APPONApp-The simple and convenient appointment app for day to day utilities of people.

1st Anwesh Das
Btech in Electrical Engineering(2017-2021)
Indian Institute of Technology
Mandi,India
b17116@students.iitmandi.ac.in

2nd Aniket Sahu
Btech in Computer Science Engineering(2017-2021)
Indian Institute of Technology
Mandi,India
b17076@students.iitmandi.ac.in

3rd Anurag Garg
Btech in Electrical Engineering(2017-2021)
Indian Institute of Technology)
Mandi,India
b17077@students.iitmandi.ac.in

4th Atyant Yadav
Btech in Computer Science Engineering(2017-2021)
Indian Institute of Technology
Mandi,India
b17037@students.iitmandi.ac.in

***Abstract*—This document is the report of our project named "APPONApp" which is an Web Application for booking appointment for many services such as Physician,Barber,Lawyer,Plumber,Electrician,Driver,Cleaning Personnel and many other professions which can be brought under the umbrella of our app and more importantly saving time as "Time is Money".**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

In today's time the two most powerful things are time and money.So we have taken an initiative through our app i.e the "APPONApp" to bring a balance between the two and hence make life simple and smooth for many people.This app saves time by being an alternative to people standing in long queues or waiting for another person's turn to be over and waiting for their turn to come and also money as it will help in generating proper business for the service provider and also for the customer to spend his money wisely according to the reviews of the service provider.

A. ReactJS for Front End

ReactJS is used in this app keeping in mind the modern day single page applications which will efficiently update and render just the right components when your data changes. Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.Since we plan to take this app to the mobile platform in near future so React is the best option as it can also render on the server using Node and power mobile apps using React Native.

B. NodeJS for backend

Node.js is a javascript runtime environment built on Chromes V8 JavaScript engine. It includes everything needed to execute a program written in JavaScript. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js package ecosystem, n.p.m., is the largest ecosystem of open source libraries in the world. It enables javascript to be used on server side which enables developers fluent in javascript to use their knowledge on the server side without the overhead of learning a new language like PHP, etc. Using npm, we can make use of available libraries to speed up our development process. We used bcrypt, json-web-tokens, etc to secure our app. We used express to create our server which interacts with database as well as the client to serve its needs.

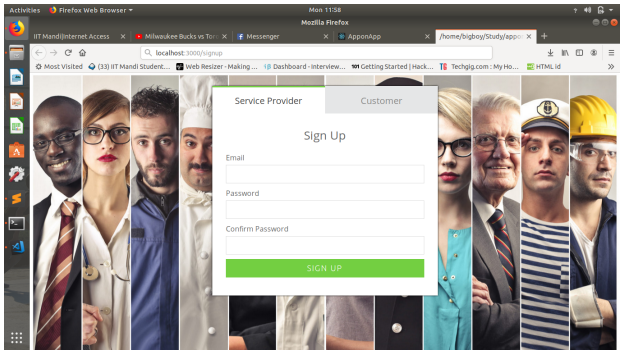


Fig. 1. SignUp with ReactJS

C. PostGreSQL for Database

- PostgreSQL, the most advanced open source database, offers many advantages over other database system. It also is available for almost every operating system with the latest stable release. PostgreSQLs Strength
- Open Source DBMS Only PostgreSQL provides enterprise-class performance and functions among current Open Source DBMS with no end of development possibilities. Also, PostgreSQL users can directly participate in the community and post and share inconveniences and bugs.

