**Boston University**

**Electrical & Computer Engineering**

**EC46**4 **Senior Design Project II**

Second Prototype Testing Plan

**AI-Enhanced Pharmacy Procurement**

by

Team 6

PAPO Group

Team Members

Joel Akerman [akermanj@bu.edu](mailto:akermanj@bu.edu)

Taha Ababou [hababou@bu.edu](mailto:hababou@bu.edu)

Manuel Segimon [manuelsp@bu.edu](mailto:manuelsp@bu.edu)

Bora Bulut [bbulut@bu.edu](mailto:bbulut@bu.edu)

Zaiyan Muhammad [monem@bu.edu](mailto:monem@bu.edu)

# 1. Introduction

The second prototype represents a significant advancement in user management and data processing capabilities. This iteration incorporates a comprehensive database designed to store user information securely, ensuring privacy and accessibility. It is equipped with a robust set of API calls that facilitate user interactions such as registration, login, logout, as well as editing and removing user profiles, thereby enhancing user experience and system functionality. Additionally, the prototype features a sophisticated package analysis algorithm, which efficiently identifies and matches relevant drugs based on their dosage and form. This innovation not only streamlines the drug selection process but also improves the accuracy and efficiency of the system, making it a powerful tool for our client.

## 1.1 Purpose

This document outlines the additions made since the first prototype testing. It highlights the specifications and tests done on the package analysis algorithm, which holds a significant part of the overall PAPO Procurement Software, as well as the database for storing user information and API implementation. All of these features will be used by all stakeholders, including developers and testers.

## 1.2 Scope

### 1.1.1 In Scope (features that will be tested in this prototype)

* **Online Database**
  + Validate the integrity of the database that stores user credentials by ensuring that it is up and running, bound to the API, and updates user information according to the changes in real time.
* **API Calls**
  + Certify that API calls create user, delete user, edit certain granted user information, log users in, log users out, and lock users after a certain number of incorrect login attempts.
* **Dosage Information Extraction Accuracy**
  + Verify that the extractDosage function correctly extracts dosage information from the input strings and handles a variety of formats and units correctly.
* **Data Classification**
  + Ensure that the classification function correctly identifies each item's basic form (e.g., Liquid, Solid) based on its properties, such as 'Total\_ML' and 'Total\_MG' fields.
* **Similarity Filtering**
  + Test the filtering process to ensure that only items with a 'True Similarity' greater than 0.5 are kept for further processing.
* **Dosage Information Enrichment**
  + Verify that the addDosageInfo function correctly adds and calculates the dosage information, enriching both the input and larger datasets as intended.
* **Specialized Filtering Logic**
  + Ensure that the algorithm correctly filters the dataset based on the form of the medication (e.g., solid or non-solid) and correctly applies specialized filtering based on dosage size and unit compatibility.
* **Total Dosage Calculation**
  + Test the calculation of total dosage values to ensure accuracy in converting and summing different units into standardized totals (e.g., total milligrams, total milliliters).
* **Dataset Cleaning**
  + Verify that the script correctly drops the original measurement columns after the total values are calculated, leaving a clean dataset for analysis.
* **Output Accuracy and Format**
  + Ensure that the final output (e.g., potential medication replacements) is accurate, properly formatted, and matches the expected criteria based on the input data.

### 1.1.2 Out of Scope

The following features will not be tested for this prototype presentation:

* User Interfaces
* Accuracy Tests with the Procurement Team at Beth Israel Medical Center

## 1.3) Roles and Responsibilities

| **Manuel** | Algorithm Developer/Head of Continuous Improvement |
| --- | --- |
| **Taha** | API Developer/QA Tester |
| **Bora** | Database Developer/QA Tester |
| **Joel** | Team Leader/Product Manager |
| **Zaiyan** | Market research |

