System Requirements Specification (SRS)

[E- Laboratory Management System]

CHAPTER 1 :

Introduction:

The Project "Online Laboratory Management system" is an Automated full stack web project for E-lab Management. It eases the tasks of the user/patient as well as lab administrator and other staff of any Diagnostic centre.

The main objective of this project is to provide the solution for an medical laboratory to provide the facilities available online as a part of E-Lab (portal for equipped for conducting tests suggested by medical experts).

This software also helps the administrator and instructor(s) to maintain proper documentation of the computing systems. This software is a web-based application and can be hosted on the internet. It also provides a clean and user-friendly interface to the users/patients.

Purpose:

The purpose of this project is to provide the solution of a local medical test laboratories to help in monitoring it's online services and provides it's existence globally through internet.

Need/motivation:

We visited a local medical test laboratory/diagnostic centre and asked them about if they have their own website/software online which keeps tracks of their individual users/patients and provide them an option to book a test slot online or at least book an appointment if their phone services are busy in some cases.

It is also difficult for the administrator to integrate entire information of all the patients/user's database who did their medical tests there. Our software solves these problems.

CHAPTER 2 :

Literature Survey:

The E-Administration of Diagnostic centre is a new attempt to speed up the process of managing physical laboratory in the industry. Presently in labs, most of the tasks are carried on manually such as lodging complaints, booking a test slot, extra lab requests etc. There are many difficulties for carrying out the lab related activities if one of the member/staff especially in the reception sector remains absent for specific reasons, also it will be easier to use the collected user data which can be used for further recommendations and improving user services.

This Software provides user an online platform to book the various diagnostic test slots available in the lab online if it’s not possible for user to physically visit the lab due to some reasons.

Objectives:

1) Helps Administrator to keep the track of the detailed information of User and provide them portal to book the test slot online.
2) Assists smooth interaction between different users.
3) Proper maintenance of available resources.
4) Helps Administrator to provide its facilities online and without physical interference.

CHAPTER 3 :

Technical Requirements:-

1) Functional Requirements ->

- It Should provide the display available services in the diagnostic centre without any of the clashes between the day, time and all available services must be visible to all.
- It should add all the services/tests selected by the registered user in his account portal.
- It should generate a report about the registered complaint to the admin and response report to the user who has submitted his queries (via email).
- User must be able to select all the tests/services he wants to take from the E-lab software.
- Secure registration and profile management facilities for different users.
- It should generate alerts via SMS and Email.

2) Non-Functional Requirements ->

Safety Requirements

If there is extensive damage to a wide portion of the database due to catastrophic failure, such as a disk crash, the recovery method restores a past copy of the database that was backed up to archival storage

Security Requirements

Security systems need database storage just like many other applications. However, the special requirements of the security market mean that vendors must choose their database partner carefully.

