InventoryPro

Software Design Document

Aiden McCollum

March 11th, 2025

CS 225, Spring 2025

Embry-Riddle Aeronautical University

Daytona Beach campus

1 Aerospace Boulevard

Daytona Beach, FL 32114

INTRODUCTION:

InventoryPro is a program that allows companies to effectively manage inventory across several different categories and projects in real time. This includes the ability to track an item during it's shipping process, sort based on whether the item is a product to be sold or just used internally, assign items to specific projects defined by the company, sort items into user-defined bins, and more.

This type of inventory management has become critical in many industries, including the aerospace field, for helping companies keep track of what is coming in and out of their warehouse [2]. Without such a system, hardware arriving at your warehouse is prone to getting lost or may be unproperly sorted to a different area in the facility. Additionally, managing hardware becomes even more complex when trying to keep track of how each product will be used and which items belong to specific projects.

With InventoryPro, not only are users able to track where items are in the purchasing process, but they will be able to sort items into different projects within their organization. This will not only ensure all items are properly organized but allow project leaders to see where in the process all their required hardware is. Users are also able to sort items into different areas in the warehouse. This adds an additional layer of organizations for the user to be able to quickly find and distribute items.

Users can log into the system, input information about parts and their projects, and update the status of their items over time. Users will then be able to leverage this information stored in a database to see items tracked throughout the company, get status updates on the projects they are working on, and more.

Inventory Pro provides a real solution to a problem that plagues rocket clubs and professional aerospace companies alike. This document provides a general description of the problem that InventoryPro aims solve and then describes how the program properly fixes the posed issue.

PROBLEM DESCRIPTION:

InventoryPro is an inventory and project management program, like those offered commercially such as Fusion by Stoke Space or Odoo [1] [2]. While these products offer very complex hardware/warehouse management, InventoryPro offers very basic inventory management while tying in basic principles from project management systems like Asana [3]. This program assumes that all users with access to the app have permission to create and edit items as well as create and view projects. This program also assumes that every item coming into the factory is either a product (to be sold) or an expense (to be used for production or for operating your facility). This

program does not include the cost of paying employees or various other shipping costs that might be part of a traditional project management system.

There are five core elements that must be included in InventoryPro to properly function as an inventory and project management platform:

1. Multi-user support and assignment functionality
2. Item creation and updating operations
3. Location creation and bin sorting
4. Project creation with personnel assignment
5. Specific Item, project, and location searching

These basic features are broken down in each subsequent category below.

Multi-user support and assignment functionality

With software centered around collaboration like inventory management systems, it is critical that users have the option to create an account and be able to log into that account in the future. When creating a user account, there are several key pieces of information that must be collected to allow users to collaborate. Table 1 below provides the essential characteristics about a user that must be collected, their type, and requirements for the value that the user can select.

Table 1: User Account Properties

User Characteristic	Characteristic Type	Characteristic Constraints
Employee ID	String	Must be unique
Employee Name	String	Must be at least 2 characters
Password	String	Must be at least 6 characters and have at least one special character and number

On top of collecting and storing this information, users should also be able to log in using the employee ID and password provided during the sign up.

To be able to properly collaborate on projects, users must be able to be assigned to projects. This kind of functionality allows users to see which tasks they are responsible for as well as provide a clearer method for tracking inventory based on a specific project. Each user can be assigned to multiple projects, but each project should only be assigned to one user. Projects don't necessarily need to be assigned to a user, but they can be. The assigned user can also be changed as needed by the users of the platform.

Item creation and updating operations

Another critical functionality for inventory management platforms is the ability to add items to the software. For many companies, items are added to the platform the moment that they are

purchased. An item has specific attributes that define it such as name, quantity, unit cost, item type (product or expense), supplier, and status. Items will typically maintain the same name and type over time, but price, quantity, supplier, and status are all subject to change over time. Therefore, a user must be able to adjust an item over time as needed.

An item's status is defined as where it is in the processing workflow. These workflows tend to vary based on application and industry, but for simplicity, it has been reduced to the workflow provided in diagram 1.

Diagram 1: Item workflow for the standard purchasing process

An item also needs to be able to optionally be assigned to a project. This means that the item would be linked to a specific project, allowing users to find specific items based on the project they are allocated to. This allows employees to effectively track similar items in varying points in their respective acquisition workflows.