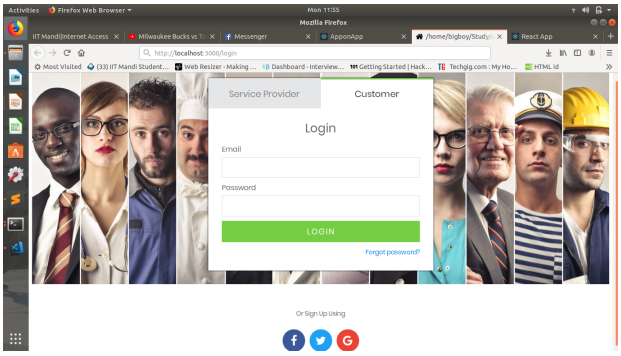


Fig. 2. Login with ReactJS

- Diverse Community One of the characteristics of PostgreSQL is that there are a wide variety of communities. Regarding PostgreSQL as Open Source DBMS, users themselves can develop modules and propose the module to the community. The development possibility is superiorly high with collecting opinions from its own global community organized with all different kinds of people. Collective Intelligence, as some might call it, facilitates transmission of indigenous knowledge greatly within the communities.
- Function SQL functions called Store Procedure can be used for server environment. Also, we support languages similar to PL/SQL in Oralcle such as PL/pgSQL, PL/Python, PL/Perl, C/C++, and PL/R.
- ACID and Transaction PostgreSQL support ACID(Atomicity, Consistency, Isolation, Durability).
- Diverse indexing techniques PostgreSQL not only provides B+ tree index techniques, but various kinds of techniques such as GIN(Generalized Inverted Index), and GiST(Generalized Search Tree), etc as well.
- Flexible Full-text search Full-text search is available when searching for strings with execution of vector operation and string search.
- Diverse kinds of replication PostgreSQL supports a variety of replication methods such as Streaming Replication , Slony-I, and cascading.
- Diversified extension functions PostgreSQL supports different kinds of techniques for geographic data storage such as PostGIS, Key-Value Store, and DBLink.

II. Technologies Used

III. Workflow

The Workflow of this app is pretty simple and can be easily understood by beginner developer.Hence we can easily add more updates and improvements in this app in near future.The ER(Entity Relation) Diagram is included in the report.

IV. Database usage and Manipulation

A. Database

We used postgresql for our database requirements because it is an open source Object Relational Database Management System (ORDBMS) that uses and extends the SQL language combined with many features. It is a highly scalable and flexible database providing many features that make it to be used as No SQL database as well. It highly extensible such that it can be extended using PostGIS, a spatial database extender for postgresql. PostGIS adds support for geographic objects allowing location queries to be run in SQL.

