**Application Name:** Dental Office (D.O.) / Dental Office Online Scheduling System
**URL**: *http://dentaloffice-app.s3-website-ap-southeast-1.amazonaws.com/*
**Developer**: Chua, Jose Paulo A.

Assumptions: Dentists already have existing time-slots

**1.) Front-end application**:
Technology used: ReactJS
Design:

| **Page** | **Route** | **Purpose** | **API calls** |
|---|---|---|---|
| About | / | Display information about the company | none |
| Services | /services | Display information about the available dental services | none |
| Book | /book | Allow logged in users to select dentists and book an appointment<br><br>If the user is not logged in, it will redirect to the login page | GET /api/booking/dentists<br><br>GET /api/booking/slots/{dentistId}<br><br>POST /api/booking/book |
| Login | /login | Allow registered users to log in<br><br>If login is successful it will redirect to the dashboard page | POST /api/login |
| Signup | /signup | Allow users to register an account | POST /api/signup |
| Dashboard | /dashboard | Allow users to view their profile and appointments | GET /api/dashboard/{userId}<br><br>DELETE /api/dashboard/{appointmentId} |

**2.) Backend Application**
Technology used: NodeJS (express)
Design:

| Method | URI | Request Body | Response |
|---|---|---|---|
| GET | /api/booking/dentists | none | dentist_id, dentist_name |
| GET | /api/booking/slots/{dentistId} | none | slot_id, slot_date, slot_time |
| POST | /api/booking/book | slot_id, user_id | INSERT query status |
| GET | /api/dashboard/{userId} | none | appointment_id, slot_date, slot_time, dentist_name |
| DELETE | /api/dashboard/{appointmentId} | none | DELETE query status |
| POST | /api/signup | user_name, user_gender, user_birthdate, user_email, user_password | INSERT query status |
| POST | /api/login | email, password | user_id, user_name, user_email, user_gender, user_birthdate |

**3.) Database**
Technology used: PostgreSQL (Relational)
Design:

Entity: **users**

| Column | Data Type | Constraints |
|---|---|---|
| id | Serial | Primary Key |
| name | Varchar (100 characters) | Not Null |
| email | Text | Not Null, Unique |
| password | Text | Not Null |
| birthdate | Date | |
| gender | Varchar (6 characters) | |

Entity: **dentists**

| Column | Data Type | Constraints |
|---|---|---|
| id | Serial | Primary Key |
| name | Varchar (100 characters) | Not Null |

Entity: **slots**

| Column | Data Type | Constraints |
|---|---|---|
| id | Serial | Primary Key |
| dentist_id | Int | Not Null, Foreign Key (dentists.id) |
| time_slot | Timestamp | Not Null |

Entity: **appointments**

| Column | Data Type | Constraints |
|---|---|---|
| id | Serial | Primary Key |
| slot_id | Int | Not Null, Foreign Key (slots.id) |
| user_id | Int | Not Null, Foreign Key (users.id) |

**4.) Deployment steps**

A. Frontend Application (Deployed to AWS S3)

1.) Build the react app into static pages
    >> npm run build
2.) Push the static pages into an s3 bucket
    >> aws s3 sync path/react/folder/build s3://s3-bucket-ipaddress --delete

B. Backend Application (Deployed to AWS EKS)

1.) Build the nodeJS application into a docker image
    >> docker build -t app .
2.) Update the docker image name to sync with the AWS ECR repository name
    >> docker tag app:latest aws-ecr-endpoint/app:latest
3.) Push the docker image into the AWS ECR repository
    >> docker push aws-ecr-endpoint/app:latest
4.) Update kubernetes config file to target the AWS EKS cluster
    >> aws eks --region ap-southeast-1 update-kubeconfig --name eks-cluster-name
5.) Deploy the ECR image to the EKS cluster
    >> kubectl apply -f deployment.yaml

C. Database (Deployed to AWS RDS)

1.) Connect to the AWS RDS endpoint using postgreSQL
    >> psql --host=rds-endpoint --port=5432 --username=postgres --dbname=postgres
2.) Configure and populate the database using sql scripts
    >> \i 'db_createTable.sql'