Location Creation and Bin Sorting

Another critical function of an inventory management system is the ability to define different locations in a warehouse and select items to store there. Location names can be defined as deemed appropriate by the user. When managing inventory, it is critical for users to be able to look up an item and accurately find where that item is stored in the warehouse. This helps speed up sales and manufacturing operations, as well as help companies maintain a high level of organization.

Users for the platform need to be able to define locations within the warehouse. To simplify the definition of a location, the platform should use a method called "bin sorting". In essence, bin

sorting is where items are categorized together and given an overarching label. This is best demonstrated in diagram 2 below.

Diagram 2: Example bin sorting diagram for a supermarket.

To utilize bin sorting, once an item has been created, users should have the option to place the item under a bin. This allows users to quickly find items by using the software to find a bin, see which items are in it, then go to the corresponding area in the warehouse and grab the item. In the software, this means the item is assigned to that bin.

Project creation with personnel assignment

Another critical part of inventory and project management software is being able to create and delegate projects [1]. For a company, regardless of it's size, there are always projects being undertaken which require specific items to operate. While the term for referring to these projects varies from company to company, every organization has different working goals that require specific items.

A solid example of a project would be the Starship program for SpaceX. The starship program could be considered a project as it requires specific components and has a leader in charge of the project. The Starship "project" is made up of dozens of sub-projects, but to keep InventoryPro simple, we will assume that the organization operates on 1 layer of projects (no sub-projects).

Each project, when created, needs to be assigned a leader. This leader can be considered a point of contact for the project and oversees all items coming in and out of the project.

Having projects within an inventory management system makes it easy for the leader of a project to track where all their items are in the purchasing process and located in the factory without

having to create a separate list outside of the application. Each item should only be assigned to one project.

Specific Item, project, and location searching

One of the most critical parts of inventory and project management is being able to efficiently find the items you are looking for! Therefore, a good inventory management system needs to allow employees to effectively search for a specific and find details about its status and location.

Additionally, employees should be able to get a birds-eye view of an organization's current inventory by examining parts on a project or location level.

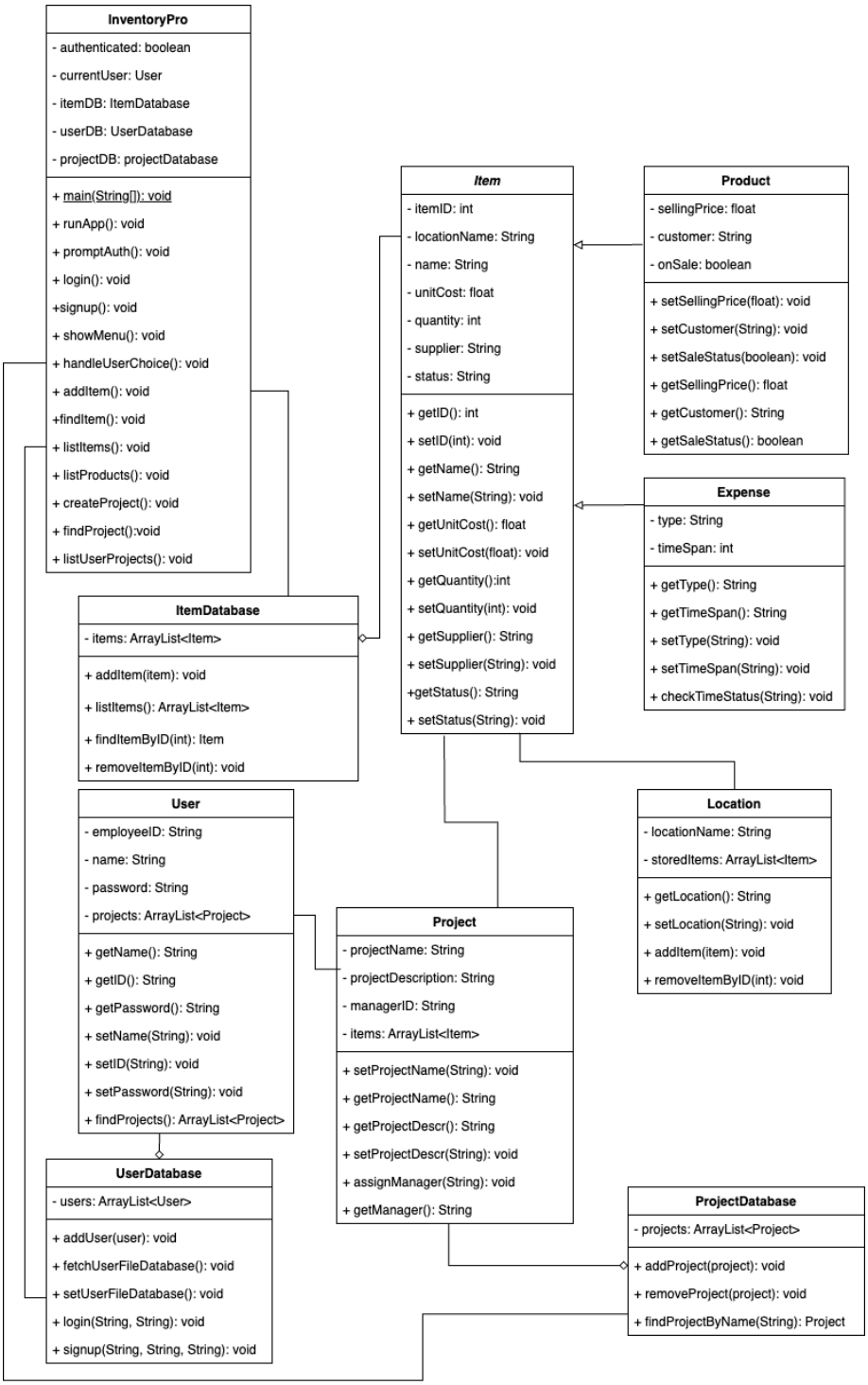
Examining parts on a project level allows leads and those interested in the project to filter through the company's entire inventory and only see the status and location of the items of relevance to the project.

