Medical Inventory Management System (MIMS)

Software & Project Documentation

Wasatch CAPS Project
Noah Lake, Jaxon Powers

Links, Development, and Sources

Project GitHub Repository

LucidChart Relationship Diagrams
LucidChart UML Diagrams
LucidChart Class Diagrams

openFDA Drug API Endpoints
openFDA Device API Endpoints

Introduction to the Medical Inventory Management System

In the realm of medical supply management, the effectiveness of inventory systems is crucial yet often lacking. Traditional systems are cumbersome, difficult to implement, and fail to meet the necessary standards of efficiency and accessibility. These shortcomings result in significant time wasted by healthcare professionals as they search for and organize medical supplies, further exacerbating the challenges of managing medical inventory.

Recognizing these critical issues, the Medical Inventory Management System (MIMS) was developed to revolutionize how medical facilities manage their inventory. MIMS streamlines the process of tracking, accessing, and utilizing medical supplies. It minimizes waste, enhances the organization of supplies, and significantly reduces the time staff spend locating specific items. By integrating advanced features like the Dynamic Product Locator (DPL), MIMS not only simplifies inventory management but also ensures that medical practitioners can quickly and efficiently scan and check out the items they need, optimizing the overall workflow within medical facilities.

Table Of Contents

Project Programming Language

The main competitors for this project were Java and Objective C. After assessing some of the benefits and weaknesses of each language, it became clear that Java would be better suited for this application.

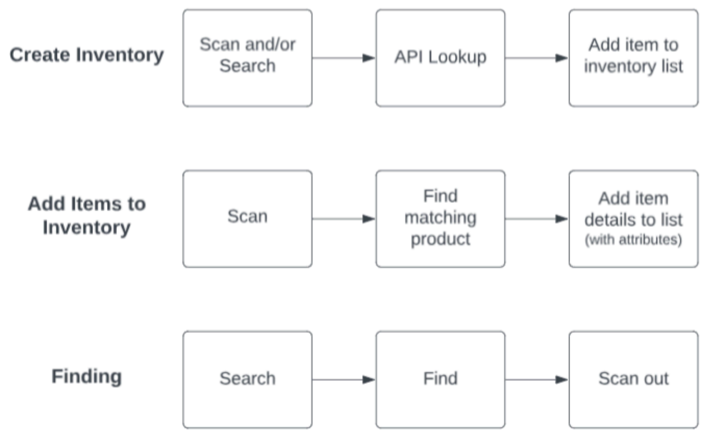
Java	Objective C
➔ Interpreted language: slower to execute code ➔ Object Oriented Programming (OOP) ➔ Closed memory access makes it so data cannot be directly accessed through the memory, which makes it less vulnerable to security risk ➔ Better at tracing errors and handling exceptions ➔ Executable on any OS as long as it has a JRE or JDK installed (can be overcome through .jar wrapping)	➔ Compiled language: faster to execute code ➔ Object Oriented Programming (OOP) ➔ Open memory access allows for more system-level customization of memory but poses a security risk ➔ Dynamic typing poses a better chance of getting a runtime error ➔ Fewer security checks ➔ Executable on Windows only

Java offers the ability to run on any operating system, which makes it easy for the user and the developers who might be switching between devices frequently.

While running a Java application requires a Java Runtime Environment (JRE) or a Java Development Kit (JDK), this problem can be easily overcome by wrapping the application once it's finished in an OS-specific file (such as .app, .exe, etc.).

Using MIMS

The Medical Inventory Management System (MIMS) is a process in which a nurse will be able to locate an item quickly and “check out” the item from the inventory.



The processes in which a user structures the inventory, adds the product to the inventory, and finds/checks out a product is very streamlined.

Dynamic Product Locator (DPL)

Basic Overview

MIMS uses a Dynamic Product Locator (DPL), which is a system in which an inventory system can be fully customized to any storage room or structure. The DPL uses a similar structure to a HashMap, where a general object may be inputted and a location is returned.

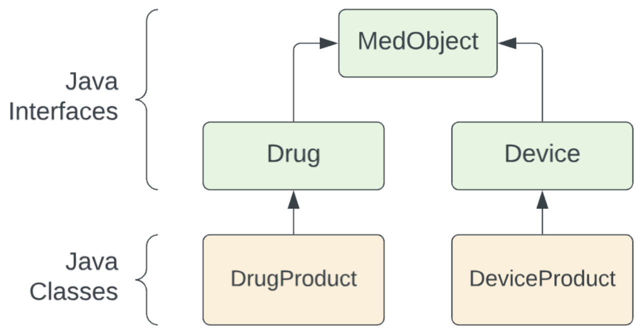
The locator tool is used anytime a user searches for a product and tells them where the product is located. For example, if a user was searching for “Advil,” they’d look it up to see where Advil can be found in the storage room, along with other attributes.

