HW 6 | Appointment Reservation System

Objectives: To gain experience with database application development, and learn how to use SQL from within Java via JDBC.

Note: Changes made after the spec release appear in *blue*.

Due dates:

- Setup: due **March 6th, 2025 at 11:59pm**
- Part 1: due **March 10th, 2025 at 11:59pm**
- Part 2: due **March 17th, 2025 at 11:59pm**

Contents:

Introduction
Setup (deliverables)
Homework Requirements
Part 1 (deliverables)
Part 2 (deliverables)
Grading
Optional: Extra Credit
Optional: Connecting to Amazon Aurora

Introduction

A common type of application that connects to a database is a reservation system, where users schedule time slots for a centralized resource. In this assignment you will program part of an appointment scheduler for vaccinations, where the users are patients and caregivers keeping track of vaccine stock and appointments.

This application runs in the command-line terminal and connects to a local SQLite database that you'll create. Optionally, you can connect to a remote Amazon Aurora database server—we encourage you to try both!

You will have two main tasks:

- Complete the design of the database schema with an E/R diagram and create table statements

- Implement the code that stores Patient information, and lets users interactively schedule their vaccine appointments. We have implemented the code that caregivers use to manage the appointment slots and inventory, which will be a useful reference for you. The implementation is broken up into two milestones, part 1 and part 2, described below.

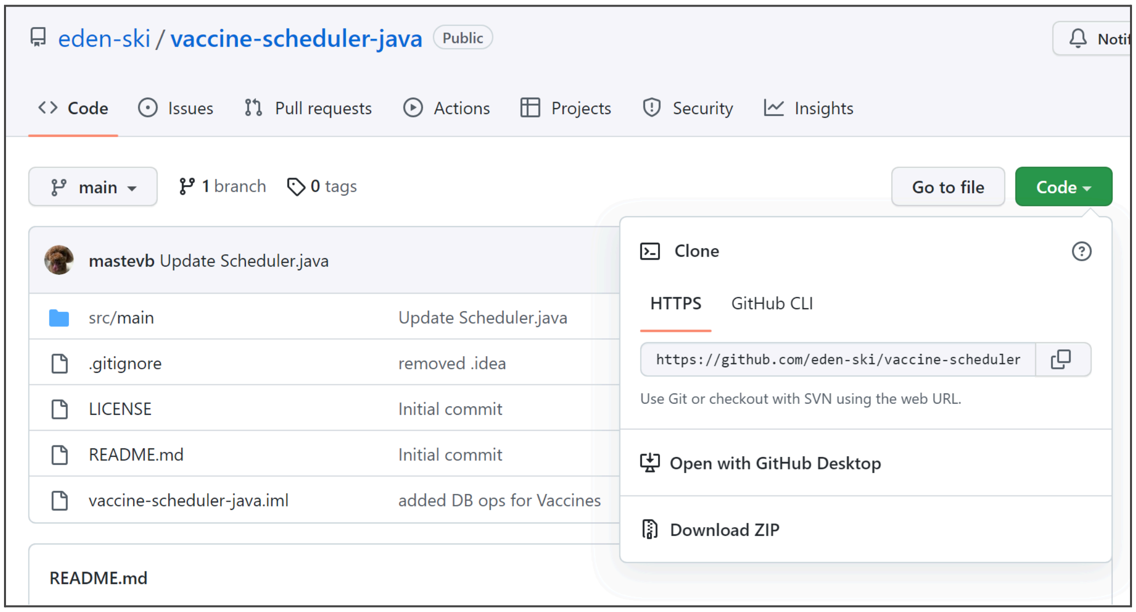
Be Careful: This homework requires writing a non-trivial amount of code; our solution is about 600 lines, including the starter code. It will take SIGNIFICANTLY more time than your previous assignments.

Setup

Make sure to finish this part of the assignment and upload your setup verification of step 2.4 for the first 5 points!

2.1 Clone the starter code

1. Navigate to the Github repository hosting the starter code: https://github.com/avk5/vaccine-scheduler-java (As you continue to work on the assignment, DO NOT manipulate the internal project structure of the assignment. This will save you some headache when it comes time to submit your code)
2. Click on the green button "Code" and select "Download ZIP" from the drop-down menu.



mastevb Update Scheduler.java

src/main	Update Scheduler.java
.gitignore	removed .idea
LICENSE	Initial commit
README.md	Initial commit
vaccine-scheduler-java.iml	added DB ops for Vaccines

```
https://github.com/eden-ski/vaccine-scheduler
```

Use Git or checkout with SVN using the web URL.