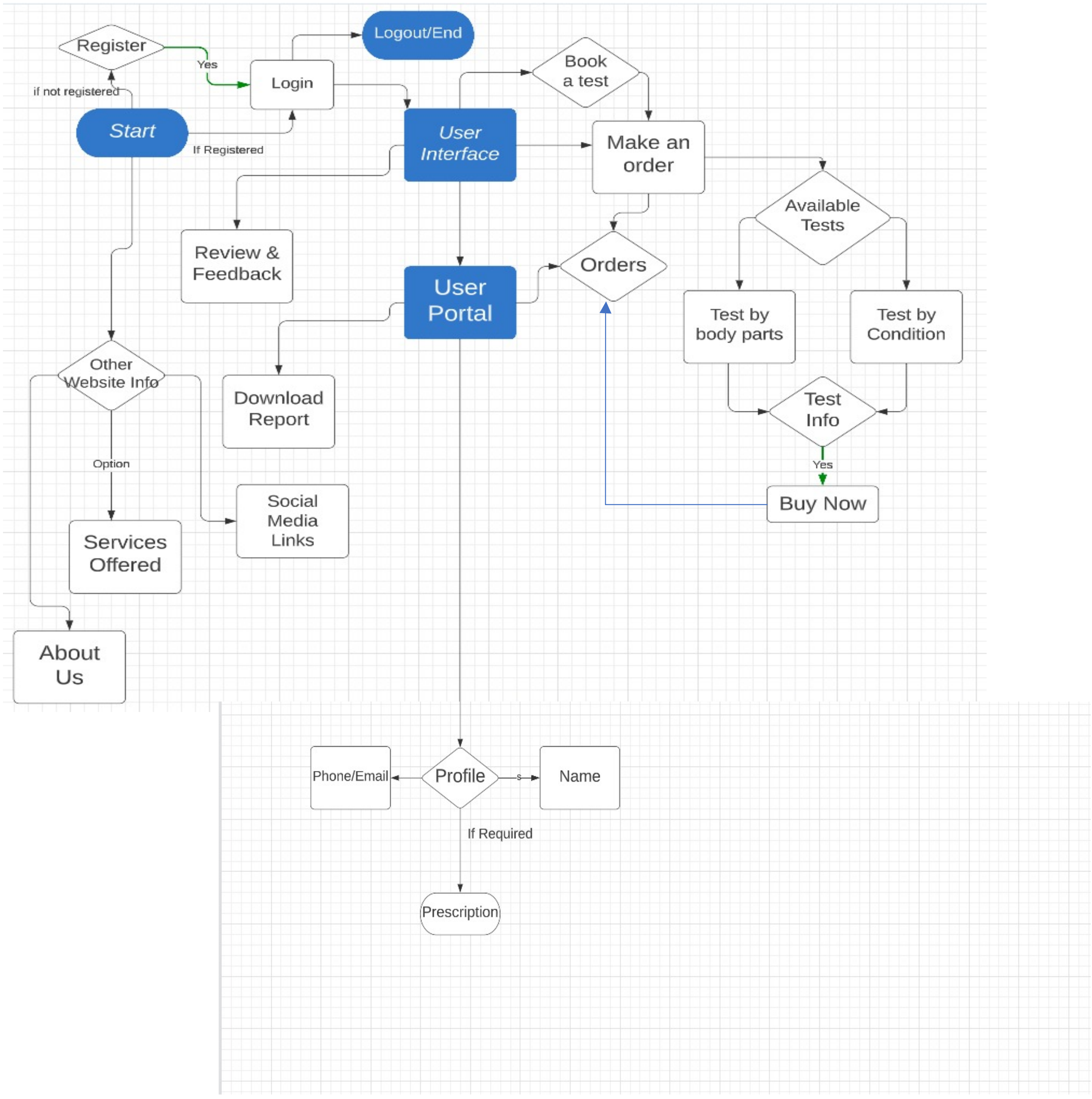
- **Hardware Requirements:**

1. A recent model of computer with at least 4gb of ram is sufficient for web full stack project. (8gb is recommended)
2. A dual-core processor (quad-core is recommended for better performance)
3. Solid-State Drive (SSD) is recommended for storage (minimum 256 gb for faster computations and efficient working).

