**C868 – Software Capstone Project Summary**

**Task 2 – Section C**

|  |  |
| --- | --- |
| **Capstone Proposal Project Name:** | http://www.idevnews.com/views/images/uploads/general/wgu_logo.png  Engine-uity Rebuilds Inventory Manager |
| **Student Name:** | Philip Parks |

Table of Contents

[Table of Contents 2](#_Toc182908739)

[Task 2 Part C – C868 Software Development Capstone 3](#_Toc182908740)

[Application Design and Testing 3](#_Toc182908741)

[Design Document 3](#_Toc182908742)

[Class Design 3](#_Toc182908743)

[UI Design 4](#_Toc182908744)

[Unit Test Plan 8](#_Toc182908745)

[Introduction 8](#_Toc182908746)

[Purpose 8](#_Toc182908747)

[Overview 8](#_Toc182908748)

[Test Plan 9](#_Toc182908749)

[Items 9](#_Toc182908750)

[Features 9](#_Toc182908751)

[Deliverables 10](#_Toc182908752)

[Tasks 10](#_Toc182908753)

[Needs 10](#_Toc182908754)

[Pass/Fail Criteria 10](#_Toc182908755)

[Specifications 10](#_Toc182908756)

[Procedures 11](#_Toc182908757)

[Results 12](#_Toc182908758)

[C4. Source Code 12](#_Toc182908759)

[User Guide 13](#_Toc182908760)

[Introduction 13](#_Toc182908761)

[Setting up and maintaining the Application 13](#_Toc182908762)

[Login 13](#_Toc182908763)

[Engines Tab 14](#_Toc182908764)

[*Search for an engine* 15](#_Toc182908765)

[*Add an engine* 15](#_Toc182908766)

[*Modify an engine* 16](#_Toc182908767)

[*Delete an engine* 17](#_Toc182908768)

[Parts Tab 18](#_Toc182908769)

[*Search for a part* 18](#_Toc182908770)

[*Delete a part* 19](#_Toc182908771)

[*Add a part* 19](#_Toc182908772)

[*Modify a part* 20](#_Toc182908773)

[Admin Tab 20](#_Toc182908774)

Task 2 Part C – C868 Software Development Capstone

# Application Design and Testing

## Design Document

### Class Design

The figures below depict the database and class design for the Engine-uity Rebuilds Inventory Manager application. A database ERD and class diagram are provided to show a common relationship between the database and models in the program that changes slightly. The User entity is used in both, but there is no relationship between the User class and the other classes. The User entity is used in the database to indicate which user is creating and modifying table information in the database. The User class in the application is used to authenticate application users. Once a user has been authenticated, an authUser object is created that holds the authenticated user information for insertion into the database when they add or modify information through the application.

A diagram of a computer program

Description automatically generated with medium confidence

The database entity relationship diagram


### UI Design

##### Low-Fidelity Primary View

The primary view will have tabs that lead to the different areas of the application. It will also have a table of engines, search functionality, an area where error or system messages can be displayed, and a button that will log the user out to the login screen.

A screenshot of a computer

Description automatically generated

##### High-Fidelity Primary View – Engines Tab

The primary view contains three tabs: Engines, Parts, and Admin. The default tab when logging in is the Engines tab. All three tabs also have access to the Log-Out button at the bottom right of the primary view.

The Engines tab has a search area where engines can be searched for by database ID (Number column) or Nomenclature. There are also buttons for adding an engine, modifying an engine, and deleting an engine.