Examining parts on a location level allows employees to verify that the warehouse is properly organized. If an employee looks up a bin identifier in the software and sees a list of parts assigned to that bin, they should be able to go into the warehouse and verify that the item is there.

This level of organization is critical for all types of organizations, whether you are part of a grocery chain or running an aerospace company. These features are parts of the standard functionality that has become expected from inventory management systems nationwide.

PROBLEM SOLUTION:

This section provides excellent insight into the structure of the program's underlying software using the Universal Modeling Language (UML). Figure 1 below shows the UML diagram for InventoryPro.



InventoryPro:

The *InventoryPro* class is the main class of the program. This class shall oversee running the entire program, collecting user inputs, and acting as the delegator for executing high level functions to trigger user interactions, item creations, and more.

Attributes:

- **authenticated** (Boolean): An attribute that indicates whether the user is logged in.
- **currentUser** (User): An attribute that stores the current user of the program.
- **itemDB** (ItemDatabase): Attribute containing an instance of the item database.
- **userDB** (UserDatabase): Attribute containing an instance of the user database.
- **projectDB** (ProjectDatabase): Attribute containing an instance of the project database.

Methods:

- **runApp()**: This method shall initialize the program by instantiating the item, user, and project databases. It shall also default set the authenticated state to false.
- **promptAuth()**: This method prompts the user to determine whether to log in or sign up.
- **login()**: This method shall prompt the user for an employee ID and password to start the log in process. These values shall be passed to the login function in the userDB.
- **signup()**: This process shall prompt the user for an employee ID, name, and password to start the sign up process. These values shall be passed to the signup function in the userDB.
- **showMenu()**: This method shall show the user the menu, consisting of options to : add an item, find an Item, list all items, list all products, create a project, find project by ID, find my projects, or the quit program.
- **handleUserChoice()**: This method handles the selections that the user makes from the showMenu() method and calls the corresponding functions to trigger that action.
- **addItem()**: This method shall prompt the user for the information needed to add an item and then pass that relevant information to the addItem function in the item database.
- **findItem()**: This method shall prompt the user to enter the ID of the item they would like to find then trigger the findItemByID() function in the item database to find the actual item. It shall then take the returned value and format it into a nice string to display for the user.
- **listItems()**: This method shall display each item in the database in a nice format.
- **listProducts():** This method shall display every item in the database that is a product.
- **createProject():** This method shall prompt the user for the information required to create a project and print out the confirmations.

- **findProject():** This method shall prompt the user to enter the name of the desired project and shall print out the project information.
- **listUserProjects()**: This method shall list every project that the user is a manager of in a clean and human readable format.