Each item inside the DPL will reference its product manager object.

Example:

- Storage Room
    - Shelf A
        - Container 1
            - Item 001
        - Container 2
            - Item 002
    - Shelf B
        - Container 1
            - Item 003
            - Item 004
    - Cold Storage Room
        - Shelf Z
            - …

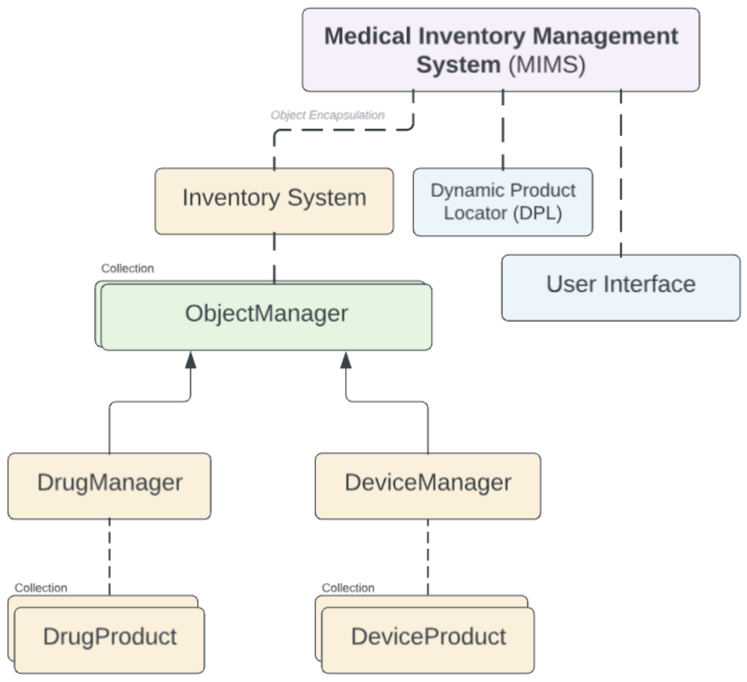
Simple Medical Object Hierarchy



Both DrugProduct and DeviceProduct objects implement interfaces derived from MedObject.

The MedObject interface contains methods that are generic, while their inheritors contain product-specific methods.

MIMS Project Hierarchy



The MIMS program has three main components: the Inventory System, the DPL, and the UI.

The InventorySystem encapsulates a collection of ObjectManager objects.

The ObjectManager interface is implemented by two classes: DrugManager and DeviceManager.

DrugManager encapsulates a collection of DrugProduct objects while DeviceManager encapsulates a collection of DeviceProduct objects.

Medical Item Identification

The best way to identify a medical item inside an inventory varies depending on the type of product it is.

A NDC (National Drug Code) is an identifier number that indicates exactly what attributes that drug has.

A UDI (Universal Device Identifier) is a device identifier that gives a lot of useful information. The FDA requires that every UDI has a DI, a fixed portion that identifies the labeler and the specific version or model of a device. Additional optional data can also be found as part of the UDI, such as the manufactured date, expiration date, lot/batch number, and the serial number.

Drug/Device API Lookup

The FDA has an API called openFDA. This API can be utilized from an HTTP connection and returns a JSON after retrieval.

The openFDA API has two structures that are used in the MIMS application. The NDC (National Drug Code) and DI (Device Identifier) lookup tools.

NDC Lookup:

Example URL Formatting: *(for advil) 66715-6434-2*
https://api.fda.gov/other/nsde.json?search=package_ndc:"66715-6434-2"

DI Lookup:

Example URL Formatting: *(for a scalpel) 00888118119018*
[insert here]

User Interface Features

Below is an outline of all UI pages and how each feature process functions. The items highlighted in YELLOW are features that will not be included in early versions of the MIMS application.

