Ahmad Mughrabi ID # 012233817

Amughr1@wgu.edu

Abstract

This is a manual covering all the diagrams relevant to the recent implementation done on December 12, 2024

The Cade

Project Documentation

Contents

[Class Diagram 3](#_Toc185120996)

[Summary 3](#_Toc185120997)

[Diagram 4](#_Toc185120998)

[Wireframes 5](#_Toc185120999)

[Details 5](#_Toc185121000)

[High Fidelity 5](#_Toc185121001)

[Mainscreen Page 5](#_Toc185121002)

[Add Inhouse Part Page 6](#_Toc185121003)

[Add Outsourced Part Page 6](#_Toc185121004)

[Update Products Page 7](#_Toc185121005)

[Report Page 7](#_Toc185121006)

[About Page 8](#_Toc185121007)

[Confirmation Page 8](#_Toc185121008)

[Low Fidelity 9](#_Toc185121009)

[Mainscreen Page 9](#_Toc185121010)

[Add Inhouse Part Page 10](#_Toc185121012)

[Add Outsourced Part Page 11](#_Toc185121013)

[Update Products Page 12](#_Toc185121014)

[Report Page 13](#_Toc185121015)

[About Page 14](#_Toc185121016)

[Confirmation Page 15](#_Toc185121017)

[Unit Testing 16](#_Toc185121018)

[Purpose 16](#_Toc185121019)

[Testing Overview 16](#_Toc185121020)

[Test Plan 16](#_Toc185121021)

[Items 16](#_Toc185121022)

[Features 17](#_Toc185121023)

[Deliverables 17](#_Toc185121024)

[Tasks 17](#_Toc185121025)

[Needs 17](#_Toc185121026)

[Pass/Fail Criteria 18](#_Toc185121027)

[Specifications 18](#_Toc185121028)

[Procedures 22](#_Toc185121029)

[Results 24](#_Toc185121030)

[DevOps 25](#_Toc185121031)

[Live URL 25](#_Toc185121032)

[Repository URL 25](#_Toc185121033)

[Quick Start Guide 25](#_Toc185121034)

# Class Diagram

## Summary

The diagram at hand shows how our new Controllers and Classes, that being the following-

* Report Controller
* Purchase
* PurchaseRepository
* PurchaseTest
* PurchaseRepositoryTest

have been implemented with their fields/methods , how they integrate and coexist with the codebase that was supplied by the customer.

## Diagram

# Wireframes

## Details

Below you’ll find the Low Fidelity and High Fidelity wireframes! The low fidelity goes displays the basic pages you’ll find that gives a good gist of what to expect from the website. How the layout of the land has been established and agreed upon. The high fidelity wireframe is a recreation of the full on frontend (the part the user interacts with) of our deployed app. It allows our end users to test out the pages and see what their day to day navigation will be like with the finished product.

## High Fidelity

Hosted Live on the Following URL -> <https://9amtech.github.io/The-Cade-High-Fidelity/index.html>

