CS 6360.003 – Database Design

Tele Medicine Database Project

Matthew Bedford (mdb190007), Marin Budic (mxb170005), Joah George (jpg190002), Shubham Singh (sxs210114)

Dr. Jalal Omer

University of Texas at Dallas

12/5/2022

Table of Contents

Introduction

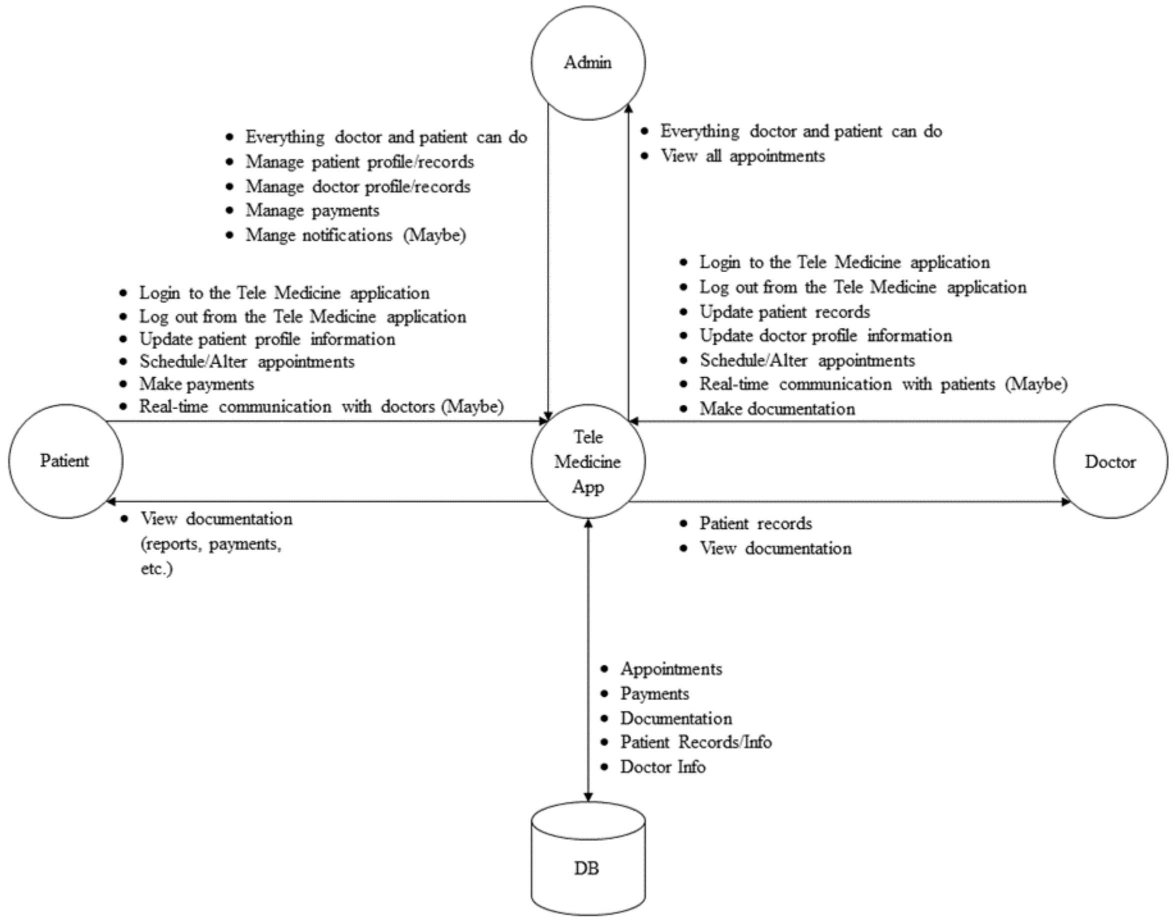
This is a report of our project which will see our team utilize a Database Management System to manage a Tele Medicine system that will provide many different uses to the patients and doctors such that Tele Medicine is able to operate. This began with a Requirement Analysis and led into conceptual and logical database design and then normalization and database implementation and testing, and then finally we bring the project together and create an application program that will allow the user to interact with our database system.

We will start off with system requirements and show our context diagram, interface requirements, and functional/non-functional requirements of the database system. After which, we will show our conceptual design of the database by providing our entity-relationship model of our database and the data dictionary and business rules of our ER model. Then, we will show our logical database schema which we derived from our ER diagram in which we used SQL statements to construct. After which we will show the functional dependencies and database normalization of our project.

Then, after describing our process of how we created our database, we will demonstrate how to install and invoke our system, and provide complex queries and views for this database. Then we will show how to build our system and show how it will interact with a user. After this, we will conclude our work, give suggestions for the future, and provide all of our references as well as an appendix of all of our work containing all of the files pertaining to this project.

System Requirements

Context Diagram (System Architecture)


Admin
• Everything doctor and patient can do
• Manage patient profile/records
• Manage doctor profile/records
• Manage payments
• Mange notifications (Maybe)
• Everything doctor and patient can do
• View all appointments
• Login to the Tele Medicine application
• Log out from the Tele Medicine application
• Update patient profile information
• Schedule/Alter appointments
• Make payments
• Real-time communication with doctors (Maybe)
• Login to the Tele Medicine application
• Log out from the Tele Medicine application
• Update patient records
• Update doctor profile information
• Schedule/Alter appointments
• Real-time communication with patients (Maybe)
• Make documentation
Patient
Tele Medicine App
Doctor
• View documentation (reports, payments, etc.)
• Patient records
• View documentation
• Appointments
• Payments
• Documentation
• Patient Records/Info
• Doctor Info
DB

System Description

The Tele Medicine application will provide patients a way to receive remote health care. Some services included in the Tele Medicine application are diagnosis, treatment, education, and counseling. The Tele Medicine application is available from anywhere in the world at any time.

Use Cases

Patient

- Login to the Tele Medicine application
- Log out from the Tele Medicine application
- Update patient profile information
- Schedule/Alter appointments
- Real-time communication with doctors (Maybe)
- Make payments
- View documentation (reports, payments, etc.)
- View services

Doctor

- Login to the Tele Medicine application
- Log out from the Tele Medicine application
- Update/Alter doctor profile information
- Add Availability
- Real-time communication with patients (Maybe)
- View documentation (reports, etc.)

Admin

- Manages web application
- Manage patient profile/records
- Manage doctor profile/records
- Manage payments Services
- Manage other services (e.g., medical tests etc.…)
- View/Modify all appointments

Requirements

Functional Requirements

Login Functional Requirements

1. The system will allow the user to log in.
2. The system will verify the username and password.
3. The system will not allow the user to log in with an invalid username or password.
4. The system will be able to remember usernames and passwords.
5. The system will allow users to create accounts.
6. The system will enable users to log out of their accounts.
7. The system will be able to distinguish between patient logins and employee logins
8. The system will allow users to log out.
9. The system will have a “forgot password” option to allow for account recovery.
10. The system will ask for an authorization key from admin.