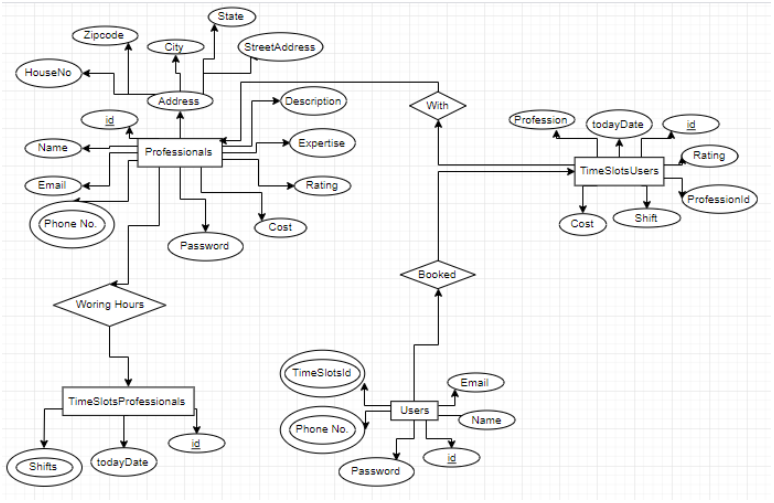


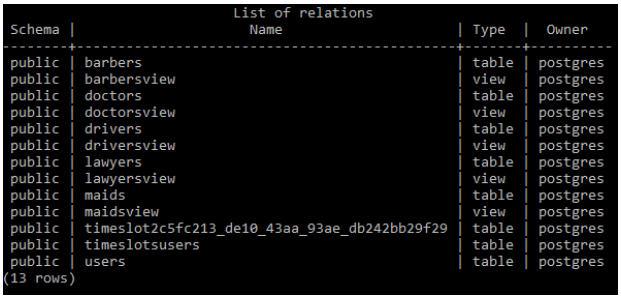
Fig. 3. ER Diagram

```
                                Table "public.doctors"
   Column    |          Type          | Collation | Nullable |      Default
-------------+------------------------+-----------+----------+-------------------
 id          | uuid                   |           | not null | gen_random_uuid()
 name        | character varying(40)  |           |          |
 email       | character varying(40)  |           | not null |
 phone       | integer[]              |           |          |
 address     | addr                   |           |          |
 expertise   | character varying(60)  |           |          |
 description | text                   |           |          |
 rating      | smallint               |           |          |
 cost        | numeric(7,2)           |           |          |
 password    | character varying(64)  |           | not null |
Indexes:
    "doctors_pkey" PRIMARY KEY, btree (id)
    "doctors_email_key" UNIQUE CONSTRAINT, btree (email)
    "doctorsrating" btree (rating)
Triggers:
    create_table_timeslots AFTER INSERT ON doctors FOR EACH ROW EXECUTE PROCEDURE create_table_timeslots()
    remove_table_timeslots BEFORE DELETE ON doctors FOR EACH ROW EXECUTE PROCEDURE remove_table_timeslots()
```

Fig. 4. Professional's Table

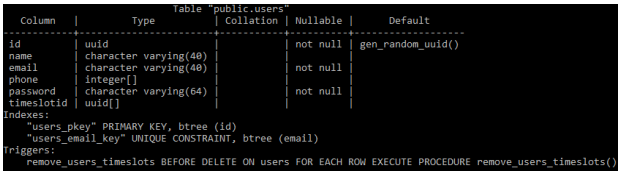
B. Tables

Our database comprises of tables for Professionals (Service Providers) such as doctors, drivers, maids, etc., Users (Clients), TimeSlots tables for each professional and a single booked TimeSlot table for users. We provide a table for each profession such as doctors, drivers, maids, etc. These tables will contain the data about each professional from that corresponding profession. Each professional will be having its own TimeSlots table that will be created dynamically, upon that professionals entry in their table, using trigger. This TimeSlots table will contain the data about that professionals working hours, cost for that particular working hour and the id of users who has booked that slot. Each professional is provided with their own TimeSlot table to make the database scalable and for optimizing the queries. If we had created a single table for every professional then we would have to search through large data set to serve the client with the available working of the service provider whose service they



```
                    List of relations
 Schema |                    Name                    | Type  |  Owner
--------+--------------------------------------------+-------+----------
 public | barbers                                    | table | postgres
 public | barbersview                                | view  | postgres
 public | doctors                                    | table | postgres
 public | doctorsview                                | view  | postgres
 public | drivers                                    | table | postgres
 public | driversview                                | view  | postgres
 public | lawyers                                    | table | postgres
 public | lawyersview                                | view  | postgres
 public | maids                                      | table | postgres
 public | maidsview                                  | view  | postgres
 public | timeslot2c5fc213_de10_43aa_93ae_db242bb29f29 | table | postgres
 public | timeslotsusers                             | table | postgres
 public | users                                      | table | postgres
(13 rows)
```

Fig. 5. Tables



```
                    Table "public.users"
   Column   |         Type          | Collation | Nullable |      Default
------------+-----------------------+-----------+----------+-------------------
 id         | uuid                  |           | not null | gen_random_uuid()
 name       | character varying(40) |           |          |
 email      | character varying(40) |           | not null |
 phone      | integer[]             |           |          |
 password   | character varying(64) |           | not null |
 timeslotid | uuid[]                |           |          |
Indexes:
    "users_pkey" PRIMARY KEY, btree (id)
    "users_email_key" UNIQUE CONSTRAINT, btree (email)
Triggers:
    remove_users_timeslots BEFORE DELETE ON users FOR EACH ROW EXECUTE PROCEDURE remove_users_timeslots()
```

Fig. 6. User's Table

want and also if a professional was updating his/her working hour we would have to lock that whole table for the transaction to complete, technically freezing our app. We provide a single table for users that contains all the details about the users who has signed up on our app. This user can book a slot, look for different services available and can get details about the professionals listed on our app. A single table for maintaining the TimeSlots booked by the users is also provided. It contains data about the slot booked, payment made and that users rating for the service provider whose service he/she took.