### Mainscreen Page

A screenshot of a computer

Description automatically generated

### Add Inhouse Part Page

A screenshot of a computer

Description automatically generated

### Add Outsourced Part Page

A screenshot of a computer screen

Description automatically generated

### Update Products Page

A screenshot of a computer

Description automatically generated

### Report Page

A screenshot of a computer

Description automatically generated

### About Page

A screenshot of a computer

Description automatically generated

### Confirmation Page

A screenshot of a computer

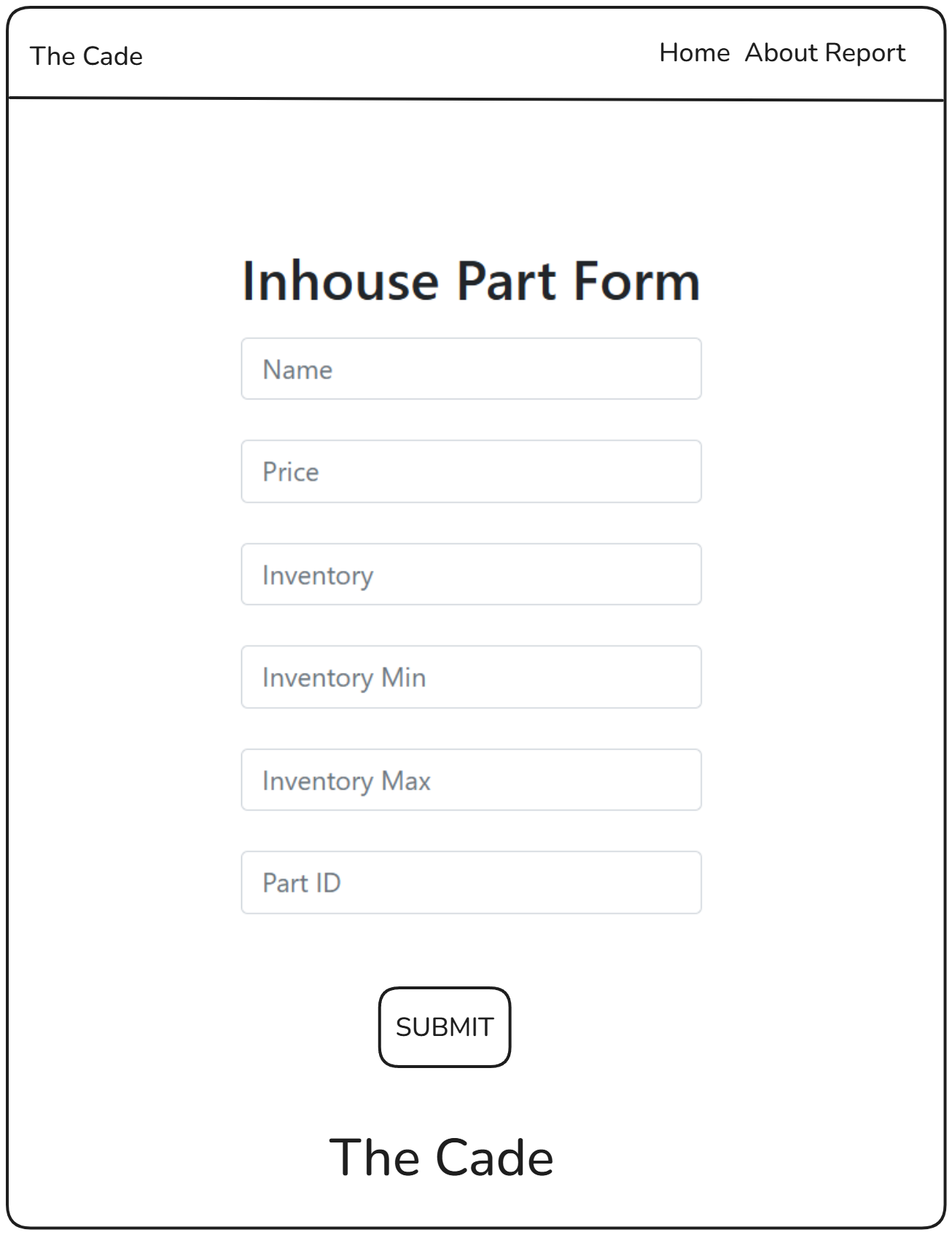
Description automatically generated

## Low Fidelity

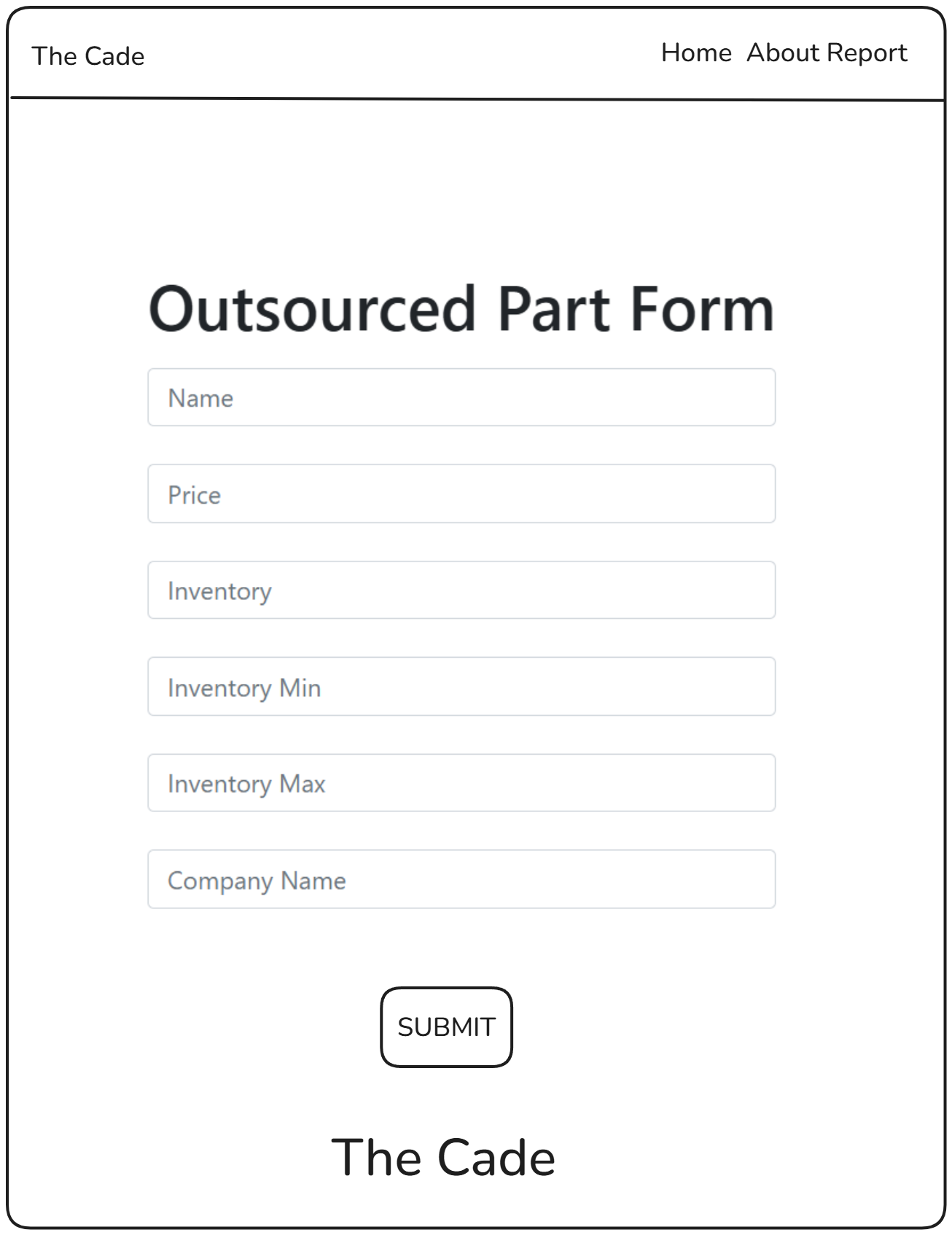
### Mainscreen Page

### 

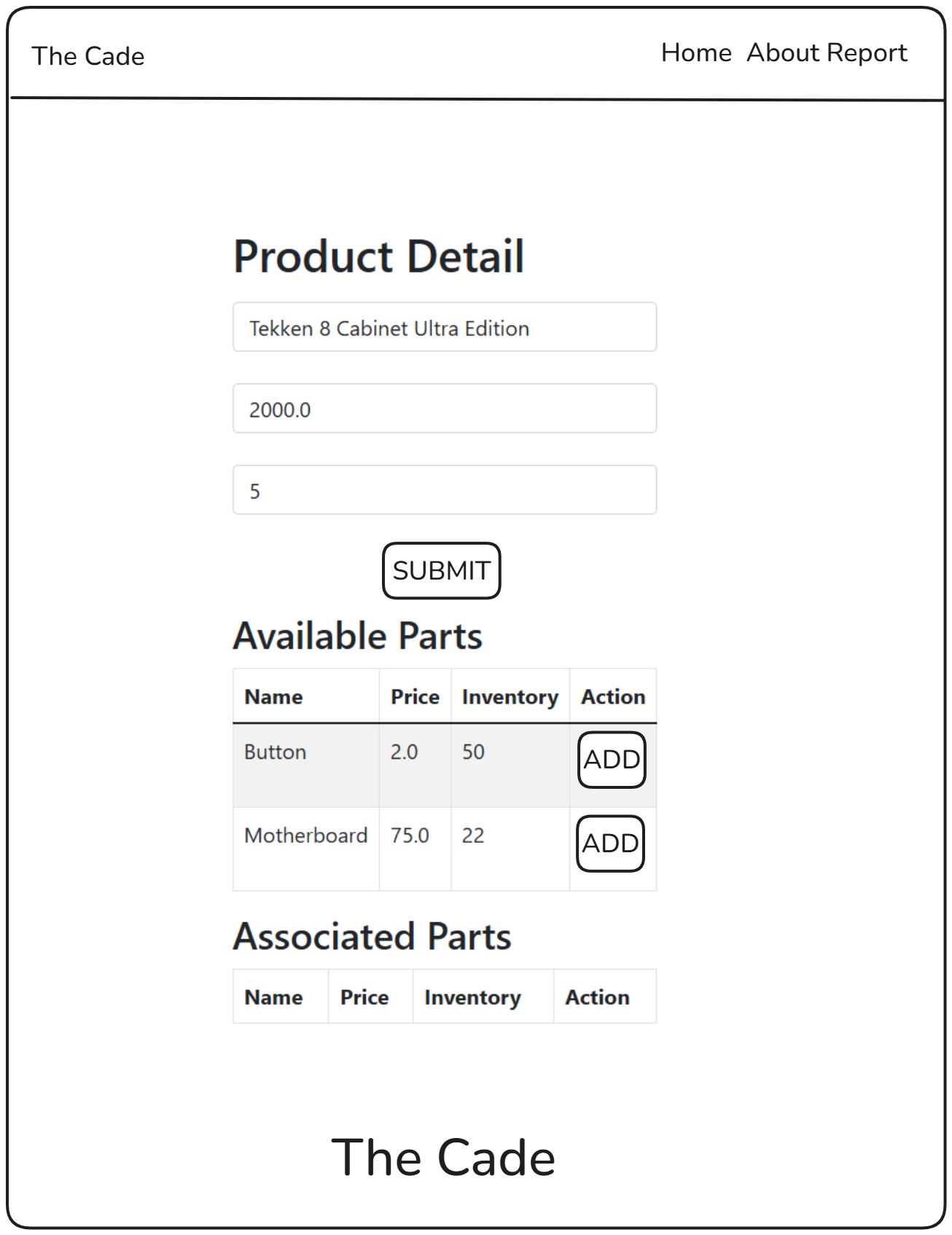
### Add Inhouse Part Page



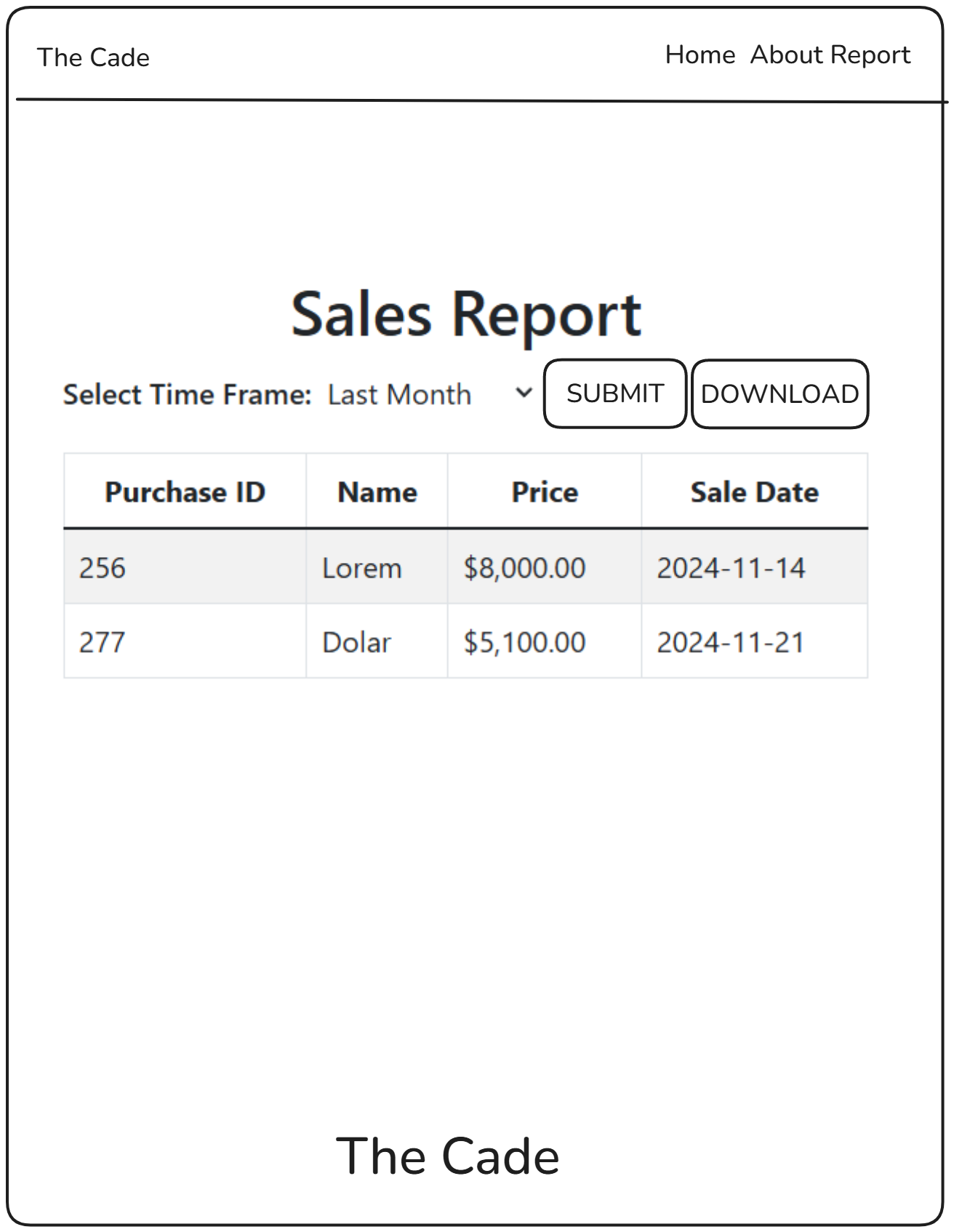
