DB Project Part1:

Web-based System for Signing Up People For COVID-19 Vaccinations

Date: May 15, 2021
Course Title: CS – 6083

Name: LEBOHANG MCCALLUM
netId: LM89

TABLE OF CONTENTS

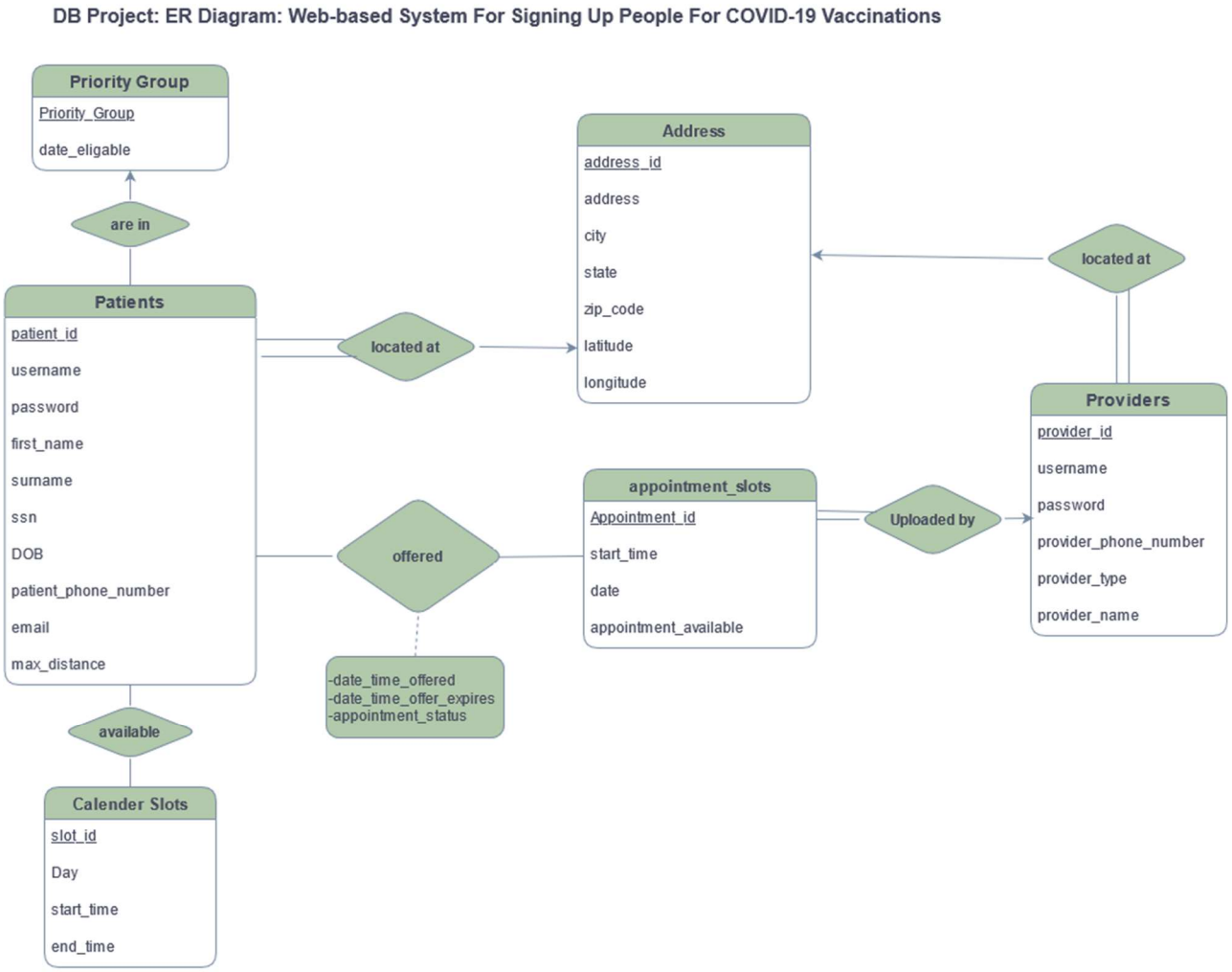
PROJECT PART I

PROJECT PART II

REFERENCES

(A) RELATIONAL SCHEMA & E-R DIAGRAM

I. Introduction & E-R Diagram



The goal of this project is to build a web-based system for signing up people for COVID-19 vaccinations. The system includes 3 types of participants namely Patients, providers, and administrators. Patients can sign up in the system and provide personal information and preferences to assist with offering them vaccination appointments. Providers are places such as hospitals, doctor's offices etc., that provide vaccinations. Providers need to sign up with their information which will allow them to upload available vaccination appointments to the system. Finally, administrators of the database system define priority groups, assigns patients to these groups, makes sure that vaccination slots are allocated to patients based on priority and time preferences, and messages patients and providers about appointments. The following report seeks to outline in detail how the system will work based on the schema design and relational model. It will also provide some sample test data and queries to observe the model in action.

II. Relational Schema

Note: Primary Keys underlined, and Tables are Bolded

- **Patients** (patient_id, username, password, first_name, surname, SSN, DOB, phone_number, email, priority_group, address_id, max_distance)
    - *Note: priority_group references priority_group in Priority_Groups*
    - *Note: address_id references address_id in Address*

- **Providers** (provider_id, username, password, phone_number, provider_type, provider_name, address_id)
    - *Note: address_id references address_id in Address*

- **Appointments** (appointment_id, provider_id, appointment_available, start_time)
    - *Note: provider_id references provider_id in Providers*

- **Calendar_slots** (slot_id, day, start_time, end_time)

- **Priority_Groups** (priority_group, date_eligable)

- **Address** (address_id, address, city, state, zip_code, latitude, longitude)

- **Offered** (appointment_id, patient_id, date_time_offered, date_time_offer_expires, appointment_status)
    - *Note: appointment_id references appointment_id in Appointments*
    - *Note: patient_id references patient_id in Patients*

- **Available** (patient_id, slot_id)
    - *Note: patient_id references patient_id in Patients*
    - *Note: slot_id references slot_id in Calendar_slots*
    -

III. Schema Description and Assumptions

Table: Patients
Purpose: Store's patients' personal information
Primary key: patient_id
Foreign key: priority_group references priority_group in Priority_Groups
Foreign key: address_id references address_id in Address
Outline of Attributes:
- patient_id: unique id identifies each patient
- username, password, email: unique login name, password, email address of user
- first_name, surname: first name, last name of user

- SSN, DOB, phone_number: social security number, birth date, phone number of user
- address_id: unique tag links the address_id associated with user in Address table.
- priority_group: identifies which vaccine eligibility group user belongs to. priority groups are added dynamically based on the eligibility guidelines and will be null when first account created/inserted
- max_distance: The distance in miles that a user is willing to travel to vaccination site. Default is 30

Table: Providers
Purpose: Store's vaccination providers private and public information
Primary key: provider_id
Foreign key: address_id references address_id in Address
Outline of Attributes:
- provider_id: unique id identifies each provider
- username, password: unique login name, password of provider
- provider_name, phone_number: name and contact phone number of provider
- provider_type: identifies type of provider i.e. hospital, clinic, school etc
- address_id: unique tag links the address_id associated with provider in Address table

Table: Appointments
Purpose: Store's all appointments and appointment availability
Primary key: appointment_id
Foreign key: provider_id references provider_id in Providers
Outline of Attributes:
- appointment_id: unique id identifies each appointment
- provider_id: unique id links the provider_id associated with a provider
- start_time: time when an appointment will begin
- appointment_available: Boolean value 0 if appointment is no longer available or 1 if appointment is available. appointment_available attribute is dynamically updated using triggers when a record is inserted/updated into the offered table based on the value of an appointment status in the Offered table which indicates the status of a particular offered appointment.

Table: Calendar_slots
Purpose: Store's fixed slots of time for a week Monday to Friday from 9 am to 6 pm
Primary key: slot_id
Outline of Attributes:
- slot_id: unique id identifies each slot
- day: indicates slot business day of the week i.e., Monday, Tuesday, Wednesday, Thursday, or Friday using integers to represent the day of the week. 2 represents Monday, 3 represents Tuesday, 4 represents Wednesday, 5 represents Thursday and 6 represents Friday

- start_time, end_time: time when calendar availability slot begins and ends. This schema has 3 slots in a day namely 9-12pm, 12-3pm and 3-6pm for availability options on any given day Monday to Friday.

Table: Priority_groups
Purpose: Store's priority group and date a group becomes eligible for vaccination
Primary key: Priority_group
Outline of Attributes:
- Priority_group: unique id identifies each group 1 to n with highest priority group starting at 1 and going to n
- date_eligable: indicates the date a priority group becomes eligible for a vaccination appointment

Table: Address
Purpose: Store's patients and providers address and geographical information
Primary key: address_id
Outline of Attributes:
- address_id: unique id identifies address for patient or provider
- address, city, state, zip_code: street and number, city, state, zip
- latitude, longitude: Geographic attitude and longitude coordinates of a patient or provider address

Table: Offered
Purpose: Store's appointments that have been offered to patients and various information about the status of those appointments.
Primary key: appointment_id, patient_id
Foreign key: appointment_id references appointment_id in Appointments
Foreign key: patient_id references patient_id in Patients
Outline of Attributes:
- appointment_id, patient_id: unique id identifies each patient
- date_time_offered, date_time_offer_expires: Once the appointment is available the patient will be informed by the date_time_offered and if the offer is not accepted before date_time_offer_expires then it will be made available in the Appointments table again as the status below will be set to 'expired'.
- appointment_status: can have the value 'offered', 'accepted', 'declined', 'cancelled', 'completed', 'no show'. Patients can accept or decline offers until date_time_offer_expires, if they do not reply before expired it is considered as declined. They can also later cancel, or may not show up, or may successfully get the shot by completing an appointment.

Table: Available

Purpose: Store's patient preference/chosen calendar slots to indicate availability for making appointment offers according to a patient's preferences and distance attribute in patient table
Primary key: patient_id, slot_id
Foreign key: patient_id references patient_id in Patients
Foreign key: slot_id references slot_id in Calendar_slots
Outline of Attributes:

- patient_id: unique id identifies each patient
- slot_id: unique id identifies each slot

IV. Functional Description

The functional description outlines some of the system demand analysis, which is how I designed the backend/database to meet the purpose of the website scheduling system

FUNCTION 1

Function: Patients can sign up in the system, and provide necessary information such as ID, name, SSN, date of birth, address, phone, and email. Patients may also indicate preferred times during the week when they would like to be scheduled for the vaccination.

Design: I designed the patients table to store all kinds of information we need for each user. I use several attributes in the patients table to store such personal information which includes social security numbers, phone numbers and email addresses etc. There is also an attribute called max_distance which can be updated in the user's profile to indicate the furthest they are willing to travel to get to a vaccination appointment. The user's email can be used to send them information about upcoming appointments etc. The only personal information about a patient that is not stored in the patients table is their address. Upon a new user registration, the address table will also be updated to include the home address of a particular patient. Since multiple people can live at the same address or building the design puts this information into a separate table to avoid redundant data and make ease of calculating distance between two address easy. An additional attribute in the address table is the latitude and longitude which is used to calculate the distance between a patient and provider.

The second part of Function 1 is that patients need to indicate preferred times they would like to be scheduled for the vaccination. This can be done using the Patients, Available and Calendar_slots table. The calendar slots table is represented by a fixed number of tuples and in this design, it is 15 records long. The records include a day of the week Monday to Friday each with three respective 3-hour time slots (5 *3 =15 records) represented by times between 9-12pm, 12-3pm and 3-6pm. The attribute 'day' are integers representing the days of the week with Monday starting at integer 2 and Friday ending at integer 6. The time slots mentioned above are then represented by a two DateTime attributes (start_time, end_time) which indicate when availability starts and ends for a particular record. Any calendar slot is then uniquely identified by a slot_id attribute. The idea is that patients will be able to select any number of tuples from the calendar_slots table and their choices will be stored in the Available table which includes the patient_id and calender_slot_id. This information can then later be used to offer patients appointments in accordance with their respective time preferences and distance if we wish.