A screenshot of a computer

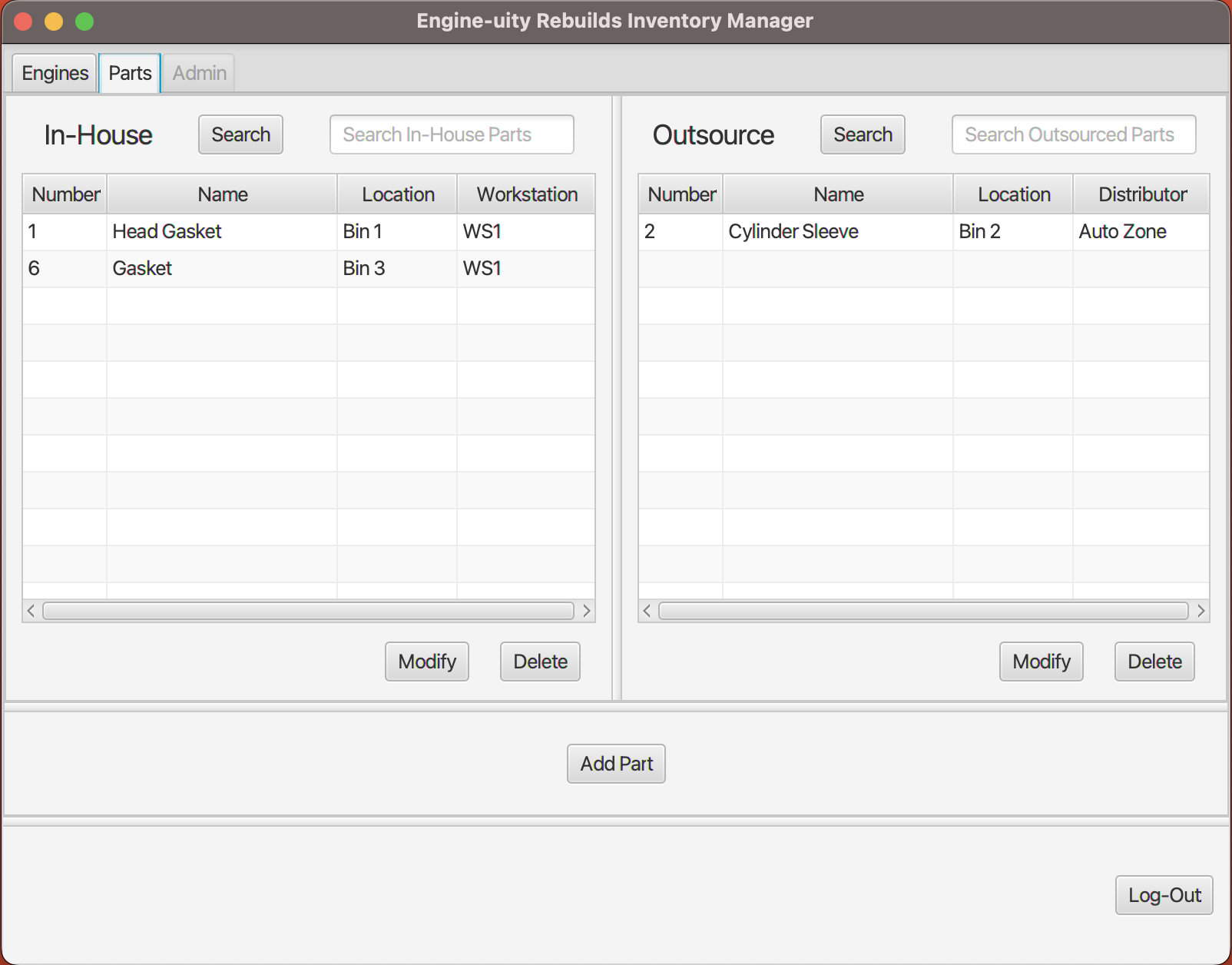
Description automatically generated

##### High-Fidelity Primary View – Parts Tab

The parts tab contains two searchable tables. The In-House table contains all parts made at a workstation located at the Engine-uity Rebuilds physical location, and the Outsource table contains all parts ordered from a distributor.

The tables also have modify and delete buttons associated with them that will perform their respective titles to a part that has been selected.

An add part button on this tab will lead users to a screen that will allow them to add a new part.



##### High-Fidelity Primary View – Admin Tab

The admin tab is only selectable if the logged-in user has administrative privileges. This view contains two tables that can add the respective sources of in-house (workstation) and outsourced parts (distributor).

To run a report, select a respective source from the choice box and click the adjacent button to populate the table with all the parts associated with the selected source.

A third table will report all engines related to a selected status when the status is selected from the choice box and the adjacent button is clicked.

Along with the three report tables, there is an area in the admin tab that allows for the addition of new users. The interface for adding a user is fairly common, with the addition of an admin check box that allows the user to assign the new user as an admin.

A screenshot of a computer

Description automatically generated

# Unit Test Plan

## Introduction

### Purpose

Several unit tests were written using JUnit, a Java testing library, to test the Inhouse class. Since the Inhouse class inherits most of its fields from the Part class, I wanted to ensure that an Inhouse part object can be properly constructed. This unit test verifies that an Inhouse part properly inherits the respective Part fields and methods necessary for creating an Inhouse object.

