Incremental Project: Life Prognosis & Management Tool

Problem Statement

Human Immunodeficiency Virus (HIV), according to the World Health Organization, is an infection that attacks the body's immune system. It leads to acquired immunodeficiency syndrome (AIDS) as the disease attacks the human immune system, weakening resilience to disease.

The introduction of Antiretroviral Therapy (ART) has helped to improve the lives of countless patients. However, HIV/AIDS negatively affects how long citizens live.

As such, we need to be able to provide accurate information to citizens to encourage awareness, early treatment, and how they are impacted regarding their expected lifespan.

This problem is to be solved with a simple command-line utility, that will collect patient information, and provide them with how long their expected lifetime is. Given that we are handling *user health data*, this system should protect user information from other patients, but should allow administrators to export data and aggregate statistics for decision-making purposes.

Objectives

At the end of this project, the student will be able to

1. Design Software Engineering Solutions that require high level Planning and implementation
2. Use OOP concepts to solve medium to complex level problems using Java
3. Use Java to solve medium level mathematical operations
4. Use Java & bash hand in hand to solve Unix problems
5. Use bash to manipulate Unix IO files
6. Work in a team of developers to solve a common challenge
7. Use Git to control the versioning of the codebase

Tech stack

1. Logic:
    a. **Java**: Core Java, no framework
    b. **Bash** to deal with files, cleaning and other related jobs
2. Data store: text files accessed using bash scripts
3. Version control and collaboration: **Git**

System Features

User Interface

The application is to be created as a USSD-like command-line application. In short, build it as a terminal application. The User Experience needs to be impeccable, and all possible scenarios tested.

Authentication & user management

This application requires two users:

1. Patient: A patient is a normal user who will be providing their health data so that the application can estimate their HIV survival rate.
2. Admin: The admin user is a privileged user who can do multiple operations on data like download/export, and user management.

Background information collection and management

The application should allow patients to provide the following data, and the application allows for adding and updating user-provided information, including:

1. First name
2. Last name
3. Date of birth
4. Whether they are HIV positive or not
5. If yes to above
    a. The time they caught the virus (same as diagnosis date)
    b. If they are on ART (Antiretroviral therapy) drugs
    c. When they began taking ART drugs (if taking)
6. Country of residence (by Country ISO Code)

Lifespan Computation

This application accepts the record of a patient and predicts how much time they have left (or remaining lifespan).

The lifespan of a user is computed as follows:

1. The lifespan of the average person is provided per country.
2. The survival rate is **5 years** if **not** on ART drugs: this means if there are no drugs, the patient will die in the 5th year after catching the virus.
3. Starting drugs will give a chance to survive 90% of the remaining time before hitting their country's lifespan limit if the patient starts the drugs **in the same year** they caught the virus.
4. HIV reduces the remaining lifespan by 10% for each additional year the patient delays starting the drugs. An example, assuming in Rwanda the lifespan is 69 years, if a 20-year-old person catches the virus and starts taking the ART drugs at 22, the remaining time will reduce to (69 - 20) * 90% * 90% * 90% because they delayed the starting the drugs.
   Explanation:
   a. The average Rwandan lives to 69 years, and so the patient was expected to have (69-20), or 49 remaining years.
   b. The patient contracted HIV, limiting their lifespan to 90% of the remaining time
   c. Since the person did not take ART drugs for two years, their remaining lifespan was further cut by 90% for each of those years.

   Note:
   a. while counting years, we are counting full years. Round up to the next full year.
   b. The above mathematical model is not an accurate medical model, it is based on a mathematical assumption

Data exporting and aggregation

The application should allow administrator users to export user information as-is for migration purposes, as necessary. The application should also be able to produce aggregated data such as various statistics that can be used for analytical and decision-making processes.

System Restrictions

You must strictly work within the following restrictions in the implementation of this project. These stated restrictions are non-negotiable, and must be adhered to

1. You **must not read** any file in Java.
   Java **cannot** access the file system to read in information. It must rely on bash for this.

2. You are not allowed to write any updated user information to files in Java.
   You are, however, allowed to use Java only when exporting files in week 3.

Deliverables: Problem Segmentation [Details for problem-solving]

Module / Week 1: Solution Design & User Management: Week 1

During this week

1. The Team of students needs to understand the problem and agree on steps to solve it, any misunderstanding should be addressed to the TAs. Ensure you are making no assumptions.

2. The team needs to come up with a clear design of the solution, including:
   a. The use case, activity, and class diagrams design of your solution
   b. Descriptions of all files that will be used as data store

3. The team needs to work on the User Management requirements: This module includes

   a. **Registration**:
      - Admin onboarding / initiating a registration by proving the email of the user.
      - The UUID code will be generated that the user should use while completing the rest of the registration.

