A PROJECT DOCUMENTATION

ON

“Web Development”

FOR

-XYZ Company

BY

KISHMAT BHATTARAI

Introduction

Problem Statement

Develop a website with CRUD operation for employee in XYZ Company with the help of backend APIs and a user-friendly frontend interface.

Project Scope

This project is basically managing the employee and their information in a software development company.

Scope:

- Creating an employee

- Reading info of employee

- Updating info of employee

- Deleting a particular employee

Project Success Criteria

Our main goal is to complete this project within allotted dead line and also with all operations. It is necessary to develop a method for capturing all the operations for employee management system is being developed, tested, and after it is rolled out.

Project Period

The duration of the project is 3 days. A proper plan to complete all the task to be completed within the targeted period is done.

Project Schedule

Task	**Duration**
Create Backend API	1 day
Create Frontend React JS App	1 day
Documentation and Deployment	1 day

Project Requirement

A sound knowledge in frontend web development with React JS framework and knowledge of HTML and CSS. In addition to that, also knowledge of creating backend API using any framework or language.

System Architecture

Backend API Layer

This layer will consist of every backend related task of creating, reading, updating and deleting the employee in the company. The respective url to for every CRUD operation is built. With the mentioned feature in backend, we can work with the user interface to perform the operation.

Database Layer

This layer consists table of employee. Employee table consist the particular employee’s id and name which can be operated to create new, delete the particular employee and edit the data of employee.

Frontend Layer

This layer with consist frontend user interface where we can see the list of the employee in the backend database and also perform CRUD operation with frontend to manipulate the data in database. Fetching the url of the backend and performing the particular operation for the particular task is done in this interface layer.

Steps:

Backend API

I created the backend API with Django Rest framework.

Prerequisite

Before starting the API part, following setup is necessary:

1. Django Framework
2. A Django Web App running on your localhost server.

So I created a django project with command

```
django-admin startproject project_name
```

After that, I created a django app inside django project with command

```
python manage.py startapp app_name
```

I created a super user from django admin after which we can get the database, where we can add the data

Python manage.py createsuperuser

After building a running project in local server, I started to create the api from backend. For that, I installed the Django Rest Framework from pip using following command.

```
pip install djangorestframework
```

After installing add rest_framework in setting file **(settings.py)**.

```
        INSTALLED_APPS = [
    ...
    'rest_framework',
]
```

Now, we created the application for the api, similar to creating another app in django:

```
python manage.py startapp api
```

Now that I have created an application named “api”, I could see a folder with different files such as admin.py, apps.py, models.py, tests.py and views.py. After the application is created I need to register that app in my settings as installed apps.

```
        INSTALLED_APPS = [
…
C‘rest_framework’,
]
```

Create a Model

First I created the model for employee then added the following code in my models.py.

```
from django.db import models

class Employee(models.Model):
    name = models.CharField(max_length=200)

    def __str__(self):
        return self.name
```

Create a Serializer

Here we are dealing with the data stored in a database which needs to be retrieved via web requests with the help of API. Therefore the best way of fetching the data is in the form of JSON. So we need to parse the data from database in JSON format. The serializer does the job. Create a file named (serializer.py)

```
from rest_framework import serializers
from .models import Employee

class TaskSerializer(serializers.ModelSerializer):
    class Meta:
        model = Employee
        fields = '__all__'
```

Configure URLs

We are configuring URLs in the following approach.

1. Create URL for api web app.
2. Link that api url to Django Project's main URL

In urls.py, I added the following URL path:

```
        urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/',include('todo_api.urls')),
]
```

Now in the api folder which is our app, I created a file urls.py and add the following paths:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.apiOverview, name="api-overview"),
    path('employee-list', views.employeeList, name=" employee-list"),
    path('employee-detail/<str:pk>/', views.employeeDetail, name="employee -
detail"),
    path('employee-create/', views.employeeCreate, name="employee-create"),
    path('employee-update/<str:pk>/', views.employeeUpdate, name="employee-
update"),
    path('employee-delete/<str:pk>/', views.employeeDelete, name="employee-
delete"),
]
```

Creating Views

We have to build models and serializer. Now our task is to create a view where we perform CRUD operations. Since we have configured the URLs.

API Overview

In our view, we will create a first view that lists all the URLs that are going to be exposed. We shall be using a decorator api_view provided by the rest framework. We need to import the EmployeeSerializer we build earlier. The employee serializer parses the request from the database and represents JSON strings We have to build models and serializer. Now our task is to create a view where we perform CRUD operations. Since we have configured the URLs.