Browsing Functional Requirements

1. The system will allow the user to select to view appointments made.
2. The system will allow the user to select to make an appointment (assuming requirement 10 is met) with a doctor as they need.
3. The system will allow the user to select to view their profile information about themselves.
4. The system will allow the user to select to view payments due or paid in the past.
5. The system will allow the user to select a chat feature with a doctor to get real time communication.
6. When a user is not logged in, they will be allowed to view a limited version of the site where they can see employees and general health information as well as general information about the service.
7. The system will not allow patients to view the features available to employees on the site.

8. Any medical forms that need to be filled out by the patient prior to an appointment will be accessible in the patient account.
9. The patient should be able to fill out these medical forms on the site and these documents will be saved to their account once completed.
10. To schedule specific appointments, the system will check to make sure that a patient has filled out required forms before an appointment can be made.
11. Once an appointment has been booked users will be able to reschedule said appointment.
12. When logged in to the system, doctors will be able to view the patients under their care as well as all medical files related to these patients.
13. The doctor will be able to manage patient appointments and view the requested information that the patient would like to go over.
14. The system will allow the doctor to prescribe medications to the patient at the doctor's discretion.
15. The system will allow doctors to send emails to their patients regarding information about an appointment or the results of an appointment.

Administrative Functional Requirements:

1. The system will allow the admin to manage records of patients and doctors to update data after an appointment.
2. The system will enable the admin to update profiles (both doctor and patient) to keep an up-to-date version of people’s information.
3. The system will ask for an authorization key from the admin every time they attempt to alter someone’s information.
4. Each account will generate an authorization key to be changed, and if the user has signed permission to change their account, then the admins will have access to their authorization key.
5. The system will enable the admin to alter payments to manage any errors that may occur.
6. The system will allow the admin to see all scheduled appointments to overlook the entire schedule.
7. The system will allow admin to manage scheduling of the real time communication between doctors and patients.
8. The system will allow the admin to send notifications to users.
9. The system will allow the admin to view the number of users in the system.
10. The system will allow the admin to remove users from the system.
11. The system will allow the admin to manage all appointments.

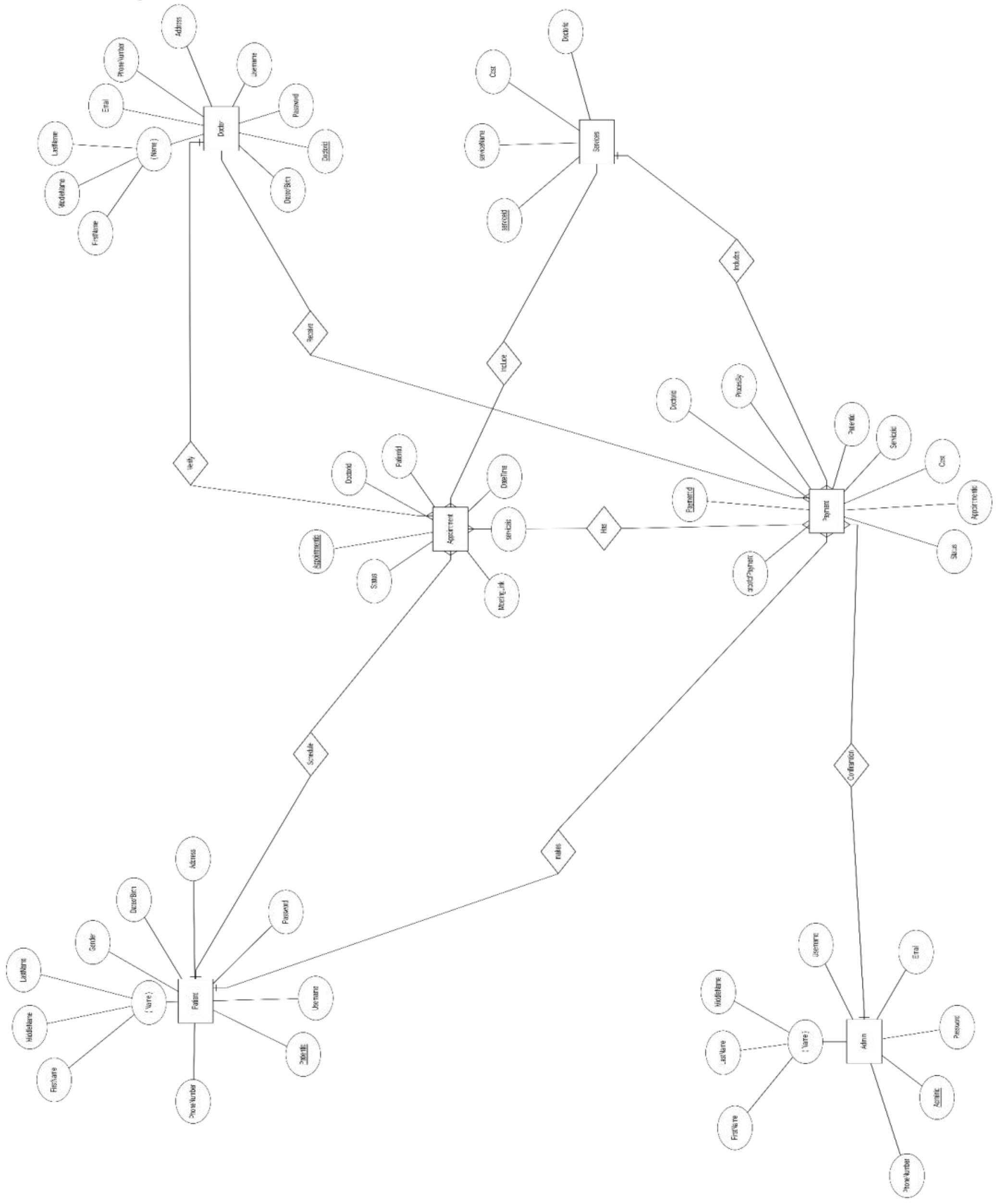
Non-Functional Requirements:

1. The system should be available every day at all hours of the day and down times should be minimal and avoided at all costs.
2. In response to going down, the system should be able to be recovered quickly and data loss should be minimized.
3. The system will not create an account for a use until they have created a strong password.
4. The user must select and answer a security question to maintain secure access to their account.

5. Notifications regarding appointment information should be sent with no more latency than 5 minutes.
6. The application should be able to handle any number of users attempting to make an appointment at once by updating the doctor's availability as soon as someone has scheduled with them.
7. The system will need to be responsive and feel good to use (no lag, little input latency, etc.) even though many users could be on the site at the same time.
8. The system should be able to handle many users being added to the system as the service expands and adopts more doctors, patients, etc.
9. The system should be secure enough so that no private user data is leaked to the public and the system will prevent users from accessing other users' data if they don’t have access to that information.
10. Submission of a medical form by a patient should not take more than 30 seconds and the patients assigned doctor as well as administrators should be able to view the submitted form within a minute of the form being submitted.

