HW 6 | Appointment Reservation System

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

Due dates:

- Setup: due **Monday Nov. 22** 11:00pm
- Part 1: no turn-in because of holiday, suggested date of completion Wednesday Nov. 24
- Final submission of part 1 and part 2: due **Friday, Dec. 3, 2021**

Contents:

Introduction
Setup
Homework Requirements
Part 1
Part 2
Grading

Introduction

A common type of application that connects to a database is a reservation system, where users schedule time slots for some 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 will run on the command line terminal, and connect to a database server you create with your Microsoft Azure account.

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.

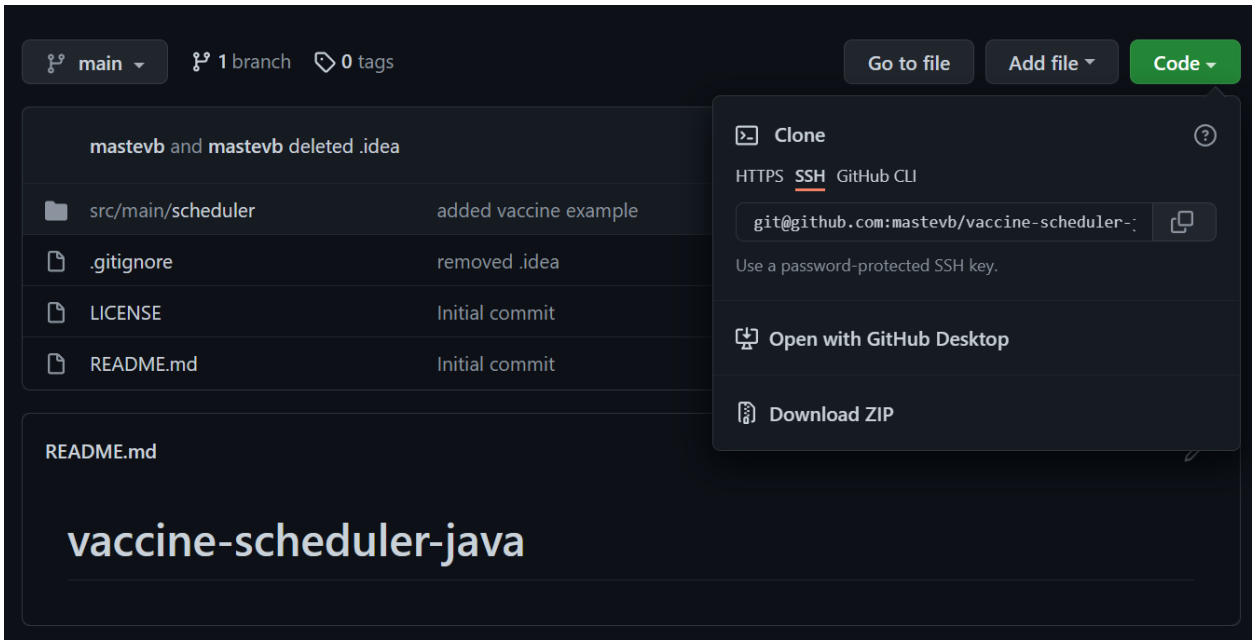
Be Careful: This homework requires writing a non-trivial amount of code; our solution is about 600 lines, including the starter code. It will take SIGNIFICANTLY more time than your previous assignments. We will show you the setup process and coding methods in section and lecture. It is critical that you follow along.

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/mastevb/vaccine-scheduler-java
2. Click on the green button “Code” and select “Download ZIP” from the drop-down menu.



3. Once your download completes, decompress 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.

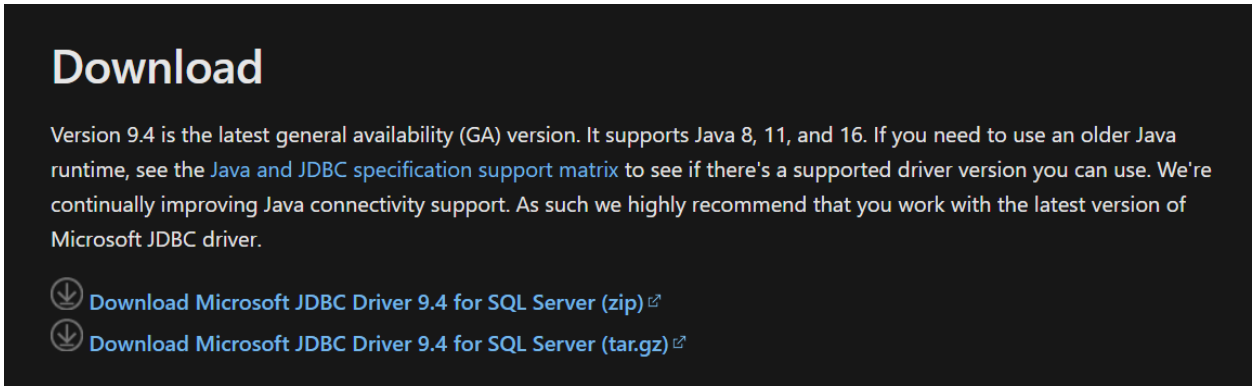
- src.main.resources/
    - **create.sql**: SQL create statements for your tables, we have included the create table code for our implementation. You should copy, paste, and run the code (along with all other create table statements) in your Azure Query Editor.