```
from rest_framework.decorators import api_view
from rest_framework.response import Response
from .serializers import EmployeeSerializer

from .models import Task
# Create your views here.

@api_view(['GET'])
def apiOverview(request):
    api_urls = {
        'List': '/employee-list/',
        'Detail View': '/employee-detail/<int:pk>/',
        'Create': '/employee-create/',
        'Update': '/employee-update/<int:pk>/',
        'Delete': '/employee-delete/<int:pk>/',
    }
    return Response(api_urls)
```

Then run your server and browse the url: http://127.0.0.1:8000/api/.

To add data in database we should register our model in admin.py file.

```
from django.contrib import admin
from .models import Employee
# Register your models here.

admin.site.register(Employee)
```

Now we need to create the respective view for the list view, create, read, update and delete the employee from database modal in our views.py file.

Employee-List

```
@api_view(['GET'])
def employeeList(request):
    employees = Employee.objects.all()
    serializer = EmployeeSerializer(employees, many=True)
    return Response(serializer.data)
```

Employee-Read

```
@api_view(['GET'])
def employeeDetail(request, pk):
    employees = Task.objects.get(id=pk)
    serializer = EmployeeSerializer(employees, many=False)
    return Response(serializer.data)
```

Employee-Create

```
@api_view(['POST'])
def employeeCreate(request):
    serializer = EmployeeSerializer(data=request.data)

    if serializer.is_valid():
        serializer.save()
    return Response(serializer.data)
```

Employee-Update

```
@api_view(['POST'])
def employeeCreate(request):
    serializer = EmployeeSerializer(data=request.data)

    if serializer.is_valid():
        serializer.save()
    return Response(serializer.data)
```

Employee-Delete

```
@api_view(['DELETE'])
def employeeDelete(request, pk):
    employee = Task.objects.get(id=pk)
    employee.delete()
    return Response("Task Deleted Successfully")
```

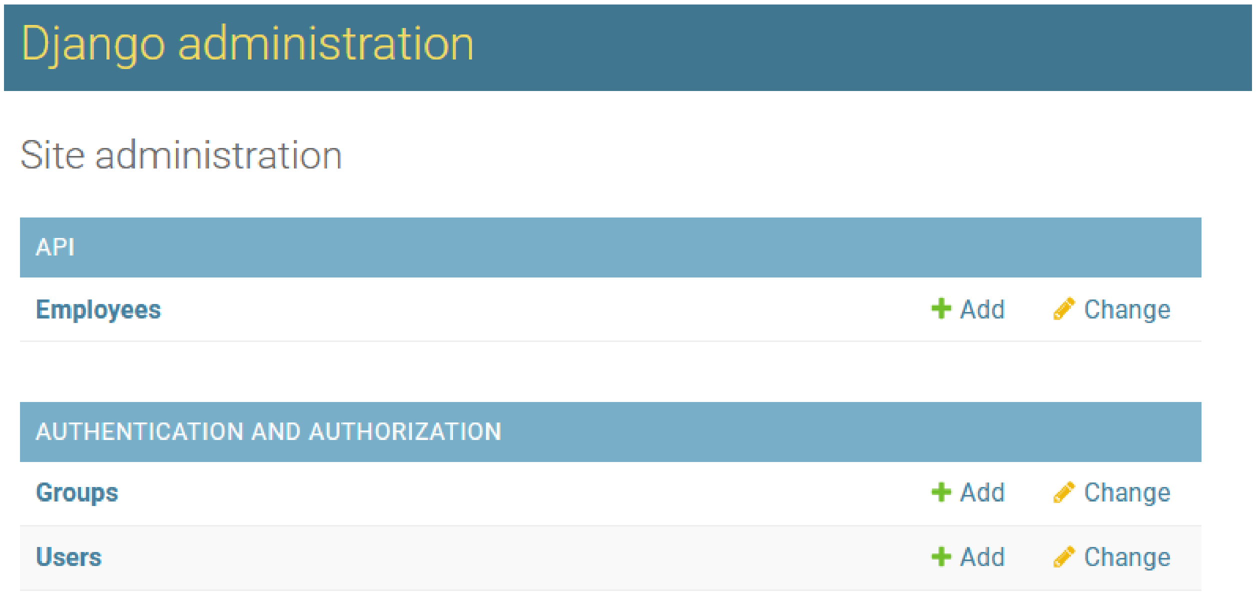
Then, after getting successfully running backend part I deployed it on heroku to get this api fetched anywhere in the world. For that I used cors-headers for letting others website to fetch the data for the api I created by adding this in settings.py file.

