Course Project—Web-based System for Signing up People for COVID-19 vaccinations

Name: Jiale Dai

ID: jd4678

Backgroubd

The gole of our course project is to build a web-based system for signing up people for COVID-19 vaccinations.

Problem Outline

In the completed system, there will be three type of participants: **Patients, providers, and administrators**. Patients can sign up in the system, and providenecessary information such as name, SSN, date of birth, address, phone, and email, and maybe additional information that can be used to determin the priority group the patient belongs to. Patients may also indicate preferred times during the week when they would like to be scheduled for the vaccination.

Providers need to sign up with information such as their 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 ahs serval locations, they need to register as different providers.

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 prociders about appointments.

To describe the process in more detail, patients sign up by providing their information, and receive a username (say, their email) and password. Their priority group in initially set to NULL, unitl the administrator assigns them to a group. Priority groups are numbered from 1 to some number N, so that group #1 has the highest priority, #2 the next highest, and so on. The administrator aslo maintains for each group a date when they qualify for vaccination, say January 1 for group #1, January 15 for group #2, etc. Patients also provider a "calendar" of their availability during the week. It is recommended to keep this feature simple — e.g., you could split each day into 4-hour blocks, and patients can select blocks where they are most likely to be available or that are convenient for them. Thus, a patient might select every weekday 8am-noon, or Tuesday noon-4pm and Friday 8am-noon, or any thiem on the weekend. Patients aslo provide a maximum distance they are willing to travel to get to the appointment, e.g., 30 miles.

Providers that have signed up periodically upload a available appointments to the system. Note that for larger providers, there may be several appointments at the same time. The administrator periodically runs an algorithm that tries to match available appointments with patients, in a way that takes eligibility, priority, availability, and distance into account. Afterwards, selected patients will be notified to offer them the the appointment, and the patients will have some limited amount of time to either accept or decline the appointment. If the patient accepts, the provider needs to be notified about the patient having accepted the appointment. An appointment may also be cancelled later, or a patient may fail to show up. All this information about what offers were sent to patients, whether they accepted or declined, if they cancelled later or did not show up, should also be stored in the database.

Assumptions

1. For simplicity, we omit any issues of payments or insurance, and assume vaccinations are free.
2. We aslo assume that administrator has some way to assign people to the correct priority group; e.g., assume that patients can upload additional documents as PDF or image files that the administrator can use, either manually or algorithmically, in comination with age, to determine the priority group.
3. You do not have to model different types of providers, such as hospitals or doctors, beyound maybe an attribute *providertype* for each procider.
4. You can split each day into 4-hour blocks, and patients can select blocks where they are most likely to be available or that are convenient for them.
5. Scheduler is same every week, though patients could later decide to update their schedule,.
6. For larger providers, there may be several appointments at the same time.
7. For now, assume that the system periodically (say, once every few hour) runs an algorithm that matches currently available apointments (new appointments, or appointments ont confirmed, rejected, or canceled by other patients) with eligible patients.

(a) Design

Entity

- patient

- provider
- appointment
- patient_scheduler
- group_date
- appointment_log
- distance

Relations

- provider upload appointment → one to many
- patient provide time slots → one to many
- patients are assiened to priority group → many to one
- patient matched with appointment → one to one
- the distance between patient and provider → one to one

Tables

patient(id, name, ssn, birth, address, phone, email, additional_info, max_distance, password, priority, status)

provider(id, name, phone, address,username, password)

appointment(id, provider_id, start_time, end_time, patient_id)

patient_scheduler(patient_id, start_time, day_of_the_week)

group_date(priority, date)

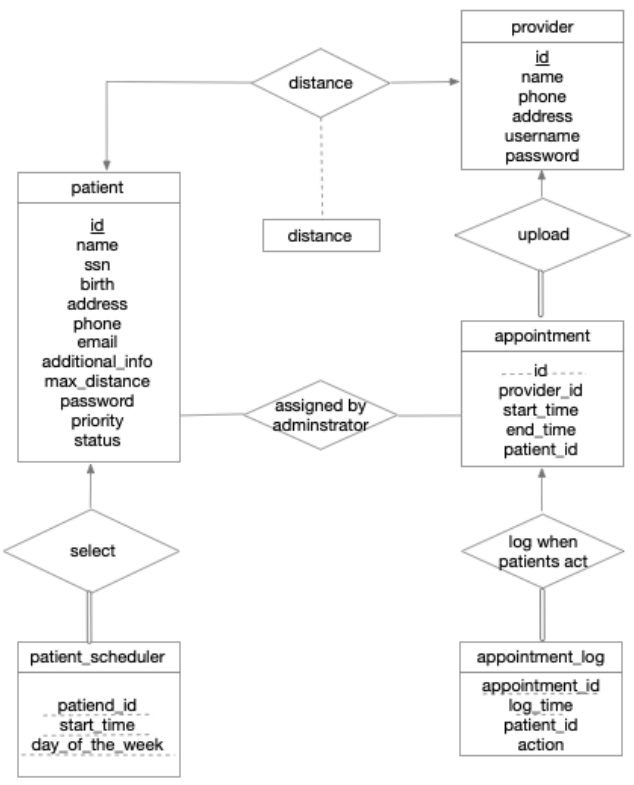
appointment_log(appointment_id, log_time, action)

distance(provider_id, patient_id, distance)

Foreign Keys

- appointment.provider_id references provider.id
- appointment.patient_id references patient.id
- appointment_log.appointment_id references appointment.id
- patient_scheduler.patient_id references patient.id
- distance.provider_id references provider.id
- distance.patient_id references patient.id