Conceptual Design of the Database

ER Diagram



Data Dictionary and Business Rules

Key Constraints

Primary Keys

Of the form: <Primary Key Attribute> in <Table>

- Patientid in Patient
- Paymentid in Payment
- Adminid in Admin

- Doctorid in Doctor
- Appointmentid in Appointment
- Serviceid in Services

Foreign Keys

Of the form: {List of Foreign Key Attributes} in <Table>

- Patientid, Doctorid, Adminid, serviceid in Payment
- Patientid, Doctorid, serviceid in Appointment
- Paymentid, Appointmentid in Has

Other Constraints

Domain Constraints

Attribute	Data Type
Patientid	int
Firstname	varchar(15)
MiddleName	varchar(15)
LastName	varchar(15)
Gender	char
DateofBirth	DATE
PhoneNumber	char(10)
Address	varchar(30)
Username	varchar(15)
Password	varchar(15)
Paymentid	int
Doctorid	int
Serviceid	int
Appointmentid	int
Status	varchar(15)
Cost	Float
Domain Constraints Continued	
proofofPayment	boolean
ProcesBy	DATE

Email	char
DateTime	DATE
MeetingLink	char
serviceName	varchar(30)
Cost	Float
Doctorid	int

Entity Integrity Constraints

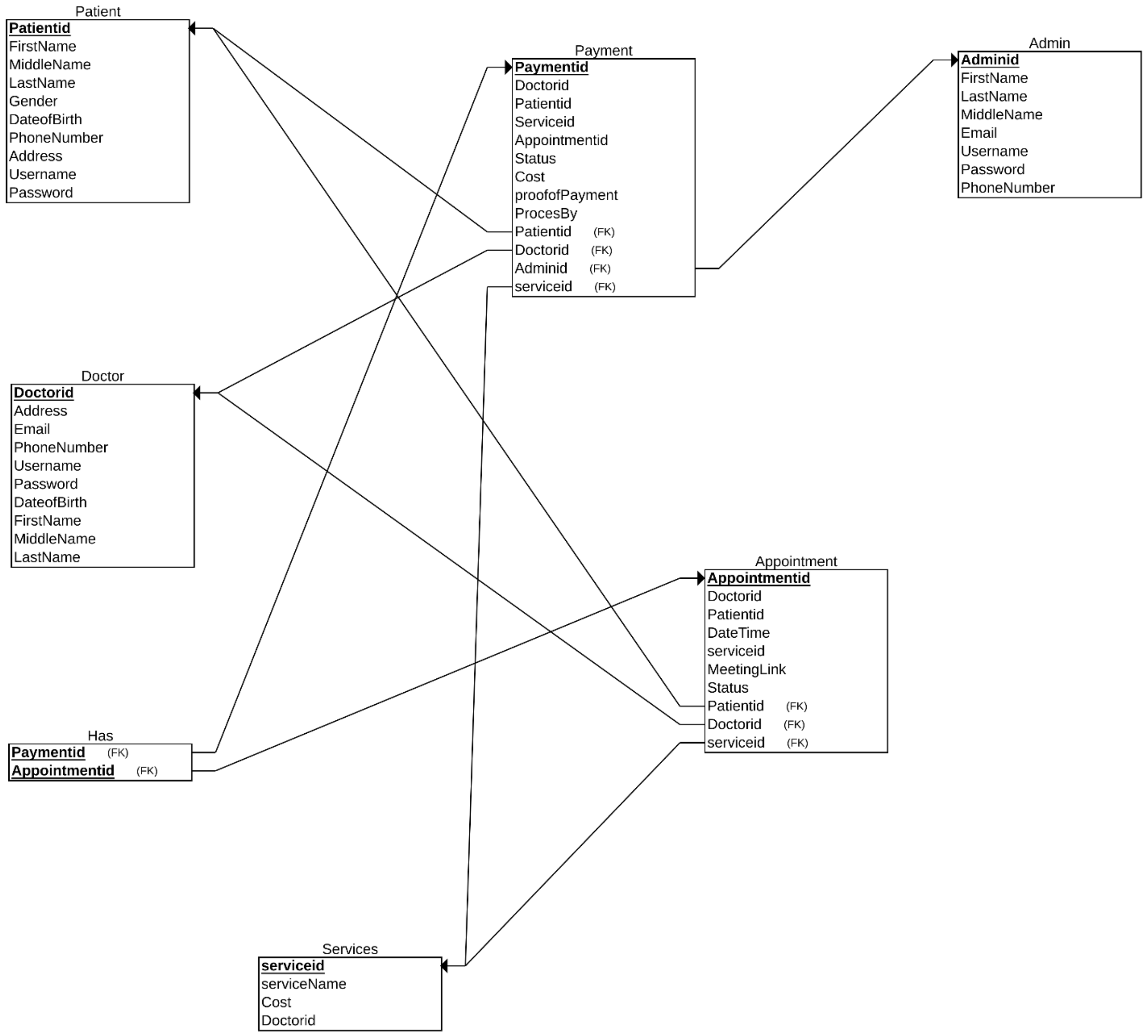
Table(Attribute)	Constraint
Patient(Patientid)	NOT NULL
Payment (Paymentid)	NOT NULL
Admin(Adminid)	NOT NULL
Doctor(Doctorid)	NOT NULL
Appointment(Appointmentid)	NOT NULL

Referential Integrity Constraints

Of the form: <Table1>(<Attribute1>) REFERENCES <Table2>(<Attribute2>)

Table1(Attribute1)	Table2(Attribute2)
Payment(Patientid)	PATIENT(Patientid)
Payment(Doctorid)	DOCTOR(Doctorid)
Payment(Adminid)	Admin(Adminid)
Appointment(Patientid)	PATIENT(Patientid)
Appointment(Doctorid)	DOCTOR(Doctorid)
Appointment(serviceid)	Service(serviceid)
Has(Paymentid)	Payment(Paymentid)
Has(Appointmentid)	Appointment(Appointmentid)

Logical Database Schema



Functional Dependencies and Database Normalization

Functional Dependencies

Patient

Patient_Id → First_Name, Middle_Name, Last_Name, Gender, Date_of_Birth, Phone_Number, Address, Username, Pass

Username → Pass

Doctor:

Doctor_Id → First_Name, Middle_Name, Last_Name, Date_of_Birth, Phone_Number, Address, Username, Pass

Username → Pass

Admin:

Admin_Id → First_Name, Middle_Name, Last_Name, Phone_Number, Username, Pass

Username → Pass

Services:

Service_Id → Service_Name, Cost_of_Service

Appointment:

Appointment_Id → DateTime, Status, Meeting_Link, Patient_Id, Doctor_Id, Service_Id, Payment_Id

Payment:

Payment_Id → Status, Proof_of_Payment, Admin_Id, Appointment_Id

Normalized Tables

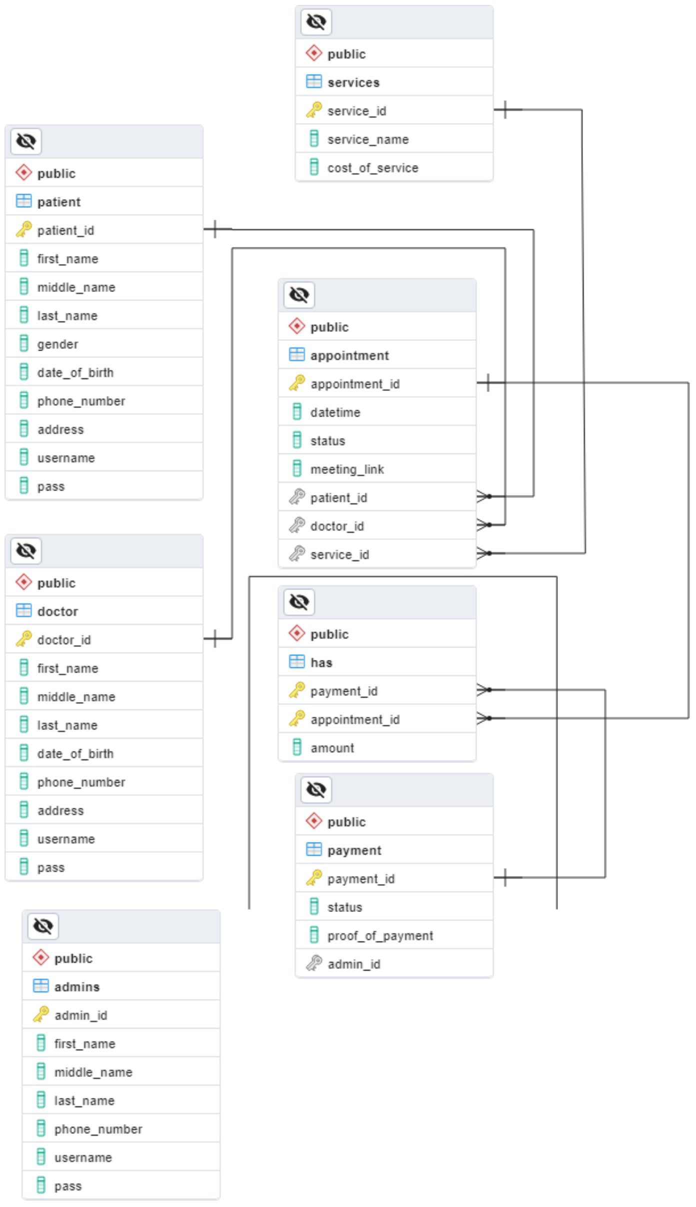
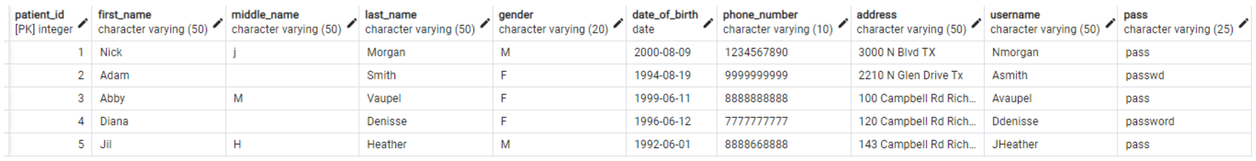


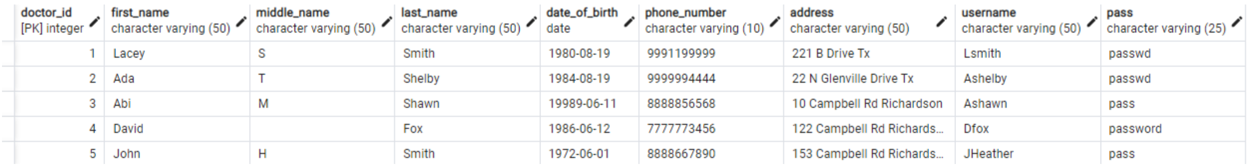
Table Instances

Patient:



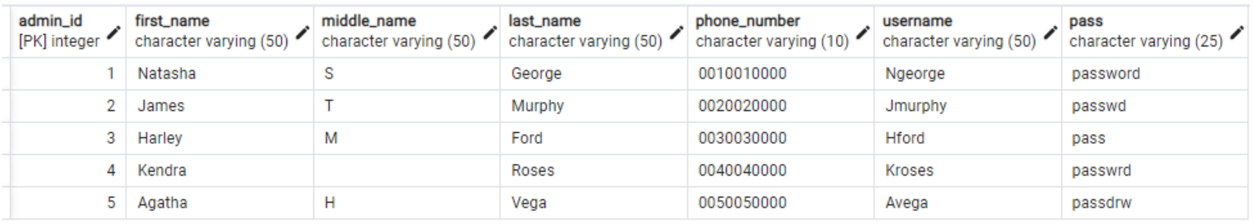
	patient_id [PK] integer	first_name character varying (50)	middle_name character varying (50)	last_name character varying (50)	gender character varying (20)	date_of_birth date	phone_number character varying (10)	address character varying (50)	username character varying (50)	pass character varying (25)
	1	Nick	j	Morgan	M	2000-08-09	1234567890	3000 N Blvd TX	Nmorgan	pass
	2	Adam		Smith	F	1994-08-19	9999999999	2210 N Glen Drive Tx	Asmith	passwd
	3	Abby	M	Vaupel	F	1999-06-11	8888888888	100 Campbell Rd Rich...	Avaupel	pass
	4	Diana		Denisse	F	1996-06-12	7777777777	120 Campbell Rd Rich...	Ddenisse	password
	5	Jil	H	Heather	M	1992-06-01	8888668888	143 Campbell Rd Rich...	JHeather	pass

Doctor:



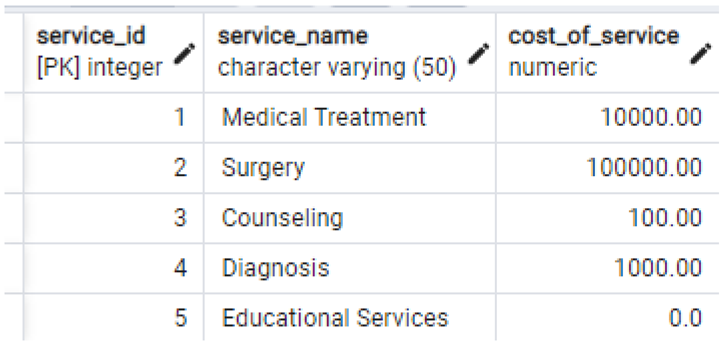
doctor_id [PK] integer	first_name character varying (50)	middle_name character varying (50)	last_name character varying (50)	date_of_birth date	phone_number character varying (10)	address character varying (50)	username character varying (50)	pass character varying (25)
1	Lacey	S	Smith	1980-08-19	9991199999	221 B Drive Tx	Lsmith	passwd
2	Ada	T	Shelby	1984-08-19	9999994444	22 N Glenville Drive Tx	Ashelby	passwd
3	Abi	M	Shawn	19989-06-11	8888856568	10 Campbell Rd Richardson	Ashawn	pass
4	David		Fox	1986-06-12	7777773456	122 Campbell Rd Richards...	Dfox	password
5	John	H	Smith	1972-06-01	8888667890	153 Campbell Rd Richards...	JHeather	pass

Admin:



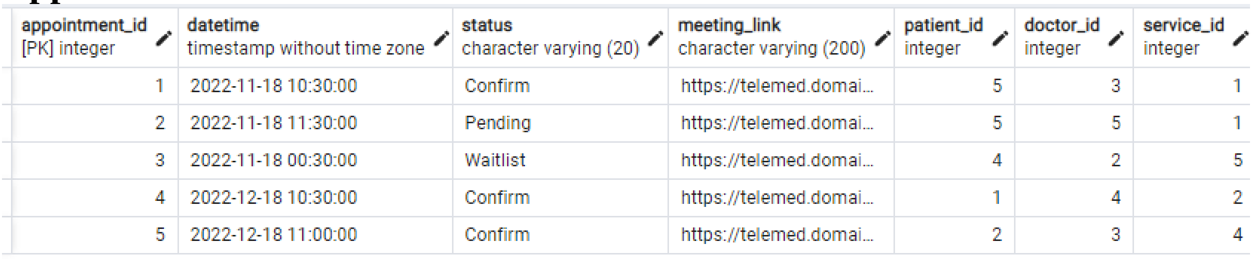
admin_id [PK] integer	first_name character varying (50)	middle_name character varying (50)	last_name character varying (50)	phone_number character varying (10)	username character varying (50)	pass character varying (25)
1	Natasha	S	George	0010010000	Ngeorge	password
2	James	T	Murphy	0020020000	Jmurphy	passwd
3	Harley	M	Ford	0030030000	Hford	pass
4	Kendra		Roses	0040040000	Kroses	passwrd
5	Agatha	H	Vega	0050050000	Avega	passdrw

Services:



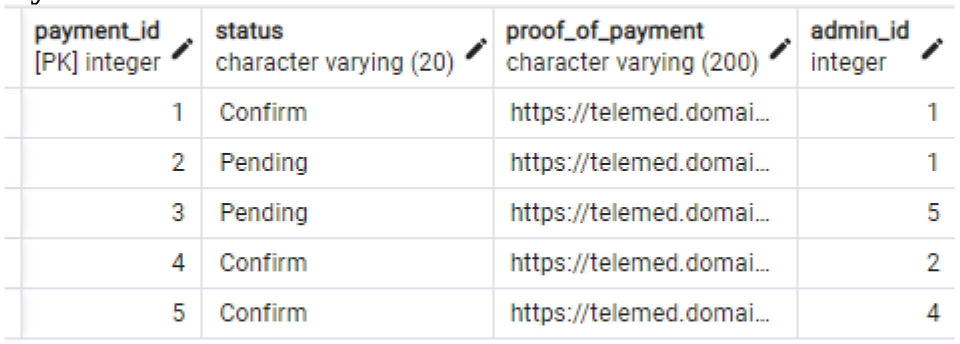
service_id [PK] integer	service_name character varying (50)	cost_of_service numeric
1	Medical Treatment	10000.00
2	Surgery	100000.00
3	Counseling	100.00
4	Diagnosis	1000.00
5	Educational Services	0.0

Appointment:



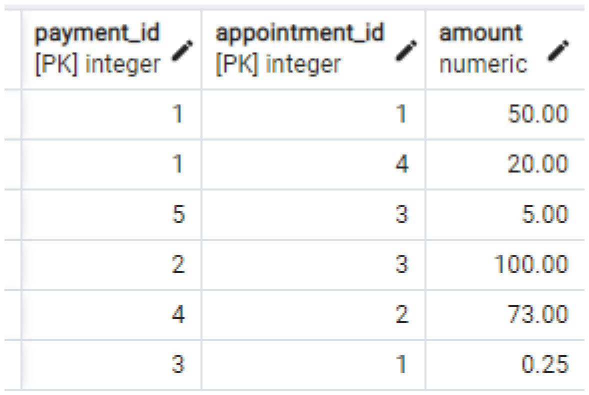
appointment_id [PK] integer	datetime timestamp without time zone	status character varying (20)	meeting_link character varying (200)	patient_id integer	doctor_id integer	service_id integer
1	2022-11-18 10:30:00	Confirm	https://telemed.domai...	5	3	1
2	2022-11-18 11:30:00	Pending	https://telemed.domai...	5	5	1
3	2022-11-18 00:30:00	Waitlist	https://telemed.domai...	4	2	5
4	2022-12-18 10:30:00	Confirm	https://telemed.domai...	1	4	2
5	2022-12-18 11:00:00	Confirm	https://telemed.domai...	2	3	4

Payment:



payment_id [PK] integer	status character varying (20)	proof_of_payment character varying (200)	admin_id integer
1	Confirm	https://telemed.domai...	1
2	Pending	https://telemed.domai...	1
3	Pending	https://telemed.domai...	5
4	Confirm	https://telemed.domai...	2
5	Confirm	https://telemed.domai...	4

Has:



payment_id [PK] integer	appointment_id [PK] integer	amount numeric
1	1	50.00
1	4	20.00
5	3	5.00
2	3	100.00
4	2	73.00
3	1	0.25

