**D424 – Software Engineering**

**Task 3**

|  |  |
| --- | --- |
| **Capstone Proposal Project Name:** | http://www.idevnews.com/views/images/uploads/general/wgu_logo.png  PurpleCat PC Store Inventory Management Application |
| **Student Name:** | Jin Luo |

**Table of Contents**

[Table of Contents 2](#_Toc150100547)

[Task 3 Design Document 4](#_Toc150100548)

[Application Design and Testing 4](#_Toc150100549)

[Class Design 4](#_Toc150100550)

[UI Design 6](#_Toc150100551)

[link to Web App Hosted 14](#_Toc150100552)

[link to GitLab Repository 14](#_Toc150100553)

[User Guide for Setting Up and Running the Application 14](#_Toc150100554)

[Introduction 15](#_Toc150100555)

[System Requirements 15](#_Toc150100556)

[Setting up Frontend (React) 15](#_Toc150100557)

[Setting up Backend (Java Spring Boot) 16](#_Toc150100558)

[User Guide for Running the Application from A User Perspective 17](#_Toc150100559)

[User Login Page 17](#_Toc150100560)

[Home Page 18](#_Toc150100561)

[Products Page 20](#_Toc150100562)

[View Products 20](#_Toc150100563)

[Add Products 21](#_Toc150100564)

[Update Products 22](#_Toc150100565)

[Delete Products 23](#_Toc150100566)

[Parts Page 24](#_Toc150100567)

[View Parts 24](#_Toc150100568)

[Add Parts 25](#_Toc150100569)

[Update Parts 26](#_Toc150100570)

[Delete Parts 27](#_Toc150100571)

[Generate Reports 30](#_Toc150100572)

[Generate Reports for Products 30](#_Toc150100573)

[Generate Reports for Parts 30](#_Toc150100574)

[Unit Test Plan 31](#_Toc150100575)

[Introduction 31](#_Toc150100576)

[Purpose 31](#_Toc150100577)

[Overview 31](#_Toc150100578)

[Test Plan 32](#_Toc150100579)

[Items 32](#_Toc150100580)

[Features 32](#_Toc150100581)

[Deliverables 32](#_Toc150100582)

[Tasks 33](#_Toc150100583)

[Needs 33](#_Toc150100584)

[Pass/Fail Criteria 33](#_Toc150100585)

[Specifications 34](#_Toc150100586)

[Procedures 35](#_Toc150100587)

[Results 38](#_Toc150100588)

**Task 3 Design Document**

# Application Design and Testing

## Class Design

This section outlines the core classes used in our inventory management application. These classes are essential for defining the structure and functionality of our application. We have four classes: Product, Part (which is an abstract class), and two subclasses of Part - InhousePart and OutsourcedPart. Each class serves a specific role in managing products and parts within the system.

* Product Class: The Product class represents the products available in our inventory. It includes attributes such as product ID, name, price, and inventory. This class is responsible for linking associated parts through a many-to-many relationship, allowing us to define the parts required for assembling products.
* Part Class: The Part class is an abstract class that serves as the parent for both InhousePart and OutsourcedPart subclasses. It contains common attributes for all parts, including part ID, name, price, inventory, maximum and minimum stock levels, and a many-to-many relationship with products.
* InhousePart Class: The InhousePart class is a subclass of Part. It has an additional attribute, part ID, to represent the internal part source. This class allows us to identify parts produced in-house and their specific properties.
* OutsourcedPart Class: The OutsourcedPart class is another subclass of Part. It includes a "companyName" attribute to specify external part supplier. This class is used to manage parts obtained from external vendors.

A screenshot of a graph

Description automatically generated

## UI Design

In this section, we present the user interface design for our Inventory Management Application. Below, you'll find images that showcase our user interface design, providing an overview of the features and pages we've incorporated into our application.

* User Login Page: Our User Login Page serves as the entry point to the application, ensuring secure access for authorized users.

A screenshot of a computer login screen

Description automatically generated

* Home Page with Main Menu: The home page serves as your central hub within the application. It offers a user-friendly interface with a main menu that provides quick navigation to other pages.