3. Once your download completes, unzip the ZIP file and retrieve the starter code.

2.2 Read through the starter code

We created the important folders and files you will be using to build your application:

- src.main.scheduler/:

    Scheduler.java:

        - This is the main entry point to the command-line interface application. Once you compile and run **Scheduler.java**, you should be able to **interact with the application.**
    - db/:
        - This is a folder holding all of the important components related to your database.
        - **ConnectionManager.java**: This is a wrapper class for **connecting to the database**. Read more in 2.3.4.
    - model/:
        - This is a folder holding all the class files for your data model.
        - You should implement all classes for your data model (e.g., patients, caregivers) in this folder. We have created implementations for Caregiver and Vaccines, and you need to complete the Patient class (which can heavily borrow from Caregiver). **Feel free to define more classes or change our implementation if you want!**
        - Our implementation uses a design pattern called the builder pattern. Refer to this article for more detail.
- src.main.resources/:
    - aurora/:
        - Ignore this folder unless you're doing the optional Aurora connection
        - **aurora-connection-manager.txt**: This is the code for an alternate connection manager class, which allows you to connect to an Amazon Aurora database instead of a SQLite database.
        - **aurora-create.sql**: SQL create statements for your Aurora tables, we have included the create table code for our implementation. You should copy, paste, and run this code (along with all other create table statements).
    - sqlite/:
        - **create.sql**: SQL create statements for your SQLite tables, we have included the create table code for our implementation. You should copy, paste, and run this code (along with all other create table statements).

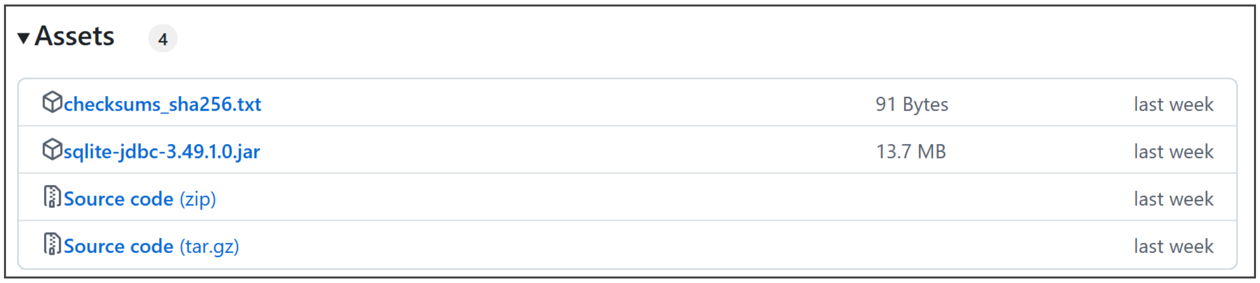
2.3 Configure your database connection

Note: if you see an error in IntelliJ saying “Project JDK is not defined”, follow the steps here to set up an SDK.

2.3.1 Installing dependencies

Our application relies on a few dependencies and external packages. You’ll need to install those dependencies to complete this assignment.

First, we need the JDBC driver to allow our Java application to connect to a SQLite database. The link can be found here. Click the link to download **sqlite-jdbc-3.49.1.0.jar**.



▾Assets 4

checksums_sha256.txt	91 Bytes	last week
sqlite-jdbc-3.49.1.0.jar	13.7 MB	last week
Source code (zip)		last week
Source code (tar.gz)		last week

We recommend that you create the folder vaccine-scheduler-java-main/lib. Copy the jar file and place it into the vaccine-scheduler-java-main/lib folder of your cloned project.