### Add Outsourced Part Page



### Update Products Page



### Report Page



### About Page



### Confirmation Page

A screenshot of a computer

Description automatically generated

# Unit Testing

## Purpose

The teams primary goal here was to determine if our newly created functionality works as per required. Dealing with monetary transactions means it’s of upmost importance to guarantee the following classes worked without fail. The results are satisfactory, all functionality has been implemented as intended. There is no remediation required.

## Testing Overview

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Expected Result** | **Actual Result** | **Pass?** |
| Purchase Class has been initialized properly. The class is able to get and set all of it’s fields. | We are able to get and set the following fields   * Id * Product * Sale Date | We are able to get and set through the Purchase Class. | Yes |
| Purchase Repository class’s query method findPurchasesBetweenDates is able to query for sales within specified Sale Dates. | If we add 3 sales, one 8 days away, 5 days away, and 2 days away and look for dates within 7 days the method will return two sales. | The method is able to find two purchases between the specified dates. | Yes |

## Test Plan

### Items

1. Development Environment
2. Testing Environment
3. Code to Test
4. Database

### Features

#### Test 1 – PurchaseTest.java Class

1. We are able to Get and Set the Id property.
2. We are able to Get and Set the Product property.
3. We are able to Get and Set the Sale Date property.

#### Test 2 – PurchaseRepositoryTest.java Class

1. We are able to get purchases within a start and end date.

### Deliverables

1. Documentation
2. Test Scripts
3. Screenshots of Test Code Samples
4. Screenshots of Test Results
5. Result Reports

### Tasks

1. Planning out what the required code shall and will do
2. Drafting up design documents to match requirements
3. Implementing the code
4. Creating unit tests to verify the code matches what requirements

### Needs

For our tests, we required the following

1. Integrated Development Environment (IDE) – Popular when working with Java, we opted to use IntelliJ IDEA 2024.1.4
2. Code to be Tested
   1. Actual Code – The actual targets of our test
      1. Purchase.java
      2. PurchaseRepository.java
   2. Dependency Code – Related to the Purchase class on the “Purchase” field.
      1. Product.java
      2. ProductRepository.java
3. Database – We used H2, an in memory / file database to act as our persistence.
4. Testing Library – For our tests we used the Java Library called Junit.