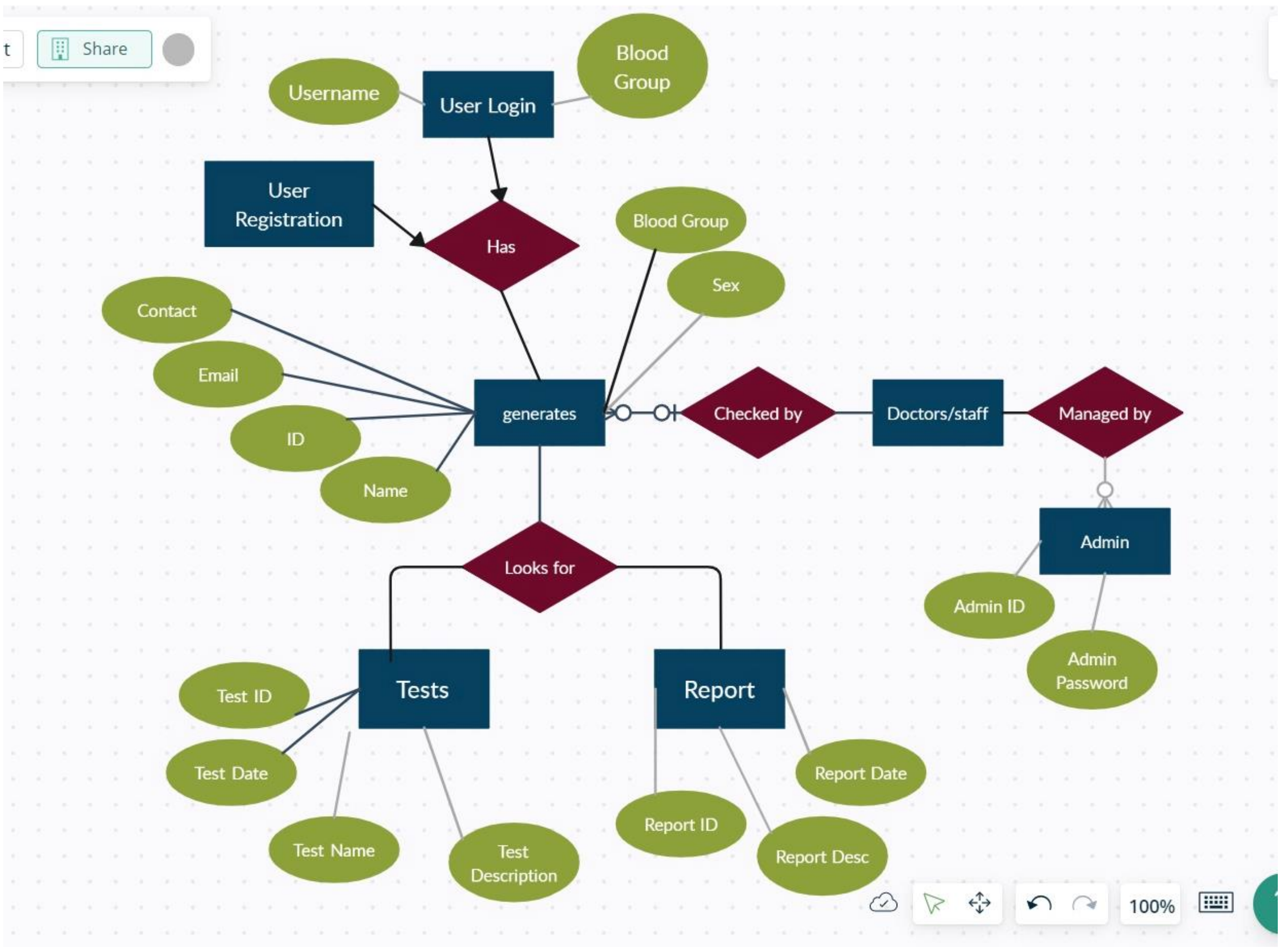
- **Software Requirements:**

1. HTML, CSS, JavaScript as front-end languages (for dynamic = thyme Leafe template or JSP) and React as framework for REST API.
2. Reference of bootstrap or tailwind for front-end development.
3. Java (Spring boot) for backend development.
4. MySQL as a database.

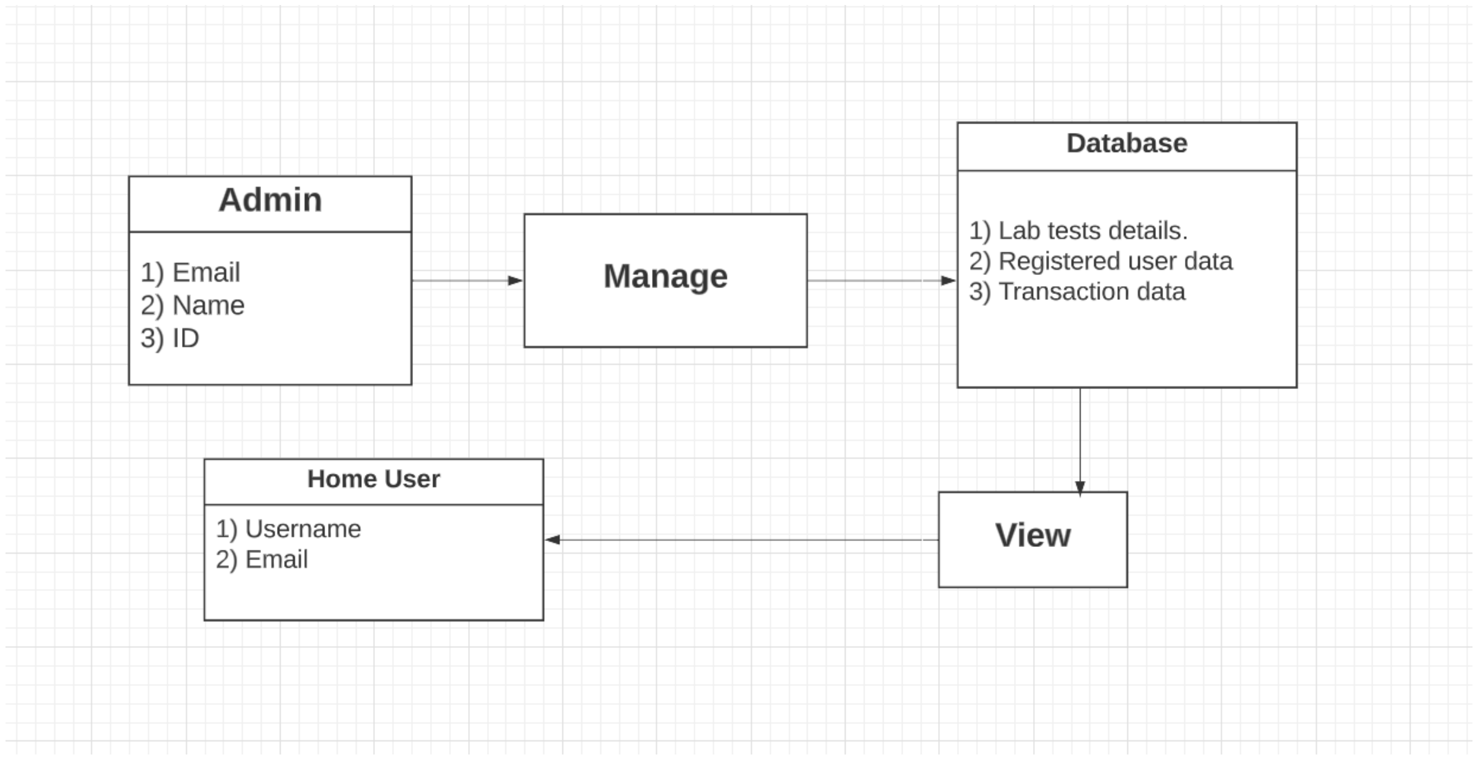
Flowchart of the Online Laboratory System