A screenshot of a computer store

Description automatically generated

* Products Management Page: This is where users can manage products, including searching, adding, updating, buying, and deleting products. Additionally, users can generate reports to gain insights into product-related data.

A screenshot of a computer

Description automatically generated

* Parts Management Page: The parts management page extends similar functionalities to products. Users can search for parts, add in-house or outsourced parts, update part information, delete parts, and generate reports.

A screenshot of a computer

Description automatically generated

* Outsourced Part Detail Page: This page offers a deep dive into the details of outsourced parts, provides in-depth information about outsourced parts.

A screenshot of a computer

Description automatically generated

* In-house Part Detail Page: Here, you'll find detailed information about in-house parts.

A screenshot of a computer

Description automatically generated

* Product Detail Page: The Product Detail Page offers a comprehensive overview of individual products. Users can add new products or update the details of existing ones. Users can associate available parts with the selected product. This page also provides a list of parts associated with the selected product.

A screenshot of a product page

Description automatically generated

A screenshot of a computer

Description automatically generated

* About Page: The About page provides information about PurpleCat PC Store.

A screenshot of a computer

Description automatically generated

# link to Web App Hosted

<https://purplecat-pc-store.netlify.app>

# link to GitLab Repository

# User Guide for Setting Up and Running the Application

**For Maintenance Purposes**

***Installation and Using the Application***

## Introduction

This user guide is designed to assist users in setting up and using the PurpleCat PC Store Inventory Management Application in their local environment. The application consists of a frontend built with React and a backend developed in Java Spring Boot, connected to a MySQL database. Users are not required to register; instead, they can use a guest login provided through the Supabase service. The guest login credentials are as follows:

* Email: ***cat1@cat.com***
* Password: ***1234***

## System Requirements

Before you start, ensure that your system meets the following requirements:

* Integrated Development Environment (IDE) such as IntelliJ IDEA and VS Code.
* Node.js and npm (Node Package Manager) for running the frontend.
* Java Development Kit (JDK) 11 or higher
* Apache Maven
* MySQL database server

## Setting up Frontend (React)

1. Download the source code.
2. Open a command prompt or terminal.
3. In the command prompt or terminal, navigate to the directory where you've stored the frontend code. (for example: “my-frontend”)
4. Run the following command to install dependencies: ***npm install***
5. Run the following command to start the React development server: ***npm start***

The frontend of PurpleCat PC Store Inventory Management Application should now be running at [http://localhost:3000](http://localhost:3000/) in your web browser.

## Setting up Backend (Java Spring Boot)

* Open your IntelliJ IDEA and open the project from the downloaded source code.
* Configure the application's database settings: Open the “***application.properties***” file located in the src/main/resources directory.
* Replace “spring.datasource.url” with the URL of your MySQL database.
* Replace “spring.datasource.username” with your MySQL database username.
* Replace “spring.datasource.password” with your MySQL database password.
* Run the application: In your IDE, navigate to the project root.
* Open a terminal window.
* Run the command: ***mvn clean install***
* Run the command: ***mvn clean spring-boot:run***

The backend of purpleCat PC Store Inventory Management Application should now be running at <http://localhost:8080> in your web browser.

* ***User Login***

Open your web browser.

1. In the address bar, enter [***http://localhost:3000***](http://localhost:3000) to access the PurpleCat PC Store Inventory Management Application.
2. Enter the following guest login credentials:
   * Email: ***cat1@cat.com***
   * Password: ***1234***
3. Click the "Login" button.
4. You will be redirected to the main page.

By following this user guide, you can set up, log in, and use the PurpleCat PC Store Inventory Management Application on your local environment.

# User Guide for Running the Application from A User Perspective

Using the Application

## User Login Page

Open your web browser.

1. In the address bar, enter [***http://localhost:3000***](http://localhost:3000) to access the PurpleCat PC Store Inventory Management Application.
2. Enter the following guest login credentials:
   * Email: ***cat1@cat.com***
   * Password: ***1234***
3. Click the "Login" button.
4. You will be redirected to the main page.

A screenshot of a computer

Description automatically generated

## Home Page

Once logged in, you will be on the Home page of the application.

