2022

YEAH (Your Events Are Here)

CAB432 Assignment 1

Eunyoung(Jasmine) Hur

N10622012

9/17/2022

Table of Contents

Introduction

Mashup Purpose & description

This project is an application that shows events in USA by category. Especially, people who like sports go to other countries to watch the game in person, and it is an application for those people. Using the SeatGeek API and the Leaflet API to search for event types, Leaflet shows where the game takes place on the map. Use the Openweather API to display the weather within 5 days and users can bring raincoats to the stadium if there is rainy.

Services used

API 1 -SeatGeek API

The SeatGeek API provides company's dataset of nearly 100,000 live events. The RESTful API returns requests in JSON, JSONP, and XML and can provide detailed information on venue lat/lon, average ticket price.

Endpoint: `https://api.seatgeek.com/2/events?taxonomies.name=sports`

Docs: https://platform.seatgeek.com/

API 2 – Leaflet API (v1.8.0)

Leaflet displays a map by displaying a series of tiles across the page. The tile layer definition has z, x, and y in it. They're replaced with actual values whenever Leaflet needs to fetch a tile.

Endpoint: `https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png`

Docs: https://leafletjs.com/reference.html

API 3 – OpenWeatherMap API

The OpenWeatherMap API provides the 5 day forecast with a 3-hour step for any location on the globe. It formats JSON and XML.

Endpoint: `api.openweathermap.org/data/2.5/forecast`

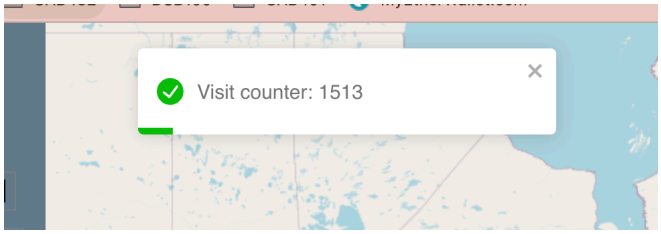
Docs: https://openweathermap.org/api

Persistence Service

The persistence service made use of S3 that is configured server-side. First, it needs to create bucket for containing body information which is counting visitor.

```
// create bucket
s3.createBucket({ Bucket: bucketName })
  .promise()
  .then(() => console.log(`Created bucket: ${bucketName}`))
  .catch((err) => {
    // We will ignore 409 errors which indicate that the bucket already exists
    if (err.statusCode !== 409) {
      console.log(`Error creating bucket: ${err}`);
    }
  });
```

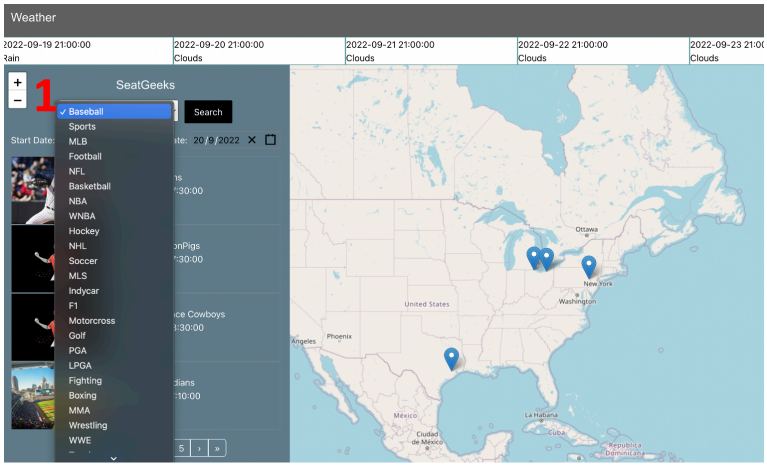
When a user accesses the main page (home page), a request is sent from the client to the server, and an S3 object's existence on the server side is verified. If there is an S3 object, the body that contains the number of visits will respond to the client side and be uploaded simultaneously.



