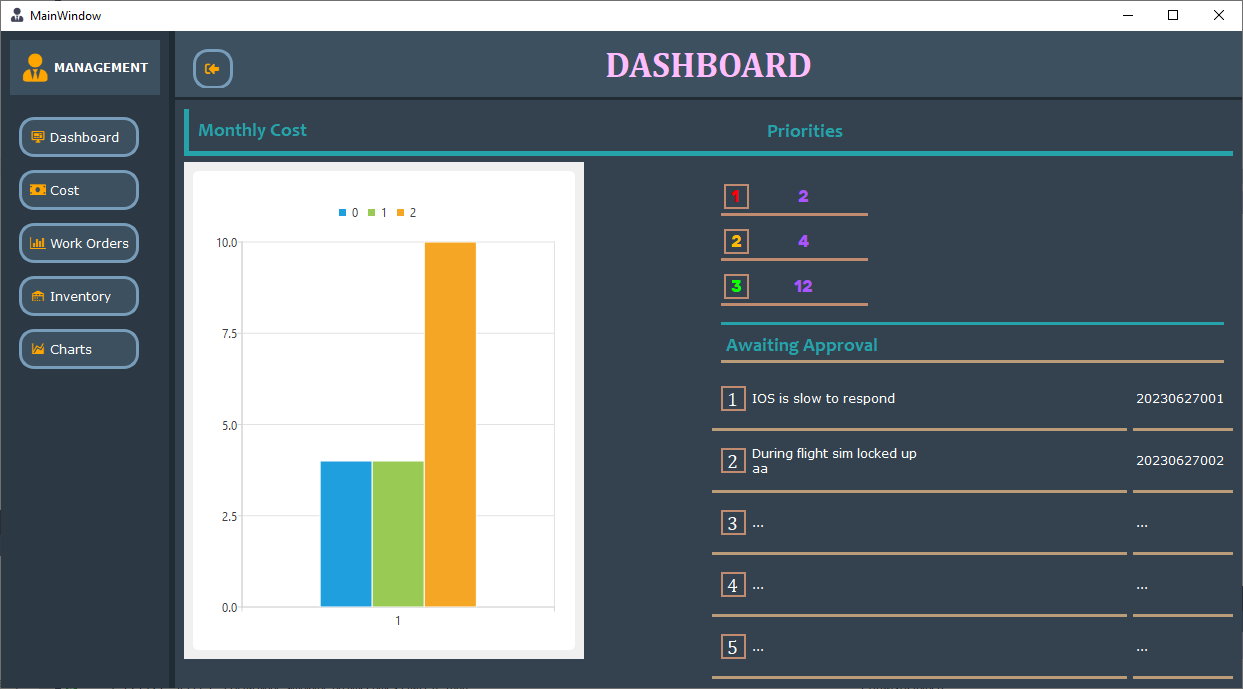


# Executive Summary

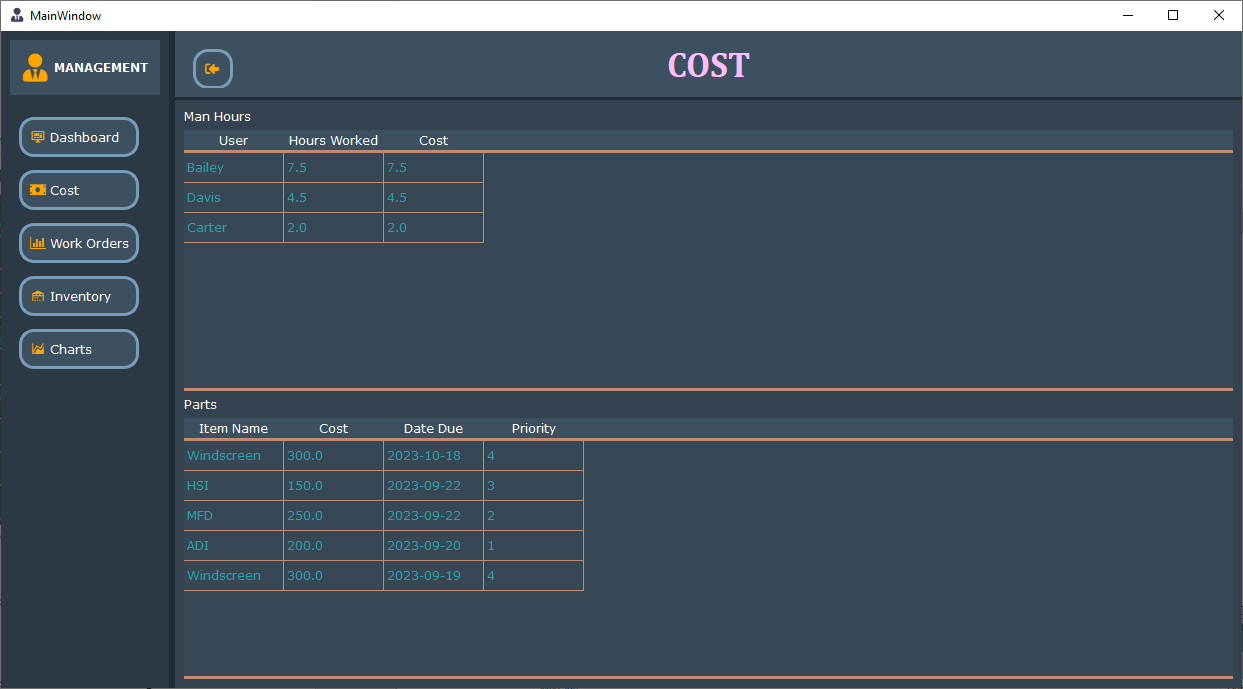
This project will allow users to add, alter, and remove inventory, add, alter, and remove work orders, and maintain repair times and cost per part repaired. View and maintain management overview dashboard.