```
CORS_ORIGIN_WHITELIST = ["http://localhost:3000",]
```

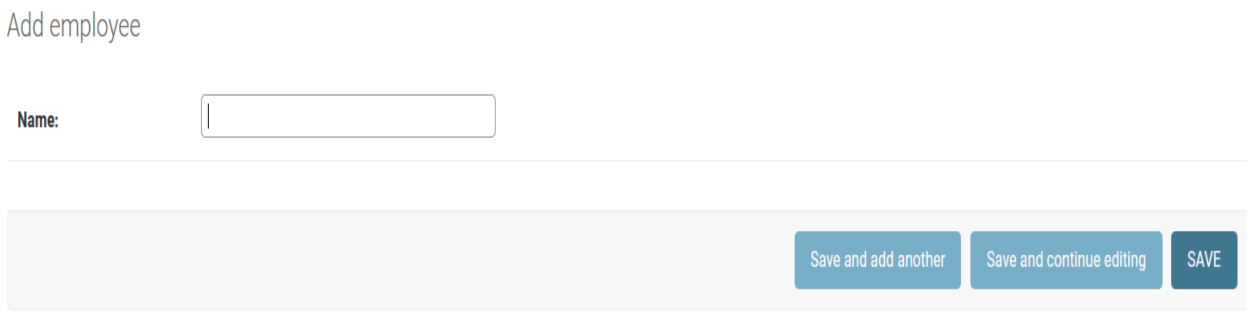
Database

For adding data in database, manipulating it and deleting it we use database, which can be accessed by using admin login page.

After logging in the admin page, we see this page



Then we can add any number of employess to the database.



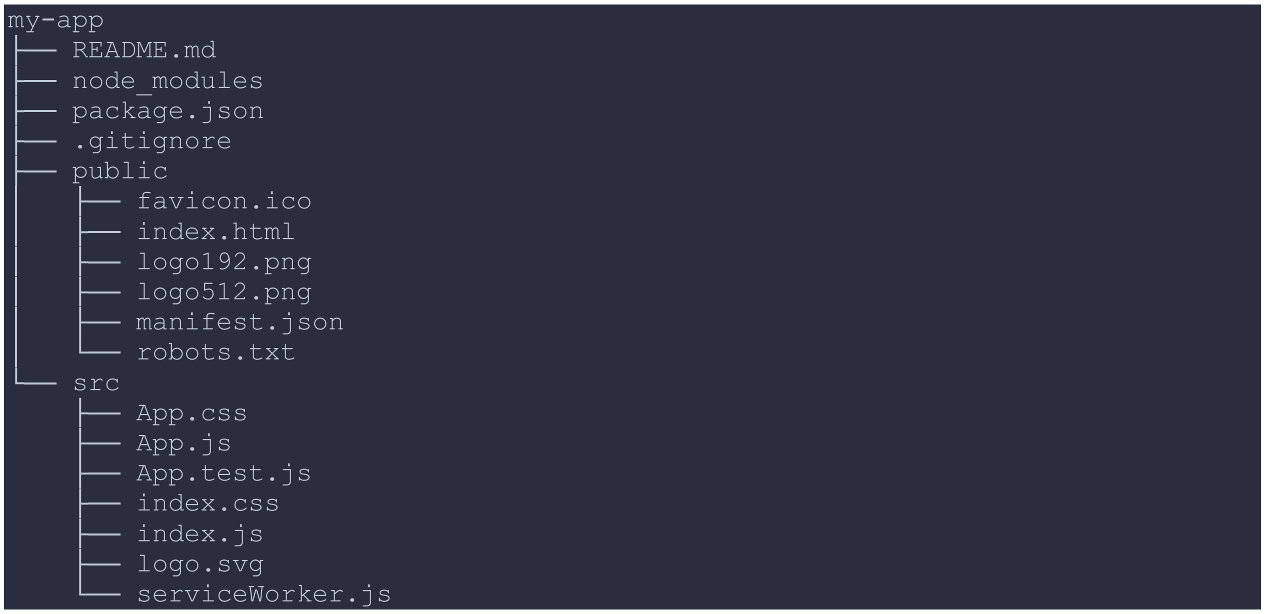
Frontend Interface(with React JS)

For starting the project we use npx which is npm package runner wheares npm is a package manager for the JavaScript programming language.

Create React App is a comfortable environment for learning React, and is the best way to start building a new single-page application in React.