E-R Diagram



(b) Database Schema

patient:

DESCRIPTION: patient table is used to hold the informaiton about the patient including the basic information (name, ssn, brith, address, phone, email), the information needed to provide after signing up (additional_info, max_distance, password), the information assigned by the system administrator (priority), and the information maintained by the server (status). All the information beside the which used to sign up is optional. Those information can be null or the specified value at the beginning. Status stands for what stage the patient currently is, with the default value "waiting", which means once the patient signed up, the system automatically mark the patient as "waiting for making the schedule".

```
CREATE TABLE `patient` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(30) NOT NULL,
  `ssn` varchar(9) NOT NULL,
  `birth` date NOT NULL,
  `address` varchar(50) NOT NULL,
  `phone` varchar(11) NOT NULL,
  `email` varchar(50) NOT NULL,
  `additional_info` text,
  `max_distance` int DEFAULT NULL,
  `password` varchar(100) NOT NULL,
  `priority` int DEFAULT NULL,
  `status` varchar(30) NOT NULL DEFAULT 'waiting' COMMENT 'waiting,scheduled, received',
  PRIMARY KEY (`id`),
  CONSTRAINT `patient_chk_1` CHECK ((((`max_distance` > 0) and (`priority` > 0)))
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

provider

DESCRIPTION: provider table is used to record the information about the provider. It just hold the basic information like name, phone and address because we assume the provider do not have to have a account. The provider always have some ways access to the administrator and provide these basic information to the administrator. And then, the administrator will add the provider's information into the database.

```
CREATE TABLE `provider` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(30) NOT NULL,
  `phone` varchar(11) NOT NULL,
  `address` varchar(100) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

appointment

DESCRIPTION: appointment table is used to record the information . It use id as primary key instead of using the combination of provider_id, start_time, and end_time, because there exists a situation where a large hospital can and handle several appointments at the same time slot. In the table, the patient_id identifies the patient that is currently make the schedule with this appointment. The patient_id can be NULL, means the appointment haven't been occupied by anyone.

```
CREATE TABLE `appointment` (
  `id` int NOT NULL AUTO_INCREMENT,
  `provider_id` int NOT NULL,
  `start_time` timestamp NOT NULL,
  `end_time` timestamp NOT NULL,
  `patient_id` int DEFAULT NULL,
  `status` varchar(30) DEFAULT 'released' COMMENT 'release, assigned, confirmed',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

patient_schedule

DESCRIPTION: The patient_schedule records the information of about the scheduler of the patient. When a patient chooses his or her time preference online, the system will insert the corresponding data to the database. Since our projcet assume the scheduler is the time slots with 4 hours long, the table just store the start_time using a int value, which represents the hour. And we can easily get the end_time by using start_time plus 4. The attribute "day_of_the _week" is also stored using a int value. (1:sunday,2:monday,3:tuesday,4:wednesday,5:thursday,6:friday,7:sunday)

```
CREATE TABLE `patient_schedule` (
  `patient_id` int NOT NULL,
  `start_time` int NOT NULL,
  `day_of_the_week` int NOT NULL COMMENT '1:sunday,2:monday,3:tuesday,4:wednesday,5:thursday,6:friday,7:saturday',
  PRIMARY KEY (`patient_id`,`start_time`,`day_of_the_week`),
  CONSTRAINT `patient_scheduler_patient_id_fk` FOREIGN KEY (`patient_id`) REFERENCES `patient` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

appointment_log

DESCRIPTION: appointment_log table keep the every operation of patients on appointments. The appointment_id refers to the id attribute in appointment table, which can be used to join appointment table to get the detail infomation. The log_time is the the time when the information is logged into the table. The attribute patient_id records which patient do the change when the log happens.

```
CREATE TABLE `appointment_log` (
  `appointment_id` int NOT NULL,
  `action` varchar(30) NOT NULL,
  `log_time` timestamp NOT NULL,
  `patient_id` int NOT NULL,
  PRIMARY KEY (`log_time`,`appointment_id`),
  KEY `appointment_log_appointment_id_fk` (`appointment_id`),
  KEY `appointment_log_patient_id_fk` (`patient_id`),
  CONSTRAINT `appointment_log_appointment_id_fk` FOREIGN KEY (`appointment_id`) REFERENCES `appointment` (`id`),
  CONSTRAINT `appointment_log_patient_id_fk` FOREIGN KEY (`patient_id`) REFERENCES `patient` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

distance

DESCRIPTION: distance table keep the record the distance btween a patient and a provider. The system updates the distance table whenever a patient or provider sign up into the system.

```
CREATE TABLE `distance` (
  `patient_id` int NOT NULL,
```

```
  `provider_id` int NOT NULL,
  `distance` double NOT NULL,
  PRIMARY KEY (`patient_id`,`provider_id`),
  KEY `distance_provider_id_fk` (`provider_id`),
  CONSTRAINT `distance_patient_id_fk` FOREIGN KEY (`patient_id`) REFERENCES `patient` (`id`),
  CONSTRAINT `distance_provider_id_fk` FOREIGN KEY (`provider_id`) REFERENCES `provider` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

(c) SQL queries

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

```
INSERT INTO patient (name, ssn, birth, address, phone, email, password)
                    VALUES
                    ("David","418194184","1998-04-07", "225 Cherry St, New York", "6462400058", "test1@nyu.edu","pwdtest1");
```