Currently, we operate out of an Excel sheet that has been damaged multiple times while adding maintenance and manager notes. Currently, inventory is impossible to create, and management cannot monitor repairs and costs to the company.

# Scenario | Management – Dashboard

Dashboard allows the Manager to see stats from the last seven days of write-ups.

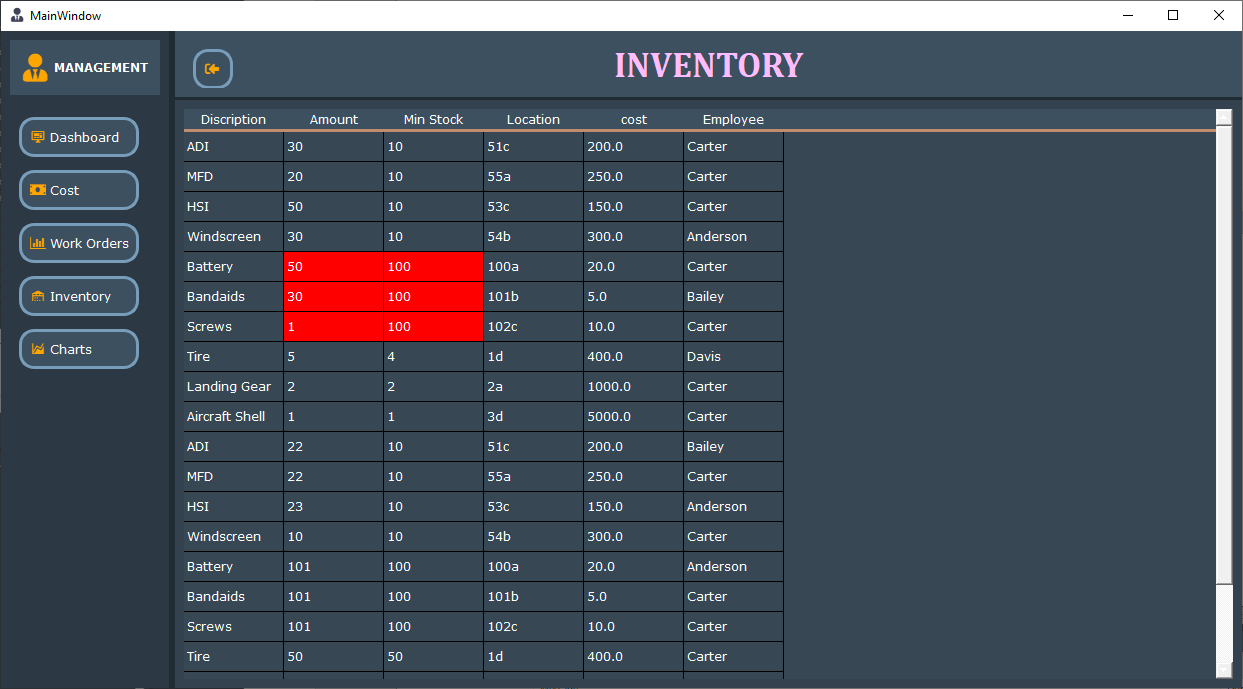
# Scenario | Management - Cost

Tool for Managers to quickly see their cost per week.

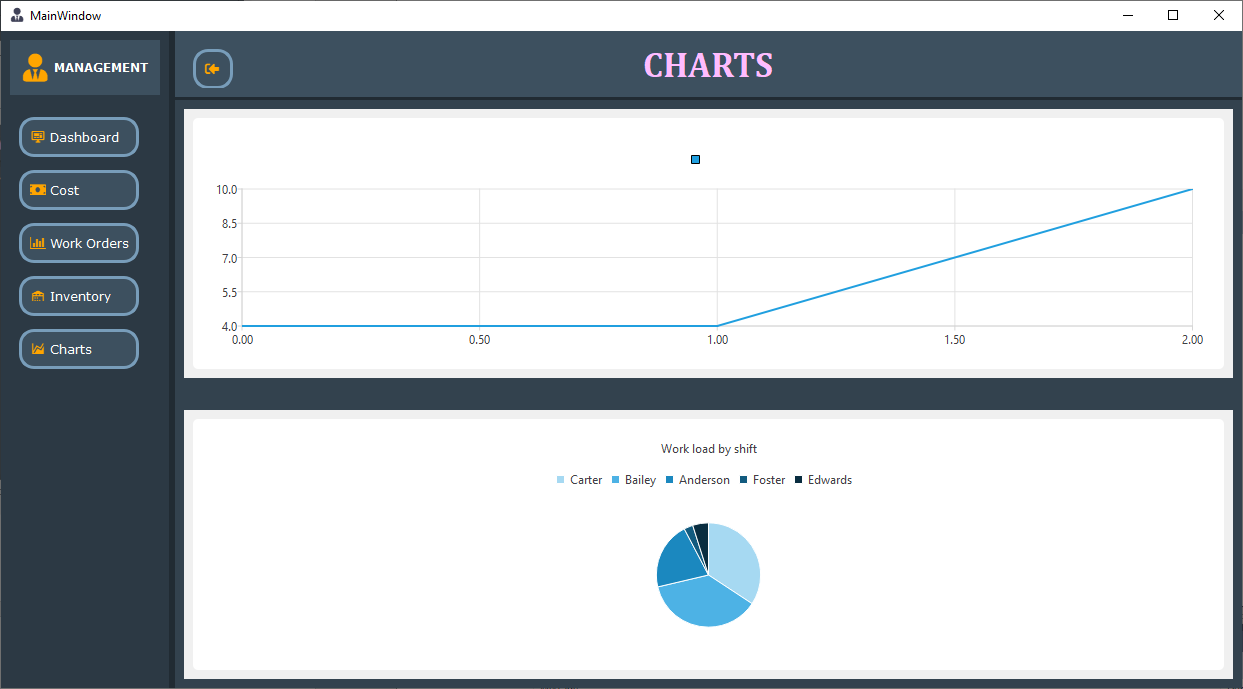
# Scenario | Management – Work Orders

 Tool for managers to quickly see their current work orders and their status of them with warnings.