        Note: Each team should come up with **user-store.txt** that contains the users. Populate the first admin user that will be used to start all other registration processes. Beware of not storing a plain password but a hashed password. You may use tools such as OpenSSL or other hashing utilities, but you may not install any additional packages aside those that come with a base Ubuntu 24.04 installation.

Procedures & steps for Registration

- The admin provides the email of the user, and a code is generated, the user primary info (email & UUID code) stored in the **user-store.txt** file, the role is by default “Patient”
- The Patient then complete the registration, provides the UUID code and:
    1. First Name
    2. Last Name
    3. Date of birth
    4. Whether they have HIV or not
        a. If yes, the time they caught the virus (same as diagnosis date)
        b. If they are on ART drugs
            i. If yes, the time they started the ART
    5. Country of residence (ISO Code)
    6. Password (must be hashed)

b. **Login**:

The user can log in with email & password: Once completed, logging in will give the user the ability to

- View Profile Data (Week 2 deliverable)
- Update profile data (Week 2 deliverable)
- download the iCalendar (.ics) schedule of the demise (bonus module in week 3)

How?

❖ Use OOP to implement this user management system.

❖ The user provides emails & password

  ➢ While **logging in**, the system should know the user type without the user manually specifying it (by the accurate use of Polymorphism)

❖ Java login method calls a bash script that checks if the user with these credentials exists, and then returns the data for the information needed to instantiate the relevant User object to Java.

OOP & Polymorphism requirement

This module needs to follow a strict polymorphic concept

a. Create an Enum defining the possible user roles in the application
b. Create an **interface** / **abstract class** named User

   All user fields that all users have in common should be here, that is:

   Attributes
   - First name
   - Last name
   - email
   - Password
   - <Any other information you may need>

c. Create a concrete/child class named Admin extending / implementing User
   Does not have any specific attributes.

d. Create a concrete/child class named Patient extending / implementing User
   Attributes

   1. Date of birth
   2. Whether they have HIV or not
   3. If yes, the time they caught the virus (same ad diagnosis date)
   4. If they are on ART drugs
   5. If yes, the time they started the ART
   6. Country of residence (ISO)

What to use Bash for

1. Create a text file to store users (e.g., user-store.txt) that has an initial admin. This is the same file that the user will be updating every time they update their information.

2. Create a bash script that manage users (e.g., user-manager.sh) (you may create multiple scripts, with as many functions as you need). Your script(s) should be able to support:

- Initiating registration by admin
- Registration by Patient
- Login by a user

Deliverables

- Admin should be able to initiate the Patient registration
- Patient should be able to complete their registration
- Patient should be able to log in
- Admin should be able to download 2 empty CSV files

    1. The file that will contain all users (empty now)
    2. A CSV file that will contain analytics (empty now)

Module / Week 2: Profile Updates/ Data Exports

By now, we already have an application where:

- Admin can initiate the registration
- A Patient can complete the registration
- A user can log in
- Admin can download empty CSV files

We now need to make sure that:

- The Patient can View their Profile Data
- The Patient can Update their profile Data
- The app should calculate the HIV survival rate
- The admin can download real user data in a CSV file

Deliverables Details

View Profile data

Implement the Patient method to retrieve profile data.

- This should return real data to the patient (profile information, and their **computed lifespan**)
- This should only be accessible by patients, and data access should be limited to the logged in patient

Update Profile data

Implement the Patient method to update profile data.

- This should allow a patient to edit their information
- This feature should only allow a **patient** to access their data

Download Users data (partial)

Now implement the admin method to download users' data

- This should return real user data from the system. This method should **only** be accessible by **admin users**.
- You only need to return a single CSV file containing the patient data. This will be improved in Week 3

Module / Week 3: Profile Updates / Data Exports

Deliverables

- A complete command line interface to interact with the application
- Export of patient data and aggregate analytics of all patients
- Testing for the whole application functionality

1. A complete command line interface

   User interaction with the application is through a command line interface. This should be an easy to navigate user journey from login/registration to exporting user data and analytics.

   - Menu items to enable navigation, apart from options available at any point in the user journey, the menu should also have options to go back, quit the app, go to their profile, and logout.
   - Users should have different menu options depending on their roles.

2. Analytics

   Admin should be able to download, amongst other data, aggregate analytics/statistics for the entire patient population.

   To accomplish this, you will augment the download feature in Week 2 to export two CSV files, the one with patient data already done, and a second file including statistics

   Note: have some data for 500 users in your database, and classify people based on their survival rate by providing some metrics like

   1. Average
   2. Median
   3. Percentile for some survival rates
   4. More statistical points you can add. [optional]

   The results of these statistics should be in the second CSV file. The first CSV file should contain all patient records. The second file, named statistics, contains all the statistics requested above.

3. Testing

   - A short demonstration video showing that the system was tested throughout all the user journeys.