### Pass/Fail Criteria

#### Test 1 – PurchaseTest.java Class

1. Pass
   1. The tests would pass in this test script if we are able to successfully get or successfully set a property on the class. The protocol for a passing result was to document our findings, and proceed to the next code set.
2. Fail
   1. A failing test would begin our strategy for discovering why the test failed and remediation. In our case we would document what caused the test to fail and how we altered the code to pass.

#### Test 2 – PurchaseRepositoryTest.java Class

1. Pass
   1. The test in this case would pass if it’s able to successfully pull 2 sales from a sample of 3 sales this month, two of which were this week, and one was last week. The search criteria to make this test pass is looking for sales between today and 7 days ago. The protocol for a passing result was to document our findings.
2. Fail
   1. A failing test would begin our strategy for discovering why the test failed and remediation. In our case we would document what caused the test to fail and how we altered the code to pass.

### Specifications

#### Test 1 – PurchaseTest.java Class

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

#### Test 2 – PurchaseRepositoryTest.java Class

A screen shot of a computer screen

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

### Procedures

1. Configure IDE for Testing
   1. Make sure your IDE of choice has a good testing module/environment
   2. Add JUnit to pom.xml
2. Creating our Test Classes
   1. Create PurchaseTest
   2. Create PurchaseRepositoryTest
3. Testing
   1. PurchaseTest Class
      1. Create @BeforeEach annotated method for setting up a purchase and product before each test method
      2. Add Getter test method for each property
      3. Add Setters test method for each property
      4. Annotate each test method with @Test
   2. PurchaseRepositoryTest Class
      1. Annotate Test Class with @SpringBootTest and @Transactional
         1. @SpringBootTest will allow you to add and use the repository for our testing purpose
         2. @Transactional makes sure that anything we add to the database will be removed after
      2. Add repositories for Product and Purchase and annotate with @Autowire
      3. Create @BeforeEach annotated method for setting up a purchase and three products before the test
      4. Add method to test findPurchaseBetweenDates
4. Iteration
   1. N/A
   2. Only needed on a fail result. Otherwise on pass/fails we documented the process, what was expected, what happened, and if it passed.
5. A screenshot of a computer

   Description automatically generatedTest 1 – PurchaseTest.java Class

### Results

#### A screenshot of a computer Description automatically generatedTest 1 – PurchaseTest.java Class

#### Test 2 – PurchaseRepositoryTest.java ClassA screenshot of a computer program Description automatically generated

## DevOps

### Live URL

<https://d424-capstone.nicebay-118f9d78.eastus.azurecontainerapps.io/mainscreen>

### Repository URL

<https://gitlab.com/wgu-gitlab-environment/student-repos/amughr1/d424-software-engineering-capstone/-/tree/working-branch?ref_type=heads>

### Quick Start Guide

1. Visit the Live URL.
2. You’ll be greeted to the main page, here you can-
   1. View and Search for Inhouse / Outsourced Parts
      1. **Add Inhouse Part** will take you to a form where you can add an inhouse part
      2. **Add Outsourced Part** will take you to a form where you can add an outsourced part
      3. **Update** will take you to the respective inhouse/outsourced part form where you can update the part details.

A screenshot of a computer

Description automatically generatedA screenshot of a form

Description automatically generatedA screenshot of a form

Description automatically generated

* 1. View and search for Products
     1. **Add Product** will take you to a form where you can add a product
        1. On this form you can add and remove the earlier created parts to said product
     2. **Buy Now** will allow you to buy the product and log it in the database automatically for reporting purposes.
     3. A screenshot of a computer

        Description automatically generated**Update** will take you to the product form where you can update the product details.
     4. **Delete** will allow you to delete the specified product. Note, you will not be able to delete the product if there are parts attached already.

1. A close-up of a computer screen

   Description automatically generatedA screenshot of a computer

   Description automatically generatedClicking the About button in the navigation will take you to our Mission Statement. It’s there to remind about who The Cade is and what your company is about.
2. Clicking the Report button in the navigation will take you to the Report page.
   1. A screenshot of a sales report

      Description automatically generatedHere you can click from the predefined report timelines to generate a report of all the given sales between two dates. It starts with this weeks sales by default.
      1. A screenshot of a computer

         Description automatically generatedHitting the download button will download the current data returned onscreen in csv format.

A screenshot of a computer

Description automatically generated