:

- The database contains the following information for each stock:
    - Name
    - Quantity
    - Supplier
    - Storage Location
    - Price Purchased
    - Date Purchased
    - Status (i.e. Depleted, Expiring, In Stock, Critically Low)
- The stock is defined to be “Expiring” when it reaches 2 months from Date Purchased.
- The stock is defined to be “Depleted” when it reaches a Quantity of 0.
- The stock with a status of “Depleted” should no longer be displayed in the database but should not be completely deleted from the database.
- The stock is defined to be “Critically Low” if the Quantity reaches the following values:
    - Diesel: 300,000 liters
    - Gasoline: 500,000 liters
    - Premium 95 Gasoline: 100,000 liters
    - Premium 97 Gasoline: 100,000 liters
    - Kerosene: 300,000 liters
- The database can be filtered according to the following attributes:

- Name
- Date Purchased
- Price Purchased

US #30

As an inventory manager, I want to add stocks so that I can update the database for new stocks.
Target Sprint: Sprint #2
Pre-condition: The inventory manager has selected to add a new stock.
Scenario: 1. The system displays all the required information the inventory manager has to input. 2. The inventory manager enters the details respectively and submits. 3. The system validates the information inputted. 4. The system adds the newly acquired stock to the database. 5. The system displays the newly updated inventory database.
Post-condition: The system displays the newly updated inventory database.
Acceptance Criteria: ● The information to be filled up for each stock consists of the following: ○ Name ○ Quantity ○ Supplier ○ Storage Location ○ Price Purchased ○ Date Purchased ● The default status of the stock is “In Stock.”

- The system uses the following rules to validate the entered information:
    - All prices should have only up to 2 decimal points.
- The system shows what is invalid if it finds something wrong with the information inputted.

US #31

As an administrator, I want to add stocks so that I can update the database for new stocks.
Target Sprint: Sprint #2
Pre-condition: The administrator has selected to add a new stock.
Scenario: 1. The system displays all the required information the administrator has to input. 2. The administrator enters the details respectively and submits. 3. The system validates the information inputted. 4. The system adds the newly acquired stock to the database. 5. The system displays the newly updated inventory database.
Post-condition: The system displays the newly updated inventory database.
Acceptance Criteria: ● The information to be filled up for each stock consists of the following: ○ Name ○ Quantity ○ Supplier ○ Storage Location ○ Price Purchased ○ Date Purchased ● The default status of the stock is “In Stock.”

● The system uses the following rules to validate the entered information: ○ All prices should have only up to 2 decimal points. ● The system shows what is invalid if it finds something wrong with the information inputted.

US #32

As an inventory manager, I want to edit stocks so that I can update the database accordingly.
Target Sprint: Sprint #2
Pre-condition: The inventory manager has selected to edit a stock.
Scenario: 1. The system displays the stock in edit mode. 2. The inventory manager edits all the attributes of the stock. 3. The system validates the information inputted. 4. The system updates the inventory database with the newly made changes. 5. The system displays the newly updated inventory database.
Post-condition: The system displays the newly updated inventory database.
Acceptance Criteria: ● The system uses the following rules to validate the entered information: ○ All prices should have only up to 2 decimal points. ● The system shows what is invalid if it finds something wrong with the information inputted.

US #33

As an administrator, I want to edit stocks so that I can update the database accordingly.
Target Sprint: Sprint #2
Pre-condition: The administrator has selected to edit a stock.
Scenario: 1. The system displays the stock in edit mode. 2. The administrator edits all the attributes of the stock. 3. The system validates the information inputted. 4. The system updates the inventory database with the newly made changes. 5. The system displays the newly updated inventory database.
Post-condition: The system displays the newly updated inventory database.
Acceptance Criteria: ● The system uses the following rules to validate the entered information: ○ All prices should have only up to 2 decimal points. ● The system shows what is invalid if it finds something wrong with the information inputted.

US #34

As a delivery manager, I can view the selling prices of the products in the system for the day so that I can keep track of their prices.
Target Sprint: Sprint #1
Pre-condition: The delivery manager has logged in.

Scenario: 1. The system displays the selling prices of the products for the day.
Post-condition: The system displays the selling prices of the products for the day.
Acceptance Criteria: ● The selling prices should all have only 2 decimal places.

US #35

As a transaction cashier, I can view the selling prices of the products in the system for the day so that I can keep track of their prices.
Target Sprint: Sprint #1
Pre-condition: The transaction cashier has logged in.
Scenario: 1. The system displays the selling prices of the products for the day.
Post-condition: The system displays the selling prices of the products for the day.
Acceptance Criteria: ● The selling prices should all have only 2 decimal places.