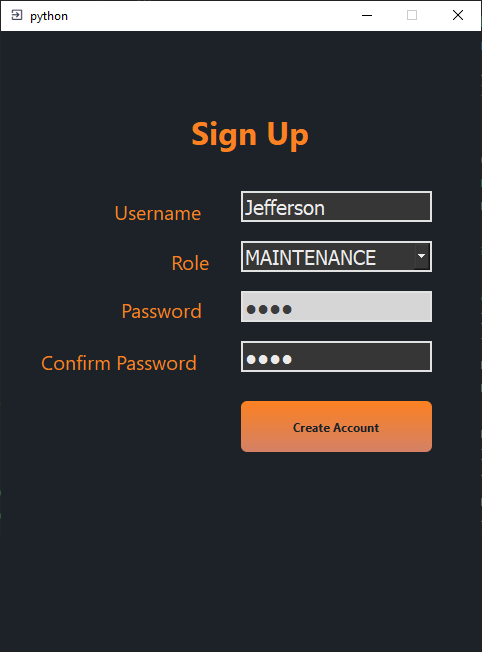
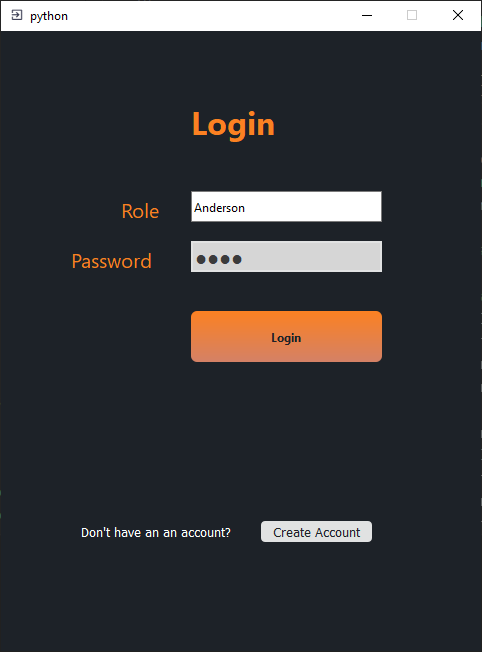
# Scenario | Management - Inventory

Tool for managers to quickly see their current inventory and stock numbers with warnings.

# Scenario | Managers - Charts

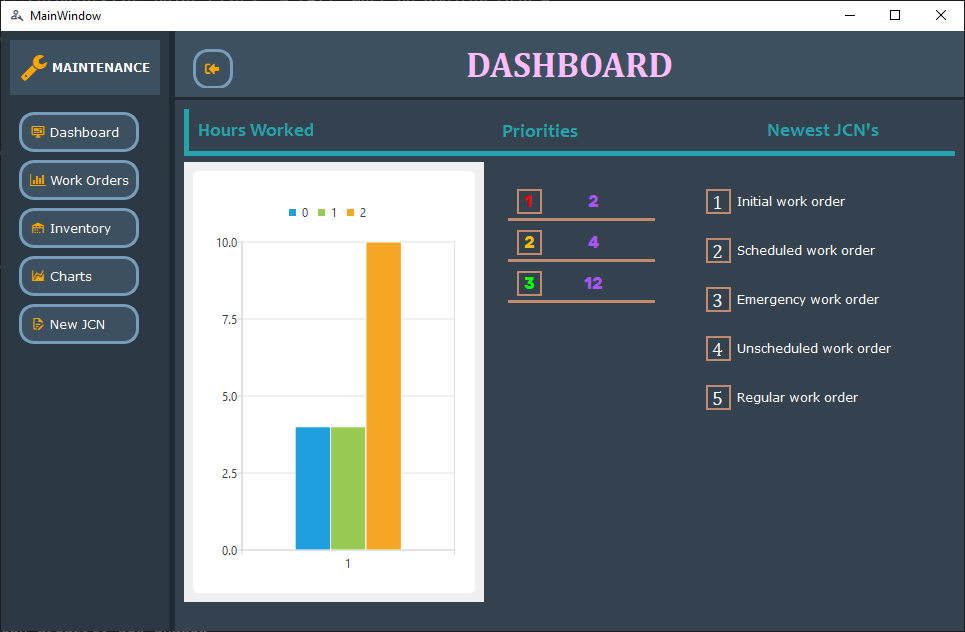
Activity charts to quickly show trends with money and work time.

# Scenario | Login



Allows the user to log in or the option to create a new account.