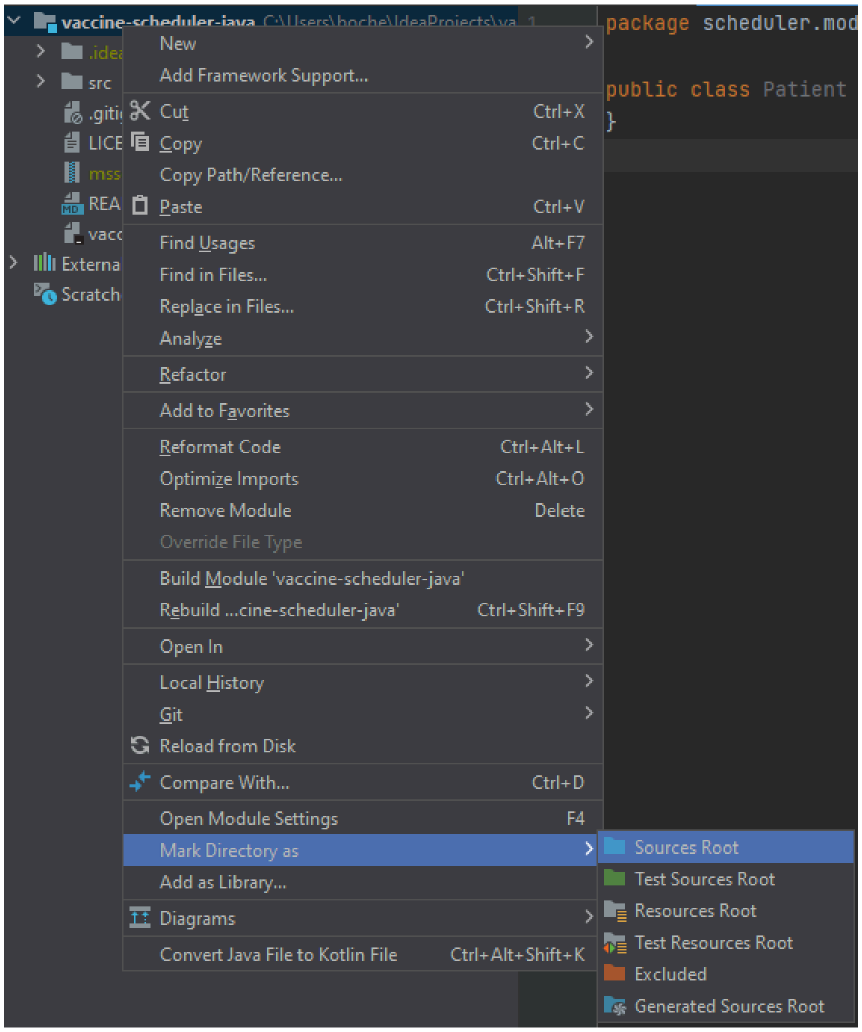
Note: If you need to download another Java version, follow this guide to change the Java version for your IntelliJ project.

2.3.2 Setting up IntelliJ

Note: We only officially support IntelliJ for this assignment. You may use another IDE, but please make sure it works before turning in your assignment!

1. Now you will open your cloned directory using your IntelliJ so that your vaccine-scheduler-java is the root directory

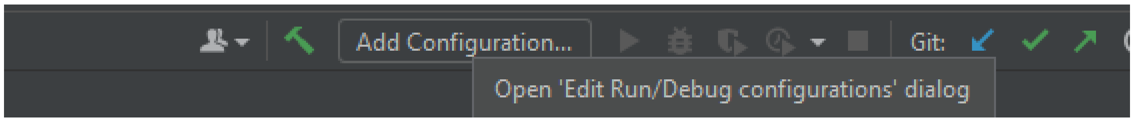
2. You will also need to mark vaccine-scheduler-java-main as the source root by right-clicking on the folder in IntelliJ and selecting “Mark Directory As” -> “Sources Root”.



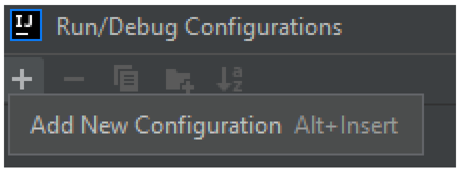
```
package scheduler.mod

public class Patient
}
```

3. Now Add Configuration
   a. Select the Add Configuration option.

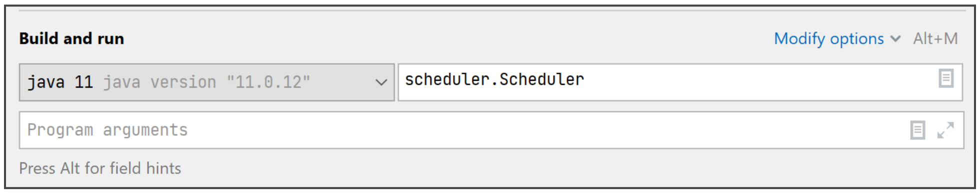


   b. Select “Add new Configuration”.



   c. Select “Application” in the list of options. Make sure you have a name for this configuration, and in the main class box write *scheduler.Scheduler* (this lets

IntelliJ know where main is located).

