# Hotel Search Web App

Installation instructions and guidelines

Sunday, 11 August 2024

**TABLE OF CONTENTS**

## App structure

For considerations of future production deploy, maintenance and development, the application has be created on top of 2 projects:

- Backend: weski-search-backend
- Frontend: weski-hotel-app

**NOTE:**
**THE INSTRUCTIONS ARE GIVEN CONSIDERING VISUAL STUDIO CODE IS USED.**

For another development environments the instructions may differ slightly.

**NOTE:**
**FOLLOW TO THE DEMO SECTION OF THIS DOCUMENT FOR LIVE INSTALLATION AND FOR FURTHER VERIFICATION OF THE LOCAL INSTALLATION INTEGRITY, OR OPEN THE FOLLOWING LINK IN THE BROWSER OF YOUR CHOICE:**

**HTTP://18.191.168.239**

## Backend Configuration

Clone the public GitHub repository containing the backend project:
**git clone https://github.com/Pankist/weski-search-backend.git**

Once the code is pulled, update the project dependencies:
**npm install**

Start the Node.js application:
**node server.mjs**

The server will start and listen on port 3000
The listened is accessible from **http://localhost:3000**

## Frontend Configuration

Clone the public GitHub repository containing the frontend project:
**git clone https://github.com/Pankist/weski-hotel-app.git**

Update the project dependencies:
**npm install**

For production environment you might consider using pm2, but for the development environment start the app by executing the following command:
**npm start**

As the app starts, pay attention to the command prompt, as the build identifies the port 3000 is already occupied by the Backend listener. The prompt will ask to confirm selection of another port by typing Yes (or Y).

The frontend app will start on the port 3001.
Verify the React app runs properly by opening in your browser:
**http://localhost:3001**

## Testing the React App

Prior to executing the project please verify the installation by running unit tests.
The Header and HotelCard components are covered with unit tests which ensure the continues development of the app.

Execute the tests by running the following command:
**npm test**

## Maintaining the projects

### WeSki search backend

- server.mjs - main backend file with all the logic.
- skiResorts.json - the hotel configuration file.
- apiEndpoints.json - JSON configuration for adding hotel search APIs. The configuration supports any required number of required APIs.

```
 [
     {
         "name": "MainAPI",
         "url": "https://gya7b1xubh.execute-api.eu-
west-2.amazonaws.com/default/HotelsSimulator"
    },
    {
         "name": "SecondaryAPI",
         "url": "https://gya7b1xubh.execute-api.eu-
west-2.amazonaws.com/default/HotelsSimulator"
    },
    {
         "name": "TertiaryAPI",
         "url": "https://gya7b1xubh.execute-api.eu-
west-2.amazonaws.com/default/HotelsSimulator"
    }
]
```

Add APIs by specifying name and the url of the service.
Make sure the payload provided by the API services is according to the initial response (refer to the company documentation)

For production environment, consider moving the configuration to a shared DB storage behind a cache with long TTL as the the responses do not frequently change.

For people searching through different geographic locations, consider having sharding by location as well as multiple cache origins.

### WeSki React Hotel App

For considerations of maintainability, ease of development and deploy, the frontend app has been created to use separate modules:
- Header.js - the module responsible for the header part of the interface
- Header.test.js - the file containing the unit tests for the header module
- HotelCard.js - the module responsible for a single hotel card of the search results
- HotelCard.test.js - the unit tests file of the corresponding module
- App.js - the main project file with the display logic and the streaming process implementation
- skiResorts.json - the hotel configuration file

The skiResorts file is the same as used by the backend and should be moved to a shared, sharded and cached service for production environment.

## Architecture Considerations

- React app uses separated modules for the UI components to support maintainability and continues development and deployment.

- For communication between the server and the client, it's been decided to use streaming instead of regular multiple fetch call.

It's been done to support the requirements:
- Fast load of results
- Multiple APIs support
- Continues load of results as the backend API calls return with data

For production environment, consider moving all the configuration files to a shared DB storage behind a cache with long TTL as the the responses do not frequently change.

For people searching through different geographic locations, consider having sharding by location as well as multiple cache origins.

## Live Demo

For simplicity of the installation, a live demo has been deployed to a production server on Amazon (AWS).

**Deployment region:** Oregon, USA
**Instance:** Amazon ec2 micro instance
**Web server:** nginx

As the demo is intended to use both projects to run on the same machine, installation had to use proxy to avoid CORS conflicts.
nginx is best fit for this matter.

Localhost:3000/search is mapped internally to api/search to support same-server installation.

Node.js is running behind pm2 process manager to support failsafe restarts of the app for the cases the servers gets restarted or moved the location.

Open the demo by accessing the URL in the browser of your choice:
**http://18.191.168.239/**

## Conclusions

Although the production server does not have sorting, the dev version does have it.

All the cards are order by price in the ascending order as the data keeps arriving.

## Follow Up and Contact Info

For any questions regarding the installation process, running the demo or the local environment, please contact Andrey Cooper at:

Email: andruka@gmail.com

Whatsapp: 05330222008
Telegram: @pankist