2. Insert a new appointment offered by provider

```
INSERT INTO appointment (provider_id, start_time, end_time)
                        VALUES
                        (1, "2021-04-21 13:30:00", "2021-04-21 17:30:00");
```

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.

```
# assume the id of the given patient is 1
with tmp as (select appointment.provider_id as provider_id, appointment.start_time as start_time, appointment.end_time as end_time from patient_schedule,appointment
where appointment.patient_id is null and dayofweek(appointment.start_time)=day_of_the_week
  and ((hour(appointment.start_time)<=patient_schedule.start_time and hour(appointment.end_time)>=patient_schedule.start_time + 4)
      or (hour(appointment.start_time)>=patient_schedule.start_time and hour(appointment.start_time)<=patient_schedule.start_time+4)
      or (hour(appointment.end_time)>=patient_schedule.start_time and hour(appointment.end_time)<=patient_schedule.start_time+4)
      or (hour(appointment.start_time)>=patient_schedule.start_time and hour(appointment.start_time)<=patient_schedule.start_time+4)))
select * from tmp natural join distance where patient_id = 1 order by distance;
```

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 a appointment.

```
with tmp as ( select id,s.status as status,priority from (select distinct status from patient) as s, patient )
select tmp.priority as priority, count(if(patient.status='received',1,NULL)) as received, count(if(patient.status='waiting',1,NULL)) as waiting,
count(if(patient.status='scheduled',1,NULL)) as scheduled
from tmp left outer join patient on tmp.status=patient.status and tmp.id=patient.id
group by tmp.priority;
```

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

```
# wheter the patient is eligible is based on his of her priority
select distinct name,date from patient join group_date on patient.priority=group_date.priority;
```

6. Output 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 tmp as (select distinct patient_id from appointment_log
group by patient_id, action
having count(action="cancelled")>=3 or count(action="absent")>=2)
select name from tmp join patient on patient_id=patient.id;
```

7. Output the name of the provider that has performed the largest number of vaccinations.

```
with tmp as (select provider_id, count(action) as count
from appointment_log join appointment on appointment_id = id group by provider_id,action
having action="completed")
select id, name from provider join (select provider_id from tmp where count=(select max(count) from tmp)) as subtable
on provider.id=subtable.provider_id;
```

(d) Populate your database with some sample data, and test the queries you have written in part(c). Make sure to input interesting and meaningful data. Limit yourself to a few patient, providers, priority classes, etc. , but make sure there is enough data to generate interesting test case for the above queries. It is suggented that you design your test data very carefully. Draw and submit a little picture of your tables that fits on one or two pages and that illustrates your test data!

Data:

patient

Aa id	name	ssn	birth	address	phone	email	additional_info	max_distance	password	priority	status

id	name	ssn	birth	address	phone	email	additional_info	max_distance	password	priority	status
1	David	418194184	1998-04-07	225 Cherry St, New York, NY	6462400058	mrlibiiuyj@iubridge.com	NULL	NULL	pwdtest1	1	waiting
2	Tony	321351674	2009-04-20	252 Sough st, New York, NY	6492400088	zsnkkgvkkk@iubridge.com	NULL	NULL	pwdtest2	2	scheduled
3	Song	319438913	2013-04-21	3049 Elliot Avenue, Seattle, WA	7471849891	porknjelet@iubridge.com	NULL	NULL	pwdtest3	3	received
4	Tom	393478991	2013-04-21	2651 Kembery Drive, Aurora, IL	3928493814	bmgqnrblsq@iubridge.com	NULL	NULL	pwdtest4	1	received

provider

id	name	phone	address	password
1	CVS	6462849914	350W 88th st, New York	NULL
2	The Brooklyn Hospital Centre	7182508000	121 Dekalb Ave, Brooklyn	NULL
3	NYU Langone Medical Center	6469297870	14 Wall St, New York, NY 10005	NULL

appointment

id	provider_id	start_time	end_time	patient_id	status
1	1	2021-04-21 13:30:00	2021-04-21 17:30:00	NULL	released
2	2	2021-04-21 13:30:00	2021-04-21 17:30:00	NULL	released
3	3	2021-04-19 15:30:00	2021-04-19 20:30:00	NULL	released

distance

patient_id	provider_id	distance
1	1	30
1	2	40
1	3	25
2	1	11
2	2	41
2	3	25
3	1	15
3	2	61
3	3	31
4	1	52
4	2	65
4	3	41

group_date

priority	date
1	2021-04-23
2	2021-04-24
3	2021-04-22

patient_scheduler

patient_id	start_time	day_of_the_week
1	12	4
1	16	2

appointment_log

appointment_id	action	log_time	patient_id
1	cancelled	2021-04-21 06:52:59	1
1	cancelled	2021-04-21 06:53:22	1
1	cancelled	2021-04-21 06:53:38	1
1	absent	2021-04-21 06:54:15	1
1	absent	2021-04-21 06:54:28	2
1	absent	2021-04-21 06:54:44	2
2	cancelled	2021-04-21 06:54:58	3
2	completed	2021-04-21 07:08:07	3
3	completed	2021-04-21 07:15:31	4

Query Result:

Note: Here I pass (1) and (2) in Part C which are all Insertion operations since I have already put the full table above.

(3) in part C: 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.