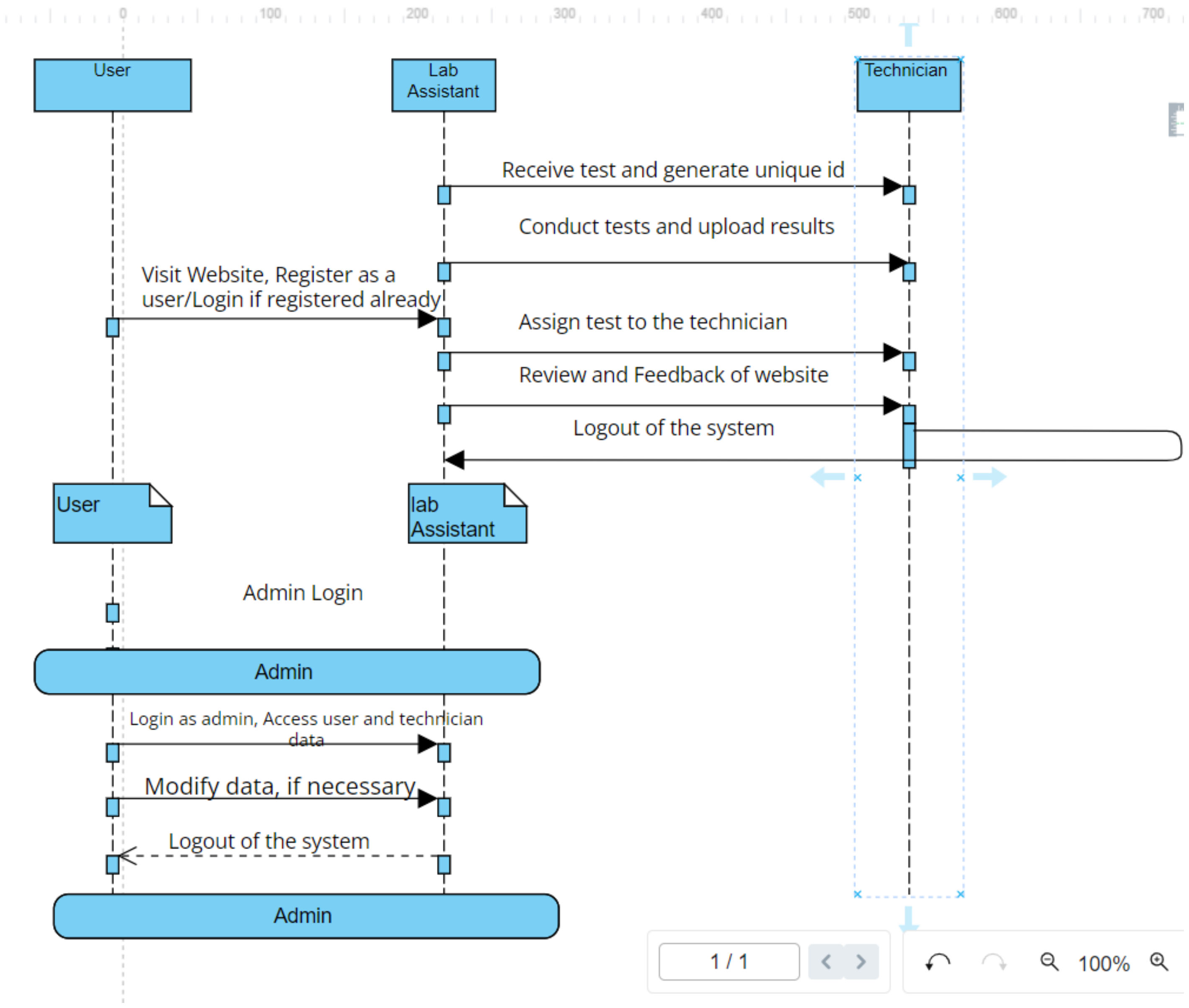
Entity-Relationship (ER) diagram for E-Laboratory System