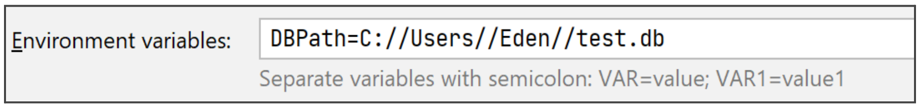

Build and run
java 11 java version "11.0.12"
scheduler.Scheduler
Program arguments

If *scheduler.Scheduler* is highlighted red and can't be located, 1) exit out of this screen and go to File -> Invalidate Caches / Restart and repeat the following steps, and 2) check that you have marked the source root mentioned in step 2.

d. You should include your environmental variables for your database connection in the format below:

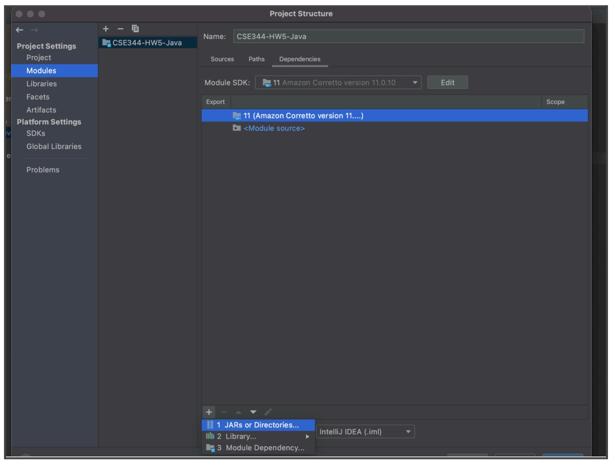
```
DBPath=
```

E.g., mine looks like:


Environment variables: DBPath=C://Users//Eden//test.db

I created a database for this project called test.db in the directory C:/Users/Eden/.

4. Next, we need to ensure that our JDBC drivers are loaded into our environment. To do this, select File -> Project Structure to open this page and go to Dependencies.


Name: CSE344-HW5-Java
Module SDK: 11 Amazon Corretto version 11.0.10

Select Modules under project settings, make sure your current configuration is selected, and then press the + and select Jars or Directories. This should prompt you to select your JDBC drivers, and once you select them they should be added to your environment. Then make sure to apply your changes.

2.3.3 Working with the connection manager

Note: There is nothing you need to do in this step.

In **scheduler.db.ConnectionManager.java**, we have defined a wrapper class to help you instantiate the connection to your SQLite database. We recommend reading about Java PreparedStatements for retrieving and updating information in your database.
Here’s an example of using ConnectionManager.

```java
// instantiating a connection manager class
ConnectionManager cm = new ConnectionManager();
Connection con = cm.createConnection();

// example 1: getting all records in the vaccine table
PreparedStatement getAllVaccines = con.prepareStatement("SELECT * FROM
vaccines");
ResultSet rs = getAllVaccines.executeQuery();
while (rs.next()) {
   System.out.println("id: " + rs.getLong(1) + ", available_doses: " +
rs.getInt(2) +
           ", name: " + rs.getString(3) + ", required_doses: " +
rs.getString(4));
}

// example 2: getting all records where the name matches “Pfizer”
PreparedStatement getPfizer = con.prepareStatement("SELECT * FROM vaccine
WHERE name = ?");
getPfizer.setString(1, "Pfizer");
ResultSet rss = getPfizer.executeQuery();
while (rss.next()) {
   System.out.println("id: " + rss.getLong(1) + ", available_doses: " +
rss.getInt(2) +
           ", name: " + rss.getString(3) + ", required_doses: " +
rss.getString(4));
}
```

Helpful resources on writing Java prepared statements:
https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html

2.4 Verify your setup

Once you're done with everything, **try to run the program and you should see the following output:**

```
Welcome to the COVID-19 Vaccine Reservation Scheduling Application!
*** Please enter one of the following commands ***
> create_patient <username> <password>
> create_caregiver <username> <password>
> login_patient <username> <password>
> login_caregiver <username> <password>
> search_caregiver_schedule <date>
> reserve <date> <vaccine>
> upload_availability <date>
> cancel <appointment_id>
> add_doses <vaccine> <number>
> show_appointments
> logout
> quit
```

If you can see the list of options above, congratulations! You have verified your local setup.