# Scenario | Maintenance - Dashboard

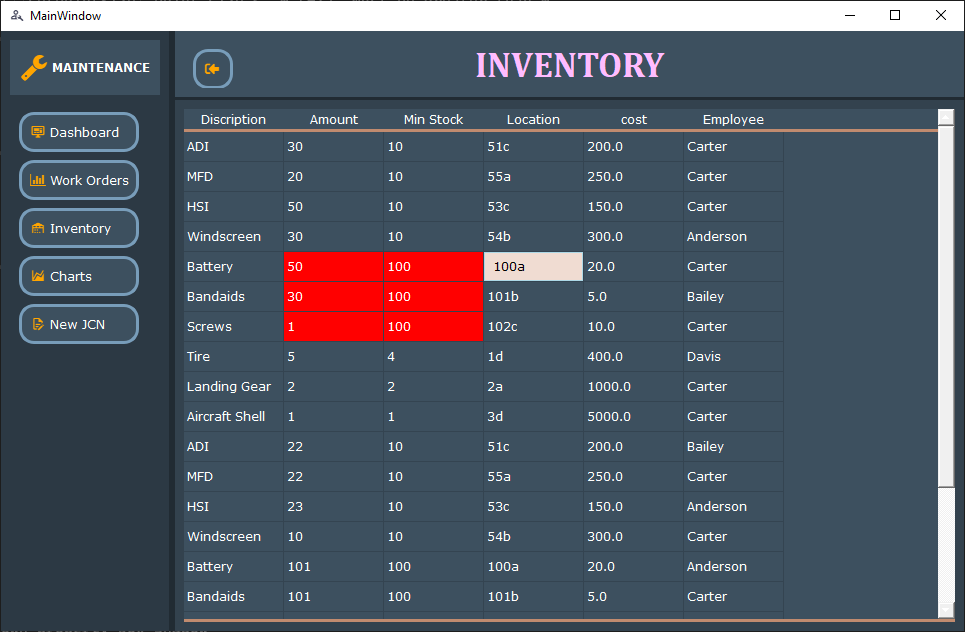
Allows the user to see the maintenance dashboard to have an overview of daily activities and priorities.

# Scenario | Maintenance -Work Orders

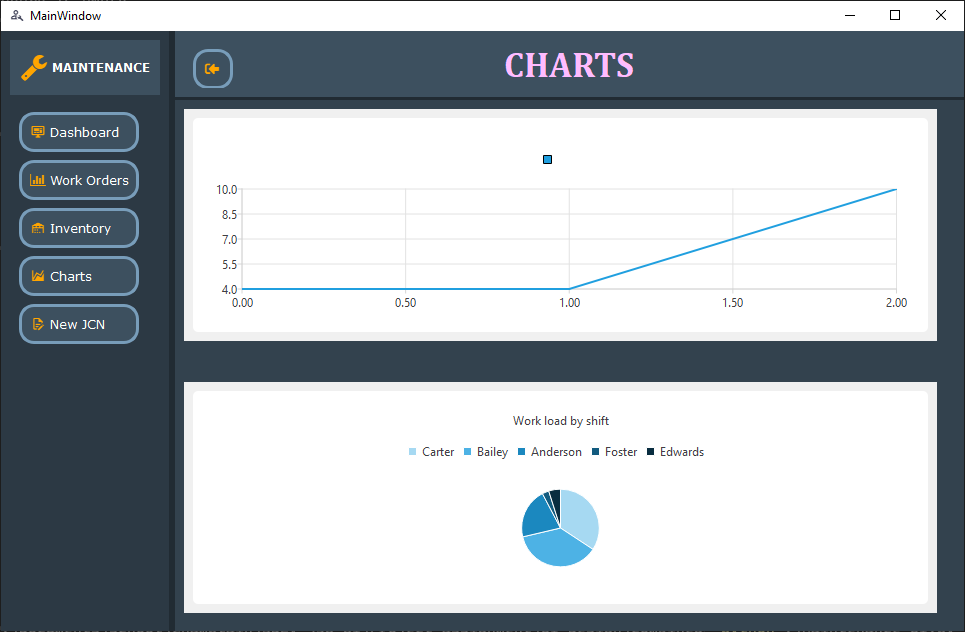


Allows maintenance to view and change work orders quickly.

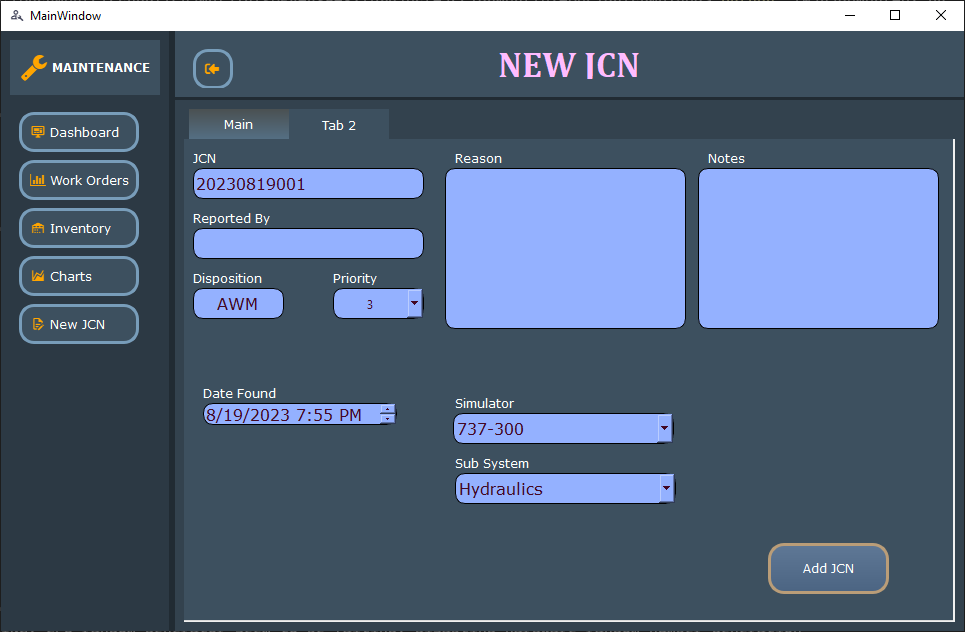
# Scenario | Maintenance - Inventory

Lets maintenance staff see the current inventory numbers to asses if they can replace an item or if they need to order it.