2.3 Configure your database connection

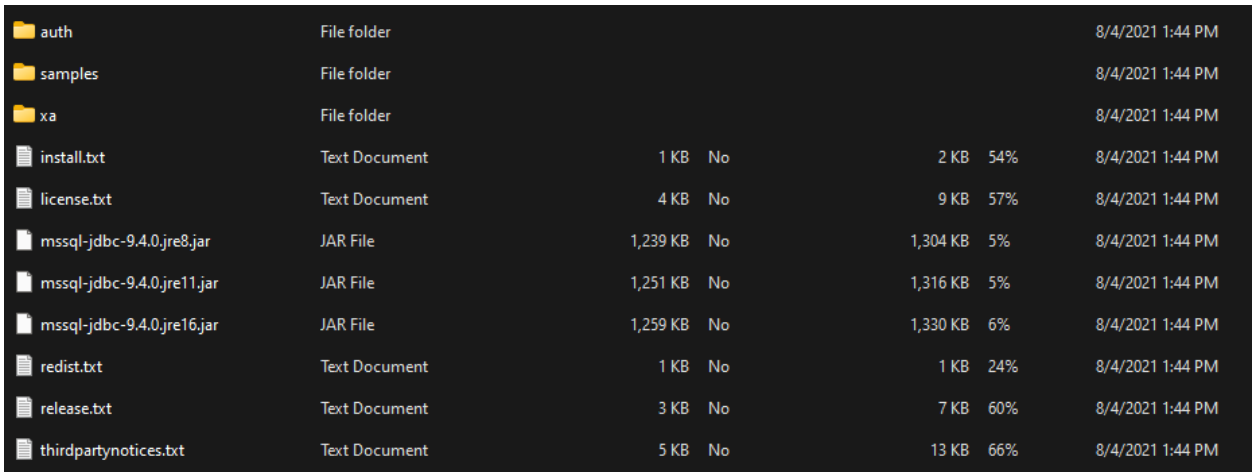
Note: if you see an error in IntelliJ saying “Project JDK is not defined”, follow the steps here to setup an SDK.

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 drivers to allow our Java application to connect to an Azure database. The link can be found here. Download the zip for JDBC 9.4 drivers for SQL server.



The zip will contain various files as listed below.



auth	File folder					8/4/2021 1:44 PM
samples	File folder					8/4/2021 1:44 PM
xa	File folder					8/4/2021 1:44 PM
install.txt	Text Document	1 KB	No	2 KB	54%	8/4/2021 1:44 PM
license.txt	Text Document	4 KB	No	9 KB	57%	8/4/2021 1:44 PM
mssql-jdbc-9.4.0.jre8.jar	JAR File	1,239 KB	No	1,304 KB	5%	8/4/2021 1:44 PM
mssql-jdbc-9.4.0.jre11.jar	JAR File	1,251 KB	No	1,316 KB	5%	8/4/2021 1:44 PM
mssql-jdbc-9.4.0.jre16.jar	JAR File	1,259 KB	No	1,330 KB	6%	8/4/2021 1:44 PM
redist.txt	Text Document	1 KB	No	1 KB	24%	8/4/2021 1:44 PM
release.txt	Text Document	3 KB	No	7 KB	60%	8/4/2021 1:44 PM
thirdpartynotices.txt	Text Document	5 KB	No	13 KB	66%	8/4/2021 1:44 PM