AI-driven research documents:

- ➔ Integrating OpenFDA API into MIMS (AI-driven research)
- ➔ MIMS Detailed UI Feature Suggestions (from ChatGPT4).pdf
- ➔ MIMS UI Feature Suggestions (from ChatGPT4).pdf

Dashboard

- Statistical Widgets
  - Par level notifications/warnings - if the user clicks on this it opens up the Par Report. (A generalized widget that shows how many warnings there are)
  - Expiration/Sterilization warnings (generalized widget that shows how many warnings there are) - if the user clicks on this, it opens up the sterilization warning
- Activity Feed
  - General display of recent inventory system logs - if the user clicks on this, it opens up the full log/stat report.
- Interactive Charts
  - Most recently used products (pie chart representing the last month of operations)
  - Other insights

Operations - the "action" page of the application. Allows staff to check in and check out any devices or drugs

- Check-In - allows a staff member to add more stock into the system
  - Search with Name or Product ID
    - If there is already a device or drug manager for the specified product, prompt the user for product quantity and expiration/sterilization dates
    - If there is not already a device or drug manager for the specified product, search the FDA API for more product information, and prompt the user for product quantity and expiration/sterilization date for the product, along with an *OPTIONAL* field for par level.

            - Instantiate a new product manager for the specified product - construct that manager with the quantity and end dates gathered in the previous dialogue.
            - Prompt the user with an option to modify the DPL configuration
    - Update Backend (Product Managers)
    - Update UI
- Check-Out - allows a staff member to scan needed products out of the system
    - The user is prompted with two options: search for a product using name/ID and scan a product (they'll have to scan out a product anyway)
        - Search for a product - if the user chooses to search for the product, then search product managers and the DPL to give information for that product, then allow the option for them to scan out
    - Scan out: the user will scan the product ID of a product's packaging and the system will remove it from the inventory
        - Find the product manager responsible for the item scanned
        - Check to make sure the item isn't expired or needs sterilization (if it does, notify the user)
- Edit product par levels
- Manually edit product inventory (or delete products)
- Run expiration and par check

Settings

- DPL
    - Room Layout - edit the layout of a room (editing the DPL)
    - Enable/disable DPL functionality
- User Interface
    - Light/dark mode
    - Text size adjustment - three main reconfigurations (small, medium, large text)
- Product Management
    - Enable/disable expiration checks
    - Enable/disable sterilization checks
    - Adjust time routine
    - Barcode scanning configuration
- Notifications
    - Par level warnings
    - Expiration/sterilization date warnings
    - Push report to an email
- Automatic Report Generator

-
    - Time routines
    - Info to include
    - Configuration of Report
- Data Autosaver
    - Time routines
    - Where to save the data files
    - What to label the data files
    - Switch - Save inventory logs?

Notifications

- Individual Notifications
    - Each notification has an enumeration (Enum) attached to it labeling the category of what it contains
        - Par Level Warnings
        - Expiration/Sterilization Warnings
        - System Notification
        - Report Generation
    - Each notification has variables holding simple booleans representing whether the notification is urgent and whether the notification has been resolved.
    - Each notification has a few options that the user can select.
        - Delete (only available if the notification isn't marked as urgent or is marked as resolved)
        - Resolve - takes the user to wherever page they need to go to resolve the notification (only available if marked as urgent)
        - Open (only on report/system notifications)
- Settings button - when clicked, forwards the user to the notifications section inside settings

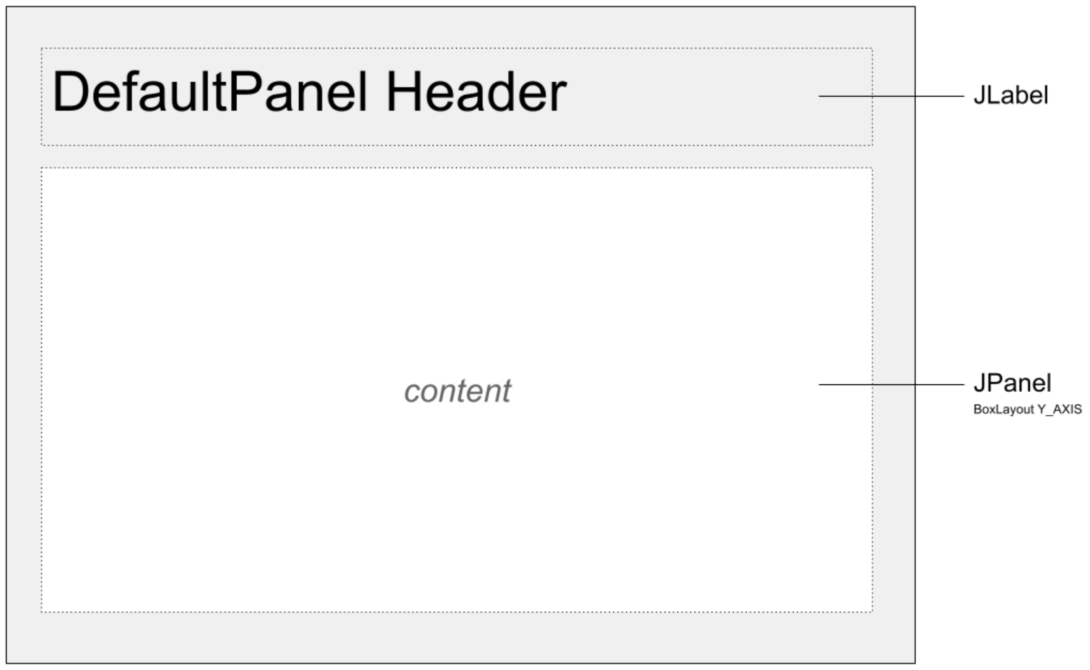
Help Dialogue - a simple dialogue box that pops up and helps the user reach out to the development team for questions, suggestions, or anything else.