```
npx create-react-app my-app

cd my-app

npm start
```

Running any of these commands will create a directory called my-app inside the current folder. Inside that directory, it will generate the initial project structure and install the transitive dependencies:



```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
```

Then I configured App.js and deleted some file such as logo.svg from src folder of the my-app.

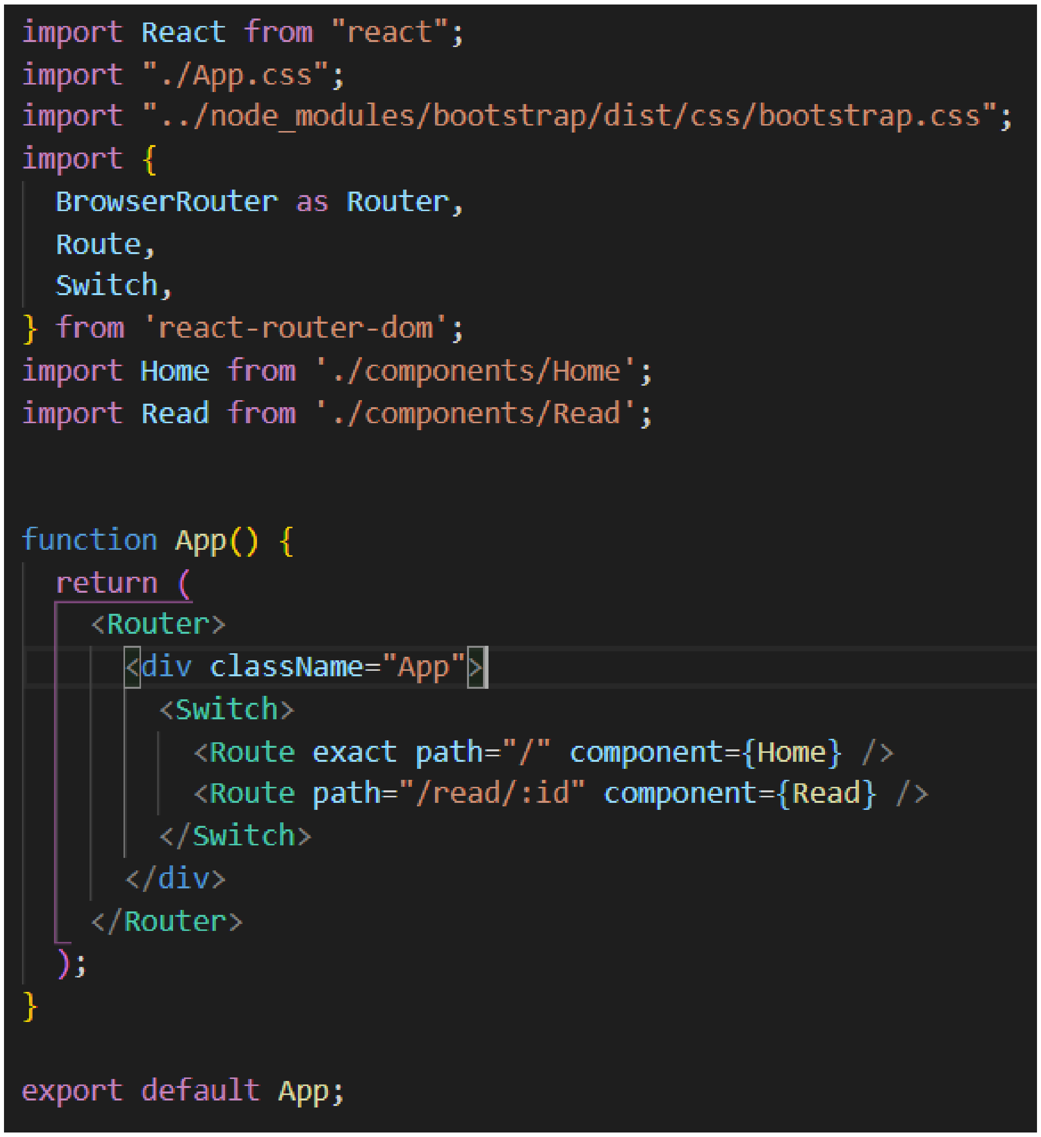
In react we solve the problem in components. I have a parent component App.js and two child components in components folder of src, where child component Read.js and Home.js are located.

I have used functional component to build my project as it has feature of Hooks which enables us to use state management hook useState and manage api call by useEffect. I have installed dependencies like axios for api call, bootstrap for css and react-router-dom for mange Routing.