# 

# 

# 

# 2. Code Overview

# Package Analysis

| **Functionality** | **File** | **Description** |
| --- | --- | --- |
| Module and Function Imports | getReplacements.py | Imports the pandas library for data manipulation and the addDosageInfo function from DosageInfo.py for adding dosage information to datasets. |
| Data Classification and Filtering | getReplacements.py | Defines a function to classify medication forms (Liquid, Solid, etc.) and filters datasets based on similarity and form. |
| Dosage Information Enrichment | getReplacements.py | Applies the addDosageInfo function from DosageInfo.py to input and data datasets to enrich them with standardized dosage information. |
| Specialized Data Filtering | getReplacements.py | Further filters the data based on the form of the medication (solid or non-solid) and other criteria like dosage size and unit compatibility. |
| Dosage Extraction | DosageInfo.py | Defines addDosageInfo function containing an inner function extractDosage which extracts numerical dosage values from a string field and assigns them to new columns in the dataset for each dosage unit. |
| Total Dosage Calculation | DosageInfo.py | Calculates total values for different dosage units (e.g., total milligrams, total milliliters) based on the extracted data to standardize dosage information across the dataset. |
| Dataset Cleaning | DosageInfo.py | Removes original measurement columns after totalizing values to simplify the dataset, leaving only relevant, consolidated dosage information alongside other non-dosage-related data. |
| Final Processing and Return | Both | In getReplacements.py, returns the final filtered dataset representing potential medication replacements. In DosageInfo.py, returns the input dataset updated with cleaned and standardized dosage information for further processing. |