```
mysql> # assume the id of the given patient is 1
mysql> with tmp as (select appointment.provider_id as provider_id, appointment.start_time as start_time, appointment.end_time as end_time from patient_scheduler,appointment
    -> where appointment.patient_id is null and dayofweek(appointment.start_time)=day_of_the_week
    ->   and ((hour(appointment.start_time)<=patient_scheduler.start_time and hour(appointment.end_time)>=patient_scheduler.start_time + 4)
    ->       or (hour(appointment.start_time)>=patient_scheduler.start_time and hour(appointment.start_time)<=patient_scheduler.start_time+4)
    ->       or (hour(appointment.end_time)>=patient_scheduler.start_time and hour(appointment.end_time)<=patient_scheduler.start_time+4)
    ->       or (hour(appointment.start_time)>=patient_scheduler.start_time and hour(appointment.start_time)<=patient_scheduler.start_time+4)))
    -> select * from tmp natural join distance where patient_id = 1 order by distance;
+-------------+---------------------+---------------------+------------+----------+
| provider_id | start_time          | end_time            | patient_id | distance |
+-------------+---------------------+---------------------+------------+----------+
|           3 | 2021-04-19 15:30:00 | 2021-04-19 20:30:00 |          1 |       25 |
|           1 | 2021-04-21 13:30:00 | 2021-04-21 17:30:00 |          1 |       30 |
|           2 | 2021-04-21 13:30:00 | 2021-04-21 17:30:00 |          1 |       40 |
+-------------+---------------------+---------------------+------------+----------+
```

(4) in part C: 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 a appointment.

```
mysql> with tmp as ( select id,s.status as status,priority from (select distinct status from patient) as s, patient )
    -> select tmp.priority as priority, count(if(patient.status='received',1,NULL)) as received, count(if(patient.status='waiting',1,NULL)) as waiting, count(if(patient.status='scheduled',1,NULL)) as scheduled
    -> from tmp left outer join patient on tmp.status=patient.status and tmp.id=patient.id
    -> group by tmp.priority;
+----------+----------+---------+-----------+
| priority | received | waiting | scheduled |
+----------+----------+---------+-----------+
|        1 |        1 |       1 |         0 |
|        2 |        0 |       0 |         1 |
|        3 |        1 |       0 |         0 |
+----------+----------+---------+-----------+
3 rows in set (0.00 sec)
```

(5)in part C: For each patient, output the name and the date when the patient becomes eligible for vaccination.

```
mysql> select distinct name,date from patient join group_date on patient.priority=group_date.priority;
+-------+------------+
| name  | date       |
+-------+------------+
| David | 2021-04-23 |
| Tony  | 2021-04-24 |
| Song  | 2021-04-22 |
| Tom   | 2021-04-23 |
+-------+------------+
4 rows in set (0.01 sec)
```

(6)in part C: Output 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.

```
mysql> with tmp as (select distinct patient_id from appointment_log
    -> group by patient_id, action
    -> having count(action="cancelled")>=3 or count(action="absent")>=2)
    -> select name from tmp join patient on patient_id=patient.id;
+-------+
| name  |
+-------+
| David |
| Tony  |
+-------+
2 rows in set (0.01 sec)
```

(7)in part C: Output the name of the provider that has performed the largest number of vaccinations.