Web-View diagram for E-Laboratory System

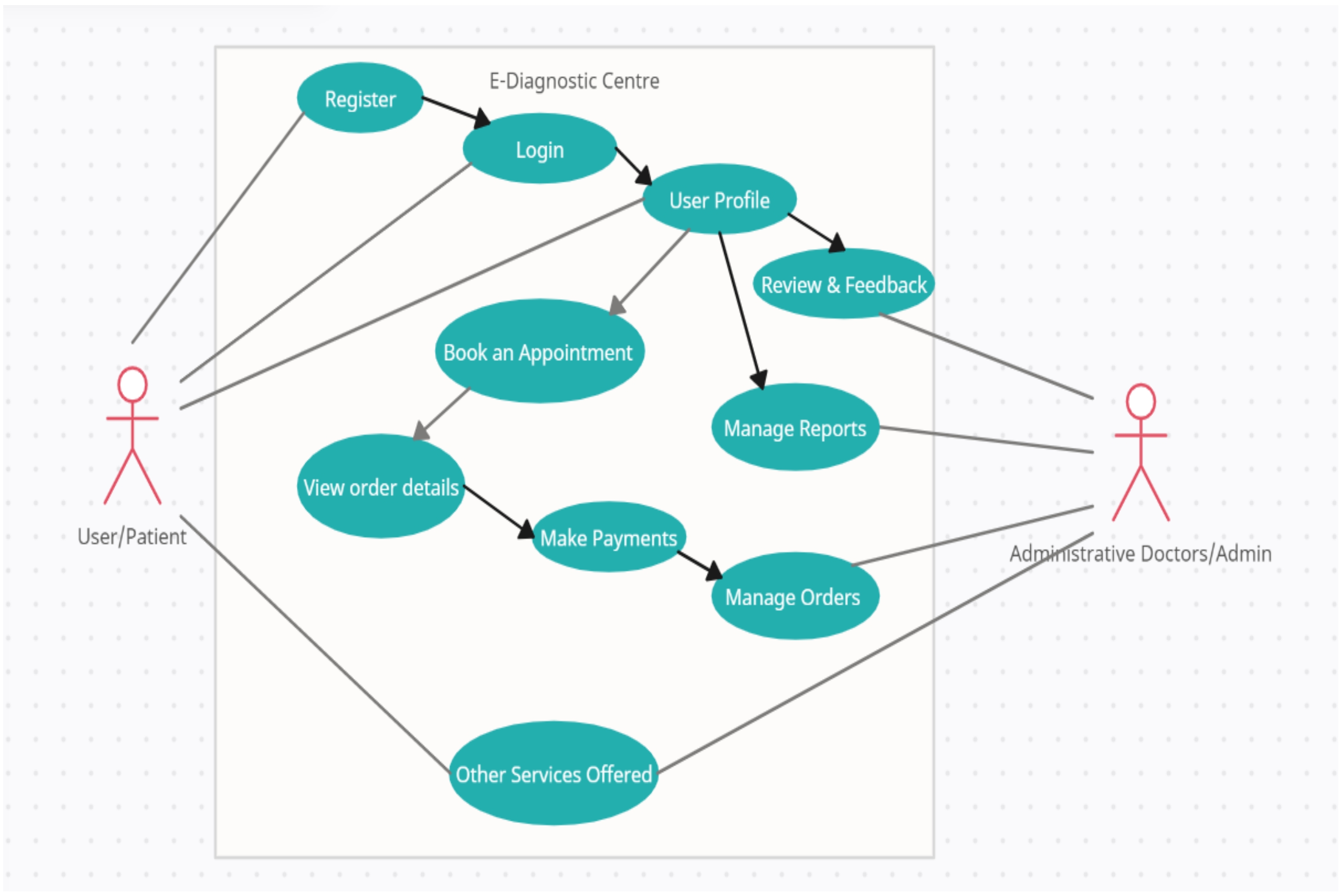


SEQUENCE DIAGRAM



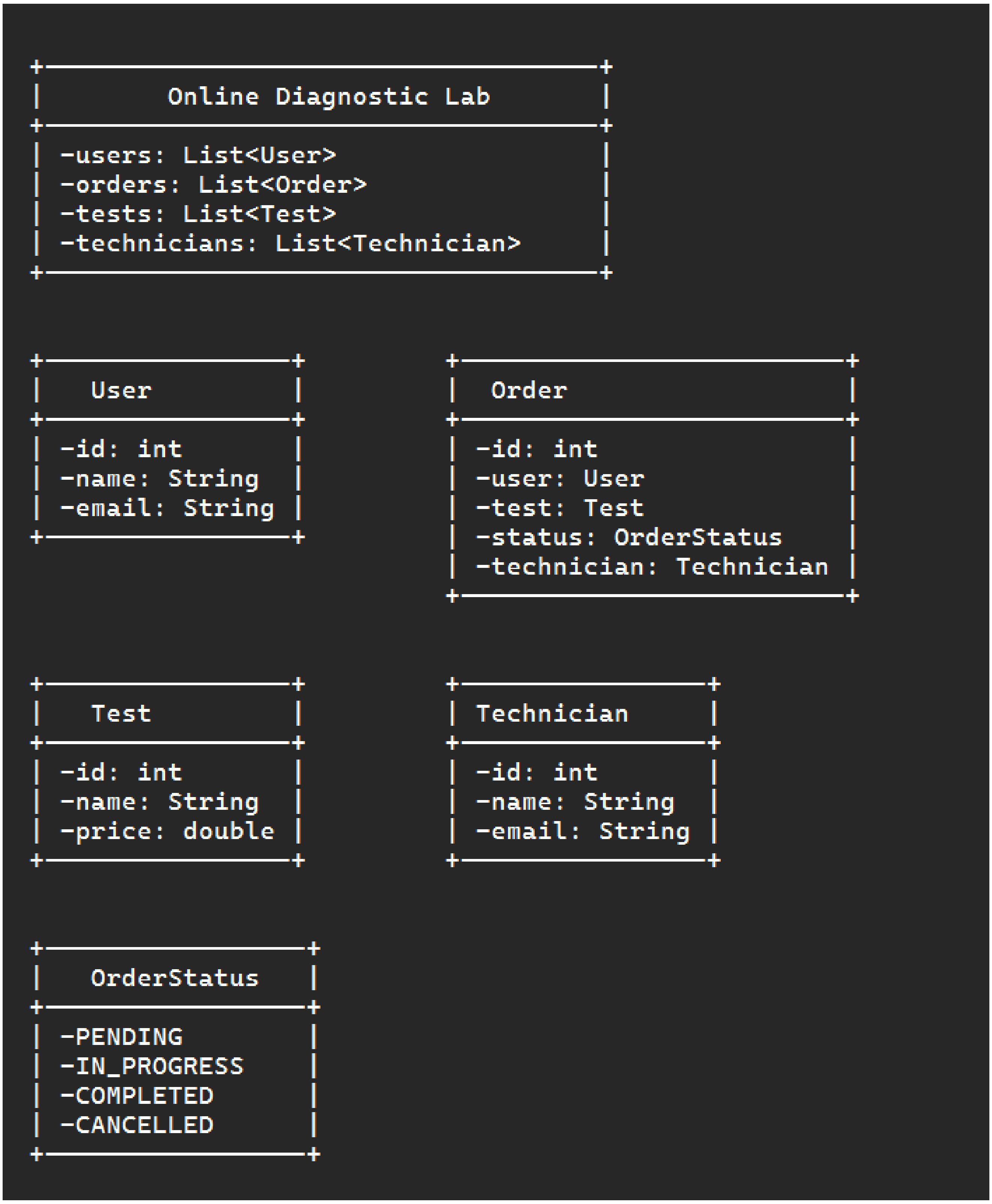
User
Lab Assistant
Technician
Receive test and generate unique id
Conduct tests and upload results
Visit Website, Register as a user/Login if registered already
Assign test to the technician
Review and Feedback of website
Logout of the system
User
lab Assistant
Admin Login
Admin
Login as admin, Access user and technician data
Modify data, if necessary
Logout of the system
Admin
1 / 1
100%