Relevant Tables:

- **Patients** (patient_id, username, password, first_name, surname, SSN, DOB, phone_number, email, priority_group, address_id, max_distance)
- **Calendar_slots** (slot_id, day, start_time, end_time)
- **Available** (patient_id, slot_id)
- **Address** (address_id, address, city, state, zip_code, latitude, longitude)

FUNCTION 2

Function: Providers need to sign up with information such as their ID, name, phone number, and address. Registered providers can then upload available vaccination slots to the system, usually at least a few days or weeks in advance. You may assume that all vaccinations by one provider happen at the same location – if a hospital or pharmacy has several locations, they need to register as different providers.

Design: I designed the Providers table to store all kinds of information we need for each registered provider. I use several attributes in the providers table to store such personal information which includes the providers login details contact information and type (such as clinic, Hospital) etc. Similar to the Patients table the address of the provider is stored in a separate table called Address. Upon a new provider registration, the address table will also be updated to include the location of a particular provider. Since multiple providers can be in the same address or building the design puts this information into a separate table to avoid redundant data and make ease of calculating distance between two address.

In addition to the providers table, we have the Appointments table which providers can upload appointments to. Each appointment has a unique 'appointment_id' and is linked to a unique 'provider_id'. This allows providers to have multiple appointments at the same time. In addition to this there is a Boolean value 'appointment_available' in the appointments table which indicates if the appointment is available (1) or if it currently being offered to a patient or has been accepted, expired etc represented by a 0. The appointment time is also indicated by a 'start_time' attribute in Appointments. The whole system of indicating appointment availability and scheduling will then be handled by administrators with the help of several triggers and stored procedures which will be outlined in Function 3 below.

Relevant Tables:

- **Providers** (provider_id, username, password, phone_number, provider_type, provider_name, address_id)
- **Address** (address_id, address, city, state, zip_code, latitude, longitude)
- **Appointments** (appointment_id, provider_id, appointment_available, start_time)

FUNCTION 3

Function: The third type of participant is the administrator of the database system, who defines priority groups, assigns patients to these groups, makes sure that vaccination slots are allocated to patients based on priority and time preferences, and messages patients and providers about appointments.

Design: The administrator functionality is not directly represented by an entity in our relational model but however the entire systems ability to manage the scheduling process can be thought of as the administrator function. In this regard for the first part of the role to assign priority groups to patients we have created a Priority group table which holds two attributes namely 'priority_group', 'date_eligable'. Priority_group represents integers from 1 to n in which an

integer is assigned according to state and federal guidelines on who is eligible for the vaccine. 1 represents the highest priority and n represents the lowest priority on a sliding scale from 1 to n. The 'date_eligable' attribute is a DateTime type that is assigned to each priority group according to again federal and state guidelines. Administrators can now assign priority groups based on this table to priority_group attribute in the patients table which by default was set to null. Then by joining the patient and priority groups table we can identify all eligible dates the vaccine according to patient's priority groups which can be used when offering appointments to patients to ensure they are allowed.

The most extensive role for administrators' function is managing the offer table and its delicate relationship between the appointments, patients, address, available and priority_group tables. In order to offer appointment appointments, we can use a helper stored procedure called 'MatchAvailableAppointments' which takes in a patient id and returns a list of available appointments that meet the criteria for the patients stored in the available table. It can then be easily checked if the patient is eligible by looking up in the patient table the priority_group attribute. Additionally, results can then be filtered for the max distance so patients are only offered appointments they will likely use. These offered appointments will be displayed in the offer table which includes attributes appointment_id, patient_id, date_time_offered, date_time_offer_expires, appointment_status which have details on the appointment. The patient can then select 'offered', 'accepted', 'declined', 'cancelled', 'completed', 'no show' in the appointment_status attribute to indicate the status of the appointment. This is important as this attribute is used to manage when an appointment should be made available or not in the appointments table. This can be achieved with the use of a trigger that whenever a record is inserted/updated in the offer table or periodically will check the status value and update the corresponding appointment in the appointment table using the appointment_id attribute to match them. If an appointment in the offer table has the value for appointment_status as 'completed', 'offered', 'accepted', or 'no show' the attribute appointment_available in appointments table should be updated to 0 to indicate it is not available. Any other status should result in the appointment_available attribute being updated to available so that it can be offered to other eligible patients. Furthermore, the date_time_offered attribute in offered indicates the latest a patient can still accept the offer. If this time expires the appointment_status should be changed to 'declined' and appoint available attribute in appointments will also be updated to 1 to indicate it is now available again.

Relevant Tables:

- **Appointments (appointment_id, provider_id, appointment_available, start_time)**
- **Offered** (appointment_id, patient_id, date_time_offered, date_time_offer_expires, appointment_status)
- **Calendar_slots** (slot_id, day, start_time, end_time)
- **Priority_Groups** (priority_group, date_eligible)
- **Address (address_id, address, city, state, zip_code, latitude, longitude)**
- **Available (patient_id, slot_id)**

V. Stored Procedures, Functions & Triggers

1. Function 'haversine' to calculate Distance:

   To measure the distance between a patient address and an appointment provider address we generated a function called haversine below. The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. the Haversine formula is fairly accurate and generally only results in an error of up to 0.5%, which for our scenario is negligible.

```
-- Function to calculate Distance
DELIMITER $$
DROP FUNCTION IF EXISTS haversine$$

CREATE FUNCTION haversine(
        lat1 FLOAT, lon1 FLOAT,
        lat2 FLOAT, lon2 FLOAT
     ) RETURNS FLOAT
    NO SQL DETERMINISTIC
    COMMENT 'Returns the distance in degrees on the Earth
             between two known points of latitude and longitude'
BEGIN
    RETURN     111.045*DEGREES(ACOS(
               COS(RADIANS(lat1)) *
               COS(RADIANS(lat2)) *
               COS(RADIANS(lon2) - RADIANS(lon1)) +
               SIN(RADIANS(lat1)) * SIN(RADIANS(lat2))
               ));
END$$
```

2. Stored Procedure 'Update_Appointment_Availability' to update availability (appointment_available) Boolean in appointments table:

   To manage the relationship between the appointments offered table and appointments table we need to periodically run the following stored procedure on all offered appointments or on inserts and updates to the Offer table which will ensure that once an appointment is declined or cancelled it can be made available in the appointments table and as such available to other patients again. It takes in an appointment id and status as an argument.

```
CREATE DEFINER= root@localhost
PROCEDURE Update_Appointment_Availability(IN appointmentId INT,
appointmentStatus VARCHAR(45))
BEGIN
    IF
        appointmentStatus = 'completed' OR status = 'offered' OR
        status = 'accepted' OR status = 'no show'
    THEN
        UPDATE db_project_schema.appointments
        SET appointment_available = 0
        WHERE appointment_id = appointmentId;
    ELSE
        UPDATE db_project_schema.appointments
        SET appointment_available = 1
        WHERE appointment_id = appointmentId;
    END IF;
END $$
```

3. Procedure 'MatchAvailableAppointments' to get available appointments for a given patient:

   The procedure finds all available (not currently assigned) appointments that satisfy the constraints on the given patient's weekly schedule, sorted by increasing distance from the user's home address. This stored procedure will be useful in the offering of appointments to patients in the system. The stored procedure makes use of our haversine distance calculating function for this purpose as well. This is essentially a more dynamic and comprehensive version of the query asked for number three in the SQL queries for this assignment. It takes a patient_id as an argument to find relevant appointments for that patient.

```
CREATE PROCEDURE MatchAvailableAppointments (IN Id INT)
BEGIN
WITH patient_info as ( SELECT pat.patient_id, padd.latitude, padd.longitude
                        FROM patients pat JOIN address padd ON
                        pat.address_id = padd.address_id
                        WHERE pat.patient_id = Id -- provide a patient id
                      )
SELECT aps.appointment_id, p.provider_id, p.provider_name, p.address_id,
       CONCAT(ad.address, ' ', ad.city, ' ', ad.state, ' ', ad.zip_code)
       as Address, aps.start_time, pin.patient_id,
      (haversine(pin.latitude, pin.longitude, ad.latitude, ad.longitude) /
8)*5 AS Distance
FROM appointments aps JOIN providers p ON aps.provider_id = p.provider_id
     JOIN address ad ON p.address_id = ad.address_id
     CROSS JOIN patient_info pin
WHERE aps.appointment_available = 1 AND EXISTS
                                        (
                                         SELECT 1

                                          FROM available avl JOIN
                                          calendar_slots csl ON avl.slot_id
                                          = csl.slot_id
                                          WHERE dayofweek(aps.start_time) =
                                          csl.day AND
                                          time(aps.start_time) between
                                          csl.start_time
                                          AND csl.end_time AND
                                          avl.patient_id = pin.patient_id
                                          )
ORDER BY Distance, aps.start_time;
END //
```

4. Triggers to call 'Update_Appointment_Availability' stored procedure upon inserts or updates into the offer table

   The trigger will call Update_Appointment_Availability after a new record has been inserted or updated into the offer table which will allow us to check for declined or cancelled appointments so that we may make them available again in the appointments table.

```
CREATE TRIGGER UpdateAppointmentAvailabilityOnInsertIntoOffered
AFTER INSERT
ON db_project_schema.offered FOR EACH ROW
BEGIN
    CALL Update_Appointment_Availability(NEW.appointment_id,
    NEW.appointment_status);
END $$


CREATE TRIGGER UpdateAppointmentAvailabilityOnUpdateIntoOffered
AFTER UPDATE
ON db_project_schema.offered FOR EACH ROW
BEGIN
    CALL Update_Appointment_Availability(NEW.appointment_id,
    NEW.appointment_status);
END $$
```

5. Trigger that checks if an account already exists before inserting a new account into the patient's table.

   The trigger checks for existing accounts by matching the new accounts social security number with the records that already exist in the table. If there is a match, then the account will not be allowed to be created as that patient already exists.

```
DELIMITER //
CREATE TRIGGER InsertNewAccount
BEFORE INSERT
ON db_project_schema.patients  FOR EACH ROW
BEGIN
    IF NEW.ssn IN (SELECT ssn FROM db_project_schema.patients WHERE ssn =
    NEW.ssn)
    THEN SIGNAL SQLSTATE '45000' SET message_text = 'Account already
    exists';
    END IF;
END //
```