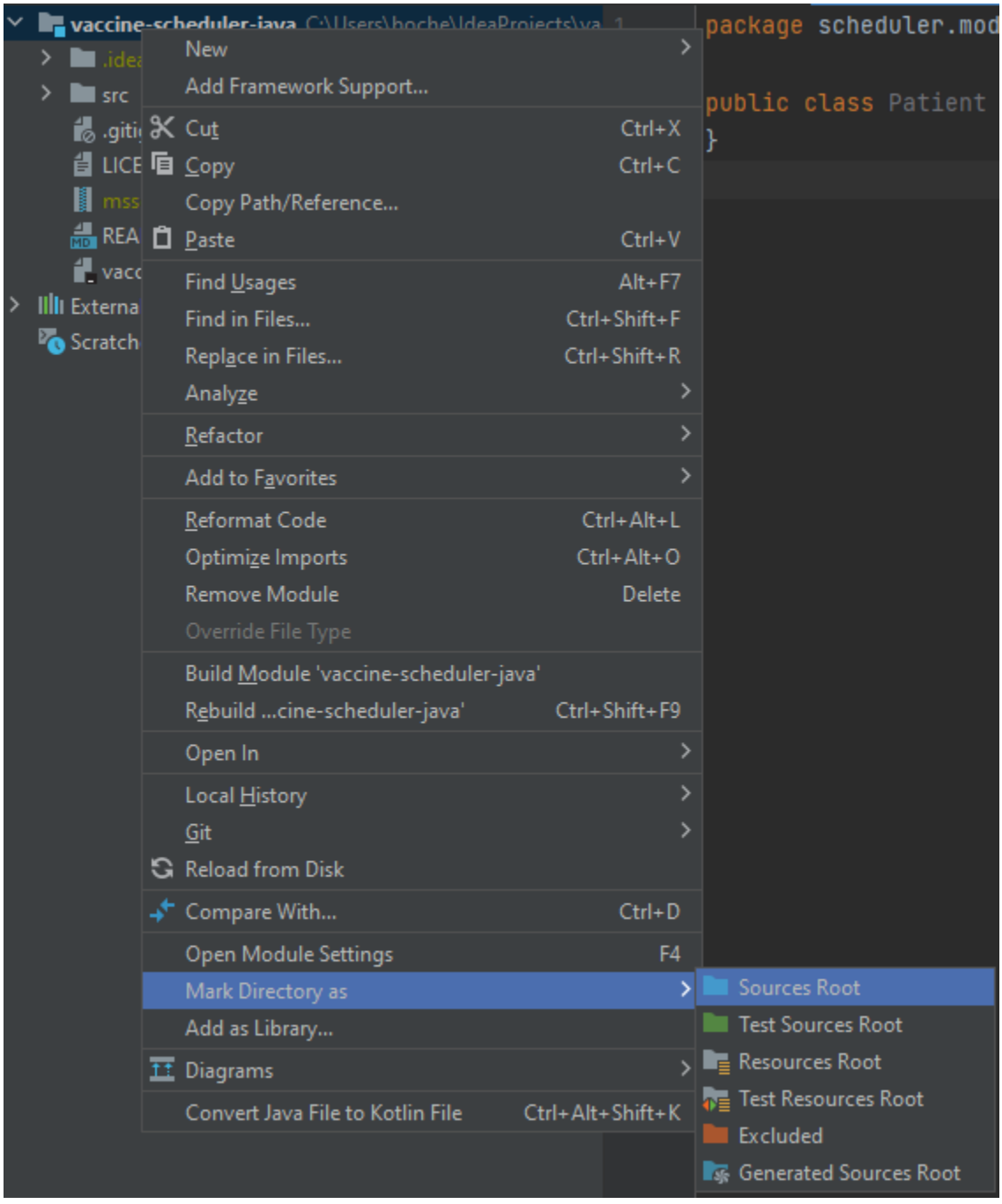
You should use the .jar that has the same JRE as your version of java. To find your java version, open a terminal in IntelliJ and type `java --version`. This should return what java version you have. After you determine what version of JDBC you'll be using, copy the jar file and place it into the root directory of your cloned directory.

Note: **JDBC is only compatible with Java 8, 11, or 16**, so you must be using one of those versions to proceed. If you need to download another Java version, follow this guide to change the Java version for your IntelliJ project.

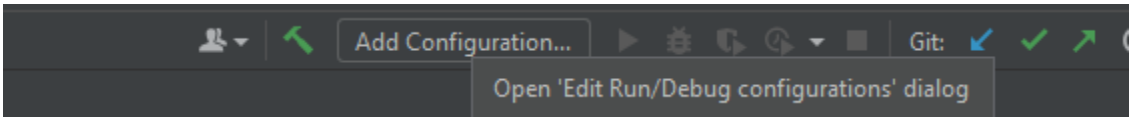
2.3.2 Setting up IntelliJ

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!