### Overview

Since, at its core, the application is an inventory manager, the logic around creating and saving parts needs to be carefully considered. Setters (mutators) are methods used to set or change a private class field. Getters (accessors) are methods used to retrieve a private class field. Setters and getters are used to maintain the privacy of class fields while still being able to access or modify the information in the fields. This test attempts to ensure that the setters and getters responsible for accessing and modifying the fields of the Inhouse class do indeed set and get the fields associated with an in-house part, especially the fields inherited from the Part class.

The unit test is written using the JUnit testing library for Java and executed in the IntelliJ IDE (Integrated Development Environment).

## Test Plan

### Items

A class with a name that indicates that a test is being performed in the class, along with the name of the class being tested (example: ClassTest), will need to be created to hold the test methods. Each test being performed will need an Inhouse object to be created, the set method associated with the field being tested that changes the value associated with the set method, and the get method associated with the field being tested that retrieves the value associated with the get method.

### Features

Each test will require an **@Test** annotation before the test function being performed on the Inhouse class and the assertEquals() method.

The following will be needed for the test cases:

* A test class that has a name that indicates the class being tested.
* The @Test annotation before each test method in the test class.
* A void return type.
* A method name that clearly describes the test being performed.
* Creating an Inhouse object that the methods being tested can be performed on.
* A call to the setter method of the field being tested with the raw parameter entered into the method that matches the field type but is not the same as the value entered in the creation of the Inhouse object.
* The assertEquals() method with the raw parameter entered into the setter method as the expected parameter and a call to the Inhouse objects getter method as the actual parameter.

### Deliverables

Passing test cases will be the only deliverables of the tests being performed.

### Tasks

To execute this unit test, the following steps are required:

1. Write the code to be tested, i.e. set up the test class.
2. Create a JUnit file in the application's root directory to house the test code.
3. Write the test code using JUnit methods.
4. Run the test.
5. Examine the output to determine pass or fail.

### Needs

The JUnit Jupiter engine 5.10.0 dependencies need to be added to the applications’ pom.xml files.

### Pass/Fail Criteria

When the test is run, all the tests should return as passing with a green check mark.

## Specifications

This is a screenshot of the test code described above. It is an excerpt from the test class. The test file is located at src/test/java/parks/inventorymanager/model/InhouseTest.java in the application's root directory.

A screenshot of a computer program

Description automatically generated

## Procedures

As seen in the code excerpt above, four separate tests verify the setters and getters of the Inhouse class.

The first test, inhousePartIdMutatorAccessorTest(), creates a new Inhouse object with an integer of ‘1’ for the id parameter. The part.setId() method is then called with an integer of ‘2’ as the id parameter to change the id of the in-house part. The assertEquals() method is then called with an expected parameter of ‘2’ and a call to the part.getId() method as the actual parameter to ensure the change occurred.

The second test, inhousePartNameMutatorAccessorTest(), creates a new Inhouse object with a String of “part” for the name parameter. The part.setName() method is then called with a String of “bolt” as the name parameter to change the name of the in-house part. The assertEquals() method is then called with an expected parameter of “bolt” and a call to the part.getName() method as the actual parameter to ensure the change occurred.

The third test, inhousePartLocationMutatorAccessorTest(), creates a new Inhouse object with a String of “location” for the location parameter. The part.setLocation() method is then called with a String of “bin 1” as the location parameter to change the physical location of the in-house part. The assertEquals() method is then called with an expected parameter of “bin 1” and a call to the part.getLocation() method as the actual parameter to ensure the change occurred.

The fourth test, inhousePartWorkstationMutatorAccessorTest(), creates a new Inhouse object with a String of “workstation” for the name parameter. The part.setWorkstation() method is then called with a String of “workstation 1” as the name parameter to change the workstation where the in-house part was made. The assertEquals() method is then called with an expected parameter of “workstation 1” and a call to the part.getWorkstation() method as the actual parameter to ensure the change occurred.

## Results

The following console output is printed when running these four tests. The green check marks indicate that the tests ran successfully, and the line at the top of the console output that reads “Tests passed: 4 of 4 tests – 34 ms” means that 4 out of 4 tests ran successfully.

A screenshot of a computer

Description automatically generated

# C4. Source Code

The source code can be found in the file inventorymanager.zip.