C. Triggers

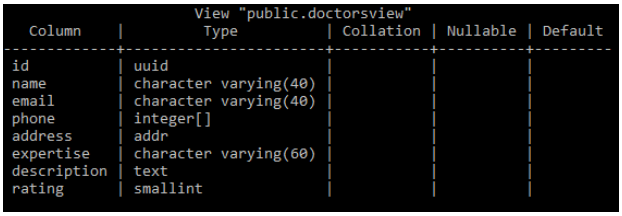
We used trigger to dynamically create a TimeSlots table for each professional upon its insertion in its corresponding table and to delete that TimeSlots table if the professional leaves our app. We have also used trigger to remove a users entry in TimeSlots table when that users leaves our app.

D. Indexes

Postgres makes use of B-Tree for indexing. Indexes are created in postgres by default for the primary keys, so each of our table makes use of that default index. Apart from that, we created index on professionals rating so to speed up searching when serving user with the professionals of high ratings. We intend to scale our app by providing text index on the expertise of professional so that a user can lookup for a professional using search works in minimum time. Each of our professional as well as user has to have a unique email so postgres also make an index on it by default.

E. Constraints

Each of our user and professional has to have a unique email that cannot be null and a password that also cannot be null.



```
                    View "public.doctorsview"
   Column    |         Type          | Collation | Nullable | Default
-------------+-----------------------+-----------+----------+---------
 id          | uuid                  |           |          |
 name        | character varying(40) |           |          |
 email       | character varying(40) |           |          |
 phone       | integer[]             |           |          |
 address     | addr                  |           |          |
 expertise   | character varying(60) |           |          |
 description | text                  |           |          |
 rating      | smallint              |           |          |
```

Fig. 7. View

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: keep-alive
Content-Length: 320
Content-Type: application/json; charset=utf-8
Date: Mon, 27 May 2019 09:17:10 GMT
ETag: W/"140-P0fitt8awBt75wPuOOUhIL0dsg8"
X-Powered-By: Express

{
    "email": "'me@gmail.com'",
    "id": "9af7ed02-770c-437c-85e7-01081a6a0592",
    "name": "'Amvesh'",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjlhZjdlZDAyLTc3MGMtNDM3Yy04NWJlLTAxMDgxYTZhMDU5MiIsImShbWUiOiInQW52ZXNoJyIsImVtYWlsIjoiJ21211QGdtYWlsLmNvbSc1LCJpYXQiOjE1NTgSNDg2MzB9.weKYtD0HO6aphHkgF3fv6cPrAgh_SxaQlb3XSG4wdU"
}
```

Fig. 8. Security

F. Views

We created views on each professions table that contains data about professionals excluding their passwords. This provides a layer of security and also enables us to open our api to other developers, who can use that data in their own applications.

G. Transactions

Transaction is used when in our app when a user books a slot, then that change has to be reflected in users TimeSlots table as well as professionals TimeSlots table so that no two users can book the same slot. Transaction is also used in situation where a professional is updating their working hours, then no user should be allowed to book a slot for that particular period or else it may lead to ambiguity in data.

H. Security

Postgres is a highly secure DBMS. For the security of our data we are keeping the passwords in the tables after salting and hashing. When a user signups or signins he/she will be supplies with a token particularly json-web-token that will contain that users basic information and will be hashed using a private key stored on our server (in the environment variable). To request information from the database, users has to provide that token, without it he/she will be denied access.

I. User defined attributes

Postgres allows for user defined custom attributes and domains that help in making database more flexible and sematic. In our database we have created a composite attribute for storing address of professionals. Postgres also allows for multivalued attributes that can stored in the form of array. We used it for storing phone numbers and timeslot ids for users. Postgres also has json data type that can be used to store flexibly structured data. We made use of it in storing professionals working slots.