Create Tables Script

```
DROP TABLE IF EXISTS Patient CASCADE;
DROP TABLE IF EXISTS Appointment CASCADE;
DROP TABLE IF EXISTS Doctor CASCADE;
DROP TABLE IF EXISTS Admins CASCADE;
DROP TABLE IF EXISTS Payment CASCADE;
DROP TABLE IF EXISTS Has CASCADE;
DROP TABLE IF EXISTS Services CASCADE;

CREATE TABLE Patient(
	Patient_ID SERIAL PRIMARY KEY,
	First_name VARCHAR (50) NOT NULL,
	Middle_name VARCHAR (50),
	Last_name VARCHAR (50) NOT NULL,
	Gender VARCHAR		(20) NOT NULL,
	Date_of_Birth DATE NOT NULL,
	Phone_Number VARCHAR(10) NOT NULL,
	Address VARCHAR (50) NOT NULL,
	Username VARCHAR (50) NOT NULL,
	Pass VARCHAR (25) NOT NULL
);

CREATE TABLE Doctor(
	Doctor_ID SERIAL PRIMARY KEY,
	First_name VARCHAR (50) NOT NULL,
	Middle_name VARCHAR (50),
	Last_name VARCHAR (50) NOT NULL,
	Date_of_Birth DATE NOT NULL,
	Phone_Number VARCHAR(10) NOT NULL,
	Address VARCHAR (50) NOT NULL,
	Username VARCHAR (50) NOT NULL,
	Pass VARCHAR (25) NOT NULL
);

CREATE TABLE Admins(
	Admin_ID SERIAL PRIMARY KEY,
	First_name VARCHAR (50) NOT NULL,
	Middle_name VARCHAR (50),
	Last_name VARCHAR (50) NOT NULL,
	Phone_Number VARCHAR(10) NOT NULL,
        Username VARCHAR (50) NOT NULL,
        Pass VARCHAR (25) NOT NULL
);

CREATE TABLE Services(
        service_id SERIAL PRIMARY KEY,
        service_Name VARCHAR(50) NOT NULL,
        cost_of_service DECIMAL NOT NULL
);

CREATE TABLE Appointment(
        Appointment_ID SERIAL PRIMARY KEY,
        DateTime TIMESTAMP NOT NULL,
        Status VARCHAR (20) NOT NULL,
        Meeting_Link VARCHAR (200) NOT NULL,
        Patient_ID INT NOT NULL,
        Doctor_ID INT NOT NULL,
        service_id INT NOT NULL,

        CONSTRAINT fk_patient_apt FOREIGN KEY(Patient_ID) REFERENCES
Patient(Patient_ID),
        CONSTRAINT fk_doctor_apt FOREIGN KEY(Doctor_ID) REFERENCES Doctor(Doctor_ID),
        CONSTRAINT fk_service_apt FOREIGN KEY(service_id) REFERENCES
Services(service_id)
);

CREATE TABLE Payment(
        Payment_ID SERIAL PRIMARY KEY,
        Status VARCHAR (20) NOT NULL,
        Proof_of_Payment VARCHAR (200) NOT NULL,
        Admin_ID INT NOT NULL,

        CONSTRAINT fk_admin_pay FOREIGN KEY(Admin_ID) REFERENCES Admins(Admin_ID)
);

CREATE TABLE Has(
        Payment_ID INT NOT NULL,
        Appointment_ID INT NOT NULL,
        Amount DECIMAL NOT NULL,
        PRIMARY KEY(Payment_ID, Appointment_ID),

        CONSTRAINT fk_pay_has FOREIGN KEY(Payment_ID) REFERENCES Payment(Payment_ID),
        CONSTRAINT fk_apt_has FOREIGN KEY(Appointment_ID) REFERENCES
Appointment(Appointment_ID)
);
```

The Database System

- Download PostgresSQL (preferably 15) from https://www.postgresql.org/download/
- When installing, use default settings (Port number should be 5432 – the default value)
- Open pgAdmin4 from start menu
- Create a master password for opening ‘pgAdmin 4’ (this can be anything, but I recommend 'password')
- Click the ‘Servers’ on the left panel

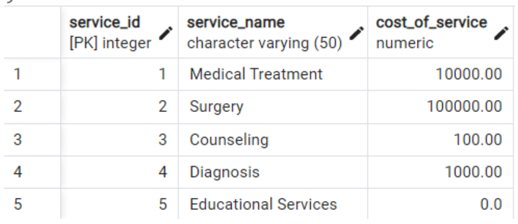
- Set the 'postgres' user password to be 'password'
- Make sure the 'PostgreSQL 15' line is selected in the left panel
- Create a new database called 'TeleMed' using default everything by going Object>Create>Database
- Right click the ‘TeleMed’ database on the left panel and select 'Query Tool' near the bottom
- In the top left of the query tool, select the folder icon to open scripts
- Open and run the `Create_Tables.sql` script
- Open and run the `Populate_Tables.sql` script
- The database is now set up!

Additional Queries and Views

Queries

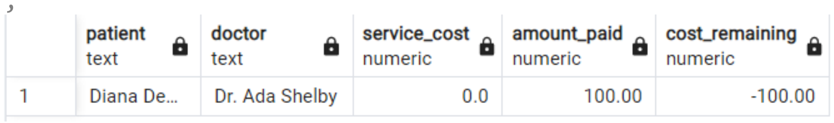
```
-- 1. Get all appointment information for a single patient
SELECT
       apt.status Appointment_Status,
       apt.meeting_link Link,
       CONCAT('Dr. ', d.first_name, ' ', d.last_name) Doctor,
       s.service_name Service
FROM Appointment apt
JOIN Doctor d ON d.doctor_id = apt.doctor_id
JOIN Services s ON s.service_id = apt.service_id
WHERE patient_id = 5
;

-- 2. Get all confirmed appointment information for a single patient
SELECT
       apt.status Appointment_Status,
       apt.meeting_link Link,
       CONCAT('Dr. ', d.first_name, ' ', d.last_name) Doctor,
       s.service_name Service
FROM Appointment apt
JOIN Doctor d ON d.doctor_id = apt.doctor_id
JOIN Services s ON s.service_id = apt.service_id
WHERE
       patient_id = 5
       AND
       apt.status = 'Confirm'
;

-- 3. Get patient, confrimed appointment time, and service by doctor
SELECT p.last_name, p.first_name, apt.datetime, s.service_name, apt.status
FROM Appointment apt
JOIN Patient p ON p.patient_id = apt.patient_id
JOIN Services s ON s.service_id = apt.service_id
WHERE
       apt.doctor_id = 3
       AND
       apt.status = 'Confirm'
;

-- 4. Get appointments and their payment info for patient
```

```
SELECT
	apt.datetime Appointment_Time,
	apt.status Appointment_Status,
	CONCAT('Dr. ', d.first_name, ' ', d.last_name) Doctor,
	s.cost_of_service Service_Cost,
	SUM(h.amount) Amount_Paid,
	s.cost_of_service - SUM(h.amount) Cost_Remaining
FROM Appointment apt
JOIN Has h ON h.appointment_id = apt.appointment_id
JOIN Payment pay ON pay.payment_id = h.payment_id
JOIN Doctor d ON d.doctor_id = apt.doctor_id
JOIN Services s ON s.service_id = apt.service_id
WHERE apt.patient_id = 1
GROUP BY
	apt.appointment_id,
	apt.datetime,
	apt.status,
	d.doctor_id,
	d.first_name,
	d.last_name,
	s.service_id,
	s.cost_of_service
;

-- 5. Get services
SELECT *
FROM Services
;
```