ItemDatabase:

The *ItemDatabase* class is made to handle all data storage and fetching for items in the program. It is tasked with storing all the items in the program at runtime and is used to keep the database updated at all times.

Attributes:

- **items** (ArrayList<Item>): This attribute shall be used for storing all the items in the program during runtime.

Methods:

- **<<constructor>>ItemDatabase():** This method shall load all the items from the item database file into the attributes of the class, where each object in the class will be instantiated with the loaded data.
- **addItem()**: This method shall add an item that has been initiated into the items list and save it to the database.
- **listItems()**: This method shall return all the items in the database.
- **findItemByID()**: This method shall return an item in the items list by matching the IDs.
- **removeItemByID()**: This method shall remove an item from the items list by matching the IDs.

UserDatabase:

The *UserDatabase* class is made to handle all data storage and fetching for users in the program. It oversees storing all the users in the program, keep the database updated at all times, and handle authentication.

Attributes:

- **users** (ArrayList<User>): This attribute shall be used for storing all the users in the program during runtime.

Methods:

- **<<constructor>>UserDatabase():** This method shall load all the items from the user database file into the attributes of the class, where each object in the class will be instantiated with the loaded data.
- **fetchUserFileDatabase():** Loads in the data from the users database file and passes the arraylist to the users attribute.
- **setUserFileData():** Writes the user data from the users attribute into the users database file.
- **addUser()**: This method shall add the passed in user to the users list and database when called.
- **login()**: The method shall validate the user's ID and password inputs and let them into the application if they match to one user record.
- **signup()**: The method shall create a user instance and add them to the user list and database.

ProjectDatabase:

The *ProjectDatabase* class is made to handle all the data storage and fetching the projects in the program. It is meant control how projects are made and updated.

Attributes:

- **projects** (ArrayList<Project>): This attribute shall be used for storing all the projects in the program during runtime.

Methods:

- **<<constructor>>ItemDatabase():** This method shall load all the items from the project database file into the attributes of the class, where each object in the class will be instantiated with the loaded data.
- **addProject()**: This method shall add an instantiated project to the projects list and database.
- **removeProject()**: This method shall remove the project from the projects list and database.
- **findProjectByName()**: This method shall find and return the target project by matching the name.

User:

The *User* class is intended to hold all the important information about a user and handle any necessary interactions for user-specific operations.

Attributes:

- **employeeID** (String): This attribute shall hold the user's employee ID.
- **name** (String): This attribute shall hold the user's name.
- **password** (String): This attribute shall hold the user's password.
- **projects** (ArrayList<Project>): This attribute shall hold the user's list of projects

Methods:

- **<<constructor>>User():** Instantiates a new user object with the following attributes defined as given parameters: employeeID, name, password.
- **getName()**: Returns the user's name.
- **getID()**: Returns the user's employee ID.
- **getPassword()**: Returns the user's password.
- **findProjects()**: Returns the list of all the user's projects.
- **setName()**: Sets the name value for the user.
- **setID()**: Sets the employee ID value for the user.
- **setPassword()**: Sets the password attribute value for the user.

Project:

The *Project* class is intended to handle all interactions dealing with projects in the program, including adding items to projects and fetching project details.

Attributes:

- **projectName** (String): This attribute holds the name of the project.
- **projectDescription** (String): This attribute holds the description of the project. This value tends to be at least one sentence.
- **managerID** (String): This attribute holds the employee ID of the user in charge of the project.
- **items** (ArrayList<Item>): This attribute holds all of the items associated with the project.

Methods:

- **<<constructor>>Project():** Instantiates a new project object with the following attributes defined as given parameters: projectName, projectDescription, managerID.
- **setProjectName():** Sets the value of the project name attribute.
- **setProjectDescr():** Sets the value of the project description attribute.
- **getProjectName():** Returns the name of the project attribute.
- **getProjectDescr():** Returns the string value of the project description attribute.

- **assignManager():** Sets the manager ID attribute to the relevant employee ID and adds the project to the user's projects attribute.
- **getManager():** Returns the employee ID of the manager of the project.

Item:

The *Item* class is an abstract class that shall store all the relevant attributes and methods that make up all items in the software. This abstract class will serve as a parent class for the Product and Expense classes discussed below.