Next, to verify that you have set up your database connection correctly, try to create a caregiver with the command "create_caregiver <username> <password>". Make sure you have created the tables in your SQLite database before executing this command.

To verify you have done the setup, take a screenshot or phone picture of this screen on your computer, along with your created caregiver on SQLite (a simple SELECT showing that you have created a caregiver would work), and upload to gradescope for 5 points.

Deliverables for Setup

Upload to Gradescope a screenshot or phone picture of the welcome prompt on your computer and the successful “create_caregiver <username> <password>” command, along with your created caregiver in SQLite.

Requirements

Your assignment is to build a vaccine scheduling application that can be deployed by hospitals or clinics and supports interaction with users through the terminal/command-line interface. In the real world it is unlikely that users would be using the command line terminal instead of a GUI, but all of the application logic would remain the same. For simplicity of programming, we use the command line terminal as our user interface for this assignment.

We need the following entity sets in our database schema design (hint: you should probably be defining your class files based on this!):

- **Patients:** these are customers that want to receive the vaccine.
- **Caregivers:** these are employees of the health organization administering the vaccines.
- **Vaccines:** these are vaccine doses in the health organization's inventory of medical supplies that are on hand and ready to be given to the patients.
- **Reservations:** these are appointments that patients will book.

In this assignment, you will need to:

- Complete the design of the database schema, with an E/R diagram and table statements **(Part 1)**;
- Implement the missing functionality from the application **(Part 1 & Part 2)**

A few things to note:

- You should handle invalid inputs gracefully. For example, if the user types a command that doesn't exist, it is bad to immediately terminate the program. **A better design would be to give the user some feedback and allow them to re-type the command. Points will be taken off if the program terminates immediately after receiving invalid input.** While you don't have to consider all possible inputs, error handling for common errors (e.g., missing information, wrong spelling) should be considered.
- We will be using an autograder to grade your implementation, so make sure that your output matches the provided specifications **exactly**. In order to check this we will allow for you to submit as many times as you would like before the due date without penalty.

1.3 How to handle passwords

You should never directly store any password in the database. Instead, we'll be using a technique called salting and hashing. In cryptography, salting hashes refer to adding random data to the input of a hash function to guarantee a unique output. We will store the salted password hash and the salt itself to avoid storing passwords in plain text. Use the following code snippet as a template for computing the hash given a password string:

```java
// Generate a random cryptographic salt
SecureRandom random = new SecureRandom();
byte[] salt = new byte[16];
random.nextBytes(salt);

// Specify the hash parameters
// HASH_STRENGTH and KEY_LENGTH have been defined for you in the starter
code
KeySpec spec = new PBEKeySpec(password.toCharArray(), salt, HASH_STRENGTH,
KEY_LENGTH);

// Generate the hash
SecretKeyFactory factory = null;
byte[] hash = null;
try {
  factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
  hash = factory.generateSecret(spec).getEncoded();
} catch (NoSuchAlgorithmException | InvalidKeySpecException ex) {
  throw new IllegalStateException();
}
```

Part 1

Design

You will first need to work on the design of your database application. Before you begin, please carefully read the assignment specification (including Part 2) and the starter code, and think about what tables would be required to support the required operations. Once you have an idea of how you want to design your database schema:

- Draw the ER diagram of your design and place it under src.main.resources (design.pdf).
- Write the create table statements for your design, create the tables in SQLite, and save the code under src.main.resources.sqlite (create.sql).

You will also need to implement the corresponding Java classes of your design. We have implemented Caregiver.java for you, but feel free to change any of the details. You will need the following classes, and you may implement more data models if you feel the necessity:

- Caregiver.java: data model for caregivers (implemented for you.)
- Vaccine.java: data model for vaccines (implemented for you.)
- Patient.java: data model for patients.
  - You will implement this class, it can be mostly based on Caregiver.java

Implementation

Congratulations! You're now ready to implement your design! For Part 1, you will need to implement the following functionalities. We have implemented account creation for caregivers as an example for you, please read through our implementation before you begin.

You must print out your results exactly in the format we specify. The "<" and ">" symbols are placeholders for your values and should not be included in the output.

You will need to implement the following operations:

- create_patient <username> <password>
  - Print "Created user <username>" if create was successful.
    - Example: `Created user p1`
  - If the username is already taken, print "Username taken, try again".