```
                List of data types
 Schema | Name | Description
--------+------+-------------
 public | addr |
(1 row)
```

Fig. 9. User defined Type

```
                Composite type "public.addr"
    Column     |          Type          | Collation | Nullable | Default
---------------+------------------------+-----------+----------+---------
houseno        | character varying(20)  |           |          |
streetaddress  | character varying(100) |           |          |
city           | character varying(50)  |           |          |
state          | character varying(25)  |           |          |
zipcode        | character varying(10)  |           |          |
```

Fig. 10. User defined Type (address)

V. Features

A. Slot availability(Customer side)

The App shows what are the available slots for a particular service in a day.This helps the customer to choose wisely the time to visit according to his convenience and gives him an option to choose from the remaining available slots if the original slot is booked.

B. Slot availability(Service provider side)

The App helps the service provider to choose what are the time slots he/she wants to provide particular service in a day.This helps the service provider to easily divide his professional and personal life according to his convenience.This also helps him to know how many customers will be there before hand.

C. One stop solution for multiple utilities

Our app gives a functionality to sort the professionals on different basis such as price, ratings etc. This feature help us to get a professional according to our requirements. Also this app allows to book for more than one services hence this app is a very large platform for a number of services and services provider.

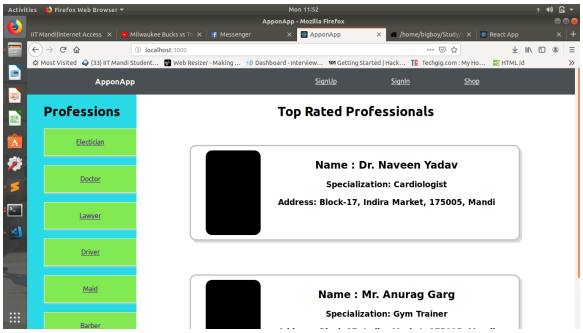


Fig. 11. Multiple profiles along with details

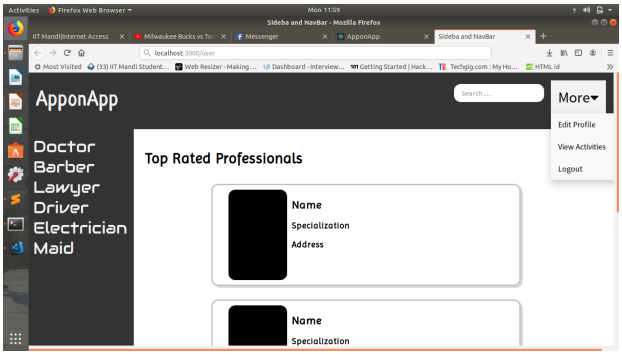


Fig. 12. Single profession Multiple Profiles

D. Large number of profiles of single profession with details

This app helps us to have a look over a number of profile simultaneously with their rate, their specialization, their address and many more. This helps us to save our time. All profiles of a particular profession are just a click away. This list of professionals are shown along with their profile photos, field of specialization, address, ratings and hence giving us an opportunity to compare among them.

E. Hassle Free Booking

The App helps the customers to book a service by viewing the available slots.There will no overlapping of booking of two persons as we are using a color coding concept.From the service provider sides we allow multi-device login and two bookings from same id will not be allowed at same slot.So this can be used properly by the service provider as well as his /her junior

VI. Uniqueness and Social Impact

This app is a first of its kind which offers to book utilities rather than hotel rooms ,rental cars or travel tickets.This app will bring a revolution in the service industry by bringing a sound balance between the two most important things time and money.This will lead to better lifestyle proper division of professional and personal life for the service providers ad reduce the pain of standing in long queues waiting for their turn for customers.

Acknowledgment

We acknowledge Dr.Arti Kashyap-our course instructor for the course(CS-309 : Information and Database systems) for allowing us to do this project which evolved throughout the timeline of this course by adding new features in our database and making this project more effective. We also acknowledge Mr.Sibaram Baral for his wonderful guidance and giving insights to the project by revewing it from time and helping us add new technologies and helping us resolving the errors with his superior professional industrial experience