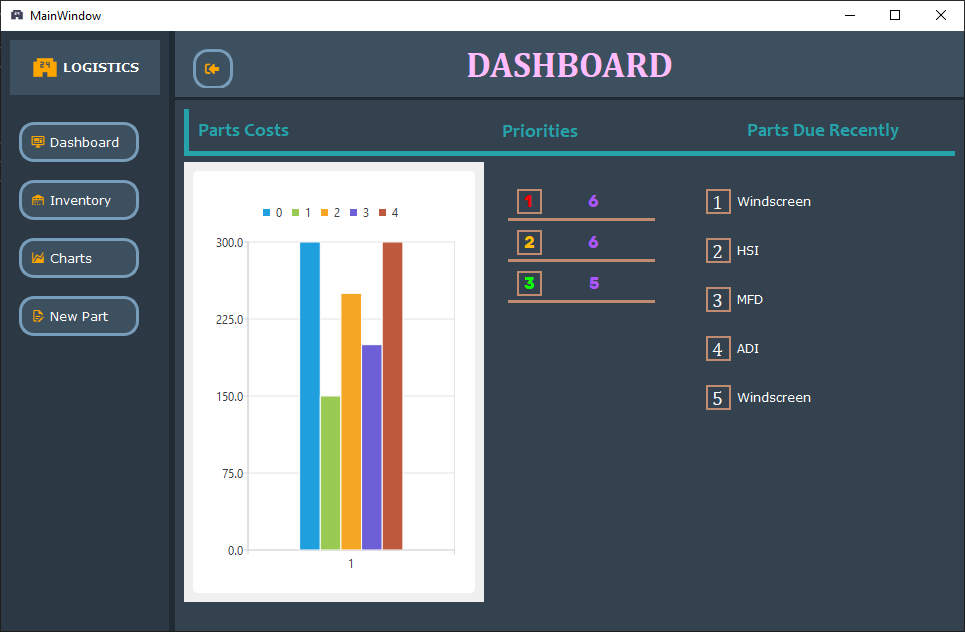
# Scenario | Maintenance - Charts

 Quick overview of week production number

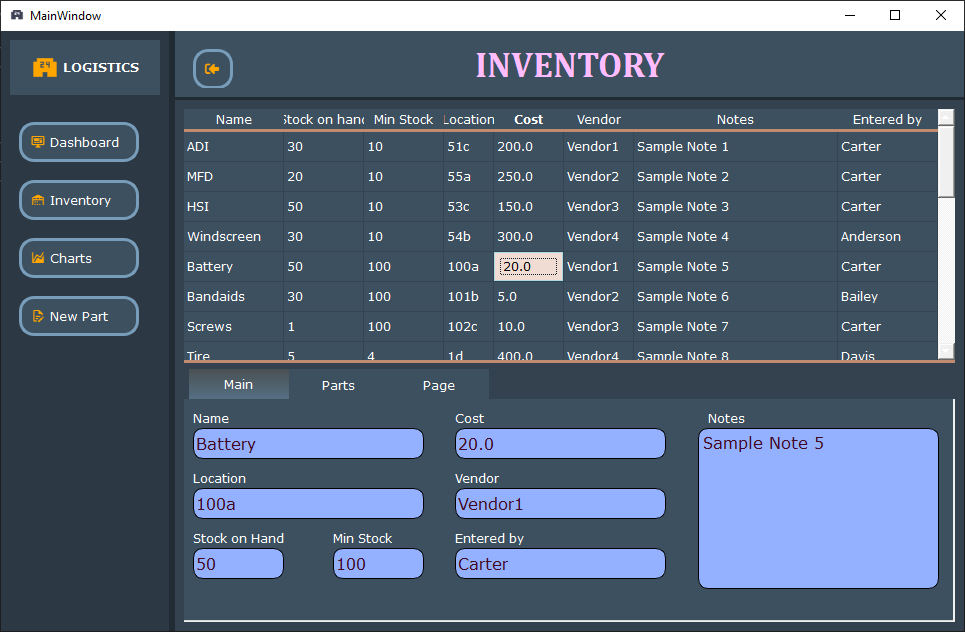
# Scenario | New JCN

 Allows maintenance to create a new work order and save it into the database.

# Scenario | Logistics - Dashboard

Quick overview of parts inventory and numbers of priorities.

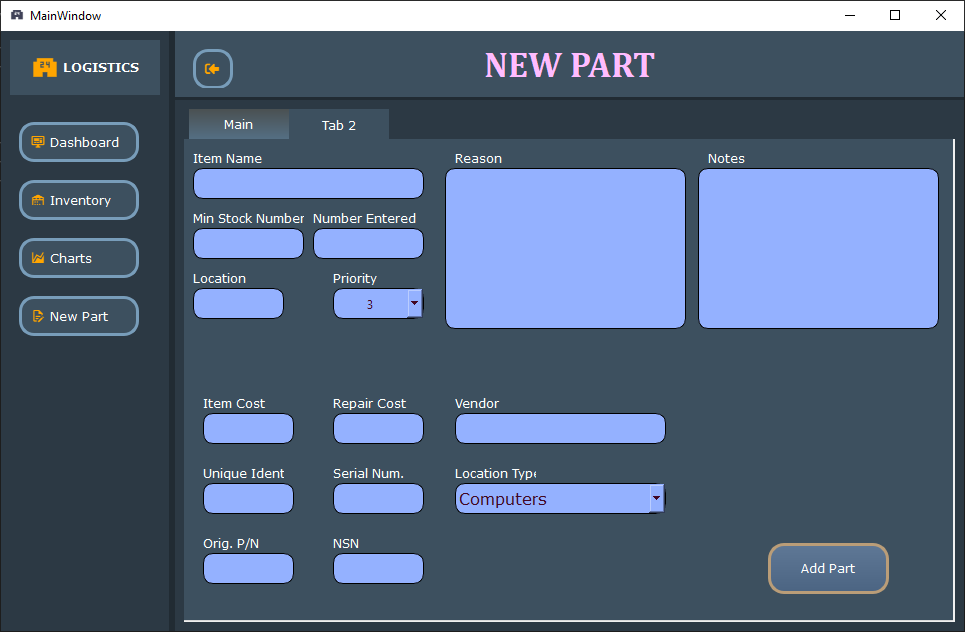
# Scenario | Logistics - Inventory

Allows users to see all inventory, location, and pricing quickly.