    - For all other errors, print “Create patient failed”.
    - **Your output must match exactly. Do not include the “< >” in your output.**
- login_patient <username> <password>
    - Print "Logged in as <username>" if login was successful.
        - Example: `Logged in as p1`
    - If a user is already logged in in the current session, you need to log out first before logging in again. In this case, print “User already logged in, try again”
    - For all other errors, print "Login patient failed"
    - **Your output must match exactly. Do not include the “< >” in your output.**

Deliverables for Part 1

You are free to define any additional files, but Part 1 should require at least the following:

- src.main.resources
    - design.pdf: the design of your database schema.
- src.main.resources.sqlite
    - create.sql: the create statements for your SQLite tables.
- src.main.scheduler.model
    - Caregiver.java: the data model for your caregivers.
    - Patient.java: the data model for your users.
    - Vaccine.java: the data model for vaccines.
    - Any other data models you have created.
- src.main.scheduler
    - Scheduler.java: the main runner for your command-line interface.

Part 2

You will implement the rest of your application in Part 2. For most of the operations mentioned below, your program will need to do some checks to ensure that the appointment can be reserved (e.g., whether the vaccine still has available doses). Again, you do not have to cover all of the unexpected situations, but we do require you to have a reasonable amount of checks (especially the easy ones).

For Part 2, you will need to implement the following operations:

- search_caregiver_schedule <date>
    - Both patients and caregivers can perform this operation.
    - Output the username for the caregivers that are available for the date ordered alphabetically by the username of the caregiver.
    - Then, output the vaccine name and number of available doses for that vaccine separated by a space.
        - For example if we had 3 available caregivers (c1, c2, c3) and 3 available vaccines (covid, flu, hpv), the output of "**search_caregiver_schedule 2023-12-03**" should be:

          ```
          Caregivers:
          c1
          c2
          c3
          Vaccines:
          covid 2
          flu 5
          hpv 3
          ```
        - **Do not include any other formatting (punctuation, titles, etc). The output should look exactly as above.**

        - If there are no available caregivers and no available vaccines (no vaccines have ever been added) your output should be:

          ```
          Caregivers:
          No caregivers available
          Vaccines:
          No vaccines available
          ```
        - If no caregivers are available, then after Caregivers: you should print "No caregivers available"
        - If there are no vaccines, then after Vaccines: you should print "No vaccines available"
    - If no user is logged in, print “Please login first”
    - For all other errors, print "Please try again"
- reserve <date> <vaccine>
    - Patients perform this operation to reserve an appointment.
    - Caregivers can only see a maximum of one patient per day, meaning that if the reservation went through, the caregiver is no longer available for that date.
    - Appointment IDs should start at 1 and be incremented by 1 each time.
    - If there are available caregivers, choose the caregiver by alphabetical order and print “Appointment ID {appointment_id}, Caregiver username {username}”.
        - For example, the output of “**reserve 2023-12-03 covid**” should be:

          ```
          Appointment ID 1, Caregiver username c1
          ```
    - If no caregiver is available, print “No caregiver is available” and return.
    - If not enough vaccine doses are available, print "Not enough available doses" and return.
    - If no user is logged in, print “Please login first” and return.
    - If the current user logged in is not a patient, print “Please login as a patient” and return.
    - For all other errors, print "Please try again".
- show_appointments
    - Output the scheduled appointments for the current user (both patients and caregivers).
    - For caregivers, you should print the appointment ID, vaccine name, date, and patient name. Order by the appointment ID. Separate each attribute with a space.

        - For example, if caregiver c1 is logged in, the output of "**show_appointments**" should look like:

          ```
          1 covid 2023-12-03 p1
          3 flu 2023-12-05 p3
          ```

    - For patients, you should print the appointment ID, vaccine name, date, and caregiver name. Order by the appointment ID. Separate each attribute with a space.
        - For example, if patient p1 is logged in, the output of "**show_appointments**" should look like:

          ```
          1 covid 2023-12-03 c1
          2 hpv 2023-12-04 c2
          ```

    - If no user is logged in, print "Please login first"
    - If no appointments exist for the user print "No appointments scheduled"
    - For all other errors, print "Please try again"
- logout
    - If not logged in, you should print "Please login first". Otherwise, print "Successfully logged out".
    - For all other errors, print "Please try again".