```
mysql> with tmp as (select provider_id, count(action) as count
    -> from appointment_log join appointment on appointment_id = id group by provider_id,action
    -> having action="completed")
    -> select id, name from provider join (select provider_id from tmp where count=(select max(count) from tmp)) as subtable
    -> on provider.id=subtable.provider_id;
+----+-----------------------------+
| id | name                        |
+----+-----------------------------+
|  2 | The Brooklyn Hospital Centre |
|  3 | NYU Langone Medical Center   |
+----+-----------------------------+
2 rows in set (0.01 sec)
```

(e) Document and log your design and testing appropriately. Submit a properly documented description and justification of your entire design, including E-R diagrams, tables, constraints, queries, procedures, and tests on sample data, and a few pages of description. This should be a paper of say 10-12 pages with introduction,explanations, E-R and other diagrams, etc., which you will then revise and expand in the second part.

NOTE: For the display the patient table fully, I print the document in A3 page, it has 7 pages because everything is scaled smaller. If in A4 format it is more then 10 pages.

I think what question(e) ask is a little duplicate with the previous question. I have answered the previous question in detail about the E-R diagrams, tbales, constraints, queries, procedures, and tests on sample data. please the refer the previous questions. Thanks!

- E-R diagram→ question (a)
- tables, constraints → question(a) & (b)
- query→ question(c)
- proedure→ question(e) below
- tests on sample data→ question(d)

Procedure

patients → sign up → select their available time slots+provide maximum distance → get notification → accept (provider get notified)/cancel/absent → info stored in the database & sent to patients

providers → sign up → upload available appointments(one to many relationship) → get notified when patients accept appointment

administrator → define priority groups → assign patient to a group → match the available appointments with patients → sent notification to patients and provider.

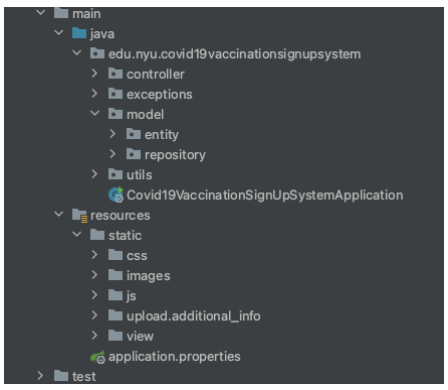
(f) Implementation

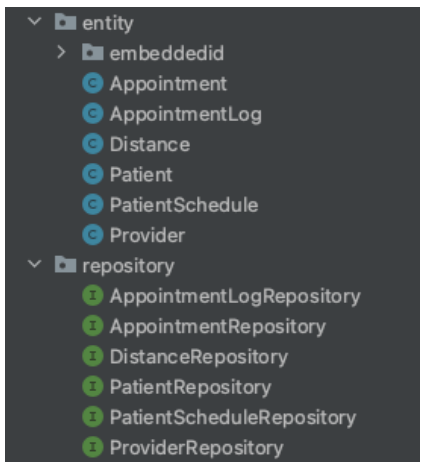
The project is implemented by using html, css, jQuery to build the front end and using Java Spring Boot to build the back end. I choose MySql as the database of the project. And back end is using Spring Data Jpa to do the operation on database. I will tell a little more about the tech stack I used below:

Key Tech Stack:

Spring MVC

I utilize SPring MVC to do the detailed application logic. MVC stands for Model- View- Controller, which is standard to design a website. The advantage of using MVC standard is that the user interface represent the content can be easily divided from the actual implemention of the project. Which is very clear for me contrate my work in certain part.





On the left side is the file structure of my project. The three main parts are the model, view and controller folder.

The model folder contains the entities and the respository. Entity is the Object mapped by the table in database. and the resposity is the data access model by which we can access and modify the data in the table.

The view folder stores all the html files in the project, it is belong to the parent folder resources which contains all the static resources such as css files, js files, images and the upload folder used to store the uploaded files.

The Controller folder contains the service part of the project. The Java files that reveive the request from the front and process the request and maybe return the corresponding response.

Restful API

Most of the api in the program is follow the standard of restful api. REST is an architectural stand for designing the APIs of Web service, REST describes how the API should look like. REST API is based on HTTPs. The difference between REST API and HTTP API is REST API should have a Resource as a targeting objection server, and exposed via URL in well organized directory structure. While HTTP API may not have resource can be less standard in URL format, and can be a general command. Usuually, GET request stands for fetch the data, POST request stands for add data, DELETE request stands for remove the data, and PATCH/PUT request means to do the update operation on the data.

Implementation Detail:

Below are some feature that I want to mention about how I implement in program.

Algorithm to Assign Appointment to Patient

The goal of the algorithm is to maximize the number of patients that can get an appointment offer, and it also mentioned that the algorithm should probably maximize the number of patients of the highest priority group that get an appointment offer, and only assign any unmatched appointment to lower prority patients. So I plan to select all the patients whose schedule matches the time slots of the appointment and haven't been assigned to any appointment. And then I will reorder the result by the patient's priority. Then I will beginning with the patients in the highest priority group and assign them the available appointment starting with the earliest date.

In this case, every person should be assigned a appointment which garantee the maximum number of patients getting appointments. And the assign work starts from the heightest priority and the earlist date, which ensure to maximize the number of patients of the highest priority group that get an appointment offer.

The code of this function is shown below. First of all, the function should be automatically run in back end in a period. The Spring framwork has the annation @Schedule, which let the program recognize the function and run it at the specified point. The cron expression is pass in the schedule annationa which can reglate the schedule rule. In my code, I use "0 30 1 ? * *" which ask the program run this function at 1:30 am every day.

The main idea to assign appointment is:

1. Go through the whole appointment table and find all appointment that haven't assign to any patients.
2. Find all patient alone with all the appointment whose time slot satisfy the patient schedule and the distance is less than the max_distance the patient can accept, sort the result based on the patient's priority in accending (from #1).
3. For every item in the result of step 1, we go through the all patient in the result of step 2 to see if the appointment can be assigned to this certain patient. We mainly check if the patient has been assigned in the previous day or if teh patient has been assigned in the previous step of the loop (since the result in step 2 will contains several records with same patientId and differet appointment, if patient is offered an appointment in the first record, we should ignore the rest of the records.

```
@Scheduled(cron = "0 30 1 ? * *") // 1:30 am every day
public void assignAppointment() {
    Set<Integer> patientSet = new HashSet<>();
    List<Map<String, Object>> result = appointmentRepository.findAvailablePatientAndAppointment();
    List<Appointment> appointmentList = appointmentRepository.findAll();
    Set<Integer> assignedPatientId = getAssignedPatientId(appointmentList);
    for (Appointment appointment : appointmentList) {
```

```
        if (appointment.getStatus().equals("released")) {
            Integer appointmentId = (Integer) appointment.getId();
            List<Map<String, Object>> tmp = filterByAppointment(appointmentId, result);
            for (Map<String, Object> each : tmp) {
                Integer patientId = (Integer) each.get("patient_id");
                if (patientSet.add(patientId) && !assignedPatientId.contains(patientId)) {
                    Integer providerId = (Integer) each.get("provider_id");
                    Timestamp startTime = (Timestamp) each.get("start_time");
                    Timestamp endTime = (Timestamp) each.get("end_time");
                    appointmentRepository.save(new Appointment(appointmentId, providerId, startTime, endTime, patientId, "assigned"));
                    Patient patient = patientRepository.findById(patientId).orElseThrow(()->new PatientNotFoundException(patientId));
                    patient.setStatus("scheduled");
                    patientRepository.save(patient);
                    break;
                }
            }
        }
    }
    System.out.println("===============Auto Assign Appointment================");
}
```

Maintain the Distance of Patient and Provider

The system require to maintain the distance between patient and provider. I think there are two choice: one is we calculate the distance when we need it, and another is we store the patientId, providerId and the distance in a distance table and keep updating the table whenever a new patient or provider registered, or anyone change address. The first choice did save the storage resources but it is harder to implement and a little time consuming to calcuate the distance every time. So, I choose the second choice which is keep a distance table and maintain it.

We need to get the distance before storing it into the distance table. I choose to use google map api to geocode the String-format address information into lantitude and longitude. And then use the geocoding result to do the distance calculation.