A screenshot of a computer

Description automatically generated

## Products Page

### View Products

* + Click on the “Products” link in the navbar, it will take you to the Products Page where you will find a list of products.

A screenshot of a computer

Description automatically generated

### Add Products

* + To add a new product into the inventory, click the "Add Product" link in the navbar.
  + Fill in the product details, including the product’s name and price.
  + Click the "Add Product" button to add the new product into the inventory.

A screenshot of a computer

Description automatically generated

### Update Products

* + To edit a product's details, click the "Update" button on the Products page. It will take you to the Product Detail page.
  + Update the product information.
  + Click the "Update" button to save the changes.

A screenshot of a computer

Description automatically generated

### Delete Products

* + To delete a product from the inventory, click the "Delete" button on the Products page.
  + Confirm the deletion when prompted.

A screenshot of a computer

Description automatically generated

## Parts Page

### View Parts

* + Click on the “Parts” link in the navbar, it will take you to the Parts Page where you will find a list of parts.

A screenshot of a computer

Description automatically generated

### Add Parts

* + To add a new part into the inventory, click the "Add Inhouse Part" or “Add Outsourced Part” link in the navbar.
  + Fill in the part details, including the part’s name, price, inventory, Max inventory, Min inventory and part inhourse ID or company name.
  + Click the "Add" button to add the new part into the inventory.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### Update Parts

* + To edit a part’s details, click the "Update" button on the Parts page. It will take you to “Outsourced Part Detail” page or “Inhourse Part Detail” page.
  + Update the part information.
  + Click the "Update" button to save the changes.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### Delete Parts

* + To delete a part from the inventory, click the "Delete" button on the Parts page.
  + Confirm the deletion when prompted.

A screenshot of a computer

Description automatically generated

Users cannot delete a part if it is associated with products.

A screenshot of a computer

Description automatically generated

## Generate Reports

### Generate Reports for Products

A screenshot of a computer

Description automatically generatedOn the Products page, you can enter a keyword in the search bar, click 'Search' to display the results on the page, and then click 'PDF Report' to navigate to the PDF report page, where you can download or print the report.

### Generate Reports for Parts

A screenshot of a computer

Description automatically generatedOn the Parts page, you can enter a keyword in the search bar, click 'Search' to display the results on the page, and then click 'PDF Report' to navigate to the PDF report page, where you can download or print the report.

By following this user guide, you can effortlessly navigate and manage your inventory within the PurpleCat PC Store Inventory Management Application. Whether you're viewing, adding, editing, deleting, or generating reports for products or parts, this guide should help you interact with the application's user-friendly interface.

# Unit Test Plan

## Introduction

### Purpose

The purpose of the Unit Test Plan is to verify the functionality of the PurpleCat PC Store Inventory Management Application. By conducting a series of unit tests, we aim to ensure that each part of the application operates as intended and meets high-quality standards. This plan describes the tests, their associated features, the deliverables, the tasks involved, technical requirements, pass/fail criteria, and remediation procedures if any issues arise during testing.

### Overview

The Unit Test Plan is an important part of the software development process for the PurpleCat PC Store inventory management application. It focuses on verifying the functionality of essential components and features of the application, including product management, part management, and report generation. Through thorough testing, we ensure that the application performs correctly in isolation and as a whole.

## Test Plan

### Items

To successfully conduct the tests, the following items are required:

* Development Environment: A development environment with appropriate software tools, including Java, Spring Boot, MySQL database system, and an integrated development environment (IDE).
* Testing Framework: JUnit, a widely used testing framework for Java applications.
* Test Data: Sample data for testing, including products list, parts list, and other information.
* Database Setup: A configured database with predefined data to replicate real-world scenarios.

### Features

The unit tests will cover the following features:

1. Product Management: Verifying the ability to add, edit, and delete products. Ensuring that data validation and management are functioning as expected.
2. Part Management: Testing the functionality of adding, editing, and deleting parts. Ensuring accurate part management and data validation.
3. Report Generation: Verifying that users can generate accurate reports based on search criteria, including product and part details.