```
npm install bootstrap

npm install axios

npm install react-router-dom
```

Parent Component

App.js only manages the routing to tell which url leads to with component.



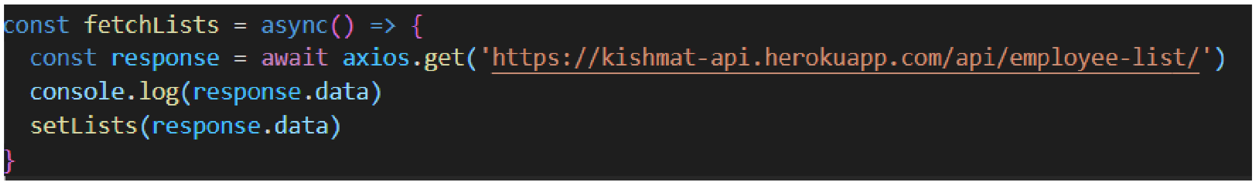
```
import React from "react";
import "./App.css";
import "../node_modules/bootstrap/dist/css/bootstrap.css";
import {
  BrowserRouter as Router,
  Route,
  Switch,
} from 'react-router-dom';
import Home from './components/Home';
import Read from './components/Read';


function App() {
  return (
    <Router>
      <div className="App">
        <Switch>
          <Route exact path="/" component={Home} />
          <Route path="/read/:id" component={Read} />
        </Switch>
      </div>
    </Router>
  );
}

export default App;
```

Route is used to manage different pages in React JS application here we have two pages for Home and Reading the particular employee.

And Home.js, one of the child component manages displaying all employee detail, creating new employee, editing one and deleting the particular employee with the help of fetching of the url created in backend.

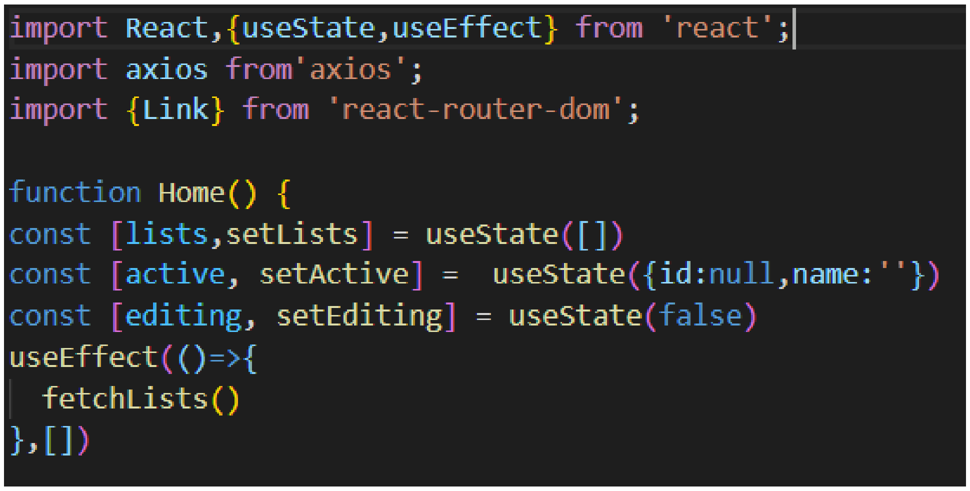
Here axios and fetch are used to get the data from api and edit and also delete the data of employee.

For example, we have fetched the list of employee and console log the data.



```
const fetchLists = async() => {
  const response = await axios.get('https://kishmat-api.herokuapp.com/api/employee-list/')
  console.log(response.data)
  setLists(response.data)
}
```

And talking about useState and useEffect, useState is used to manage the state and useEffect is perform certain task when page is rendered.