US #36

As an inventory manager, I can view the selling prices of the products in the system for the day so that I can keep track of their prices.
Target Sprint: Sprint #1
Pre-condition: The inventory manager has logged in.
Scenario: 1. The system displays the selling prices of the products for the day.
Post-condition: The system displays the selling prices of the products for the day.
Acceptance Criteria: ● The selling prices should all have only 2 decimal places.

US #37

As an administrator, I can view the selling prices of the products in the system for the day so that I can keep track of their prices.
Target Sprint: Sprint #1
Pre-condition: The administrator has logged in.
Scenario: 1. The system displays the selling prices of the products for the day.

Post-condition: The system displays the selling prices of the products for the day.
Acceptance Criteria: ● The selling prices should all have only 2 decimal places.

US #38

As an inventory manager, I can edit the selling prices of the products in the system for the day so that I can update the prices accordingly.
Target Sprint: Sprint #1
Pre-condition: The inventory manager has logged in.
Scenario: 1. The system displays the selling prices of the products for the day. 2. The inventory manager has chosen to edit the selling prices of the products. 3. The system displays the selling prices in edit mode. 4. The inventory manager edits the prices. 5. The system validates the information entered. 6. The system updates the prices. 7. The system displays the newly updated selling prices for the day.
Post-condition: The system displays the newly updated selling prices for the day.
Acceptance Criteria: ● The system uses the following rules to validate the entered information: ○ The selling prices should all have only 2 decimal places.

US #39

As an administrator, I can edit the selling prices of the products in the system for the day so that I can update the prices accordingly.
Target Sprint: Sprint #1
Pre-condition: The administrator has logged in.
Scenario: 1. The system displays the selling prices of the products for the day. 2. The administrator has chosen to edit the selling prices of the products. 3. The system displays the selling prices in edit mode. 4. The administrator edits the prices. 5. The system validates the information entered. 6. The system updates the prices. 7. The system displays the newly updated selling prices for the day.
Post-condition: The system displays the newly updated selling prices for the day.
Acceptance Criteria: ● The system uses the following rules to validate the entered information: ○ The selling prices should all have only 2 decimal places.

US #40

As an administrator, I want to validate and edit the roles of the users in the system so that I can limit the access of each user accordingly.
Target Sprint: Sprint #1
Pre-condition: The administrator has selected the editing of roles in the menu of features.

Scenario:

1. The system displays a users database.
2. The administrator changes the status of some users to “Accepted.”
3. The system updates the database with the newly made changes.
4. The system displays the newly updated user database.

Post-condition: The system displays the newly updated user database.

Acceptance Criteria:

- The database contains the following information for each transaction:
  - Email
  - Name
  - Username
  - Role
  - Status (i.e. Accepted, Rejected, Pending)
- The administrator can only edit the status of the users. The admin cannot edit the other fields.
- The administrator cannot edit their own role.
- Once the status of a user is “Accepted,” they are granted limited access to the web application according to their initially registered role.
- Once the status of a user is “Rejected” or “Pending”, they are not granted access to the web application according to their initially registered role.

US #41

As a user, I want to view and edit the details of my account so that I can see and edit them accordingly.

Target Sprint: Sprint #1

Pre-condition: The user has selected the viewing of their details.

Scenario:

1. The system displays the details of the user.
2. The user chooses to edit their details.
3. The system displays the details in edit mode.
4. The user changes their details and submits.
5. The system updates the database of the newly made changes.
6. The system displays the newly updated details of the user.

Post-condition: The system displays the newly updated details of the user.

Acceptance Criteria:

- The details contain the following information for each user:
    - Email
    - Name
    - Username
    - Role
    - Password
- The user can edit everything except their role.
- The system uses the following rules to validate the entered information:
    - The email should not have been used by other users in the system.
    - The username has not yet been used by other users in the system. It should also contain at least one character from any of the following groups: lower case, upper case, numbers, and punctuations.
    - The password should at least have 12 characters. It should also contain characters in all of these groups: lower case, upper case, numbers, and punctuations.

US #42

As an administrator, I want to delete accounts so that I can erase unnecessary users from the system.
Target Sprint: Sprint #1
Pre-condition: The administrator has chosen to delete accounts in the users database.
Scenario: 1. The system displays the users database. 2. The administrator has chosen to delete a certain account. 3. The system deletes the user from the database. 4. The system displays the newly updated database.
Post-condition: The system displays the newly updated database.
Acceptance Criteria: ● The administrator can delete everyone except themself. ● The system uses the following rules to validate the entered information: ○ The email should not have been used by other users in the system. ○ The username has not yet been used by other users in the system. It should also contain at least one character from any of the following groups: lower case, upper case, numbers, and punctuations. ○ The password should at least have 12 characters. It should also contain characters in all of these groups: lower case, upper case, numbers, and punctuations.