# User Guide

## Introduction

This guide will include setup and maintenance instructions for IT personnel and a general user guide that will outline how to use the different functions of the application once the setup has been executed.

## Setting up and maintaining the Application

There is a file in the inventorymanager.zip file that is titled inventory.sql. The following will guide you in setting up the schema in the MySQL database provided in the virtual environment.

* Extract the zip file.
* Open the MySQL workbench and select the JavaConnection.
* Create a new query.
* Click on the ‘open a script file in this editor’ button shaped like a file in the upper left corner of the query tab.
* Navigate to where the zip file was extracted and select the inventory.sql file.
* Click on the lightning bolt to execute the statements in the inventory.sql file.

The schema that the application will interact with should now be in the database. If it is not visible in the schema area, click the refresh button on the top left corner of the schema area, and the inventory schema should appear.

The schema includes two initial user accounts:

1. Admin – The admin account username is ‘admin’, and the password is ‘admin’. (These should be changed to reflect the primary admin username and password before being used in an active work environment)
2. Test – The test account username is ‘test’, and the password is ‘test’. This account is a non-admin account to test the non-admin functions of the application.

The schema also includes initial data for application setup and testing.

There is a login-activity.txt file that records all login activity. The document records the following information:

* Whether the login was successful or not.
* The date, time, and time zone of the login attempt.
* The username associated with the login attempt.
* If the login failed due to a bad password, the record will include the password entered.

## Login

A screenshot of a login

Description automatically generated

1. *Enter the username “admin” in the username text field.*
2. *Enter the password “admin” in the password text field.*
3. *Click the submit button in the lower right-hand corner of the screen.*

Once logged in, the primary view will load with the engine tab selected.

To log out of the application, click the ‘Log-Out’ button at the bottom right of the primary view to be returned to the log-in view.

## Engines Tab

Four functions can be performed from the engines tab:

1. Search for a specific engine by nomenclature or number.
2. Add an engine to the table.
3. Modify an engine in the table.
4. Delete an engine from the table.

A screenshot of a computer

Description automatically generated

### *Search for an engine*

To search for an engine in the table, enter the nomenclature (name) or number of an engine into the text field and click the ‘Search’ button.

### *Add an engine*

To add an engine to the table, click on the ‘New Engine’ button, and the following view will appear:

A screenshot of a computer

Description automatically generated

All engines entered into the table must have at least three fields filled out before it can be saved:

1. The Engine Name field. (The engine name field is the engine’s Nomenclature. The engine nomenclature usually consists of the engine manufacturer and series. Ex. Cat 6, for an engine of the 6th series manufactured by Caterpillar)
2. The Serial Number field.
3. The Status field.

The other fields are optional but highly recommended. Once these three fields have been filled out, click on the ‘Save’ button to save the new engine to the table.

Click on the ‘Cancel’ button to cancel the creation of a new engine and return to the engine tab of the primary view.

### *Modify an engine*

To modify an engine in the table of the engine tab, select an engine from the table by clicking on it, then click the ‘Modify’ button. The following view will appear:

A screenshot of a computer

Description automatically generated

The information for the engine will populate the text fields on the left. All of the fields can be modified. New engines, by default, will have no parts associated with them.

To add parts to the engine, click on the desired part in the ‘All Parts’ table and then click the ‘Add Part to Engine’ button. The ‘Engine Parts’ table will update with the added part.

To remove parts from the engine, click on the desired part in the ‘Engine Parts’ table and then click on the ‘Remove Part from Engine’ button. The ‘Engine Parts’ table will update with the changes.

The ‘Save’ button will save changes made to the text fields. Changes to parts associated with an engine are saved as soon as the ‘Add Part to Engine’ or ‘Remove Part from Engine’ buttons are clicked.

Click on the ‘Cancel’ button to cancel any changes made to the text fields and return to the engine tab of the primary view.

### *Delete an engine*

To delete an engine from the engine table, select the engine you wish to delete and click on the ‘Delete’ button. Keep in mind that this action cannot be undone.

## Parts Tab

There are two tables on the parts tab:

1. In-house – a table displaying all parts made at a workstation.
2. Outsource – a table displaying all parts ordered from a distributor.

Each table has the following functionality:

1. Search for a specific part by name or number.
2. Modify a part in the table.
3. Delete a part from the table.

A part of either source can be added by clicking the ‘Add Part’ button.