### Deliverables

The Unit Test Plan will include the following deliverables:

* Test Scripts: A collection of test scripts written in JUnit, including test classes and methods.
* Test Results: A detailed report summarizing test outcomes, including successful tests and any failed tests with relevant error messages.

### Tasks

The testing process consists of the following tasks:

1. Environment Setup: Ensure that the development environment is correctly configured with all necessary tools and dependencies.
2. Test Script Preparation: Confirm the availability of test scripts covering product management, part management, and report generation.
3. Database Initialization: Initialize the database with predefined test data, including products list and parts list.
4. Test Execution: Execute the test scripts using JUnit, monitor test results, and capture any errors.
5. Results Analysis: Analyze test results, document issues, and evaluate test outcomes.

### Needs

* Software Requirements: Ensure that all necessary software and dependencies, including Java, Spring Boot, JUnit, and any related libraries, are correctly installed.
* Database Setup: Ensure that the database is properly configured with the required schema, tables, and initial test data.
* Access to Source Code: Access to the application's source code and test scripts to review and update them as needed during testing.
* Test Data: Prepare test data to populate the database, including products list and parts list.

### Pass/Fail Criteria

The criteria for determining the success or failure of each test are as follows:

* Product Management:
  + Pass: Products can be added, edited, and deleted without errors. The data is correctly validated, and the products list is accurate.
  + Fail: Product management may encounter issues such as validation errors, data not saving, or incorrect product details.
* Part Management:
  + Pass: Parts can be added, edited, and deleted without errors. The data is correctly validated, and the parts list is accurate.
  + Fail: Part management may encounter issues such as validation errors, data not saving, or incorrect part details.
* Report Generation:
  + Pass: Users can generate accurate reports based on search criteria, and the reports are displayed correctly.
  + Fail: Report generation may result in problems such as inaccurate reports, search criteria not functioning, or errors in the report generation process.

If a test fails, we will:

* + Debug and identify the cause of the failure.
  + Modify the source code to fix the issue.
  + Execute the test and analyze test results to verify that the problem is resolved.

## Specifications

Provide sample code that represents what testing code was used. Screenshots are acceptable.

The Unit Test Plan will include sample code snippets that demonstrate the testing procedures and assertions. These code snippets will be part of the test documentation.

javaCopy code

@TestpublicvoidtestProductCreation(){ Productproduct=newProduct("Sample Product", 99.99, 10); assertNotNull(product); assertEquals("Sample Product", product.getName()); assertEquals(99.99, product.getPrice(), 0.001); assertEquals(10, product.getInventory()); }



## Procedures

Provide a detailed list of the steps you used to complete the testing process. Be sure to mention if iterations were/are part of the process used and when pass/fail results were provided.

* Product Management Testing Process:
* Test Add Product (Iteration):
* The functionality of adding a product was tested by adding a new product.
* An initial test was conducted to verify successful product addition.
* The test result was provided immediately after the test, indicating a "Pass."
* An iteration occurred when testing the addition of a product with incomplete information.
* After multiple iterations, the test results were provided, indicating "Fail" for incomplete product information.
* Test Edit Product (Iteration):
* The functionality of editing a product was tested by modifying an existing product.
* An initial test was conducted to verify successful editing of the product.
* The test result was provided immediately after the test, indicating a "Pass."
* An iteration occurred when testing the editing of a product with invalid data.
* After multiple iterations, the test results were provided, indicating "Fail" for invalid product data.
* Test Delete Product (Not Iteration):
* The functionality of deleting a product was tested by removing a product.
* An initial test was conducted to verify successful product deletion.
* The test result was provided immediately after the test, indicating a "Pass."
* Part Management Testing Process:
* Test Add Part (Iteration):
* The functionality of adding a part was tested by adding a new part.
* An initial test was conducted to verify successful part addition.
* The test result was provided immediately after the test, indicating a "Pass."
* An iteration occurred when testing the addition of a part with incomplete information.
* After multiple iterations, the test results were provided, indicating "Fail" for incomplete part information.
* Test Edit Part (Iteration):
* The functionality of editing a part was tested by modifying an existing part.
* An initial test was conducted to verify successful part editing.
* The test result was provided immediately after the test, indicating a "Pass."
* An iteration occurred when testing the editing of a part with invalid data.
* After multiple iterations, the test results were provided, indicating "Fail" for invalid part data.
* Test Delete Part (Iteration):
* The functionality of deleting a part was tested by removing a part.
* An initial test was conducted to verify successful part deletion.
* The test result was provided immediately after the test, indicating a "Pass."
* An iteration occurred when testing the deletion of a part that was associated with products.
* After multiple iterations, the test results were provided, indicating "Fail" for deleting a part that was associated with products.
* Report Generation Testing Process:
* Test Generate Product Report (Not Iteration):
* The functionality of generating product reports was tested by generating a product report.
* An initial test was conducted to verify successful report generation.
* The test result was provided immediately after the test, indicating a "Pass."
* Test Generate Part Report (Not Iteration):
* The functionality of generating part reports was tested by generating a part report.
* An initial test was conducted to verify successful report generation.
* The test result was provided immediately after the test, indicating a "Pass."