User Interface Objects

DefaultPanel

Rather than utilizing a custom JPanel for each page inside the MIMS application, a class called *DefaultPanel* extends JPanel and has default traits that almost every page inside the application follow.

Layout of the DefaultPanel UI Object:

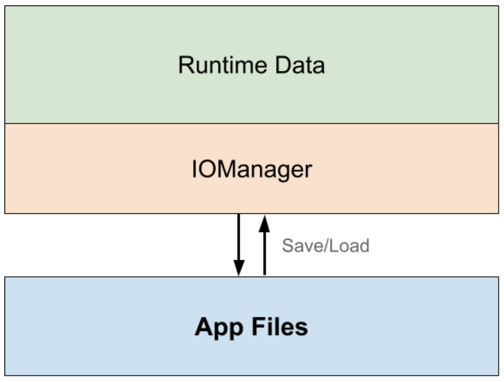


When calling the **add()** method of a DefaultPanel, the component included in the method parameters will not be added to the inherited JPanel, but rather to the JPanel *content.* Similar method calls that influence panel behavior will also be overridden and forwarded to the content panel.

The DefaultPanel object also has several available methods to customize behavior such as **setHeader(String text)**.

Application Data Management

All systematic variables, application settings, inventory configurations, and environment variables are located in the application folder labeled *MIMS_Data*.



The *IOManager* class handles the conversion of essential runtime variables to system files saved in the app folder. It's the "middle man" of saving data.

The *MIMS_Data* folder is under the *AppData/Roaming* directory. To find the directory on your Windows Machine, use Win+R, run the command "appdata," and look in the *Roaming* folder. Example AppData File Path: C:\Users\noahl\AppData\Roaming\MIMS_Data

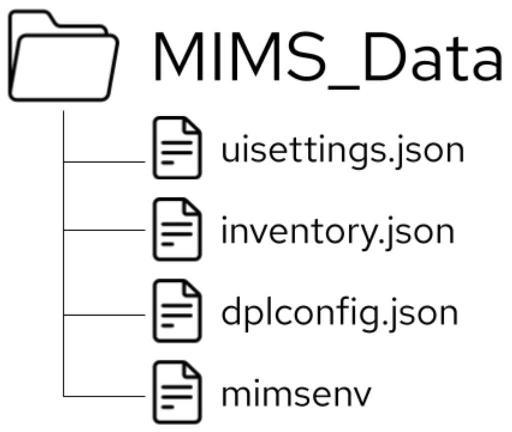
When the application autosaves, the IOManager will fetch all variables from the system and convert them into a JSON file. Upon app launch, the IOManager will pull from the app folder and construct objects with values read from existing files.

This application uses the open-source FasterXML Jackson, a lightweight JSON parser built for Java. The module can quickly convert large data arrays in the MIMS application into JSON-parsed files and can load them back into class constructors upon launch.

Maven Dependency: *com.fasterxml.jackson.core:jackson-core:2.14.2*

Upon application launch, the IOManager will check if the MIMS_Data folder exists. If it doesn't exist, the app will create one. Only the *mimsenv* file will be created with all default launch settings. In this case, the other JSON files will be instantiated when the application autosaves data.

Files Inside the MIMS_Data Directory



uisettings.json - Holds all settings pertaining to the user interface. Only saved to when a user alters and saves their changes.

inventory.json - Holds all settings and data pertaining to the inventory stock. Includes ObjectManagers and every product in their management. This will typically be the largest file.

dplconfig.json - Holds the Dynamic Product Locator's (DPL) arrangement of every container and product associated with it.

mimsenv - A simple plaintext file that holds systematic variables which are read upon launch. Variables stored on this file influence main behavior of the application, such as development or production mode.