```
import React,{useState,useEffect} from 'react';
import axios from'axios';
import {Link} from 'react-router-dom';

function Home() {
const [lists,setLists] = useState([])
const [active, setActive] =  useState({id:null,name:''})
const [editing, setEditing] = useState(false)
useEffect(()=>{
  fetchLists()
},[])
```

And another child component deals with reading of the particular employee data. When we click on the button the it takes to new url which takes to new Component Read.js which is managed by Route.

Read.js read the data of employee by get request which the help of axios by using employee id passed in url as params. The read data is displayed in the user interface.

In this way, we completed the task of CRUD operation in backend with Django Rest Framework and front end with React JS.

Deployment

API Deployment

I deployed API on heroku. For that, the following commands are required:

```
heroku login

git init

heroku git: clone a <heroku-appname>

git add .

git commit –m “message”

git push heroku master
```

After deployment the resulted url is: https://kishmat-api.herokuapp.com/api/employee-list/

React App Deployment and Hosting

I hosted the code on github master and deployed on gh-pages. For that, I use the following command and changes in the file.

```
npm install gh-pages –save-dev
```

At first I created a github repository for example “repo-name”.

Add properties in package.json:

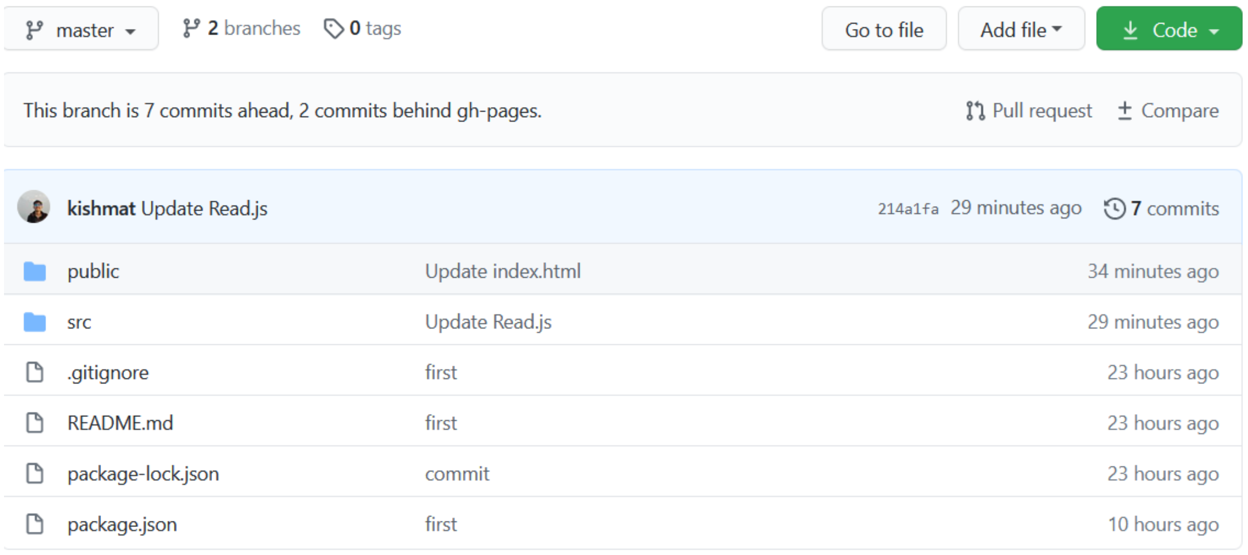
The first property we need to add at the top level homepage second we will define this as a string and the value will be "http://{username}.github.io/{repo-name}" {username} is your GitHub username, and {repo-name} is the name of the GitHub repository you created.

Second in existing scripts property we need to add predeploy and deploy.

```
"scripts": {

      //...

      "predeploy": "npm run build",

      "deploy": "gh-pages -d build"

      }
```

Then I opened git in the required directory and use the following command in git-bash terminal:

```
git init

git remote add origin <github-repo-link>

npm run deploy

git add .
git commit -m "Your awesome message"

git push origin master
```



This branch is 7 commits ahead, 2 commits behind gh-pages.

kishmat Update Read.js 214a1fa 29 minutes ago 7 commits

public	Update index.html	34 minutes ago
src	Update Read.js	29 minutes ago
.gitignore	first	23 hours ago
README.md	first	23 hours ago
package-lock.json	commit	23 hours ago
package.json	first	10 hours ago

After deployment on github pages, the link to the site is:

Link: https://kishmat.github.io/employee-app/