```
// geocode the address into lantitude and longitude by google map
public static Map<String, Object> getGeoCodingInfoFromAddress(String address) throws URISyntaxException, IOException {
    RestTemplate restTemplate = new RestTemplate();
    final String baseUrl = "https://maps.googleapis.com/maps/api/geocode/json?address=" + formatAddress(address) + "&key="+ PropertiesUtil.getProperty("google-map-apk");
    URI uri = new URI(baseUrl);
    ParameterizedTypeReference<Map<String, Object>> typeRef = new ParameterizedTypeReference<Map<String, Object>>() {
    };
    ResponseEntity<Map<String, Object>> result = restTemplate.exchange(uri, HttpMethod.GET, null, typeRef);
    return (Map<String, Object>) ((Map<String, Object>) ((List<Map<String, Object>>) result.getBody().get("results")).get(0).get("geometry")).get("location");
}

private static String formatAddress(String address) {
    address = address.trim().replaceAll(" +", " ").replaceAll("\n", "").replaceAll(" ", "+");
    return address;
}
```

```
// get the distance from the lantitude and longitude of the two loaction
public static double GetDistance(double lon1, double lat1, double lon2, double lat2) {
    double radLat1 = rad(lat1);
    double radLat2 = rad(lat2);
    double a = radLat1 - radLat2;
    double b = rad(lon1) - rad(lon2);
    double s = 2 * Math.asin(Math.sqrt(Math.pow(Math.sin(a / 2), 2) + Math.cos(radLat1) * Math.cos(radLat2) * Math.pow(Math.sin(b / 2), 2)));
    s = s * EARTH_RADIUS;
    return Double.parseDouble(String.format("%.2f",s * 0.00062137119));
}
private static double rad(double d) {
    return d * Math.PI / 180.0;
}
```

```
// get the distance between to address
public static double GetDistance(double lon1, double lat1, double lon2, double lat2) {
    double radLat1 = rad(lat1);
    double radLat2 = rad(lat2);
    double a = radLat1 - radLat2;
    double b = rad(lon1) - rad(lon2);
    double s = 2 * Math.asin(Math.sqrt(Math.pow(Math.sin(a / 2), 2) + Math.cos(radLat1) * Math.cos(radLat2) * Math.pow(Math.sin(b / 2), 2)));
    s = s * EARTH_RADIUS;
    return Double.parseDouble(String.format("%.2f",s * 0.00062137119));
}

public static double getDistance(String from, String to) throws URISyntaxException, IOException {
    Map<String, Object> fromInfo = getGeoCodingInfoFromAddress(from);
    Map<String, Object> toInfo = getGeoCodingInfoFromAddress(to);
    return GetDistance((Double) fromInfo.get("lng"), (Double) fromInfo.get("lat"), (Double) toInfo.get("lng"), (Double) toInfo.get("lat"));
}
```

Notification by Email

I develop a notification service for patient get appointment information when they accept the appointment offered by the system. Once the front receive the reponse from back end for the accept appointment request, the front end will sent a POST Request to the NotificationController which will process the request and sent email to the patient address.

Figure right side is how the notification looks like.

```
@PostMapping(path = "/patient/confirmation")
public void sendConfirmation(@RequestBody Map<String, String> patientInfo) throws IOException {
    String fromEmail = PropertiesUtil.getProperty("spring.mail.username");
    String fromName = "Reservation Confirmed";
    Patient patient = patientRepository.findById(Integer.parseInt(patientInfo.get("patient_id"))).orElseThrow(()->new PatientNotFoundException(patientInfo.get("patient_id")));
    String toEmail = patient.getEmail();
    String toName = patient.getName();
    Appointment appointment = appointmentRepository.findById(Integer.parseInt(patientInfo.get("id"))).orElseThrow(()->new RuntimeException());
    Provider provider = providerRepository.findById(appointment.getProviderId()).orElseThrow(()->new ProviderNotFoundException());
    String emailContent = "Dear "+patient.getName()+":\n\tYour Reservation is Confirmed!\n=======================Reservation Info=========================\nAddress: "+
                          provider.getAddress()+ "\nstart time:\t"+ TimeUtils.parseTimeStamp(appointment.getStartTime()).toString()
                          +"\nend time:\t"+TimeUtils.parseTimeStamp(appointment.getEndTime()).toString();
    final Email email = DefaultEmail.builder()
            .from(new InternetAddress(fromEmail, fromName))
            .to(Lists.newArrayList(new InternetAddress(toEmail, toName)))
            .subject("Covid19 Vaccination Reversion")
            .body(emailContent)
            .encoding("UTF-8").build();
    emailService.send(email);
}
```

File Upload and Download

When patients sign in to their account, they can choose to uplaod their additional information as a form of file, and access the file by downloading it. The file is uploaded under the url: resources/static/upload/additional_info, and the url will be kept in the patient table. Once the file is uploaded or changed, the front end will refresh the file link to show the current file name. Below is the FileController in back end.