In summary, iterations were an important part of the testing process, and test results were provided after each iteration, indicating whether the tests passed or failed. These iterations were essential for identifying and addressing issues with the application's functionality.

## Results

Here you will describe and provide examples of the testing results. If you were using a testing package include a screenshot of the interface. Screenshot work best.

The test results for the PurpleCat PC Store Inventory Management Application will be obtained after executing the test scripts using JUnit. The results will include detailed descriptions and screenshots to illustrate the testing outcomes, covering both successful tests and tests that encountered issues.

* Product Management Test Results:
* *Add Product Test*:testAddProduct()
  + Purpose: This test ensures that products can be added successfully.
  + Result: The test passed when a new product was successfully added, and the server returned a status code of 200.
  + Pass/Fail Criteria: Pass
  + Remediation: No remediation required.
* *Edit Product Test: testEditProduct():*
  + Purpose: This test verifies that product information can be updated successfully.
  + Result: The test passed when an existing product was successfully updated, and the server returned a status code of 200.
  + Pass/Fail Criteria: Pass
  + Remediation: No remediation required.
* *Delete Product Test: testDeleteProduct():*
  + Purpose: This test ensures that products can be deleted successfully.
  + Result: The test passed when a product was successfully deleted, and the server returned a status code of 200.
  + Pass/Fail Criteria: Pass
  + Remediation: No remediation required.
* Part Management Test Results:
* *Add Part Test: testAddPart():*
  + Purpose: This test ensures that parts can be added successfully.
  + Result: The test passed when a new part was successfully added, and the server returned a status code of 200.
  + Pass/Fail Criteria: Pass
  + Remediation: No remediation required.
* *Edit Part Test: testEditPart():*
  + Purpose: This test verifies that part information can be updated successfully.
  + Result: The test passed when an existing part was successfully updated, and the server returned a status code of 200.
  + Pass/Fail Criteria: Pass
  + Remediation: No remediation required.
* *Delete Part Test: testDeletePart():*
  + Purpose: This test ensures that parts can be deleted successfully.
  + Result: The test passed when a part was successfully deleted, and the server returned a status code of 200.
  + Pass/Fail Criteria: Pass
  + Remediation: No remediation required.
* Report Generation Test Results:
* *Generate Product Report Test: testGenerateProductReport():*
  + Purpose: This test verifies the generation of accurate product reports based on search keyword criteria.
  + Result: The test passed when a product report was successfully generated, and the server returned a report with the expected name and description.
  + Pass/Fail Criteria: Pass
  + Remediation: No remediation required.
* *Generate Part Report Test: testGeneratePartReport():*
  + Purpose: This test ensures the generation of accurate part reports based on search keyword criteria.
  + Result: The test passed when a part report was successfully generated, and the server returned a report with the expected name and description.
  + Pass/Fail Criteria: Pass
  + Remediation: No remediation required.

In summary, the unit tests for product management, part management, and report generation all passed successfully. Since all tests have passed, no remediation is required at this stage. These tests confirm that the application's core features are functioning as expected.



*C2. Provide a link to where the web app is hosted with HTML code (if applicable).*

*C3. Provide a link to the GitLab repository of the code indicating the version included in this submission.*