Attributes:

- **itemID** (int): This attribute holds the program generated random number for the ID of the item.
- **locationName** (String): This optional attribute holds the item's physical location in the warehouse.
- **name** (String): This attribute holds the name of the item.
- **unitCost** (float): This attribute holds the cost per item for the item.
- **quantity** (int): This attribute holds the quantity of items.
- **supplier** (String): This attribute holds the name of the supplier.
- **status** (String): This attribute holds the current status of the item, which can be one of the following values: Purchased, In Transit, Arrived, In Stock, or Out of stock.

Methods:

- **getID():** Returns the item's ID attribute.
- **getLocationName():** Returns the name of the location the item is stored at.
- **getName():** Returns the name of the item.
- **getUnitCost():** Returns the cost per item of the item.
- **getQuantity():** Returns the number of this item that exist.
- **getSupplier():** Returns the supplier of the item.
- **getStatus():** Returns the current status of the item.
- **setID():** Sets the item's ID attribute.
- **setLocationName():** Sets the name of the location the item is stored at.
- **setName():** Sets the name of the item.
- **setUnitCost():** Sets the cost per item of the item.
- **setQuantity():** Sets the number of this item that exist.

- **setSupplier():** Sets the supplier of the item.
- **setStatus():** Sets the current status of the item.

Product:

The *Product* class is responsible for all items that are being sold by the company. This class inherits the attributes from its parent *Item* class, but consists of additional attributes and methods specific to the sale and distribution of an item.

Attributes:

- **sellingPrice** (float): This attribute is the price that the item is being sold for.
- **customer** (String): This attribute is the name of the costumer who will be buying the item.
- **onSale** (Boolean): This attribute indicates whether the item is being sold for a discount.

Methods:

- **<<constructor>>Product():** Instantiates a new Product object with the following attributes defined as given parameters: sellingPrice, customer, onSale.
- **setSellingPrice():** Sets the price that the item is being sold at.
- **setCustomer():** Sets the value of the customer attribute.
- **setSaleStatus():** Sets the status of whether the item is for sale or not.
- **getSellingPrice():** Returns the price that the item is being sold at.
- **getCustomer():** Returns the value of the customer attribute.
- **getSaleStatus():** Returns the status of whether the item is for sale or not.

Expense:

The *Expense* class is responsible for all items that are a consumable for the company. This class inherits the attributes from its parent *Item* class, but consists of additional attributes and methods specific to items that are consumed within the company.

Attributes:

- **type** (string): This attribute describes the type of expense the item is.
- **timespan** (int): This attribute defines how long the expense is expected to last in days. For a subscription expense, this would be the duration of the expense. For a consumable expense, this would be how long the item is expected to last.

Methods:

- **<<constructor>>Expense():** Instantiates a new expense object with the following attributes defined as given parameters: type, timespan.
- **getType():** Returns the value of the type attribute.
- **getTimespan():** Returns the number of days the product is expected to last.
- **setType():** Sets the value of the type attribute.
- **setTimespan():** Sets the number of days the product is expected to last.
- **checkTimeStatus():** Determines whether the item will expire soon. If the item's time span is less than 7 days, the method will return a string saying "expiring soon". Otherwise, it will return "Not expiring soon".

Location:

The *location* class is intended to help indicate where items are physically stored by allowing users to define names of locations and the items at each location. This class shall allow items to be added and removed from a location.

Attributes:

- **locationName** (String): This attribute is the name of the location.
- **Items** (ArrayList<item>): This attribute holds all the items in the location.

Methods:

- **<<constructor>>Location():** Instantiates a new location object with the following attributes defined as given parameters: locationName.
- **getLocationName():** Returns the name of the location.
- **setLocationName():** Sets the name of the location.
- **addItem():** Adds a passed in item to the items list and changes the location attribute the item to the name of the location.
- **removeItemByID():** Removes the target item from the items list by matching IDs.

REFERENCES:

[1] Odoo. (n.d.). *Warehouse & inventory management in Odoo*. Retrieved from https://www.odoo.com/app/inventory

[2] Stoke Space. (n.d.). *Fusion: Inventory and Project Management Solution*. Retrieved from https://www.stokefusion.com/product/

[3] Asana. (n.d.). *Asana: Project Management.* Retrieved from https://asana.com/features/project-management

*[**Shall be non-empty in P1 and updated as needed with each deliverable**.]*

APPENDICES:

This is optional, but may include external sources, source code, or other related material.

*[**Shall be completed as needed with each deliverable**.]*