# Scenario | Logistics - Charts

Charts for weekly costs and items in stock.

# Scenario | Logistics – New Part

Allows the creation of new parts to be added to the inventory.

# Reports

Charts are outlined in the above scenarios and provide an overview and detailed look into aspects of the work orders and daily costs.

# Testing Plan

Summary: Testing was conducted by separately testing each component and function. After all testing on functions was completed, I brought the code together and then tested each functional operation of the GUI. The database testing was done by error-checking each call to the database. After all, data was received correctly, and then testing was completed.

Unit Testing: Tested each component, function, and module to ensure it worked correctly by itself. After bugs were found, they were troubleshoot and fixed prior to integration.

Integration Testing: After unit testing was completed, code was integrated into the main body of each script. After the integration was completed, refactoring began.

Refactor: Common code was removed from the main scripts to provide unitality functions and classes. The common library allowed me to work from one code base and minimize future changes.

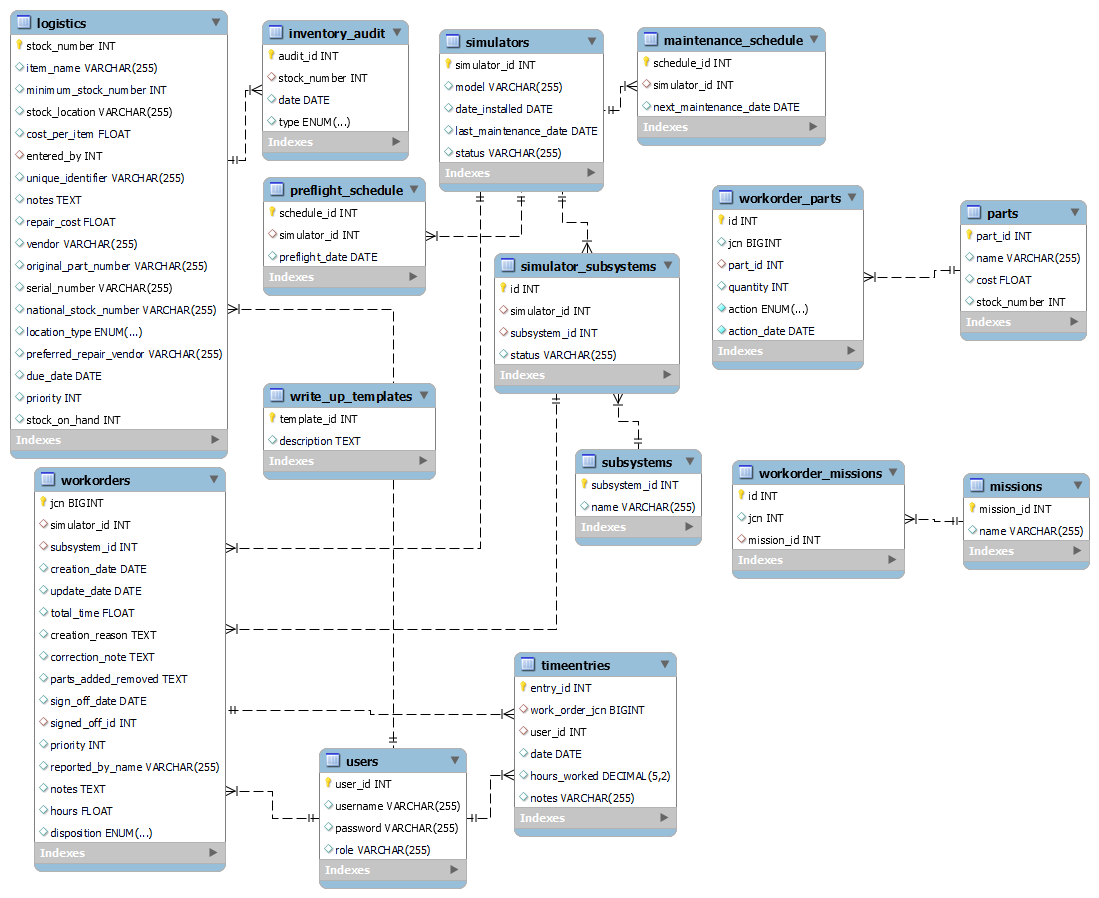
Database Testing: After stored procedures were created, each function call that was made was tested to ensure proper data was returned.

Acceptance Testing: When all other testing was completed. I ran through scenarios of common tasks a user would complete to ensure all code was functioning correctly.

# System Architecture

Overview:  
 The high-level architecture is you start the application with the login screen. After verifying your credentials, you will be taken to the appropriate screen (i.e., Maintenance, Management, Logistics). Calls are made to the database when the application first loads to pull in information. Pushed to the database are only made at the user’s request, then verification happens prior to pushing to the database to protect the database.

**Diagram:**



**Frontend:**

These components will sit on the user’s computers.

**Backend:**

Backend components can sit on any computer that is included in the ‘config.encrypted’ file.

**Database:**

The database uses MySQL. I am using stored procedures to clean up the code required on the front end and to hide some of the functioning of the database. Data is packed into several tables to separate users and functions.

**Communications:**

The connection between the front and backend is done through the execute\_query() function and the key files.

**Security:**

Security is done through encrypted configuration, encrypted public key files, and functions to clean user input.