A screenshot of a computer

Description automatically generated

### *Search for a part*

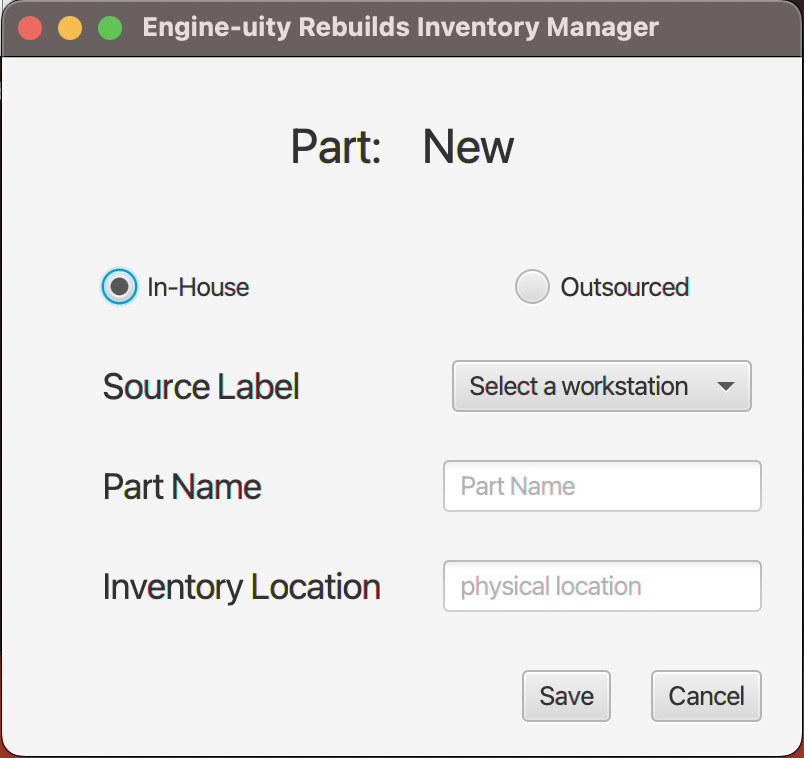
To search for a part in either table, enter the name or number of a part into the respective text field and click the ‘Search’ button.

### *Delete a part*

To delete a part from either table, select the part you wish to delete and click the ‘Delete’ button below the respective part table. Keep in mind that this action cannot be undone.

### *Add a part*

To add a new part from either source, click the ‘Add Part’ button at the bottom center of the parts tab. The following view will appear:



The figure above has the ‘In-House’ radio button selected. The ‘Source Label’ and selection options to the right of it will change depending on the selected radio button. If ‘In-House’ is selected, there will be a list of workstations to select from. If ‘Outsourced’ is selected, there will be a list of distributors to select from.

Every part must have at least a source selected and a name entered into the ‘Part Name’ text field to save a new part. After that, click on the ‘Save’ button to save a new part to the respective table of the part tab.

Click on the ‘Cancel’ button to cancel any changes made to the text fields and return to the engine tab of the primary view.

### *Modify a part*

To modify a part, select a part from either table of the parts tab by clicking on the part then click the ‘Modify’ button under the respective table. The view that loads is identical to the view to add a part with the following differences:

* The label at the top of the view will display the part number next to the ‘Part:’ label instead of the word ‘New’.
* The radio button that corresponds to the part being modified will be selected.
* The source of the part will be selected from the source selection options.
* The name of the part will be in the ‘Part Name’ text field.
* The location of the part will be in the ‘Inventory Location’ text field (if one has been entered).

All of these options can be modified. To save the changes made, click on the ‘Save’ button.

Click on the ‘Cancel’ button to cancel any changes made to the part and return to the engine tab of the primary view.

## Admin Tab

The admin tab is only selectable by a logged-in user with administrative privileges. The admin tab has six functions:

* Add a workstation.
* Display a list of in-house parts made at a selected workstation.
* Add a distributor.
* Display a list of outsourced parts obtained from a selected distributor.
* Display a list of engines by a selected status.
* Add a new user.

Workstations, distributors, and users can only be added through the application. The aforementioned can only be modified or deleted from the MySQL database by the database admin.

Select the desired report criteria from the drop-down menu and click the adjacent button to run any of the reports. The respective table will populate with a list matching the selection.

A screenshot of a computer

Description automatically generated