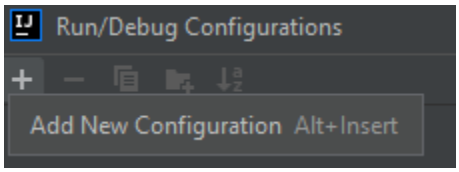
1. Now you will open your cloned directory using your IntelliJ so that your vaccine-scheduler-java is the root directory
2. You will also need to mark vaccine-scheduler-java-main as the source root by right-clicking on the folder in IntelliJ and selecting “Mark Directory As” -> “Sources Root”.



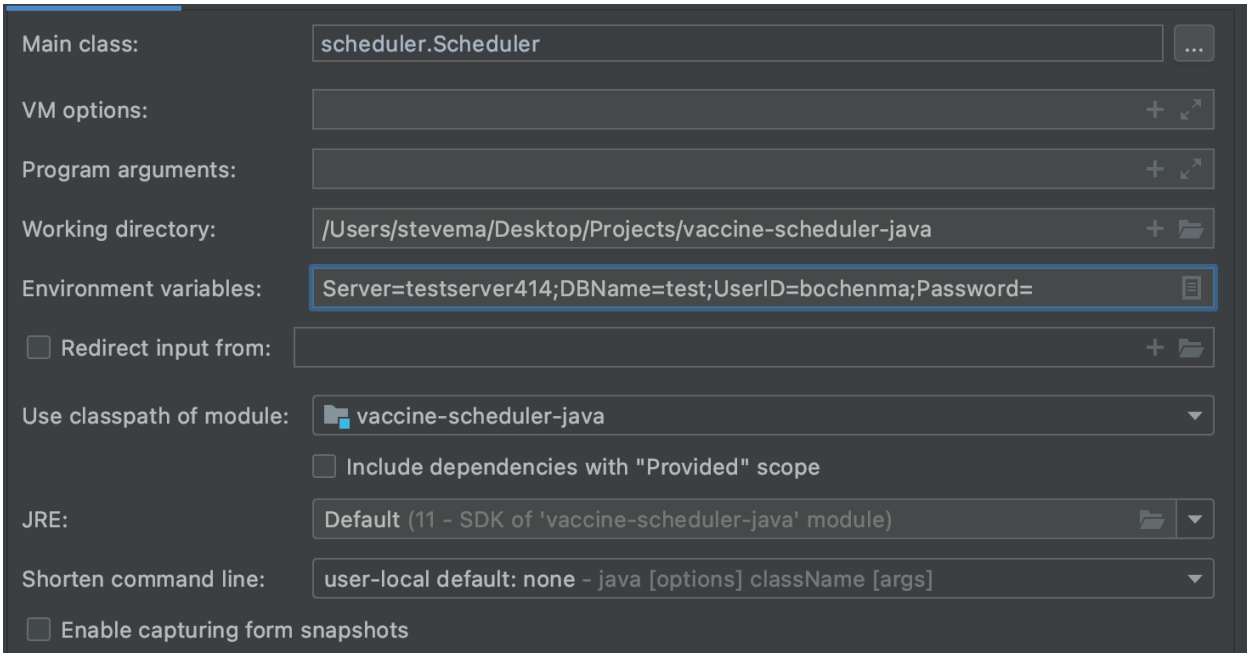
3. Now Add Configuration
    a. Select the Add Configuration option.



    b. Select “Add new Configuration”.



    c. Select “Application” in the list of options, and the following screen will appear. Your options might look a little bit different based on your IntelliJ version, but the majority should be the same.
    d. Make sure you have a name for this configuration and the java is set to the correct version, and in the main class box write *scheduler.Scheduler* (this lets IntelliJ know where main is located).