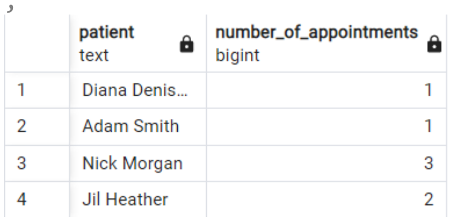
	service_id [PK] integer	service_name character varying (50)	cost_of_service numeric
1	1	Medical Treatment	10000.00
2	2	Surgery	100000.00
3	3	Counseling	100.00
4	4	Diagnosis	1000.00
5	5	Educational Services	0.0

```
-- 6. Get pending payments for admin to confirm
SELECT
	CONCAT(p.first_name, ' ', p.last_name) Patient,
	CONCAT('Dr. ', d.first_name, ' ', d.last_name) Doctor,
	s.cost_of_service Service_Cost,
	SUM(h.amount) Amount_Paid,
	s.cost_of_service - SUM(h.amount) Cost_Remaining
FROM Payment pay
JOIN Has h ON h.payment_id = pay.payment_id
JOIN Appointment apt ON apt.appointment_id = h.appointment_id
JOIN Patient p ON p.patient_id = apt.patient_id
JOIN Doctor d ON d.doctor_id = apt.doctor_id
JOIN Services s ON s.service_id = apt.service_id
WHERE
	pay.admin_id = 1
	AND
	pay.status = 'Pending'
GROUP BY
	pay.payment_id,
	p.patient_id,
	p.first_name,
	p.last_name,
        d.doctor_id,
        d.first_name,
        d.last_name,
        s.service_id,
        s.cost_of_service
;
```



	patient text	doctor text	service_cost numeric	amount_paid numeric	cost_remaining numeric
1	Diana De...	Dr. Ada Shelby	0.0	100.00	-100.00

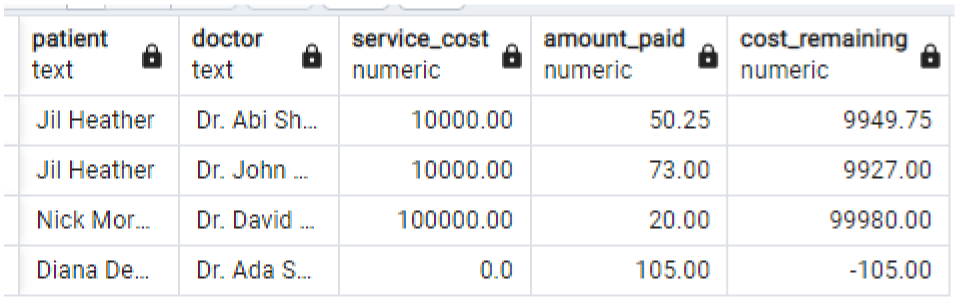
```
-- 7. Get number of appointments per patient
SELECT
        CONCAT(p.first_name, ' ', p.last_name) Patient,
        COUNT(apt.appointment_id) Number_of_Appointments
FROM Patient p
JOIN Appointment apt ON apt.patient_id = p.patient_id
GROUP BY
        p.patient_id,
        p.first_name,
        p.last_name
;
```



	patient text	number_of_appointments bigint
1	Diana Denis...	1
2	Adam Smith	1
3	Nick Morgan	3
4	Jil Heather	2

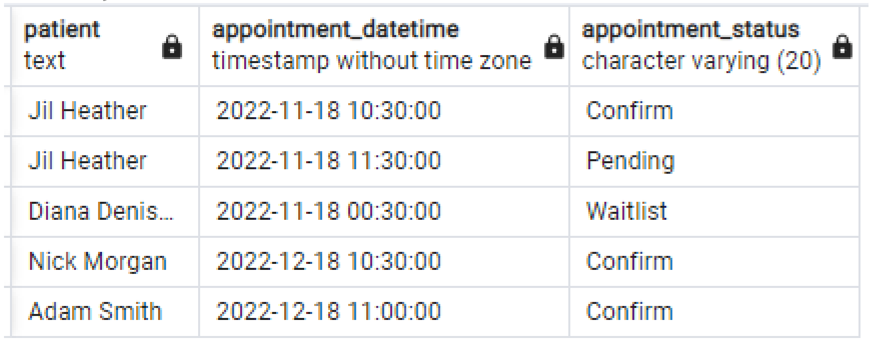
Views

```
CREATE OR REPLACE VIEW PatientPaymentInformation
AS
        SELECT
                CONCAT(p.first_name, ' ', p.last_name) Patient,
                CONCAT('Dr. ', d.first_name, ' ', d.last_name) Doctor,
                s.cost_of_service Service_Cost,
                SUM(h.amount) Amount_Paid,
                s.cost_of_service - SUM(h.amount) Cost_Remaining
        FROM Appointment apt
        JOIN Has h ON h.appointment_id = apt.appointment_id
        JOIN Payment pay ON pay.payment_id = h.payment_id
        JOIN Patient p ON p.patient_id = apt.patient_id
        JOIN Doctor d ON d.doctor_id = apt.doctor_id
        JOIN Services s ON s.service_id = apt.service_id
        GROUP BY
                p.patient_id,
                p.first_name,
                p.last_name,
                d.doctor_id,
                d.first_name,
                d.last_name,
                s.service_id,
                s.cost_of_service
        ;
```



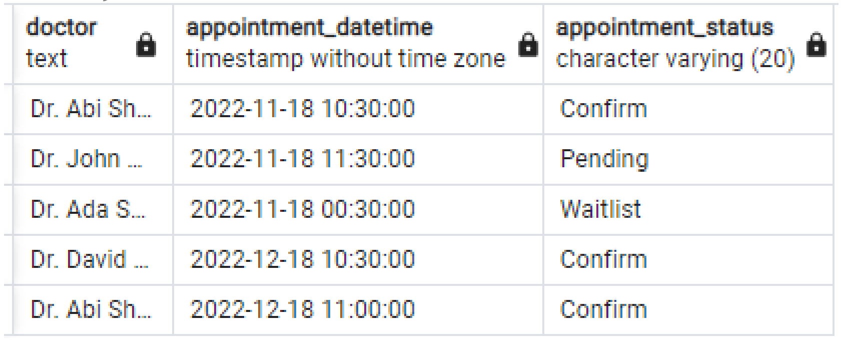
patient text	doctor text	service_cost numeric	amount_paid numeric	cost_remaining numeric
Jil Heather	Dr. Abi Sh...	10000.00	50.25	9949.75
Jil Heather	Dr. John ...	10000.00	73.00	9927.00
Nick Mor...	Dr. David ...	100000.00	20.00	99980.00
Diana De...	Dr. Ada S...	0.0	105.00	-105.00

```
CREATE OR REPLACE VIEW PatientAppointments
AS
       SELECT
              CONCAT(p.first_name, ' ', p.last_name) Patient,
              apt.datetime Appointment_DateTime,
              apt.status Appointment_Status
       FROM Appointment apt
       JOIN Patient p ON p.patient_id = apt.patient_id
       ;
```