Deliverables for Part 2

When you're finished, please turn in the entire repository by compressing the project folder into a zip file, then uploading it on Gradescope.

Grading

Your grade for this homework will be worth 125 points, divided as:

- Setup (5 points)
  - Finish setup through step 2.4 and upload your verification to Gradescope

- Part 1 (45 points)
  - Design (15 points)

    Your database design including the files design.pdf and create.sql
  - Implementation (30 points)

    Your working implementation of the Part 1 functions:

    create_patient, login_patient

- Part 2 (75 points)
  - Implementation (75 points)

    The remainder of the functions for Patient:
    search_caregiver_schedule, reserve, show_appointments, logout

Additionally, you may receive up to 12 points of extra credit for implementing one of the options below

Optional Extra Credit

You can do either one of the following extra tasks by the final due date for up to 12 extra credit points.

1. **(4 Points)** Add guidelines for strong passwords. In general, it is advisable that all passwords used to access any system should be strong. Add the following check to only allow strong passwords:
   a. At least 8 characters.
   b. A mixture of both uppercase and lowercase letters.
   c. A mixture of letters and numbers.
   d. Inclusion of at least one special character, from “!”, “@”, “#”, “?”.
      i. If the password entered is not strong, print
         1. “Create patient failed, please use a strong password (8+ char, at least one upper and one lower, at least one letter and one number, and at least one special character, from "!", "@", "#", "?")”
         2. “Create caregiver failed, please use a strong password (8+ char, at least one upper and one lower, at least one letter and one number, and at least one special character, from "!", "@", "#", "?")”
2. **(8 Points)** Both caregivers and patients should be able to cancel an existing appointment. Implement the cancel operation for both caregivers and patients. Hint: both the patient’s schedule and the caregiver’s schedule should reflect the change when an appointment is canceled. Regardless of who canceled it, it should update the vaccine table, remove it from the appointments, and add it back as an availability.
   cancel <appointment_id>
   a. Both patients and caregivers can perform this operation.
   b. If the appointment id is valid print “Appointment ID <appointment_id> has been successfully canceled”
   c. If the appointment id is invalid print “Appointment ID <appointment_id> does not exist”
   d. If no user is logged in, print “Please login first”
   e. For all other errors, print "Please try again"

Optional: Connecting to Amazon Aurora

Imagine your vaccine scheduler application is now being deployed in the real world, serving many users across different locations. A local database won't be enough—everyone needs access to shared, up-to-date information, no matter where they are. This is where a cloud database like Amazon Aurora becomes essential!

To make your application scalable and accessible, you can optionally connect it to a remote Amazon Aurora database server. In the following steps, you'll learn how to establish this connection.

Note: The files you'll need are in the **/main/src/resources/aurora** directory.

1. Create an Amazon Aurora database using the Amazon PostgreSQL Setup Guide from homework 3.
    a. Create the tables in **aurora-create.sql** using VSCode or psql, then create the additional tables you created. **Note:** Postgres does not support the BINARY(n) datatype; use BYTEA instead.
2. Download the PostgreSQL JDBC JAR here.
    a. Copy the jar file and place it into the vaccine-scheduler-java-main/lib folder of your cloned project.
    b. Add the jar to the dependencies by editing the project structure.
3. Edit the runtime configuration to add more environmental variables. You should add your environmental variables for your Aurora connection in the format below:

    ```
    Username=; Database=; Password=; Endpoint=
    ```

    The values they equal will be the values you picked when creating your Aurora database.
4. Replace the content of **ConnectionManager.java** with the content of
5. Congrats, you're done! Now, when you interact with the program, data will be persisted in your remote Amazon Aurora database. To see the content of your tables, you can use VSCode or psql.

Note 2: If you choose this option, you still need to have your create.sql implemented so it works with your solution locally.

Common Questions

Q: My IDE is not recognizing my imports, what should I do?
A: You will likely need to set up source root for your IDE. Check the documentation for your specific IDE on how to set up source roots.

Q: I’m getting an error: “Exception in thread "main" java.lang.NullPointerException: Cannot invoke "java.sql.Connection.close()" because "this.con" is null”.
A: This indicates that there is something wrong with your database connection setup. Verify that you have followed our instructions and read the error messages carefully.

Q: I’m getting a DB error when trying to create a caregiver for my setup.
A: Make sure you have created your tables on SQLite.

🎉🎉🎉 **Congrats you made it to the end!** 🎉🎉🎉