(B) CREATE DATABASE

```
CREATE SCHEMA `db_project_schemas`;
USE `db_project_schema`;

DROP TABLE IF EXISTS 'patients';
CREATE TABLE `patients` (
  `patient_id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `first_name` varchar(45) NOT NULL,
  `surname` varchar(45) NOT NULL,
  `SSN` int NOT NULL,
  `DOB` date NOT NULL,
  `phone_number` bigint NOT NULL,
  `email` varchar(45) NOT NULL,
  `max_distance` int NOT NULL,
  `priority_group` int DEFAULT NULL,
  `address_id` int DEFAULT NULL,
  PRIMARY KEY (`patient_id`),
  KEY `priority_group_idx` (`priority_group`),
  KEY `address_id_patientsFK_idx` (`address_id`),
  CONSTRAINT `address_id_patientsFK` FOREIGN KEY (`address_id`) REFERENCES `address` (`address_id`),
  CONSTRAINT `priority_group_patientsFK` FOREIGN KEY (`priority_group`) REFERENCES `priority_groups` (`priority_group`));
ALTER TABLE `db_project_schema`.`patients`
CHANGE COLUMN `max_distance` `max_distance` INT NULL DEFAULT 30 ;

DROP TABLE IF EXISTS 'providers';
CREATE TABLE `providers` (
  `provider_id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `phone_number` bigint NOT NULL,
  `provider_type` varchar(45) NOT NULL,
  `provider_name` varchar(45) NOT NULL,
  `address_id` int DEFAULT NULL,
  PRIMARY KEY (`provider_id`),
  KEY `address_id_providerFK_idx` (`address_id`),
  CONSTRAINT `address_id_providerFK` FOREIGN KEY (`address_id`) REFERENCES `address` (`address_id`));

DROP TABLE IF EXISTS 'appointments;
CREATE TABLE `appointments` (
  `appointment_id` int NOT NULL AUTO_INCREMENT,
  `start_time` datetime NOT NULL,
  `appointment_available` tinyint(1) NOT NULL DEFAULT '1',
  `provider_id` int DEFAULT NULL,
  PRIMARY KEY (`appointment_id`),
  KEY `provider_id_appointmentsFK_idx` (`provider_id`),
  CONSTRAINT `provider_id_appointmentsFK` FOREIGN KEY (`provider_id`) REFERENCES `providers` (`provider_id`));

DROP TABLE IF EXISTS 'priority_groups';
CREATE TABLE `priority_groups` (
  `priority_group` int NOT NULL,
  `date_eligable` date DEFAULT NULL,
  PRIMARY KEY (`priority_group`));

DROP TABLE IF EXISTS 'address';
CREATE TABLE `address` (
  `address_id` int NOT NULL AUTO_INCREMENT,
  `address` varchar(255) NOT NULL,
  `city` varchar(45) NOT NULL,
  `state` varchar(45) NOT NULL,
  `zip_code` int NOT NULL,
  `latitude` double DEFAULT NULL,
  `longitude` double DEFAULT NULL,
  PRIMARY KEY (`address_id`));

DROP TABLE IF EXISTS `available`;
CREATE TABLE `available` (
  `patient_id` int NOT NULL,
  `slot_id` int NOT NULL,
  PRIMARY KEY (`patient_id`,`slot_id`),
  KEY `slot_id_availbleFK_idx` (`slot_id`),
  CONSTRAINT `patient_id_availableFK` FOREIGN KEY (`patient_id`) REFERENCES `patients` (`patient_id`),
```

```
  CONSTRAINT `slot_id_availbleFK` FOREIGN KEY (`slot_id`) REFERENCES `calendar_slots` (`slot_id`));

DROP TABLE IF EXISTS `calendar_slots`;
CREATE TABLE `calendar_slots` (
  `slot_id` int NOT NULL AUTO_INCREMENT,
  `day` int NOT NULL,
  `start_time` time NOT NULL,
  `end_time` time NOT NULL,
  PRIMARY KEY (`slot_id`));

DROP TABLE IF EXISTS `offered`;
CREATE TABLE `offered` (
  `appointment_id` int NOT NULL,
  `patient_id` int NOT NULL,
  `appointment_status` varchar(45) NOT NULL DEFAULT 'offered',
  `date_time_offered` datetime NOT NULL,
  `date_time_offer_expires` datetime NOT NULL,
  PRIMARY KEY (`appointment_id`,`patient_id`),
  KEY `patient_id_offeredFK_idx` (`patient_id`),
  CONSTRAINT `appointment_id_offeredFK` FOREIGN KEY (`appointment_id`) REFERENCES `appointments` (`appointment_id`),
  CONSTRAINT `patient_id_offeredFK` FOREIGN KEY (`patient_id`) REFERENCES `patients` (`patient_id`)
)
```

(C) SQL TEST QUERIES

1. Create a new patient account, together with email, password, name, date of birth, etc.

```
-- Update patient first
INSERT INTO patients (username, password, first_name, surname, SSN,
DOB, phone_number, email, max_distance)
VALUES ('Database_Man', md5('ILoveHomework'), 'Trevor', 'Noah',
124745569, '1990-07-20', 2124566466, 'TrevoNoah@gmail.com', 30);

-- Update patients address
INSERT INTO address (address, city, state, zip_code, latitude,
longitude)
VALUES ('4 Washington Square Village', 'New York', 'NY', 10012,
40.7289655430854, -73.9970805658389);

-- update patients auto incremented address_id in the patient table
as well
UPDATE patients
SET address_id = (SELECT address_id FROM Address WHERE address = '4
Washington Square Village'
                        AND city = 'New York' AND city = 'New York'
                        AND state = 'NY'
                        AND zip_code = '10012' LIMIT 1)
WHERE SSN = 124745569;
SELECT * FROM db_project_schema.patients;
```

patient_id	username	password	first_name	surname	SSN	DOB	phone_number	email	max_
4	Prof89	Kabi	Teboho	Moja	124782145	1954-11-21	2122609030	moja45@nyu.edu	10
5	KingLeo100	GoodGrades	Dipak	Chopra	124785549	1951-05-18	2124563258	crystals100@gmail.com	15
6	George87	AllstonFranklin	Jill	Roberts	124778987	1980-04-06	6468954758	jillyRoberts@yahoo.com	30
7	Freedom100	RobertBruce	William	Wallace	124774585	1970-02-25	6468745523	scotsman21@gmail.com	10
8	England100	BestKing	Edward	Longshanks	125447896	1971-02-05	6648445123	englishman@yahoo.com	20
18	Database_Man	572786cd9e8efd34026428f0cc941c54	Trevor	Noah	124745569	1990-07-20	2124566466	TrevoNoah@gmail.com	30

2. Insert a new appointment offered by a provider.
   Query:

```
INSERT INTO appointments(start_time, appointment_available,
provider_id)
VALUES ('2021-05-07 12:30:00', 1, 4);
SELECT * FROM db_project_schema.appointments;
```

appointment_id	start_time	appointment_available	provider_id
18	2021-03-05 10:00:00	0	3
19	2021-03-05 10:30:00	0	3
20	2021-03-05 11:00:00	0	3
21	2021-03-05 11:30:00	0	3
24	2021-05-07 12:30:00	1	4

3. Write a query that, for a given patient, finds all available (not currently assigned) appointments that satisfy the constraints on the patient's weekly schedule, sorted by increasing distance from the user's home address.
   Query: Note: haversine is function defined to calculate distance

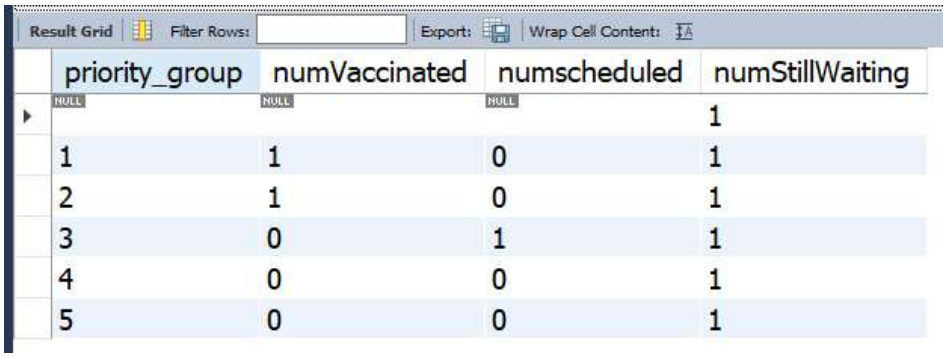
```
WITH patient_info as ( SELECT pat.patient_id, padd.latitude,
padd.longitude
                        FROM patients pat JOIN address padd ON
                        pat.address_id = padd.address_id
                        WHERE pat.patient_id = 1 -- provide a patient
                        id
                      )
SELECT aps.appointment_id, aps.start_time, pin.patient_id,
      (haversine(pin.latitude, pin.longitude, ad.latitude,
      ad.longitude) / 8)*5 AS Distance
FROM appointments aps JOIN providers p ON aps.provider_id =
p.provider_id
     JOIN address ad ON p.address_id = ad.address_id
     CROSS JOIN patient_info pin
WHERE aps.appointment_available = 1 AND
      EXISTS
            (
            SELECT 1
            FROM available avl JOIN calendar_slots csl
            ON avl.slot_id = csl.slot_id
            WHERE dayofweek(aps.start_time) = csl.day AND
            time(aps.start_time) between csl.start_time AND
            csl.end_time AND avl.patient_id = pin.patient_id
            )
ORDER BY Distance, aps.start_time;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

appointment_id	start_time	patient_id	Distance
6	2021-05-04 10:00:00	1	0.9005485475063324
7	2021-05-04 15:30:00	1	0.9005485475063324
8	2021-05-04 17:00:00	1	0.9005485475063324
10	2021-05-04 12:00:00	1	5.816815495491028
11	2021-05-04 18:00:00	1	5.816815495491028
9	2021-05-05 09:30:00	1	5.816815495491028
12	2021-05-05 13:30:00	1	8.163979053497314
13	2021-05-05 14:00:00	1	8.163979053497314
14	2021-05-05 14:30:00	1	8.163979053497314

4. For each priority group, list the number of patients that have already received the vaccination, the number of patients currently scheduled for an appointment, and the number of patients still waiting for an appointment.

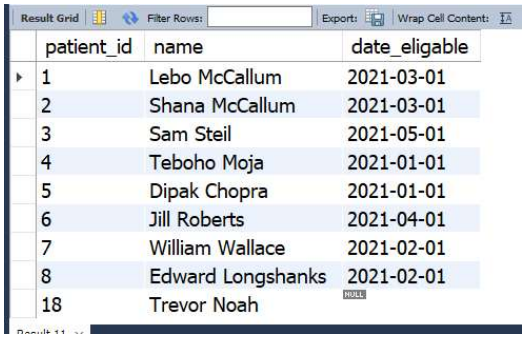
```
WITH v1 AS (
            SELECT pat.priority_group,
            COUNT(CASE WHEN (ofr.appointment_status =  'completed') THEN 1
END) as numVaccinated,
            COUNT(CASE WHEN (ofr.appointment_status = 'accepted') THEN 1
END) as numscheduled
            FROM offered ofr RIGHT JOIN patients pat ON ofr.patient_id =
pat.patient_id
            GROUP BY pat.priority_group
            ),
V2 AS (
        SELECT pat3.priority_group, COUNT(*) as numStillWaiting
        FROM (SELECT pat2.patient_id,
                COUNT(CASE WHEN (ofr2.appointment_status ='completed')
THEN 1 END) as Vaccinated,
                COUNT(CASE WHEN (ofr2.appointment_status = 'accepted') THEN
1 END) as scheduled
             FROM offered ofr2 RIGHT JOIN patients pat2 ON ofr2.patient_id
             = pat2.patient_id
             GROUP BY pat2.patient_id
             HAVING Vaccinated = 0 AND scheduled = 0) t1
             LEFT JOIN patients pat3 ON t1.patient_id = pat3.patient_id
             GROUP BY pat3.priority_group
        )
SELECT V1.priority_group, V1.numVaccinated, V1.numscheduled, V2.numStill-
Waiting
FROM V1 RIGHT JOIN V2 ON V1.priority_group = V2.priority_group
ORDER BY 1;
```



priority_group	numVaccinated	numscheduled	numStillWaiting
NULL	NULL	NULL	1
1	1	0	1
2	1	0	1
3	0	1	1
4	0	0	1
5	0	0	1

5. For each patient, output the ID, name, and date when the patient becomes eligible for vaccination.

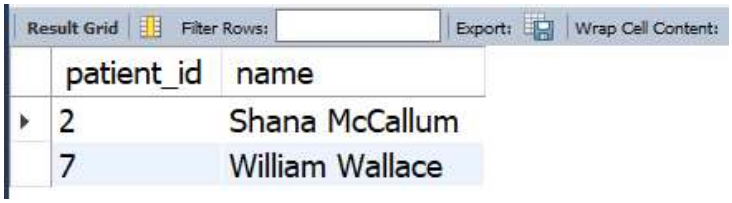
```
SELECT p.patient_id, CONCAT(p.first_name, ' ', p.surname) AS name,
date_eligable
FROM patients p LEFT JOIN priority_groups pg ON p.priority_group =
pg.priority_group;
```



patient_id	name	date_eligable
1	Lebo McCallum	2021-03-01
2	Shana McCallum	2021-03-01
3	Sam Steil	2021-05-01
4	Teboho Moja	2021-01-01
5	Dipak Chopra	2021-01-01
6	Jill Roberts	2021-04-01
7	William Wallace	2021-02-01
8	Edward Longshanks	2021-02-01
18	Trevor Noah	

6. Output the ID and name of all patients that have cancelled at least 3 appointments, or that did not show up for at least two confirmed appointments that they did not cancel

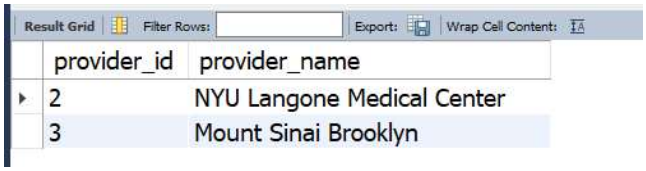
```
WITH V1 AS (
            SELECT patient_id, appointment_status,
            Count(appointment_status) as count_status
            FROM offered o
            GROUP BY appointment_status, patient_id
            HAVING (appointment_status = 'cancelled' AND count_status
            >= 3) OR (appointment_status = 'no show' AND count_status
            >= 2)
            )
SELECT patient_id, CONCAT(p.first_name, ' ', p.surname) AS name
FROM Patients p
WHERE p.patient_id IN (SELECT patient_id FROM V1);
```



patient_id	name
2	Shana McCallum
7	William Wallace

7. Output the ID and name of the provider(s) that has performed the largest number of vaccinations