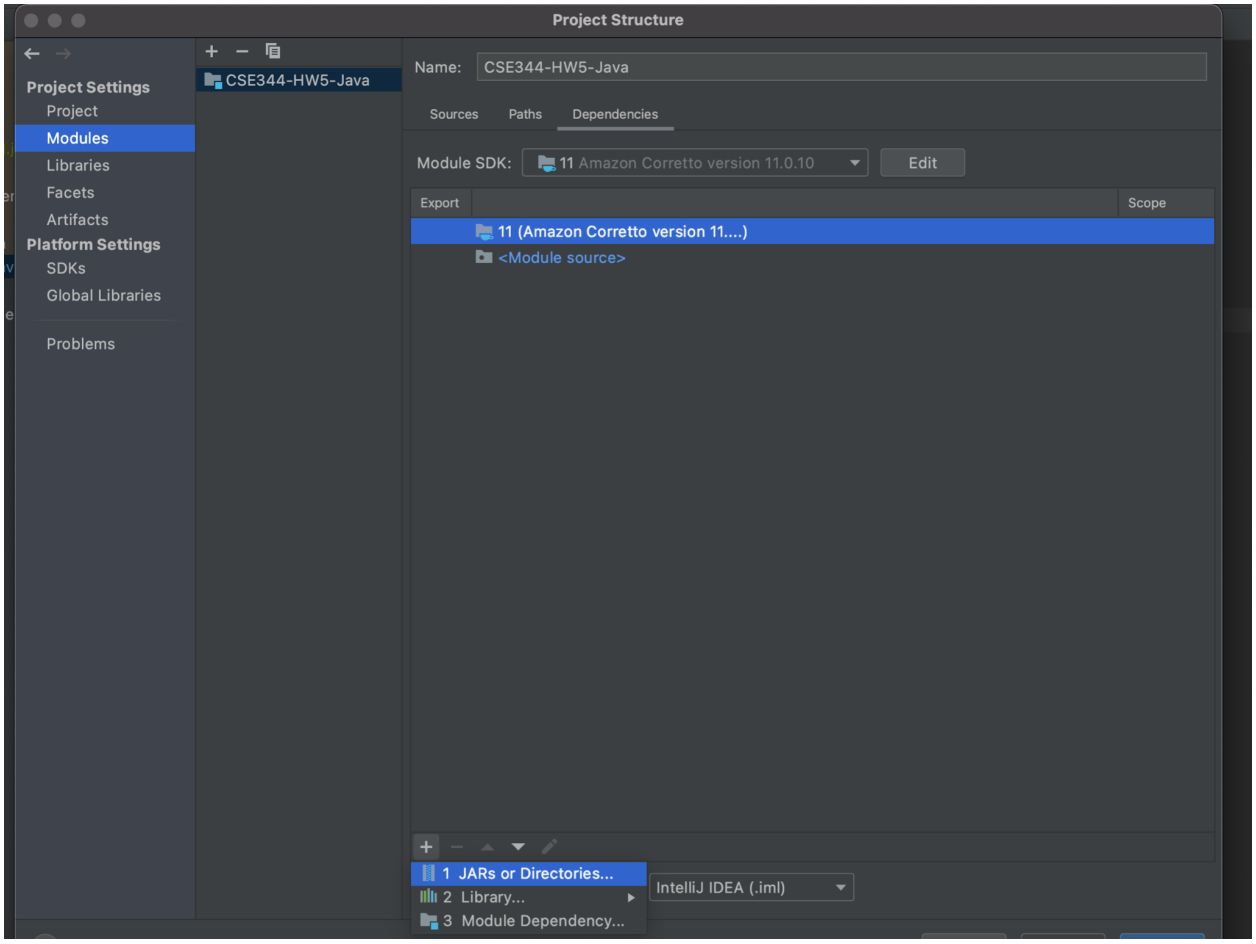


    If *scheduler.Scheduler* is highlighted red and can’t be located, 1) exit out of this screen and go to File -> Invalidate Caches / Restart and repeat the following steps, and 2) check that you have marked the source root mentioned in step 2.
    e. You can also include your environmental variables here for your server login in the format below with your corresponding Azure information (Refer to 2.3.3 for

how to retrieve credentials, if you choose to include your environmental variables here you can skip 2.3.3a and/or 2.3.3b).

```
Server=;DBName=;UserID=;Password=
```

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.



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.

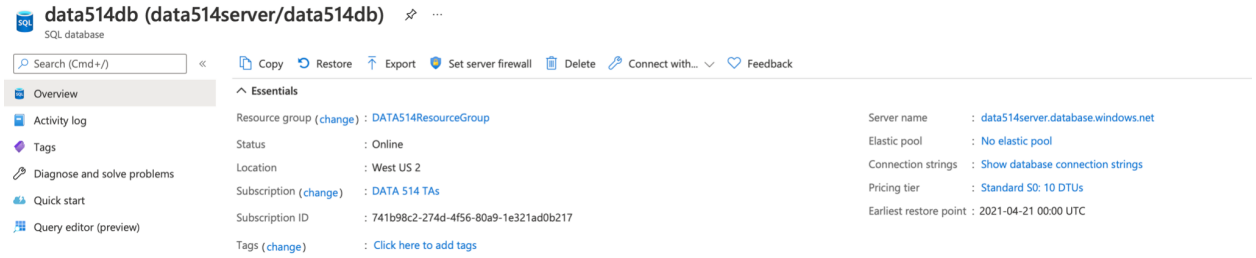
2.3.3 Setting up credentials

The first step is to retrieve the information to connect to your Microsoft Azure Database. The Server and DB names can be found in the Azure portal.