# 

# 

# 

# 

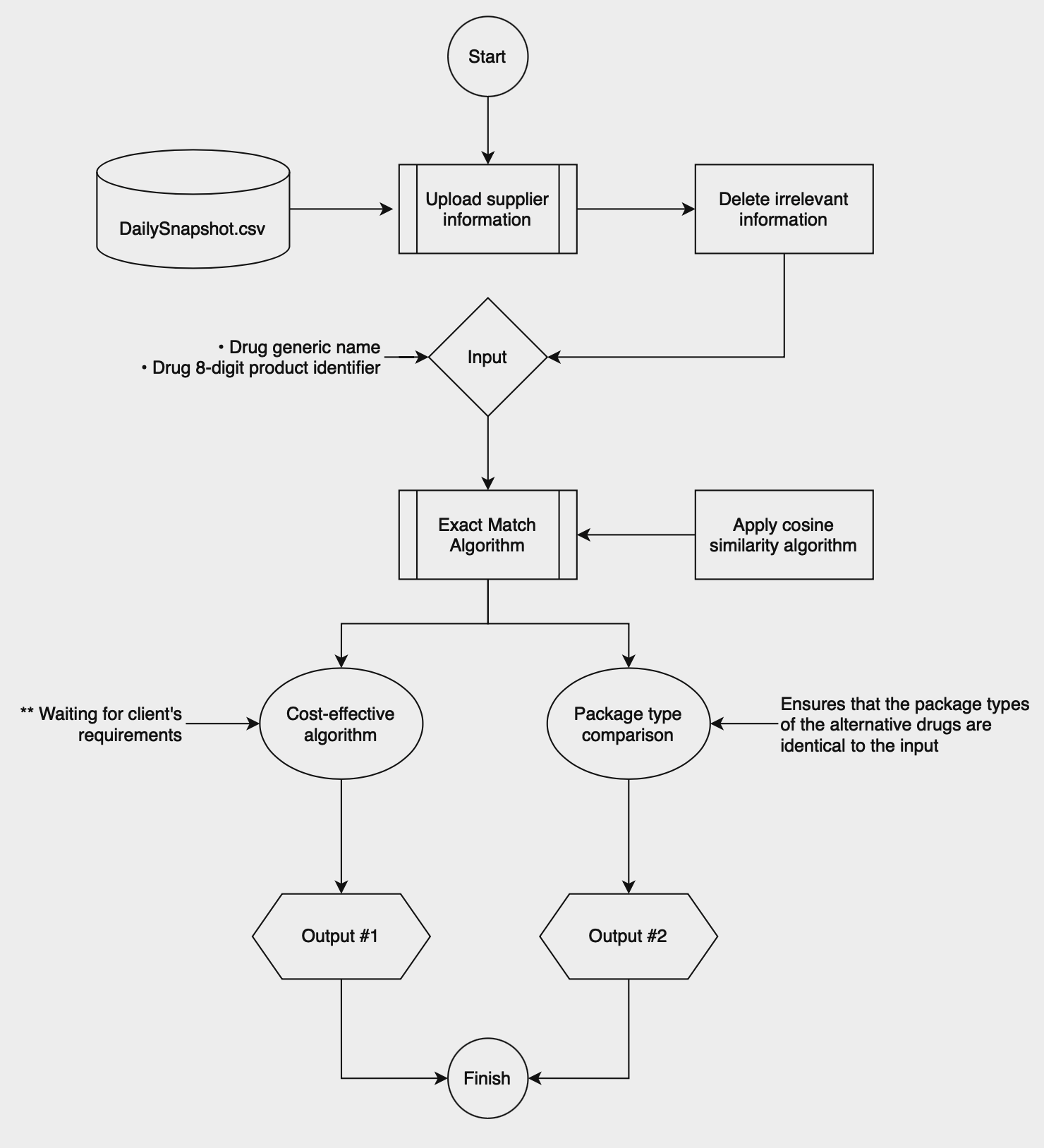
# API Implementation

# 

| **Feature** | **Description** |
| --- | --- |
| Hashing and Verifying Passwords | Uses bcrypt to securely hash passwords before storage and verify stored password hashes against user-provided passwords. |
| User CRUD Operations | Supports creating, retrieving (by name or ID), and deleting users in the database. |
| Data Validation Schemas | Utilizes Pydantic models to validate data for user operations (create, get, delete), login, logout, and file management (upload, get, delete) in API requests. |
| API Route Definitions | Defines FastAPI routes for user and file management operations, including database connectivity checks and session validation for protected routes. |
| Session Management | Generates, inserts, validates, and deletes session cookies for managing user login sessions, ensuring secure access and operation within the application. |
| Database Connection Management | Establishes and manages database connections using asyncpg, including setup and teardown of the connection pool for handling queries. |
| File Management Operations | (Assumed based on schemas) Allows for uploading, retrieving, and deleting files, likely involving operations similar to user management but for file data. |

# 

## 2.1 Visualization



*Figure A. Prototype Flowchart*

**3. Test Methodology**

3.1 Pre-Testing Setup Procedure

1. Database Preparation:
   1. Login to AWS with admin credentials. Find the PostgreSQL database created to store user information. Ensure it is up and running, security protocols are set to accept incoming connections, and it is made ‘publicly available.
2. Install Python Modules
   1. Ensure that essential Python modules are installed. This includes:
      1. **Scikit-learn:** A machine learning library used for data mining and data analysis.
      2. **Pandas:** A library providing high-performance, easy-to-use data structures and analysis tools.
      3. **Re (Regex):** A module for string searching and manipulation using regular expressions.
      4. **Numpy:** A library for large, multi-dimensional arrays and matrices and a collection of mathematical functions to operate on these arrays.
      5. And more!

These modules can be installed using pip, the Python package manager, with the command pip install scikit-learn pandas numpy regex.

1. Data and Script Preparation:
   1. Place the client's data file, 'Daily Snapshot.csv', in the newly created directory. This file contains the data set required for the testing.
2. Environment Verification:
   1. Ensure API is up and running and that it respondes in a timely manner.

By following these enhanced setup procedures, the testing environment will be optimally prepared, ensuring a smooth and efficient testing process.

3.2 Testing Procedure

1. Using Postman send a request to the root of the API to see if it is active
2. Using Postman send a request to the API to create a new user in the database using the “/user/create” path and the proper parameters (username, password, center, permission, employeeid, islocked). Then confirm that the new user has been properly added to the database.
3. Using Postman send a request to the API to login the new user using the “/session/login” path and the proper parameters (username, password). Then save the session cookie that is returned.
4. Using Postman send a request to the API to check the validity of the session cookie using the “/session/is\_session\_cookie\_valid” path and the proper parameters (session\_cookie).
5. After making sure that the API environment is working properly, select a 9 drugs included in the CSV given by our client.
6. For each selected drug, using Postman a request to the API to get drug replacements using the path “/drugs/replacements” and the proper parameters (session\_cookie, drugid). Copy paste the output into your jupiter notebook to have a better visualization of the output.