```
with t1 as(
        SELECT pr.provider_name, pr.provider_id, Count(*) as
        numOfVaccinationsCompleted, DENSE_RANK () OVER (ORDER BY
        Count(*) DESC) as ranking
        FROM offered o JOIN appointments app ON o.appointment_id =
        app.appointment_id RIGHT JOIN providers pr ON app.provider_id
        = pr.provider_id
        GROUP BY o.appointment_status, pr.provider_name
        HAVING o.appointment_status = 'completed'
        )
SELECT t1.provider_id, t1.provider_name
FROM t1
WHERE ranking = 1;
```



provider_id	provider_name
2	NYU Langone Medical Center
3	Mount Sinai Brooklyn

(D) POPULATED DATABASE WITH SAMPLE DATA

Patients Table:

patient_id	username	password	first_name	surname	SSN	DOB	phone_number	email	max_distance	priority_group	address
1	Obelmac	Yo_mama	Lebo	McCallum	128987748	1989-04-09	3474978967	obels@gmail.com	30	3	1
2	Zimshady89	Family	Shana	McCallum	124984587	1989-08-08	3475896478	shana@gmail.com	30	3	1
3	Sammygirl	Nala	Sam	Steil	124985879	1994-05-31	3475896645	stiels@yahoo.com	15	5	3
4	Prof89	Kabi	Teboho	Moja	124782145	1954-11-21	2122609030	moja45@nyu.edu	10	1	4
5	KingLeo100	GoodGrades	Dipak	Chopra	124785549	1951-05-18	2124563258	crystals100@gmail.com	15	1	5
6	George87	AllstonFranklin	Jill	Roberts	124778987	1980-04-06	6468954758	jillyRoberts@yahoo.com	30	4	2
7	Freedom100	RobertBruce	William	Wallace	124774585	1970-02-25	6468745523	scotsman21@gmail.com	10	2	6
8	England100	BestKing	Edward	Longshanks	125447896	1971-02-05	6648445123	englishman@yahoo.com	20	2	6
18	Database_Man	572786cd9e8efd34026428f0cc941c54	Trevor	Noah	124745569	1990-07-20	2124566466	TrevoNoah@gmail.com	30	NULL	18

Providers Table:

provider_id	username	password	phone_number	provider_type	provider_name	address_id
1	United.Health	SaveLives	4785549865	Clinic	United HealthCare	7
2	Purple98	StudentsUnite	2124578521	Hospital	NYU Langone Medical Center	8
3	BrooklynPride	BestBurrow	4587789654	Hospital	Mount Sinai Brooklyn	9
4	WomenUnite	FirstLady	4521145874	Hospital	Brigham and Women's Hospital	10
5	Harlem100	SoulCity	8547558779	Clinic	Family Health Center of Harlem	11

Appointments Table:

appointment_id	start_time	appointment_available	provider_id
1	2021-05-03 09:00:00	0	1
2	2021-05-03 09:30:00	0	1
3	2021-05-03 10:00:00	0	1
4	2021-05-03 15:00:00	0	1
5	2021-05-03 17:30:00	0	1
6	2021-05-04 10:00:00	1	2
7	2021-05-04 15:30:00	1	2
8	2021-05-04 17:00:00	1	2
9	2021-05-05 09:30:00	1	5
10	2021-05-04 12:00:00	1	5
11	2021-05-04 18:00:00	1	5
12	2021-05-05 13:30:00	1	3
13	2021-05-05 14:00:00	1	3
14	2021-05-05 14:30:00	1	3
15	2021-02-01 09:00:00	0	2
16	2021-03-05 09:00:00	0	3
17	2021-03-05 09:30:00	0	3
18	2021-03-05 10:00:00	0	3
19	2021-03-05 10:30:00	0	3
20	2021-03-05 11:00:00	0	3
21	2021-03-05 11:30:00	0	3
24	2021-05-07 12:30:00	1	4

Calendar_slots Table:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

slot_id	day	start_time	end_time
1	2	09:00:00	12:00:00
2	2	12:00:00	15:00:00
3	2	15:00:00	18:00:00
4	3	09:00:00	12:00:00
5	3	12:00:00	15:00:00
6	3	15:00:00	18:00:00
7	4	09:00:00	12:00:00
8	4	12:00:00	15:00:00
9	4	15:00:00	18:00:00
10	5	09:00:00	12:00:00
11	5	12:00:00	15:00:00
12	5	15:00:00	18:00:00
13	6	09:00:00	12:00:00
14	6	12:00:00	15:00:00
15	6	15:00:00	18:00:00
NULL	NULL	NULL	NULL

Priority_groups Table:

Result Grid | Filter Rows:

priority_group	date_eligable
1	2021-01-01
2	2021-02-01
3	2021-03-01
4	2021-04-01
5	2021-05-01
NULL	NULL

Address Table:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

address_id	address	city	state	zip_code	latitude	longitude
1	1 Washington Sq...	New York	NY	10012	40.7289655430854	-73.9970805658389
2	89 Franklin Street	Boston	MA	2134	42.35901503391019	-71.1322881023161
3	Main Street	Philadelp...	PA	19127	39.97407150268555	-75.1395263671875
4	3 Washington Sq...	New York	NY	10012	40.7289655430854	-73.9970805658389
5	18 E 14th St	New York	NY	10003	40.73620456301941	-73.99127012389293
6	280 Pleasant Ave	New York	NY	10029	40.79483590382659	-73.93171324696924
7	206 W 21st St	New York	NY	10011	40.74310369474567	-73.99628393791191
8	14 Wall St	New York	NY	10005	40.71669387817383	-74.00263977050781
9	3201 Kings Hwy	New York	NY	11234	40.61889882398311	-73.94235173304202
10	75 Francis St	Boston	MA	2115	42.33640978008255	-71.1061721734808
11	1824 Madison Ave	New York	NY	10035	40.80201758665251	-73.94283125076778
12	2 Washington Sq...	New York	NY	10012	40.7289655430854	-73.9970805658389
18	4 Washington Sq...	New York	NY	10012	40.7289655430854	-73.9970805658389
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Offered Table:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

appointment_id	patient_id	appointment_status	date_time_offered	date_time_offer_expires
1	1	accepted	2021-04-23 09:00:00	2021-04-30 09:00:00
2	1	offered	2021-04-23 09:00:00	2021-04-30 09:00:00
3	1	offered	2021-04-30 09:00:00	2021-04-30 09:00:00
4	5	offered	2021-04-23 09:00:00	2021-04-30 09:00:00
5	5	offered	2021-04-23 09:00:00	2021-04-30 09:00:00
15	4	completed	2021-02-01 09:00:00	2021-02-01 18:00:00
16	2	cancelled	2021-03-01 09:00:00	2021-03-02 09:00:00
17	2	cancelled	2021-03-01 09:00:00	2021-03-02 09:00:00
18	2	cancelled	2021-03-01 09:00:00	2021-03-02 09:00:00
19	7	no show	2021-03-01 09:00:00	2021-03-02 09:00:00
20	7	no show	2021-03-01 09:00:00	2021-03-02 09:00:00
21	7	completed	2021-03-01 09:00:00	2021-03-02 09:00:00
NULL	NULL	NULL	NULL	NULL

Available Table:

Result Grid | Filter Rows:

patient_id	slot_id
1	1
1	2
1	3
1	4
1	5
1	6
1	7
1	8
1	9
1	10
1	11
1	12
1	13
1	14
1	15
2	1
2	2
2	3
2	4
2	5
2	6
4	10
4	11
4	12

available 1

Result Grid | Filter Rows:

patient_id	slot_id
4	12
4	13
4	14
4	15
5	7
5	8
5	9
7	1
7	2
7	3
7	4
7	5
7	6
7	7
7	8
7	9

patient_id	slot_id
7	8
7	9
7	10
7	11
7	12
7	13
7	14
7	15
8	1
8	2
8	3
8	10
8	11
8	12
NULL	NULL

PROJECT PART II

(E) CHANGES TO RELATIONAL SCHEMA

VI. Introduction & Updated Relational Schema

Part II of the project seeks to implement a web-based user interface to support the effective scheduling of COVID-19 vaccinations. Patients will now be able to register, login, and edit their availability and other information as well as accept, decline, and cancel any appointments that are offered to them. Providers are also able to sign up, and to add appointments to the database. (As per the instructions I have assumed that providers cannot cancel appointments once they are uploaded, but only patients can). Additionally, the system has an included feature that automatically and periodically assigns available appointments to suitable patients when they access the schedule appointments tab on the website. This report will outline all major changes to relational schema as well as updates that have been made to existing SQL stored procedures, triggers, and functions to run the website. The report will additionally outline the testing and functionality implemented in the prototype to showcase the website.

UPDATED SCHEMA:

Note: Primary Keys underlined, and tables that were altered from the original Schema in Part I are highlighted in green.

- **Patients** (patient_id, ~~username~~, password, first_name, surname, SSN, DOB, phone_number, patientEmail, priority_group, max_distance, patientAddress, patientState, patientCity, patientZipCode, patientLatitude, patientLongitude)
    - *Note: priority_group references priority_group in Priority_Groups*
    - *Note: address_id references address_id in Address*

- **Providers** (provider_id, ~~username~~, password, phone_number, provider_type, provider_name, providerEmail, providerAddress, providerCity, providerState, providerZipCode, patientLatitude, patientLongitude)
    - *Note: address_id references address_id in Address*

- **Appointments** (appointment_id, provider_id, appointment_available, start_time)
    - *Note: provider_id references provider_id in Providers*

- **Calendar_slots** (slot_id, day, start_time, end_time, day_text)

- **Priority_Groups** (priority_group, date_eligable)

- ~~**Address** (address_id, address, city, state, zip_code, latitude, longitude)~~

- **Offered** (appointment_id, patient_id, date_time_offered, date_time_offer_expires, appointment_status)
    - *Note: appointment_id references appointment_id in Appointments*
    - *Note: patient_id references patient_id in Patients*

- **Available** (patient_id, slot_id)
    - *Note: patient_id references patient_id in Patients*
    - *Note: slot_id references slot_id in Calendar_slots*

VII. OUTLINE OF REVISED SCHEMA DESIGN

- The most significant change in the relational schema was the removal of the Address relation which was effectively moved to the Patient and Provider relations, respectively. The change was important since the function of making offers of appointments and checking the distance constraint for patients is used regularly on the website and so consistently doing a join with address table for all these operations would result in high optimization costs as the table may need to be fetched and joined consistently.
- A provider email address attribute was added to the provider table since the website will use email addresses for login purposes and communication. This made the username attributes in both patient and provider tables redundant and as such it has been removed. The patient table already included an email attribute
- Finally, I added the day_text attribute to calendar_slots relation. Previously we used numbers to indicate days of the week, but we needed a way to display the day as text on the website, so users can identify the day of the week for which they are updating their preferences in the available table using the calendar_slots relation.

VIII. UPDATES TO STORED PROCEDURES, FUNCTIONS &TRIGGERS

Below is a list of existing triggers, functions and stored procedures retained from project part I which we use in the implementation of the website:

- Function 'haversine' is used to calculate distance between a patient and provider using their respective latitudes and longitudes
- Stored Procedure 'Update_Appointment_Availability' is called by triggers on updates and inserts into the offered table to ensure appointment is marked available or not in the respective appointments table
- Procedure 'MatchAvailableAppointments' provides a table with a list of available appointments for a particular patientId and the distance to the appointments sorted in descending order.

- We have two triggers discussed in part I that call 'Update Appointment Availability' stored procedure upon inserts or updates into the offer table

New stored procedures to improve functionality of the website:

1. Stored Procedure 'patientsThatNeedAppointments':
   This returns a table containing multiple patientId's matched with available appointmentId's that meet the patients' constraints. The list only includes patients who do not currently have a completed, accepted or offered appointments. This is used as an input into our matching algorithm which will try looking at patients still requiring appointments and available appointments to determine how to maximize offers.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`patientsThatNeedAppointments`()
BEGIN
SELECT DISTINCT patientStillNeedApps.patientId, aps.appointment_id,
aps.start_time as appointmentTime, patientStillNeedApps.distancePreference
FROM appointments aps JOIN (WITH CompletedScheduledOffered AS(
                                                              SELECT
pat.priorityGroup, pat.patientId
                                                              FROM offered
ofr RIGHT JOIN patient pat ON ofr.patientId = pat.patientId
                                                              WHERE
ofr.appointment_status =  'completed' OR

ofr.appointment_status = 'accepted' OR

ofr.appointment_status = 'offered'
                                                              )
                            SELECT pat2.patientId, avl.slot_id, cs.day,
cs.start_time, cs.end_time, cs.day_text, pat2.distancePreference
                            FROM patient pat2 JOIN available avl ON
pat2.patientId = avl.patientId JOIN calendar_slots cs ON cs.slot_id =
avl.slot_id
                            WHERE pat2.patientId NOT IN (SELECT
CompletedScheduledOffered.patientId
                                                         FROM
CompletedScheduledOffered)
                        ) as patientStillNeedApps
WHERE aps.appointment_available = 1 AND EXISTS
                                    (SELECT 1
                                     FROM available avl JOIN
calendar_slots csl ON avl.slot_id = csl.slot_id
                                     WHERE dayofweek(aps.start_time) =
csl.day AND
                                           time(aps.start_time)
between csl.start_time
                                           AND csl.end_time AND
avl.patientId = patientStillNeedApps.patientId)
ORDER BY patientStillNeedApps.patientId;
END
```