USE CASE DIAGRAMS



UML – CLASS DIAGRAM

1) UML Class for Diagnostic Centre:



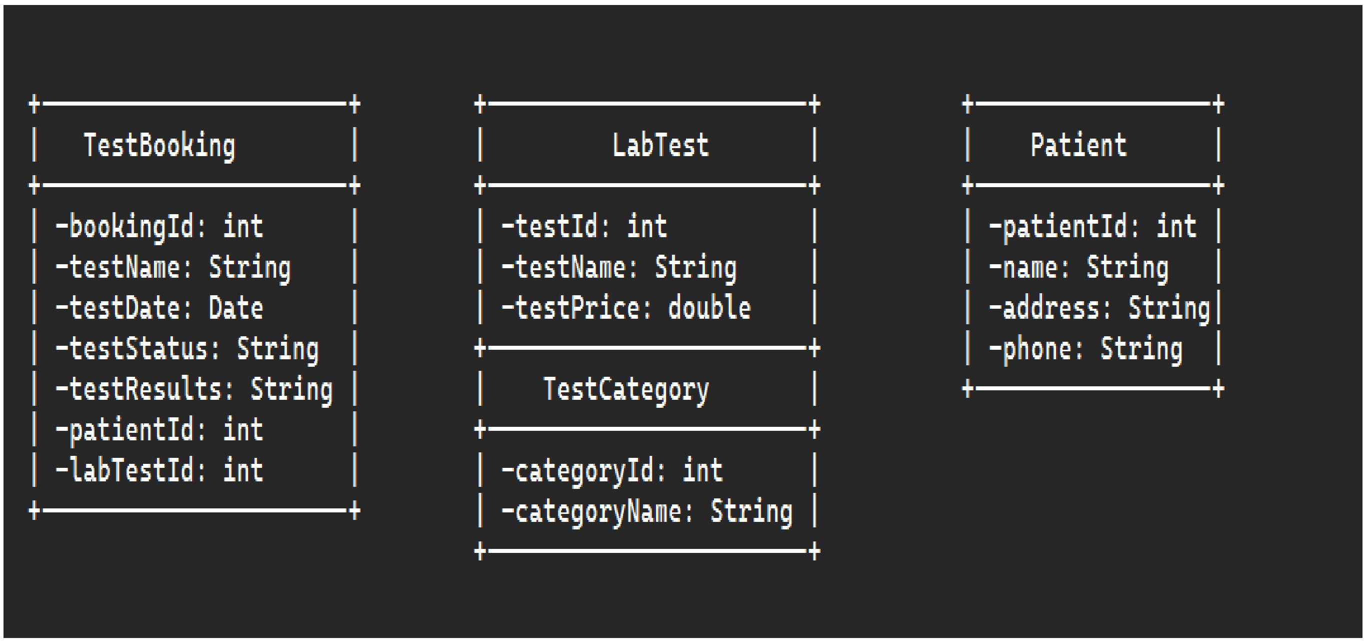
```
+-------------------------------------+
|         Online Diagnostic Lab       |
+-------------------------------------+
| -users: List<User>                  |
| -orders: List<Order>                |
| -tests: List<Test>                  |
| -technicians: List<Technician>      |
+-------------------------------------+


+-----------------+          +--------------------------+
|   User          |          |  Order                   |
+-----------------+          +--------------------------+
| -id: int        |          | -id: int                 |
| -name: String   |          | -user: User              |
| -email: String  |          | -test: Test              |
+-----------------+          | -status: OrderStatus     |
                             | -technician: Technician  |
                             +--------------------------+


+-----------------+          +-----------------+
|   Test          |          | Technician      |
+-----------------+          +-----------------+
| -id: int        |          | -id: int        |
| -name: String   |          | -name: String   |
| -price: double  |          | -email: String  |
+-----------------+          +-----------------+


+------------------+
|   OrderStatus    |
+------------------+
| -PENDING         |
| -IN_PROGRESS     |
| -COMPLETED       |
| -CANCELLED       |
+------------------+
```