## Source Code Structure

All files are contained in one folder.

|  |  |
| --- | --- |
| **Code Directory** | |
| **Directory** | **Usage** |
| UI-Design\combined | The folder contains all files for using this program in all aspects. |
| UI Files | login.ui  Logistics-UI.ui  Maintenance-UI.ui  Management-UI.ui  createaccount.ui |
| Python Scripts | login.py  Logistics-UI.py  Maintenance-UI.py  Management-UI.py  login\_utilities.py  ui\_utilities.py  utilities.py  database\_utilities.py |
| Other Files | public.key  config.encrypted  encryption.key |
| UI-Design\encryption | encrypt\_file.py |
| *Highlighted rows indicate directories containing source code.* | |

# Executables

The project utilizes Python scripts version 3.9 and PyQt6 Gui’s to use all functions that would normally be handled in an executable file. The scripts contain the logic, user interface, and connections to the database.

### User Interface:

The project uses Python scripts and not executables to allow users to access data for management, maintenance, and logistics. The GUI controls are made using QT Designer.

### Utility Scripts:

For added functionality and code reuse, utility scripts have been written to contain all useful common code.

# Code Architecture

The architecture of this system is designed to allow users with correct access to manage flight simulator write-ups, logistics, and people. I am using version 3.9 of Python and PyQt6 for all GUIs. The GUI is for the user, the database holds the data, and the scripts connect the two. The goal was to have a responsive interface and to allow for changes and updates to the database.

User Interface:

The project relies on PyQt6 and calls made through its signals and slots between interface components and Python functions.

Utility:

I extracted as much as possible into helper scripts that are used in every GUI application. The reason is to hide some functionality from the user but more so to have a common area where code is stored, used, and updated.

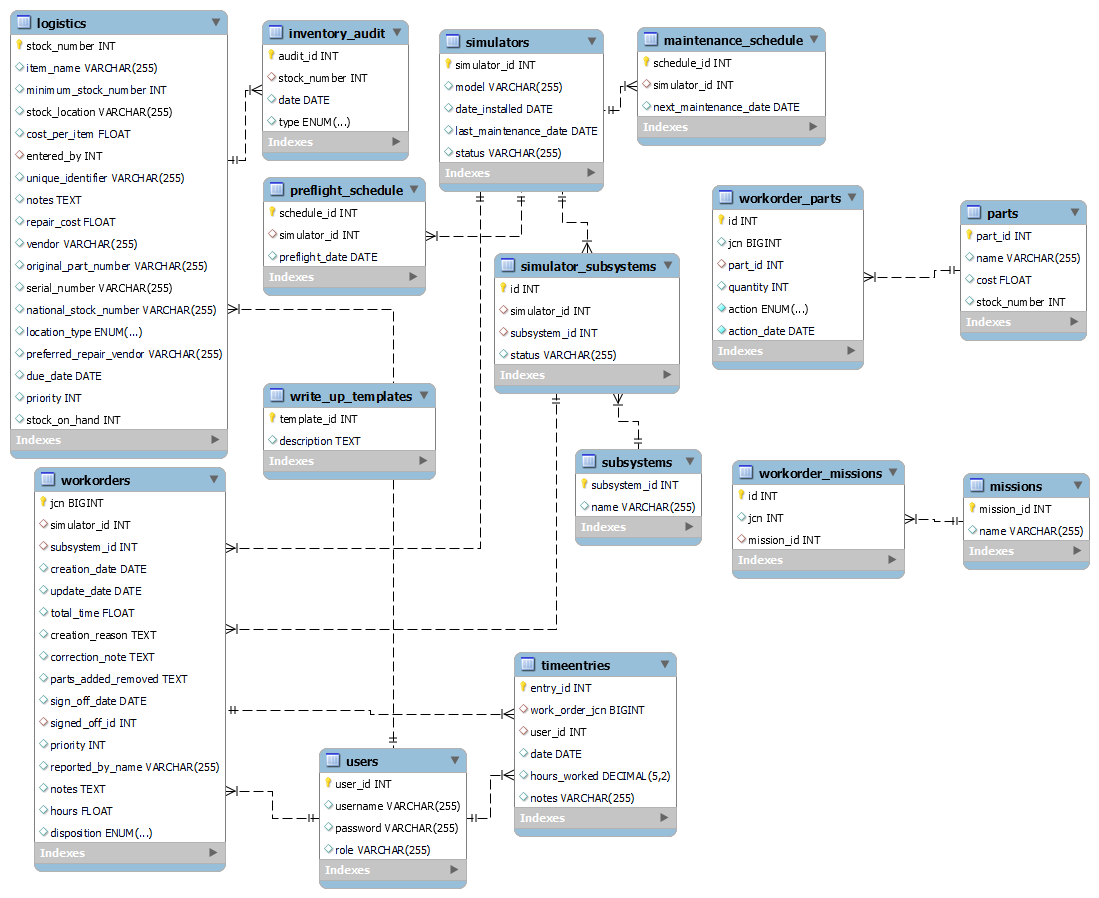
Database:

The database is written for MySQL, which came with a learning curve just to install it on two different types of systems. After initial planning was completed, the database was updated throughout the course until a stable version was reached.

Stored Procedures:

Stored procedures were created to hide some of the functionality of the database from the user and to lower the amount of code that was needed in the scripts to pull out certain types of data. This is especially true for all the charts. Very specific types of data were needed to create the charts, and it was less complicated to do it on the database side than in the Python scripts.

## Database design:



## Views, Stored Procedures, and User Defined Functions

GetHoursWorkedPerPerson: Used for managers to view work hours.

GetInventoryData: Used by managers and logistics to see current inventory in charts.

GetTechSummary: Used by managers to get details about technicians’ daily activities.

GetWorkOrderCountPerDay: used by managers and maintenance to get chart details for work orders.

ShowPartsData: used by logistics to see parts count data.

External Files & Data

Config.encrypted: The file is used to set up the connection from the Python script to the database. If your database changes locations, this file needs to be updated to match.

Encryption.key: The file is used for decryption in the utilities.py file.