2. Stored Procedure 'getAvailableAppointments':
   This is a simple procedure which returns a list of all currently available appointments which is the second input into our matching algorithm.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `getAvailableAppointments`()
BEGIN
    SELECT appointment_id FROM appointments WHERE appointment_available =
1;
END
```

3. Procedure 'MatchAvailableAppointmentsLimit' to get appointments to offer a given patient:
   The procedure extends on the original 'MatchAvailableAppointments' procedure by taking in an argument now which will limit the number of rows returned. This returned table will be used to generate new offers for patients who try to schedule an appointment. The limit argument that is given to this procedure is one of the outputs of our matching algorithm which informs what is the maximum number of appointments we should offer to this patient to maintain an optimal allocation.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `MatchAvailableAppointmentsLimit`(IN Id
INT, IN maxToOffer INT)
BEGIN
WITH patient_info as ( SELECT pat.patientId, pat.patientLatitude,
pat.patientLongitude, pat.distancePreference
                        FROM patient pat
                        WHERE pat.patientId = Id -- provide a patient id
                      )
SELECT aps.appointment_id, p.providerName, p.providerAddress, p.providerCity,
p.providerState, p.providerZipCode, aps.start_time, pin.patientId,
      ROUND((haversine(pin.patientLatitude, pin.patientLongitude,
p.providerLatitude, p.providerLongitude) / 8)*5 , 2) AS distance,
      DATE_ADD(now(),interval 240 hour) as dateOfferExpires
FROM appointments aps JOIN provider p ON aps.providerId = p.providerId
     CROSS JOIN patient_info pin
WHERE aps.appointment_available = 1 AND (ROUND((haversine(pin.patientLatitude,
pin.patientLongitude, p.providerLatitude, p.providerLongitude) / 8)*5 , 2)  <
pin.distancePreference) AND EXISTS
                                        (SELECT 1
                                         FROM available avl JOIN calendar_slots csl
ON avl.slot_id = csl.slot_id
                                         WHERE dayofweek(aps.start_time) = csl.day
AND
                                                time(aps.start_time) between
csl.start_time
                                                AND csl.end_time AND avl.patientId
= pin.patientId)
ORDER BY Distance, aps.start_time
LIMIT maxToOffer;
END
```

4. Procedure ‘updateStaleAppointments’:
   We have a quite simple procedure which is run periodically before look ups into the appointments table. The procedure is simple and essentially just checks for any appointment dates that have passed and checks to make sure their availability is indicated as not available. This ensures we never offer appointments that are in the past or recently expired. The event of updating staleAppointments will run every 30 minutes.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `updateStaleAppointments`()
BEGIN
    UPDATE appointments
    SET appointment_available = 0
    WHERE start_time < now();
END
```

```
 CREATE EVENT updateOldAppointments
    ON SCHEDULE EVERY 30 MINUTE
    DO
      CALL updateStaleAppointments();
```

5. Procedure ‘updatePrioirtyGroup()’ and Event check ‘priorityGroups’:
   We have a quite simple procedure which is run periodically That checks the age of all the patients and allocates the priority groups for the patients according to their age and our decided priority groups. The procedure will run every 30 minutes.

```
CREATE PROCEDURE updatePrioirtyGroup()
BEGIN
    UPDATE patient
    SET priorityGroup = 1
    WHERE ((DATEDIFF(CURDATE(), dob)) / 365.25) >= '60';

    UPDATE patient
    SET priorityGroup = 2
    WHERE (DATEDIFF(CURDATE(), dob)) / 365.25 BETWEEN '50' AND '59';

    UPDATE patient
    SET priorityGroup = 3
    WHERE (DATEDIFF(CURDATE(), dob)) / 365.25 BETWEEN '30' AND '49';

    UPDATE patient
    SET priorityGroup = 4
    WHERE (DATEDIFF(CURDATE(), dob)) / 365.25 < '30';
END $$
```