```
@Controller
@RequestMapping(path = "/api")
public class FileController {
    @PostMapping(path = "/file")
    public @ResponseBody String uploadSingleFile(@RequestParam("file") MultipartFile multipartFile) throws JsonProcessingException {
        System.out.println("file controller called");
        if (multipartFile.isEmpty()) {
            throw new IllegalArgumentException("no file is selected");
        }
        FilePath.getInstance().setUrl(null);
        ObjectMapper objectMapper = new ObjectMapper();
        Map<String, String> res = new HashMap<>();
        res.put("uploadStatus","fail");
        try {
            String contentType = multipartFile.getContentType();
            String originalFilename = multipartFile.getOriginalFilename();
            byte[] bytes = multipartFile.getBytes();

            String filePath = PropertiesUtil.getProperty("upload-path");
            File parentPath = new File(filePath,"additional_info");
            try {
                File destFile = new File(parentPath,originalFilename);
                FileUtils.writeByteArrayToFile(destFile,multipartFile.getBytes());
                FilePath.getInstance().setUrl(destFile.getPath());
                res.put("uploadStatus","success");
                res.put("url",destFile.getPath());
            } catch (Exception e) {
                e.printStackTrace();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return objectMapper.writeValueAsString(res);
    }
}
```

Transation and Concurrency

Although this is a course and may not have much user, I still consider about how the system work in concurrency. In this program, when patient or provider want to write or update the information they only operate on their own account, so there is less likely to occur the race condition. I think the race condition is likely to happen when patient accept or decline a appointment offer with admin operating on appointment in the background at the same time. Spring provide the annotion @Transcional. By default, transcation will roll bace RunningtimeException or Error.