2) **UML class for Test slot booking :**



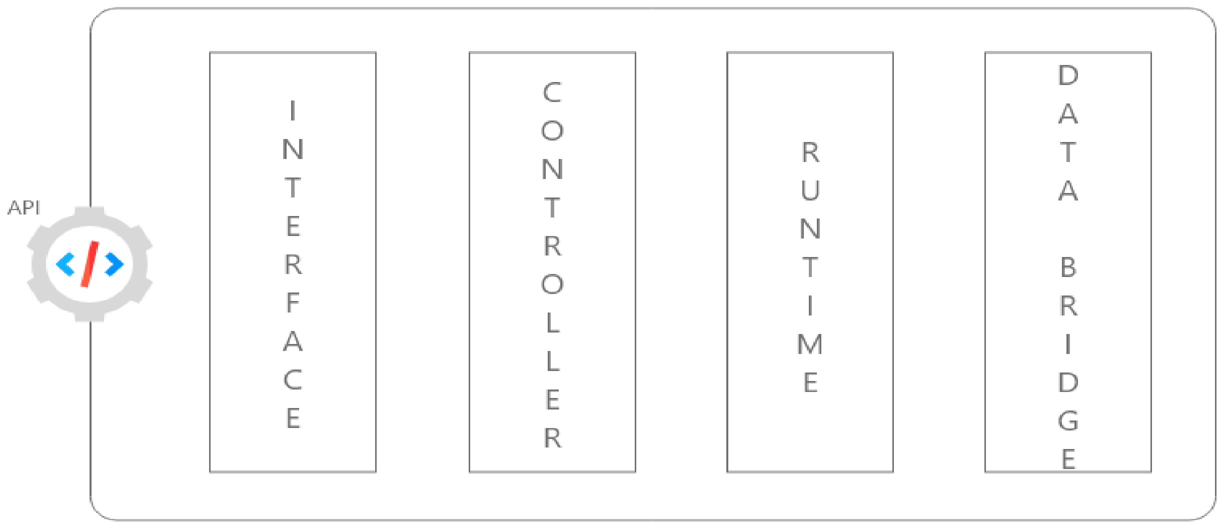
```
+--------------------+      +--------------------+      +-----------------+
|   TestBooking      |      |       LabTest      |      |    Patient      |
+--------------------+      +--------------------+      +-----------------+
| -bookingId: int    |      | -testId: int       |      | -patientId: int |
| -testName: String  |      | -testName: String  |      | -name: String   |
| -testDate: Date    |      | -testPrice: double |      | -address: String|
| -testStatus: String|      +--------------------+      | -phone: String  |
| -testResults: String|     |    TestCategory    |      +-----------------+
| -patientId: int    |      +--------------------+
| -labTestId: int    |      | -categoryId: int   |
+--------------------+      | -categoryName: String |
                            +--------------------+
```

API ARCHITECTURE

API Architecture

API architecture refers to the technical framework of developing a software interface that exposes backend data and application functionality for use in external applications. An API architecture consists of components for external interfacing, traffic control, runtime execution of business logic, and data access.