The server name would be “data514server,” and the database name would be “data514db” for the database shown in the screenshot below. **YOU NEED TO CHANGE THIS ACCORDING TO YOUR DATABASE!**

The User ID and Password are what you used to log in to your query editor.

If you’re having trouble finding that information, please make a discussion post, or contact us through email!



Once you’ve retrieved the credentials needed, you can set up your environment variables. Ignore the next sections for setting up environment variables if you already have done the setup in step 7 of 2.3.2 Setting up Intellij.

2.3.3a Setting up environment variables - MAC USERS ONLY

Open your terminal and type:

```
export Server={}
export DBName={}
export UserID={}
export Password={}
```

Where “{}” is replaced by the respective information you retrieved from step 1.

Do note that those environment variables are only added for the current session. If you’d like to add those environment variables permanently, copy the exports and paste them into “.bash_profile”.

2.3.3b Setting up environment variables - WINDOWS USERS ONLY

Open a command prompt and type:

```
setx Server ”x”
setx DBName ”x”
setx UserID ”x”
setx Password ”x”
```

Where each “x” is replaced by the respective information you retrieved from step 1.
These environment variables will be saved permanently, and a “SUCCESS: Specified value was saved.” should be returned for each variable saved. To see these changes, close the command prompt and open a new command prompt, and type:

```
set
```

This will return a list of your environment variables, and you should be able to see your new variables within this list.

2.3.4 Working with the connection manager

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

```
// 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
vaccine");
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
```

To verify you have done the setup, take **a screenshot or phone picture of this screen on your computer, and upload to gradescope for 5 points**.

Some of these operations will not work! It is your job in the rest of the assignment to finish implementing the entire system.

Requirements

Your assignment is to build a vaccine scheduling application (with a database hosted on Microsoft Azure) 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.

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.
- **After executing a command, you should re-route the program to display the list of commands again.** For example:
    - If a patient 'reserves' their vaccine for a date, you should update your database to reflect this information and route the patient back to the menu again.

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:

```
// 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 on Azure, and save the code under src.main.resources (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. It is up to you to decide how you want the user to interact with your system. TAs will be interacting with your command-line interface, and we will give credits to all reasonable designs, so don't worry too much about the details.
We have implemented account creation for caregivers as an example for you, please read through our implementation before you begin.

You're allowed to choose your own messages to display, but please make sure to supply enough information to the user regarding specific situations (e.g., when create failed). Refer to our implementation as an example.

You will need to implement the following operations:

- create_patient <username> <password>
- login_patient <username> <password>
    - If a user is already logged in in the current session, you need to logout first before logging in again.

Deliverables for Part 1

~~We ask you to submit the whole repository once you're done with part 1.~~ **Note for Autumn 2021 quarter**: Because of the Thanksgiving holiday, we don't want to pressure you on any particular due date for Part 1. Complete part 1 on the way to part 2, but make sure you have finished part 1 roughly around Nov. 24 or you will be significantly behind on the homework.

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

- src.main.resources
    - design.pdf: the design of your database schema.
    - create.sql: the create statement for your 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, along with the number of available doses left for each vaccine.
- reserve <date> <vaccine>
    - Patients perform this operation to reserve an appointment.
    - You will be randomly assigned a caregiver for the reservation on that date.
    - Output the assigned caregiver and the appointment ID for the reservation.
- 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.
    - For patients, you should print the appointment ID, vaccine name, date, and caregiver name.
- Logout

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

We will test your solutions by downloading your repo from gradescope and running the application on our machines. Your grade for this homework will be worth 100 points, divided as:

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

- Part 1 (50 points)
    - Design (30 points)

        Your database design including the files design.pdf and create.sql

    - Implementation (20 points)

        Your working implementation of the part 1 functions:

        create_patient, login_patient

- Part 2 (45 points)
    - Implementation (45 points)

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

Additionally, you may receive up to 10 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 10 extra credit points.

1. 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 “!”, “@”, “#”, “?”.
2. 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.

```
> cancel <appointment_id>
```