```
@Transactional
@Modifying
@Query(value = "update appointment set patient_id = null, status = 'release' where patient_id = ?", nativeQuery = true)
void releaseAppointment(Integer id);
```

Other basic features

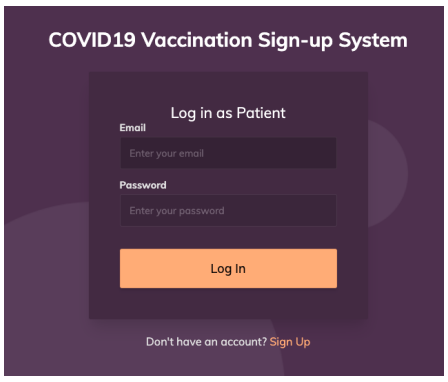
Here are some basic feature that support the system running.

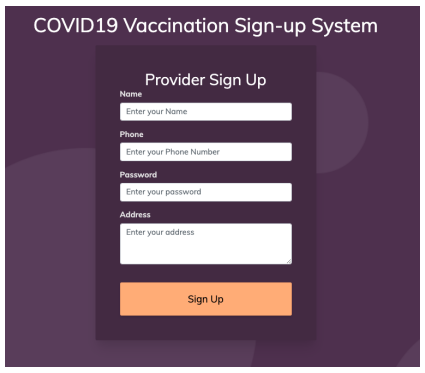
Sign In and Sign Up

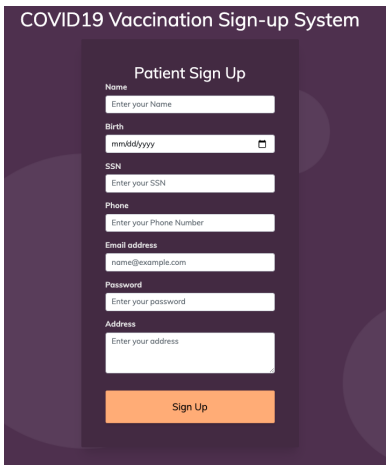
Sign in function is use use patient or provide's login information to check if there is such a record in the database. If yes, then allow user to login and response with the user id.

I put the id information into the url and passing it to the detailed information page.

Sign up function just check the if the unique informaiton (patient email, provider phone, etc). If the user already in the database, then alert a error, otherwise, add the sign up information into the database.

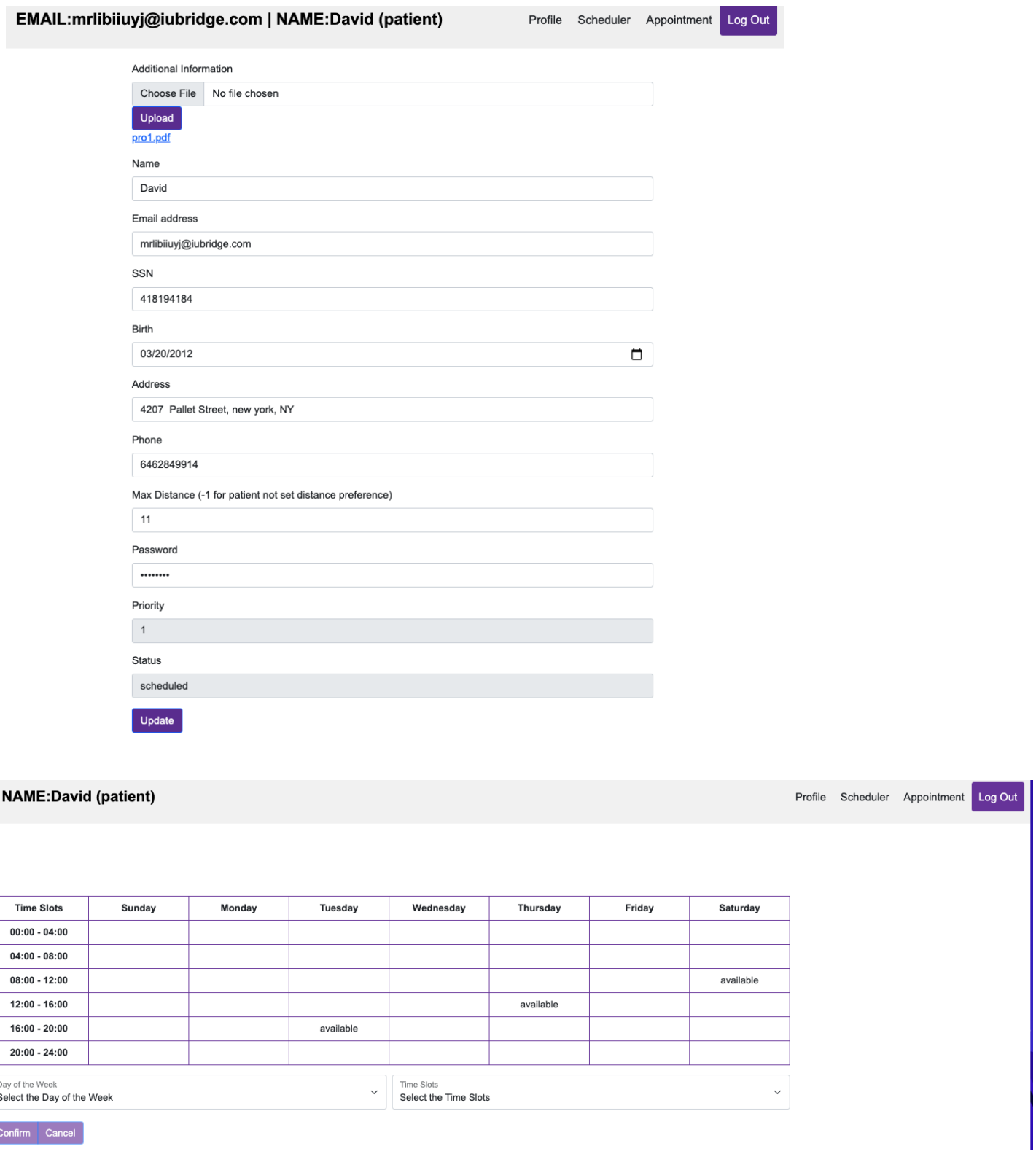


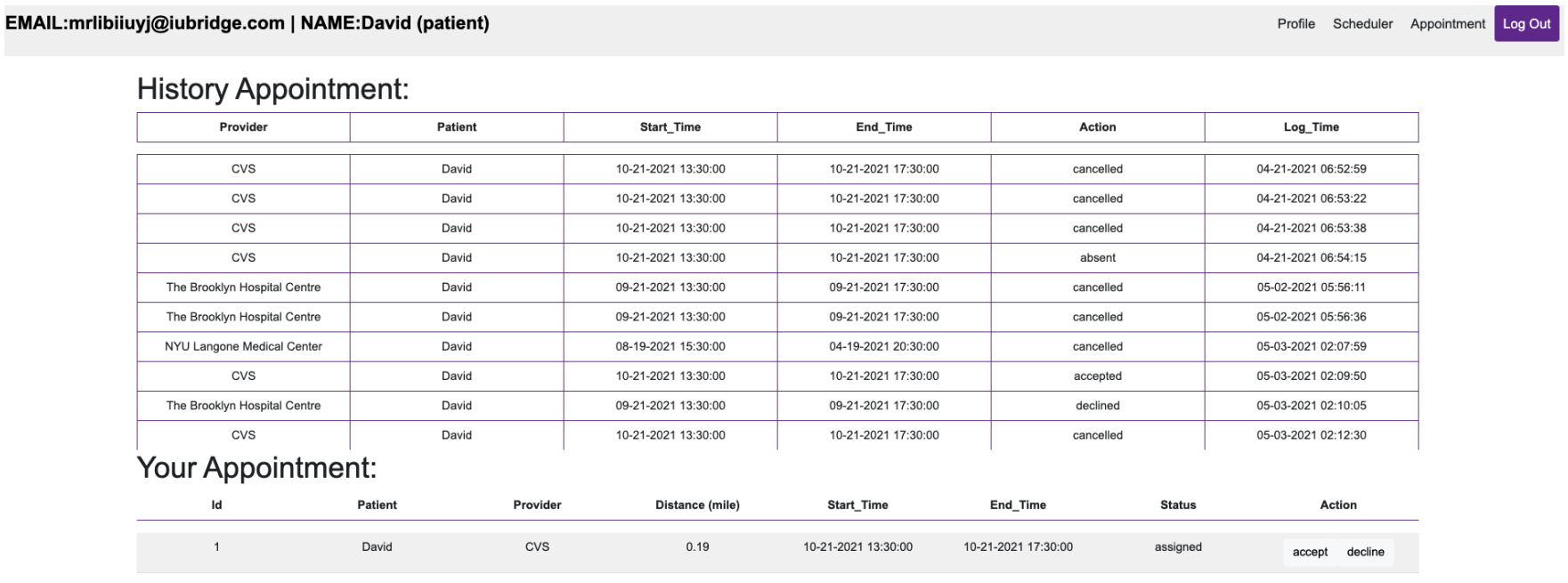




Patient information page

patient information page can divided into three part. In profile part, patient can change the personal information. In scheduler, the patient can arrange the avaliable time slot in a week. And in appointment, the patient can see the history log and and do operation on the new appointment offer.





Provider inforamtion page

The provider information page as two part. The profile section is samiliar as the patient's, provider can modify information there. And in Appointment page, provider can upload new appointment, check the non-expired appointments (the appointments can be sort in differetn fliter condition). And also, the provider can operation on the expired appointment to set the status into absent or completed. Both the operation will be logged into log page. If the patient is absent, he will be released into the waiting poll for the next round matching.