**Measurable Criteria**

| **Data Cleaning and Preprocessing Accuracy** | Evaluate the accuracy of the data cleaning and preprocessing steps in the Python code. |
| --- | --- |
| **Efficiency in Data Transformation** | Measure the time taken to execute data transformation steps, including splitting descriptions and reordering columns. |
| **Similarity Score Precision** | Assess the precision of TF-IDF vectorization and cosine similarity calculations in identifying similar drug forms and sizes. |
| **Filtering and Sorting Capability** | Determine the efficiency and accuracy of the system in filtering and sorting pharmaceutical products based on similarity scores. |
| **Scalability with Large Datasets** | Test the system's performance with increasingly large datasets to assess scalability. |
| **Response Time in Data Analysis** | Measure the response time from data input to the display of results. |
| **Feedback and Iterative Improvement** | Implement a system to collect feedback on the prototype’s performance and use this feedback for iterative improvements. |

**Score sheet**

The following active ingredient-drug code combinations will be tested:

| **Description** | **Drug Code** |
| --- | --- |
| entecavir ORAL TABLET 0.5 MG | 10000077 |
| entecavir ORAL TABLET 1 MG | 10000082 |
| METHADONE HCL 10 MG/5 ML SOL 5 | 10083467 |
| METHADONE HCL 5 MG/5 ML SOL 50 | 10083468 |
| SAXAGLIPTIN 2.5 MG TAB 30 | 10281732 |
| SAXAGLIPTIN 5 MG TAB 30 | 10281664 |
| FLUPHENZINE HCL 2.5 MG TAB 100 | 10262230 |
| FLUPHENZINE HCL 5 MG TAB 100 ( | 10262212 |
| ISOSORBIDE DINITRATE 5 MG TAB | 10208756 |
| ISOSORBIDE DINITRATE 20 MG TAB | 10208855 |

| **Score sheet** | **Achieved?** |
| --- | --- |
| 1. Database is able to run and connect to the API system |  |
| 1. API calls add, remove, edit users |  |
| 1. User login and logout work as intended |  |
| 1. Updates on the database are shown in real time and accurately |  |
| 1. Package effective algorithm returns the appropriate drugs |  |

**4. Conclusion**

The current prototype demonstrates capabilities in integrating a robust database with an API system, facilitating seamless user interactions including addition, removal, and editing of user profiles. Another key feature of this prototype is its real-time database updates, which guarantees that users have access to the latest information without any discrepancies. Moreover, the package analysis algorithm introduces a new layer of customization and efficiency. It allows users to sort drugs based on package types and dosage levels, enabling a more tailored approach to drug selection.

**4.1 Future milestones**

#### 4.1.1 Platform Integration

* Objective: Develop and launch a dedicated platform to feature our product, enhancing accessibility and user engagement.
* Action Plan: Design a user-friendly interface that accommodates all the current functionalities and allows for future expansions. Ensure the platform supports various devices and browsers for maximum reach.

#### 4.1.2 Real-Time Supplier Data

* Objective: Achieve real-time integration with supplier databases to provide up-to-the-minute data on pharmaceuticals, including availability and pricing.
* Action Plan: Establish partnerships with suppliers for API access. Develop a system for continuous data synchronization and implement fail-safes for data accuracy.

#### 4.1.3 Cost-Effectiveness Algorithm

* Objective: Introduce a cost-effectiveness algorithm to assist users in making financially viable choices without compromising on quality.
* Action Plan: Research and develop an algorithm that considers drug efficacy, cost, and user health insurance coverage. Test the algorithm for accuracy and user satisfaction.

#### 4.1.4 Collaboration with Procurement Team

* Objective: Ensure the highest accuracy in drug procurement and inventory management through regular checks and balances with the procurement team.
* Action Plan: Establish a routine review process with the procurement team to verify the accuracy of drug information and availability. Use feedback to refine database and algorithm accuracy.

Appendix

* GitHub repository:

<https://github.com/BU-EC463>