```
CREATE EVENT checkPriorityGroup
    ON SCHEDULE EVERY 30 MINUTE
    DO
      CALL updatePrioirtyGroup();
```

IX. MAXIMUM MATCHING ALGORITHM DESIGN AND DESCRIPTION

Adapted from code provided by Neelam Yadav (reference: https://www.geeksforgeeks.org/maximum-bipartite-matching/) I created an algorithm that optimizes the number of patients that will receive appointment offers via my website. It uses the intuition set forth by matching in a bipartite graph. Essentially the algorithm tries to create a matching bipartite graph in such a way that no two edges share an end point (i.e., conflicting appointment). The result is a maximum size (number of edges) such that I can determine there is enough appointments that meet all the existing patient's constraints (preferences and distance) that still do not have a scheduled appointment.

If there are enough appointments for all patients given their constraints (meaning the algorithm returned the maximum number of patients that can get an appointment as equal to the number of patients still waiting for an offer), we will offer them the maximum number of appointments by taking all the available appointments and dividing it by the number of patients seeking appointments. If this is not the case, we will offer them some arbitrary limited number of appointments (2 in the case of my website using Procedure 'MatchAvailableAppointmentsLimit') to ensure there is enough appointments for the rest of the patients. The algorithm can be set to run on a particular priority group or groups in the event we would want to maximize a particular patient category before offering appointments to the rest of the patients. The algorithm is dynamically run when a patient clicks the schedule appointment tab on the website the first time. Consequently, the algorithm will only be run for that patient again once they have accepted, declined, cancelled, or have expired all existing offers. Below is an outline of how the algorithm functions in the 'scheduleAppointment.php' file.

Figure 1 and figure 2 below:

Below is a graphical illustration of how the algorithm works using a flow diagram. The source is the pool of patients while the second set of circles represent their unique appointment preferences (identified by distance, patientId and timeslots). In figure 1 the patients and their preferences can then have outgoing edges to multiple appointments (represented by the 3rd set of circles) indicating appointments that meet the patients' constraints. Finally, the algorithm will take these preferences into account and try to figure out if there is a way to match the 2nd and 3rd sets in such a way that each patient can have an appointment that if it conflicts with another patient's appointment matched at least one of them will have an alternative appointment that is also not conflicting with another patient's appointment (figure 2).

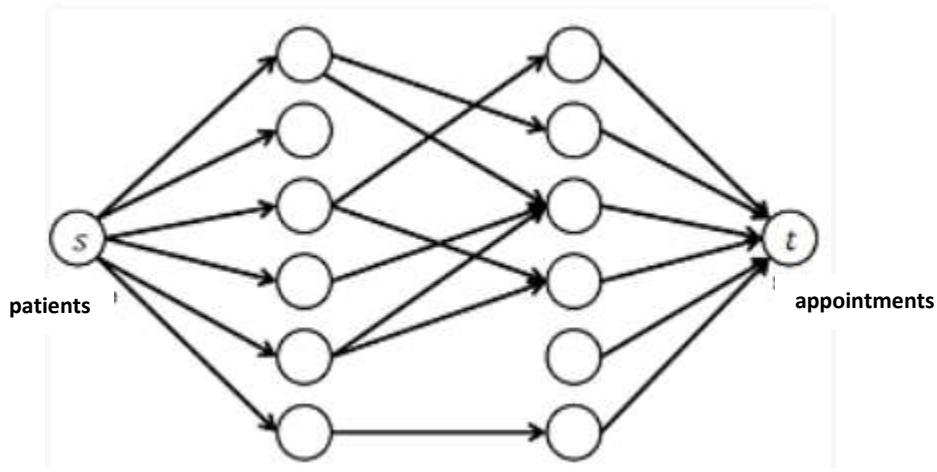
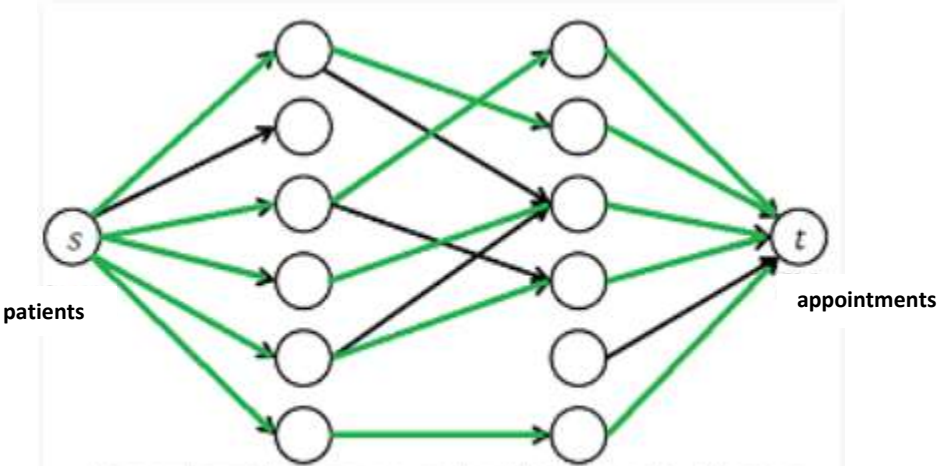


Figure 1 (adapted from: https://www.geeksforgeeks.org/maximum-bipartite-matching/)



The maximum flow from patient to appointment is therefore 5. A maximum of 5 people can get appointments which indicates appointment scarcity

Figure 2 (adapted from: https://www.geeksforgeeks.org/maximum-bipartite-matching/)

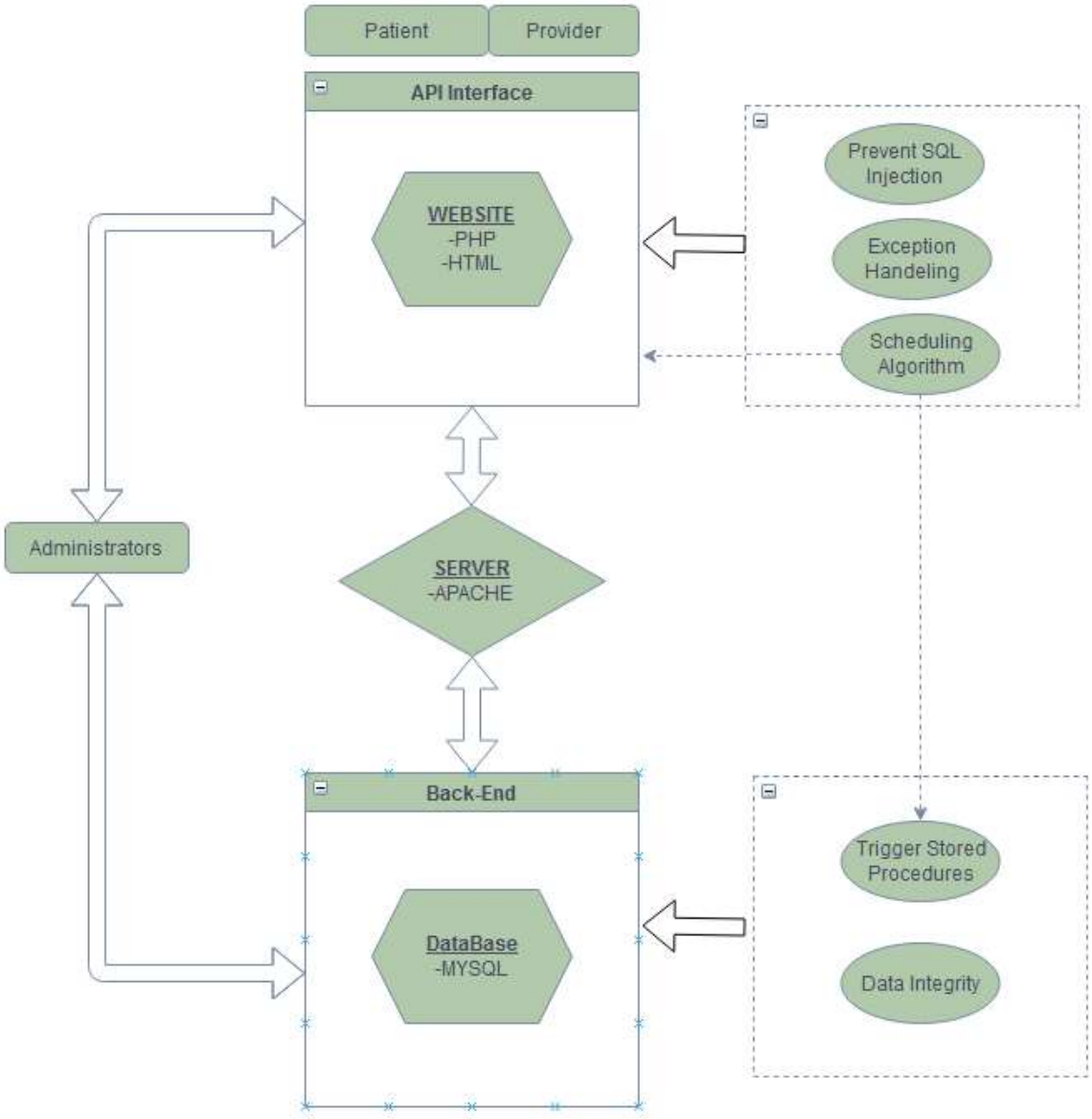
The algorithm takes in an argument which is equivalent to the above graph in the form of Edmonds matrix which is a 2D array 'Graph[M][N]' with M rows (for M patients) and N columns (for N appointments). The value Graph[i][j] is 1 if i'th patients preferences match to the j'th job, otherwise 0. Conflicting appointment preferences matched with available appointments can be seen highlighted in green below as an example. The algorithm will recursively check when there is a conflicting appointment if an alternative match can be found and if not will reduce the maximum number of people who can get an appointment due to this conflict.