Encrypt\_file.py: The file is used to create a new location, password, and encryption key. (i.e., creates the config.encrypted and encryption.key file)

Programming Language | Python3.9 PyQt6

Python3.9:  
 General purpose language, easy to use and write.

PyQt6:

Used for GUI interfaces and ease of construction of Python scripts.

MySQL Connector:

Used to connect to the MySQL database

PyQt6 Charts:  
 Used to create charts after testing was completed, I found PyQt6.Charts are easier to use than matplotlib.

Project Classes

Classes within the project are used to abstract reusable pieces of code. Classes are also used to group related values, known as properties. The project utilizes these classes:

### Login | login.py

Lets the user enter a username, password, or switch to create an account.

### CreateAcc | login.py

Allows the user to create an account as set their type.

### MainWindow | Maintenance-UI.py, Management-UI.py, Logistics-UI.py

The class creates the UI using the .ui file and allows the user to view and enter changes to the database.

### Validator | utilities.py

Checks for bad user input.

Project Modules

Modules are used for procedural-based code that does not require state data like class modules do. Complete the introduction to modules.

### Decrypt\_config | utilities.py

Inputs the config file and returns unencrypted database configuration data.

### authenticate | login\_utilites.py

Checks entered user credentials with stored credentials in the database.

### Get\_user\_role | login\_utilites.py

Returns the role type of the user to know what scripts to bring up and what actions they are allowed to perform.

### Get\_user\_id | login\_utilites.py

Returns the user id for support to other database calls that need the user’s id.

### Execute\_query | database\_utilities.py

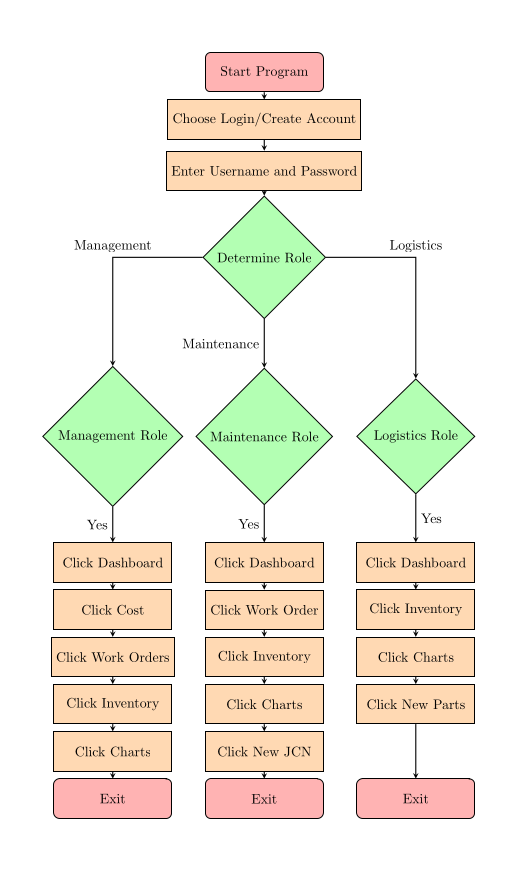
Used to retrieve information from the database.

### Execute\_insert\_query | database\_utilities.py

Used to add information to the database.

Program Start and End Flow

Program control flow is the same for Management, Maintenance, and Logistics.



Summary

The system documentation provides an overview of how the scripts work. It describes the features and actions and is responsive to user input. The following is a summary of that documentation. The document overview describes the three key features that the user will interact with and includes the main login section.

Each role is documented to show the user how to use the application and what features are available to them. For instance, if the user is marked as a Manager, they will have access to views and features unavailable to maintenance or logistics users. The documentation’s purpose is to streamline the user’s learning process and answer any questions they could have.

The System Documentation is there to support the user in learning the application and its role. There is common uses across those roles, but the document will still be split into roles to make referencing tasks easier for each person.

# APPENDIX B (BUILD AND RELEASE PROCESS)

Two tools are used to build the application. The first part is the database, MySQL, and the other is the framework PyQT6, a Python binding’ for the cross-platform toolkit created by Qt. PyQt6 provides the front end, and MySQL provides the back end, with MySQL Workbench as the main interface to the database.

The release process will be as simple as downloading and replacing the original contents of the folder containing the application.

# APPENDIX C (CLIENT INSTALLATION INSTRUCTIONS)

Detail how a client machine or device is prepared to utilize the project.

**Install Python 3.9 on Ubuntu 22.04**

1. Update System Packages:

`sudo apt update`

`sudo apt upgrade`

2. Install Prerequisites:

`sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl-dev libreadline-dev libffi-dev libsqlite3-dev wget libbz2-dev`

3. Download Python 3.9:

`wget https://www.python.org/ftp/python/3.9.7/Python-3.9.7.tgz`

4. Extract and Change directory:

`tar -xf Python-3.9.7.tgz`

`cd Python-3.9.7`

5. Configure:

`./configure --enable-optimizations`

6. Compile:

`make -j 4` # The number after -j can match the number of CPU cores.

7. Install Python 3.9:

`sudo make altinstall`

6. Verify Installation:

`python3.9 --version`

Install packages through PIP for python3

`pip install bcrypt==4.0.1’

`pip install QtAwesome==1.2.3`

`pip install mysql-connector-python==8.0.33`

`pip install cryptography==41.0.1`

`pip install PyQt6==6.5.1`

`pip install PyQt6-Charts==6.5.0`

`pip install PyQt6-Charts-Qt6==6.5.1`

`pip install PyQt6-Qt6==6.5.1`

`pip install PyQt6-sip==13.4.0`

# APPENDIX D (DEVELOPER SETUP INSTRUCTIONS)

Developer instructions will mirror user install instructions. The PyQt6 install will also provide you with the Qt Designer application to change the GUI through a WYSIWYG editor.