patient text	appointment_datetime timestamp without time zone	appointment_status character varying (20)
Jil Heather	2022-11-18 10:30:00	Confirm
Jil Heather	2022-11-18 11:30:00	Pending
Diana Denis...	2022-11-18 00:30:00	Waitlist
Nick Morgan	2022-12-18 10:30:00	Confirm
Adam Smith	2022-12-18 11:00:00	Confirm

```
CREATE OR REPLACE VIEW DoctorAppointments
AS
       SELECT
              CONCAT('Dr. ', d.first_name, ' ', d.last_name) Doctor,
              apt.datetime Appointment_DateTime,
              apt.status Appointment_Status
       FROM Appointment apt
       JOIN Doctor d ON d.doctor_id = apt.doctor_id
       ;
```



doctor text	appointment_datetime timestamp without time zone	appointment_status character varying (20)
Dr. Abi Sh...	2022-11-18 10:30:00	Confirm
Dr. John ...	2022-11-18 11:30:00	Pending
Dr. Ada S...	2022-11-18 00:30:00	Waitlist
Dr. David ...	2022-12-18 10:30:00	Confirm
Dr. Abi Sh...	2022-12-18 11:00:00	Confirm

User Application Interface

We used a three tier architecture to implement the system user interface. The front end (which is what the user interacts with) has many features listed below. The front end passes information to the API. The API typically has some checks on the information before passing the information to the database in the form of a query.

Logging In As Admin

An admin has the ability to log in to the system. Once the user inputs the information, the information is sent to the API and then used to query Admins table in the database to find an admin that matches the credentials provided. If no admin is found, then the user is notified of invalid username or password and not logged in. If an admin is found, the user is logged in to the admin dashboard.

Viewing All Patients

An admin has the ability to view all patients currently in the database by clicking ‘Patient’ in the left panel. This is done in SQL by simply getting all records in the Patient table and then displaying them in the front end.

Creating a New Patient

An admin has the ability to create a patient by clicking ‘New’ in the ‘Patient’ section in the left panel. Once the user inputs the information, the information is sent to the API which will check for empty fields. If there are empty fields, then the user is notified that there is an invalid field and no patient is created. If there are no empty fields, then the API passes the information to the database by using an insert query on the Patient table.

Editing a Patient

An admin has the ability to edit patient information by clicking the green icon next to the patient in the ‘Patient’ section in the left panel. Once the user has the desired edits to the patient, the information is sent to the API. If there are empty fields, then the user is notified that there is an invalid field and the patient is not updated. If there are no empty fields, then the API passes the information to the database by using an update query using ‘patient_id’ and the Patient table to find the correct patient.

Viewing All Doctors

An admin has the ability to view all patients currently in the database by clicking ‘Doctor’ in the left panel. This is done in SQL by simply getting all records in the Doctor table and then displaying them in the front end.

Creating a New Patient

An admin has the ability to create a doctor by clicking ‘New’ in the ‘Doctor’ section in the left panel. Once the user inputs the information, the information is sent to the API which will check for empty fields. If there are empty fields, then the user is notified that there is an invalid field and no doctor is created. If there are no empty fields, then the API passes the information to the database by using an insert query on the Doctor table.

Editing a Doctor

An admin has the ability to edit doctor information by clicking the green icon next to the patient in the ‘Doctor’ section in the left panel. Once the user has the desired edits to the doctor, the information is sent to the API. If there are empty fields, then the user is notified that there is an invalid field and the doctor is not updated. If there are no empty fields, then the API passes the information to the database by using an update query using ‘doctor_id’ and the Doctor table to find the correct patient.

Creating an Appointment

An admin has the ability to create an appointment by selecting ‘Add Appointment’ in the ‘Dashboard’ section in the left panel. The ‘Patient’ field gets all patients from the database and displays their first and last name to be selected. The ‘Doctor’ field gets all doctors from the database and displays their first and last name to be selected. The ‘Service’ field gets all services from the database and displays the service name to be selected. These fields use a simple query that gets all of the information from the respective type in the database. Once the user inputs the information, the information is passed to the API. The information passed is not the names of the patient, doctor, and service, but rather the IDs of the selected items. If there are empty fields, then the user is notified that there is an invalid field, and the appointment is not created. If there are no empty fields, then the API passes the information to the database by using an insert query on the Appointment table and uses the IDs provided as foreign keys.

Log Out

An admin can log out of the system by clicking the logout icon in the top right corner. This will return them to the login screen. No SQL is needed for this operation.

Conclusions and Future Work

In conclusion, this project delivered to us a deep understanding of everything that goes on behind the scenes with all of the separate databases we interact with on the daily. It’s a lot more complex than out data just goes into a list and comes out of one. There are entire ER Diagrams and Database Schema that make up all of the rules of how and where our data will be stored. Then, there is ultimately the dependencies of relations that define what information impacts what.

Alongside all of our newfound database knowledge which spans much more, one of the biggest things we had to learn from was the connection between the front end and the back end. The front end, which users see and interact with, has to have a way to work with the backend which handles data in the database. This was the hardest part for us, but once we established the connection and functionality, we were able to start implementing features for the user application interface to interact with the database. As mentioned above, this included the features logging in as admin, viewing all patients, creating a new patient, editing a patient, viewing all doctors, creating a new patient, editing a doctor, creating an appointment, and finally the ability to log out.

This leads into the future work. There are many more features which can be added to this. First of all, we can implement more functionality for a patient or doctor to login in the system. This patient can view their appointments they have scheduled and can also view payments. We can also add a feature for admin to view payments as well. Showing appointments that are scheduled can also be implemented for doctors so that they may see what appointments they have for the day and maybe even other appointments that are scheduled if that is important to them. As is evident by the list above, there are many more things that can be added for the system to have further interaction with the database. For the most part, this involves logging in for student and doctor tables to be added which is already shown capable by the admin table, then just giving them some admin capabilities that only show results for themselves based on their keys. This proves that these functionalities are capable of being implemented as future work. Finally, there can always be more work done on a User Interface in order to make it more visually appealing or easily interactable for the user.

At last, it is evident that more features can be implemented for the entire stack of the project. Frontend and backend can work in conjunction to provide more features, and the frontend development of the interface can be adjusted to fit improving artistic standards or use standards.

References

inettutor. *Telemedicine Online Platform Free Bootstrap Template*. 23 October 2021. <https://www.inettutor.com/programming-tutorial/bootstrap/telemedicine-online-platform-free-bootstrap-template/>.

W3Schools. *CSS*. n.d. <https://www.w3schools.com/css/default.asp>.

—. *HTML*. n.d. <https://www.w3schools.com/html/>.

—. *JavaScript*. n.d. <w3schools.com/js/default.asp>.

Appendix

Link to the zip file: https://github.com/knives-knife/TeleMedCS6360/tree/main