```
$Graph = array(array(0, 1, 1, 0, 0, 0),
                array(1, 0, 0, 1, 0, 0),
                array(0, 0, 1, 0, 0, 0),
                array(0, 0, 1, 1, 0, 0),
                array(0, 0, 0, 0, 0, 0),
                array(0, 0, 0, 0, 0, 1));
```

Output :
Maximum number of patients that can get an appointment is 5

Finally given the output of our algorithm we can offer the maximum number of appointments available to patients (number of appointments divided by patients waiting to be scheduled) if the maximum number of patients that can get an appointment is equal to the number of patients seeking appointments. Otherwise, we limit the number offered using the stored procedure 'MatchAvailableAppointmentsLimit' discussed previously. The code for this algorithm can be found in the scheduleAppointment.php file.

(F) SYSTEMS, TOOLS AND APPLICATIONS FOR PROJECT IMPLEMENTATION



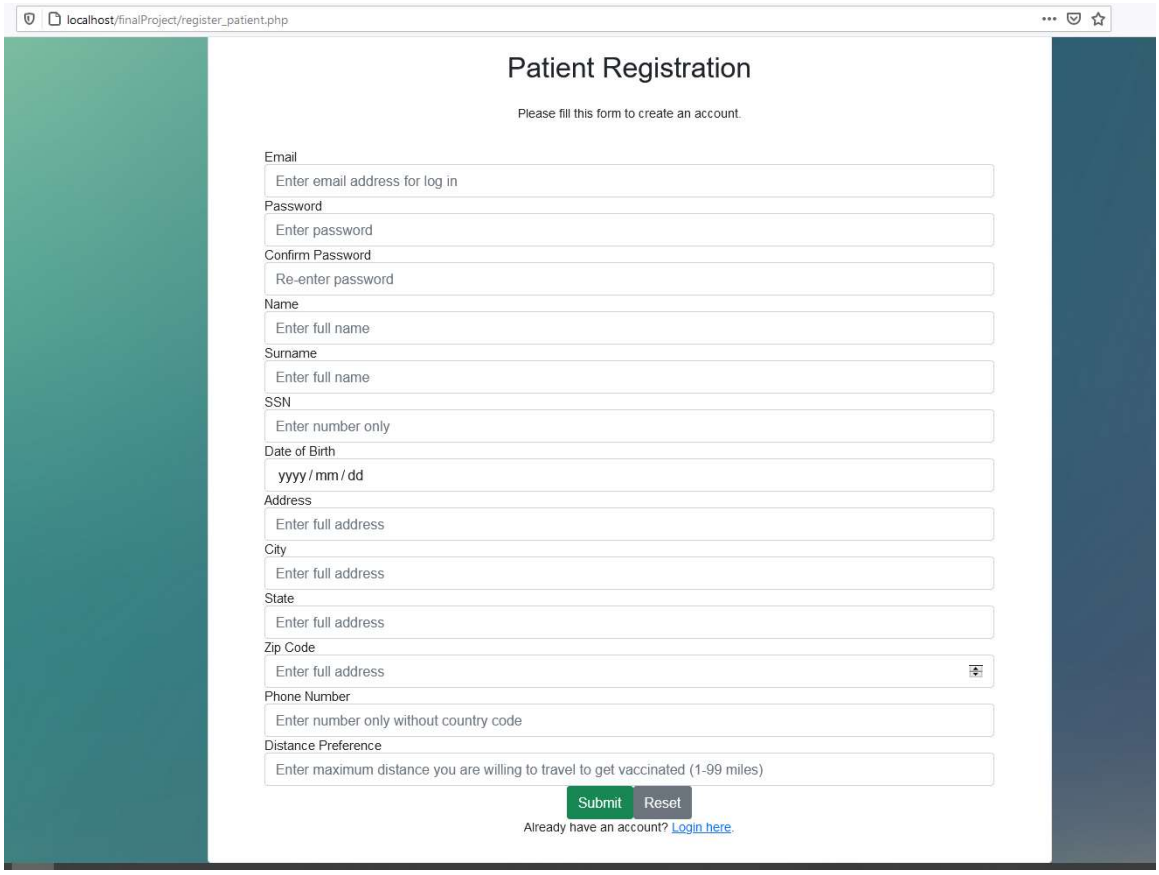
The above diagram outlines the service and backend model for the scheduling website. Front-end design was done using PHP and HTML with prepared statements and exception handling to prevent SQL injection and crashes. The website was hosted on an apache server which connects to the back-end database in MYSQL. Cross-Site Scripting site attack prevention is provided by apache by editing the configure file as such “Header always set X-XSS-Protection "1; mode=block". This prevents Cross-Site Scripting (Also known as XSS) which is a client-side attack by injecting malicious scripts to the web application. The integrity of the data is handled by stored procedures as well as backend exception handling. I will go through the front-end design of the website below outlining the backend interaction and handling as I give brief description. It is assumed administrators can interact with both the front end and back end to perform their tasks. Finally the use of html special chars is used in PHP to make sure data sent is sanitized when sending back to the website.

X. SERVICE & CONTROLLER: FUNCTIONAL DESIGN BACK-END FRONT-END INTERACTION

Register/Create Patient and Provider Accounts:

- From Index page user/patient can select sign up which will redirect to an html registration form to insert into the database via prepared statement.
- Checks weather users email already exists in the database and if not then add user to database
- Check's password requirements must be at least 7 characters long
- Check's weather registering user provides all information required for non-nullable fields in database.
- Encrypts user password using php function password_hash()
- The patient priority group is updated periodically by a stored procedure and an event every 30 minutes based on the age of the patients.

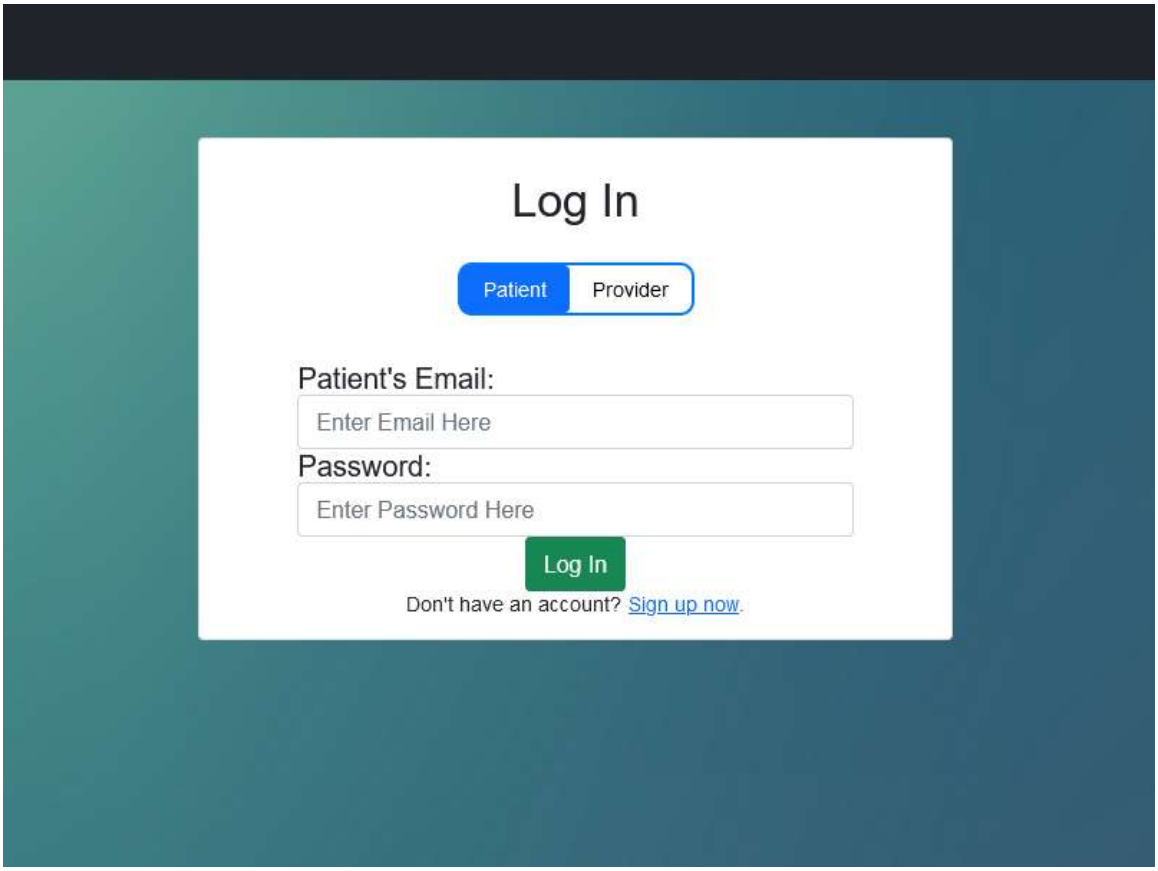
Relevant files: Index.php, register_patient.php, register_provider.php

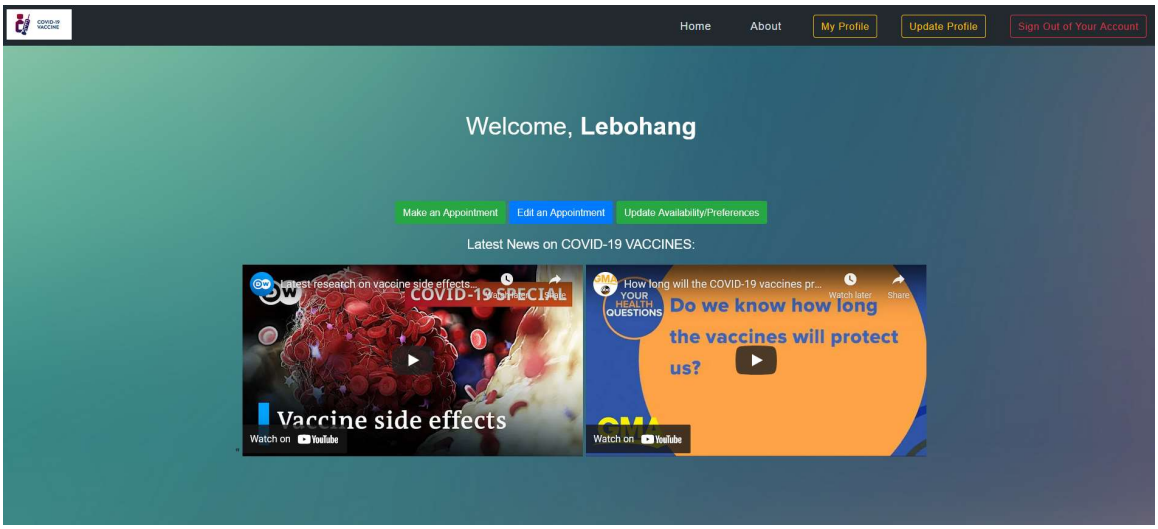


Patient and Provider Login and Authentication:

- Crosschecks user email and encrypted password with database if successful logs user in and starts a session that contains user/provider id.
- Successful login redirects to patient/provider homepage which displays latest news on COVID-19 vaccinations.
- User session id is checked on all webpages and redirect to index page user not supposed to be on a page.
- User session ended if logout clicked on any page.
- Home button will always redirect user back to their relevant homepage using the session id

Relevant files: Index.php, logout.php, patient.php, provider.php

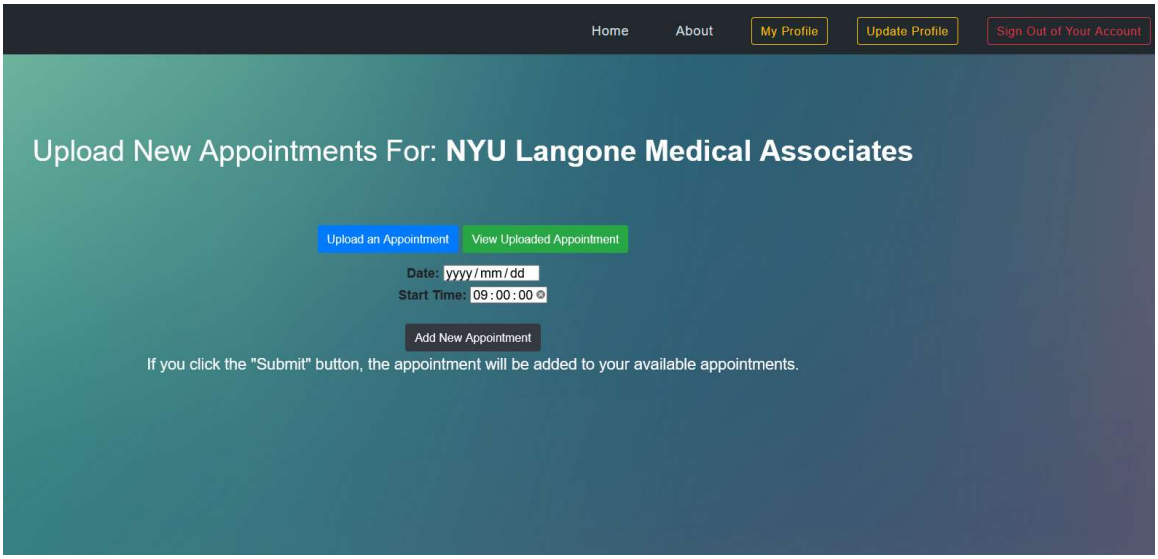




Providers upload appointments:

- HTML form used to request date and start time for a new appointment to be uploaded. If successful it is uploaded using prepared statements to the database.

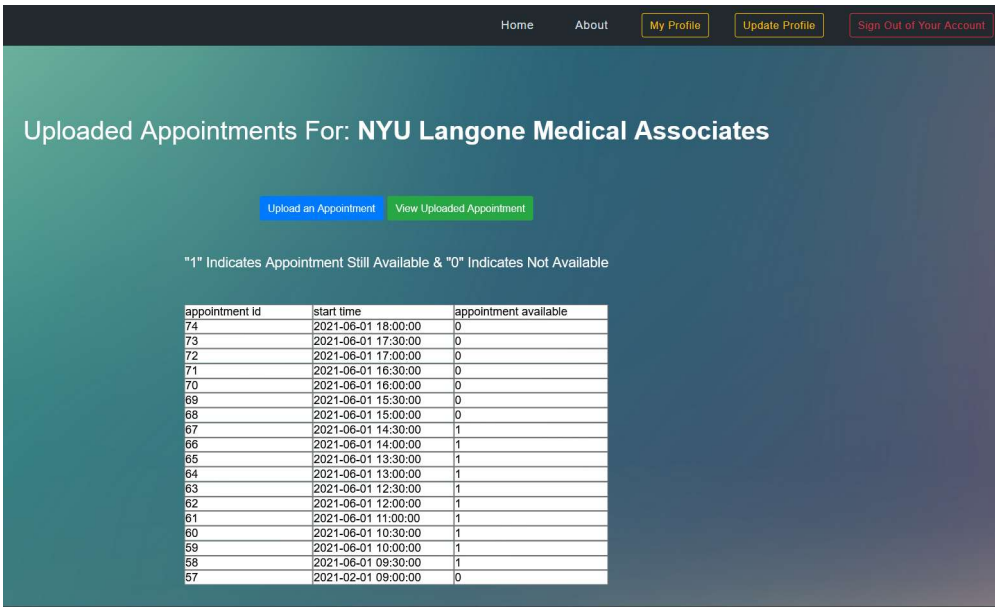
Relevant files: uploadAppointment.php, landing.php



Providers view uploaded appointments:

- Fetches and displays all provider appointments matching provider id in session and displays a table indicating if appointment is available or has been offered/scheduled.

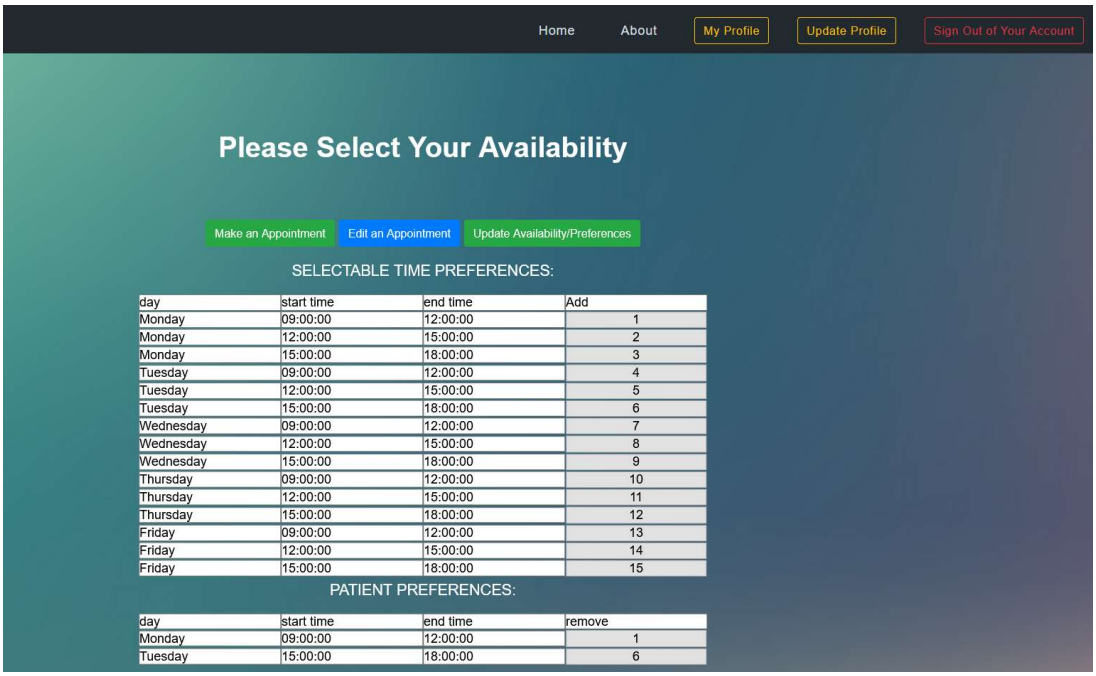
Relevant files: uploadAppointment.php, landing.php



Patients update availability/preferences:

- Fetches and displays all available calendar slots which are selectable to the user
- User can add/remove calendar slots to their preferences also displayed on the webpage which will dynamically update their preferences in the database. If user does not update the system will assume no time available.

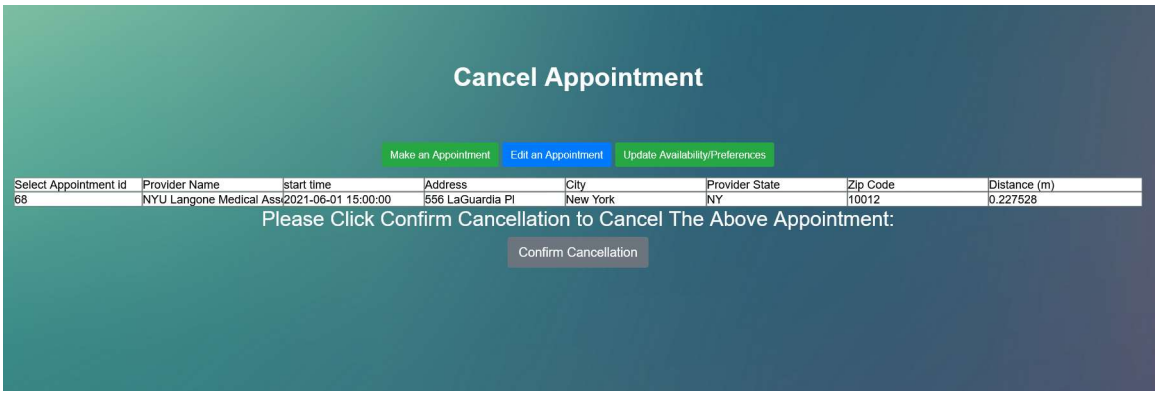
Relevant files: patientPreferences.php, landing.php



Patients edit accepted appointments:

- Fetches from database and displays all appointments which the user had accepted in the scheduling tab.
- User can cancel appointment which will update the status of the appointment in the database back to indicate it is available. This is done on the back end using triggers on the offered table which checks for updates and insertions.

Relevant files: patientPreferences.php, landing.php



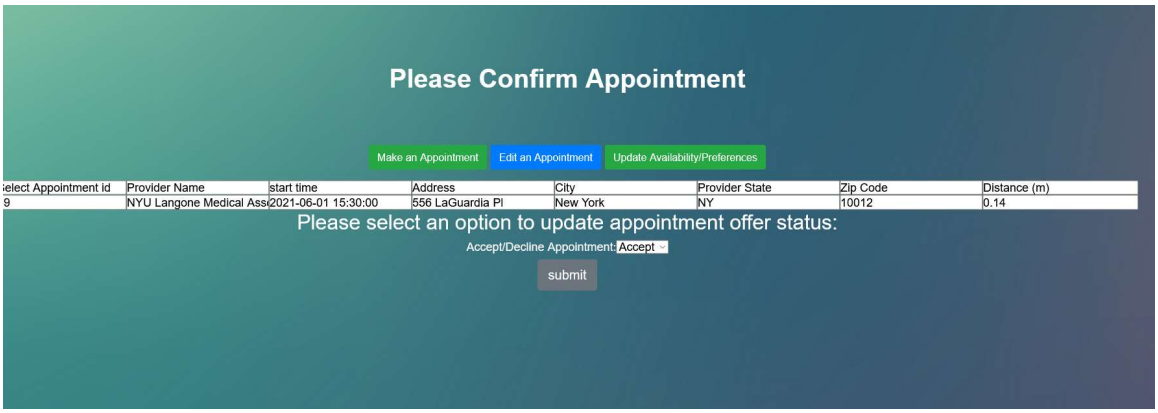
Patients Schedule appointments:

- When the user clicks the schedule tab the webpage first fetches a prepared statement that checks if the patient in session already has offers for appointments. If yes, then the page fetches their offers for appointments and displays a selectable list. The patient can now select appointments which will redirect them to acceptAppointment.php and they can accept or decline the appointment. The decision they make will update the database via prepared statements and use triggers to ensure the appointment status is updated in the appointments table.
- If the user does not have any existing appointment offer after checking it will run the maximum matching algorithm and determine how many appointments to

offer the patient. The appointments will be displayed and updated in the offer table which can then be accepted o declined. Currently if there are not enough appointments for all patients the algorithm will limit the number of offers to 2. The algorithm can be modified to either run for a particular priorityGroup or groups or for all patients seeking offers that do not already have offers.

- The appointments have an expire time on the after which they will be made available again to other patients. This is set arbitrarily for the prototype to 10 days after the offer was made.

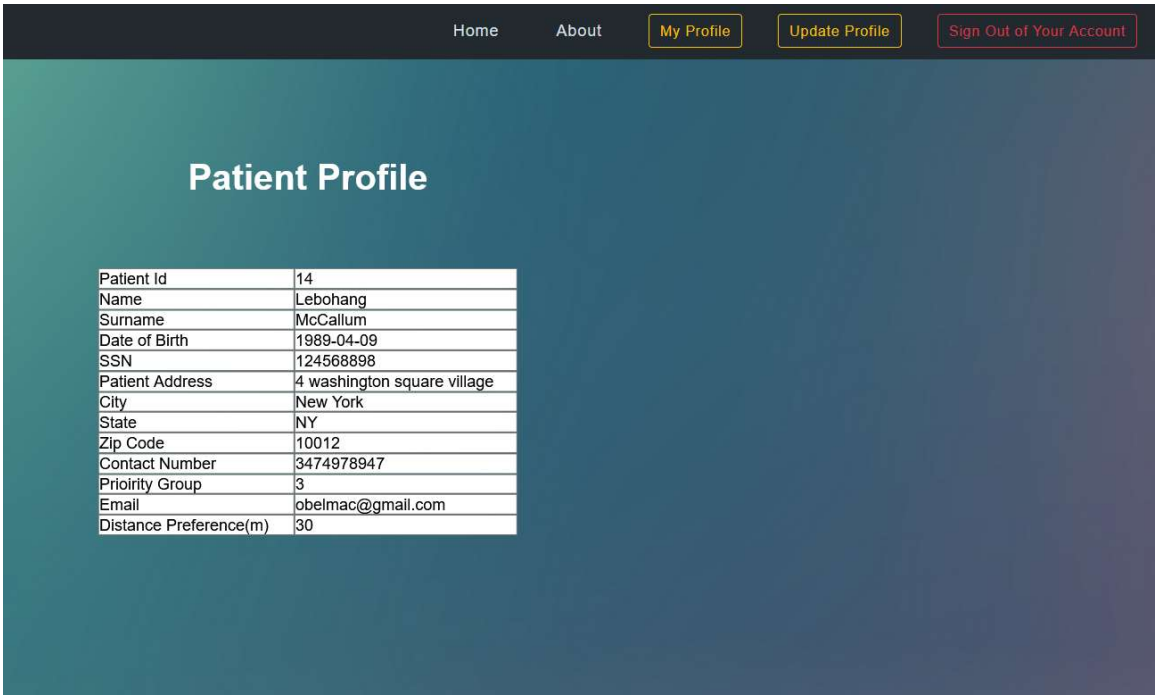
Relevant files: scheduleAppointment.php, patientAcceptAppointment.php, patientAcceptDeclineAppointment.php, landing.php



Patients view profile details:

- The patient view profile tab fetches and displays all information relevant to the session id of the patient and displays this for them to see.

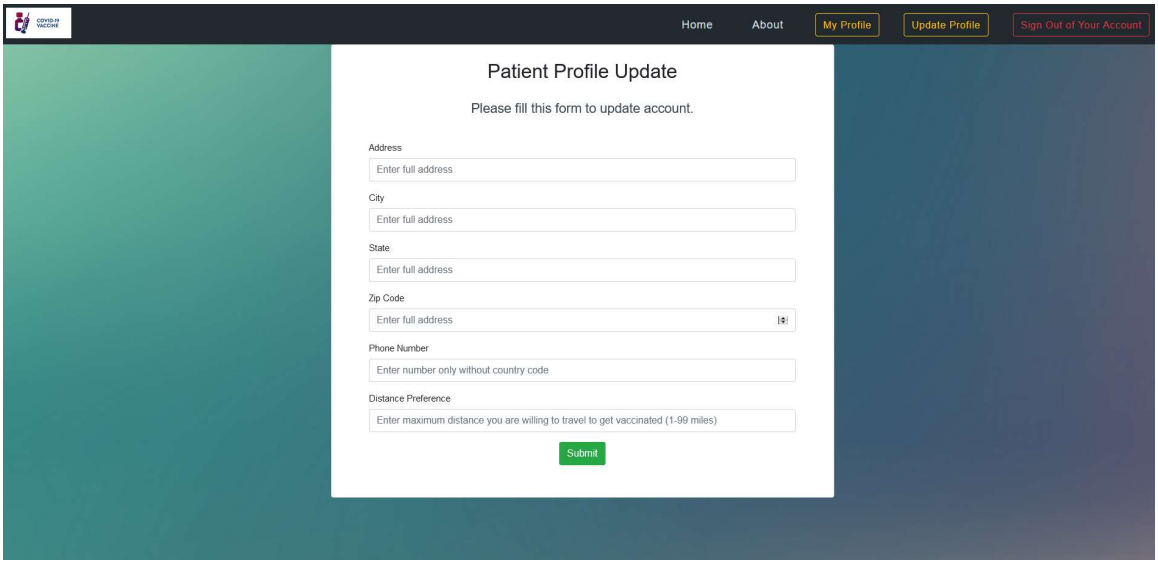
Relevant files: viewProfile.php, landing.php



Patients update profile:

- The patient update profile provides an html form in which users can update relevant details about themselves such as address and distance preference for appointments. These are updated via a prepared statement in into the database.
- Some patient details are not included and are assumed to not be editable for example the name and SSN as well as patient email since these are unique to a patient and generally immutable. For example, email is used for login so should not be easily changeable. If the patient would like to change these details if there was a mistake, they would need to contact the administrators.

Relevant files: patientUpdateProfile.php, landing.php


Home
About
My Profile
Update Profile
Sign Out of Your Account
Patient Profile Update
Please fill this form to update account.
Address
Enter full address
City
Enter full address
State
Enter full address
Zip Code
Enter full address
Phone Number
Enter number only without country code
Distance Preference
Enter maximum distance you are willing to travel to get vaccinated (1-99 miles)
Submit

(G) Algorithm References

References used to modify and create my scheduling algorithm:

http://www.cs.cornell.edu/~wdtseng/icpc/notes/graph_part5.pdf
http://www.youtube.com/watch?v=NlQqmEXuiC8
http://en.wikipedia.org/wiki/Maximum_matching
http://www.stanford.edu/class/cs97si/08-network-flow-problems.pdf
http://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/07NetworkFlowII-2×2.pdf
http://www.ise.ncsu.edu/fangroup/or766.dir/or766_ch7.pdf