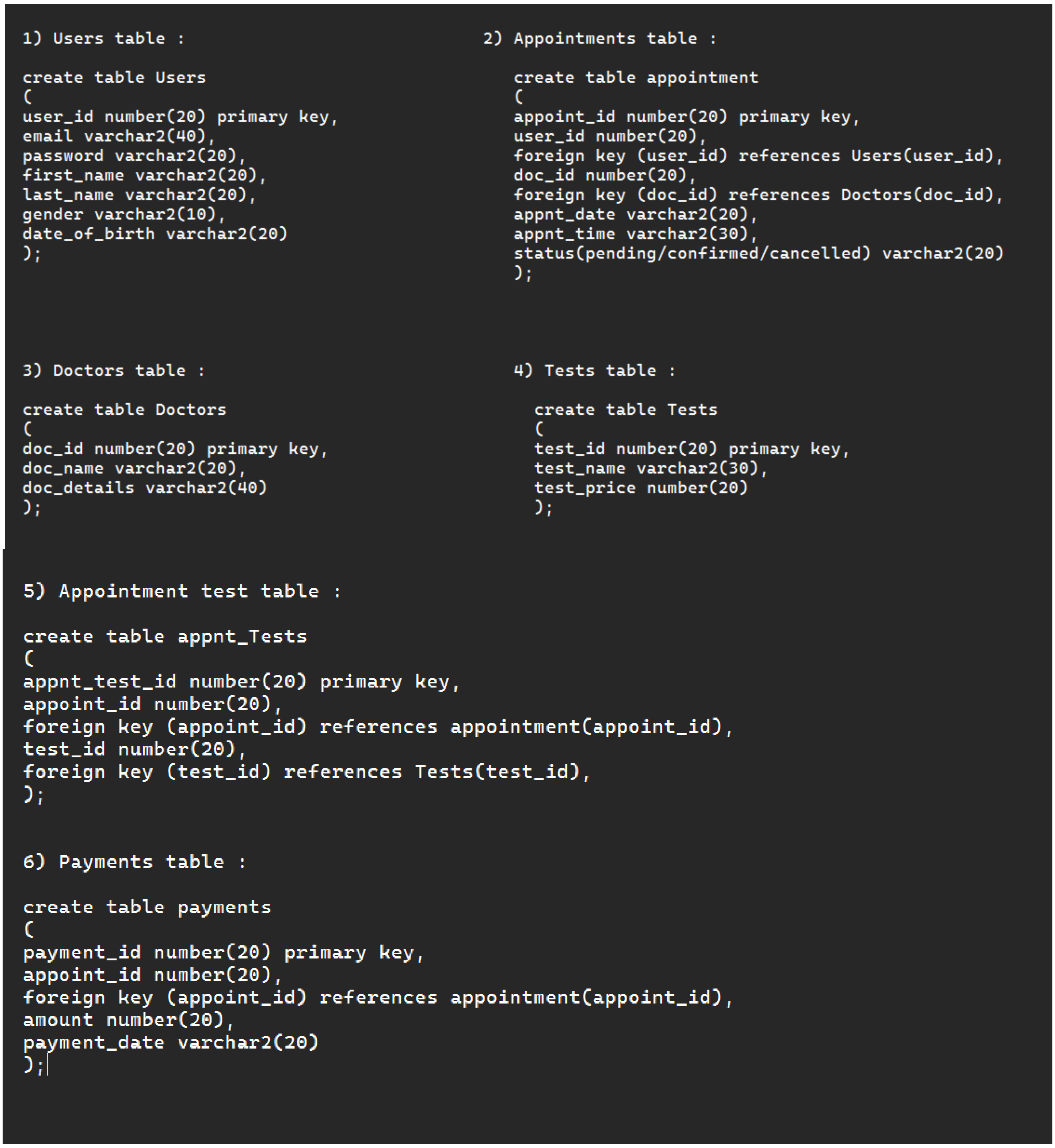
Four building blocks of an API :



We’ll have to create following sub packages to the project to use Rest-API:

- DAO - The DAO (data access layer) provides an interface to connect with the database and access the data stored in the database. A single DAO class can deal with queries retrieving different types of entities.
- Repository - This layer is similar to the DAO layer which connects to the database and accesses the data. However the repository layer provides a greater abstraction compared to the DAO layer. Every class is responsible for accessing and manipulating one entity. This tutorial will use the repository layer.
- Service - This layer calls the DAO layer to get the data and perform business logic on it. The business logic in the service layer could be - performing calculations on the data received, filtering data based on some logic, etc.
- Model - The model contains all the Java objects that will be mapped to the database table using. The DAO will fetch the data from the database and populate the respective model with that data and return it to the service layer and vice versa.
- Controller - This is the topmost layer, called when a request comes for a particular REST API. The controller will process the REST API request, calls one or more services and returns an HTTP response to the client.

DATABASE SCHEMA



```
1) Users table :

create table Users
(
user_id number(20) primary key,
email varchar2(40),
password varchar2(20),
first_name varchar2(20),
last_name varchar2(20),
gender varchar2(10),
date_of_birth varchar2(20)
);
```

```
2) Appointments table :

   create table appointment
   (
   appoint_id number(20) primary key,
   user_id number(20),
   foreign key (user_id) references Users(user_id),
   doc_id number(20),
   foreign key (doc_id) references Doctors(doc_id),
   appnt_date varchar2(20),
   appnt_time varchar2(30),
   status(pending/confirmed/cancelled) varchar2(20)
   );
```

```
3) Doctors table :

create table Doctors
(
doc_id number(20) primary key,
doc_name varchar2(20),
doc_details varchar2(40)
);
```

```
4) Tests table :

  create table Tests
  (
  test_id number(20) primary key,
  test_name varchar2(30),
  test_price number(20)
  );
```

```
5) Appointment test table :

create table appnt_Tests
(
appnt_test_id number(20) primary key,
appoint_id number(20),
foreign key (appoint_id) references appointment(appoint_id),
test_id number(20),
foreign key (test_id) references Tests(test_id),
);
```

```
6) Payments table :

create table payments
(
payment_id number(20) primary key,
appoint_id number(20),
foreign key (appoint_id) references appointment(appoint_id),
amount number(20),
payment_date varchar2(20)
);
```