Mashup Use Cases and Services

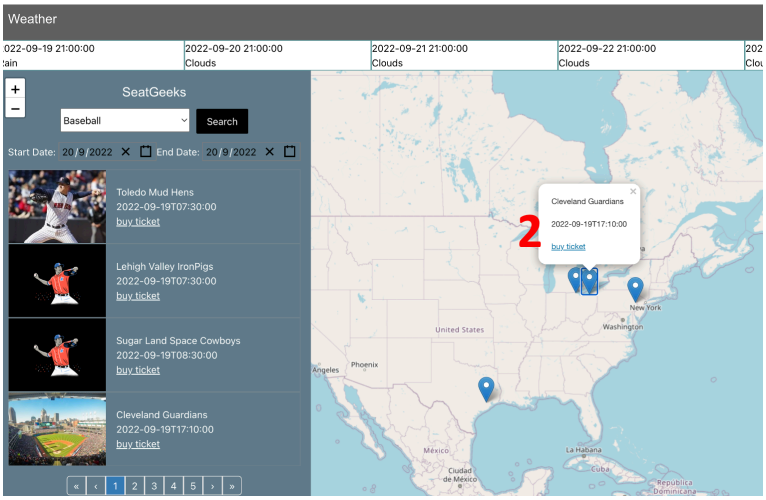
Event Search in a Foreign City

As an	Football event lover
I want	To go to the U.S. to see the game in person
So that	I can buy tickets for the performance that I want on the date and watch it.



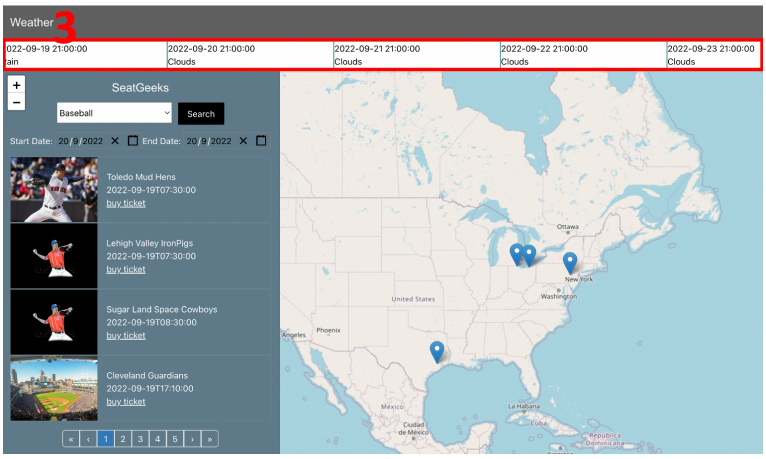
Finding event location on Map

As a	Football event lover
I want	to check where the football event is held in the U. S
So that	I can plan my route.

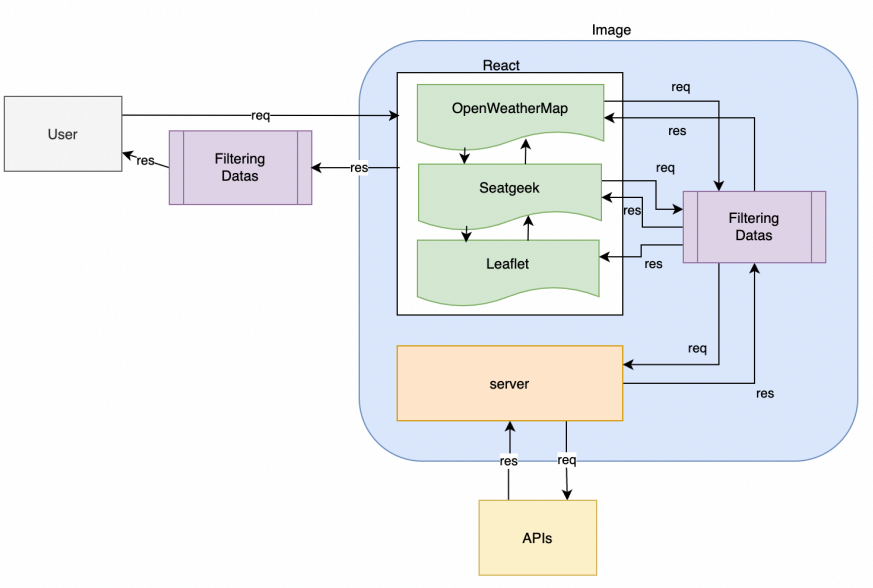


Weather checking

As a	Football event lover
I want	to check the recent weather
So that	I can check whether I should pack my raincoat or not.



Technical breakdown



This project aimed to provide users with the information they wanted in concisely. What is included in the docker image and architecture is displayed in the structure above.

Client-side

Client (react) sends a request to the server through fetch and then according to the user's action. Filter and transmits it to the server according to the the information received from the user. The information obtained from the server side is converted to JSON and presented as an object array. Filter it and display it to the user.
The data loading time was the most challenging aspect of Mashup. The data would not load instantly and came out 1-2 seconds later, thus it had to deal with this by using if conditions when if conditions are false, data doesn't render the screen while brining the data from server side.

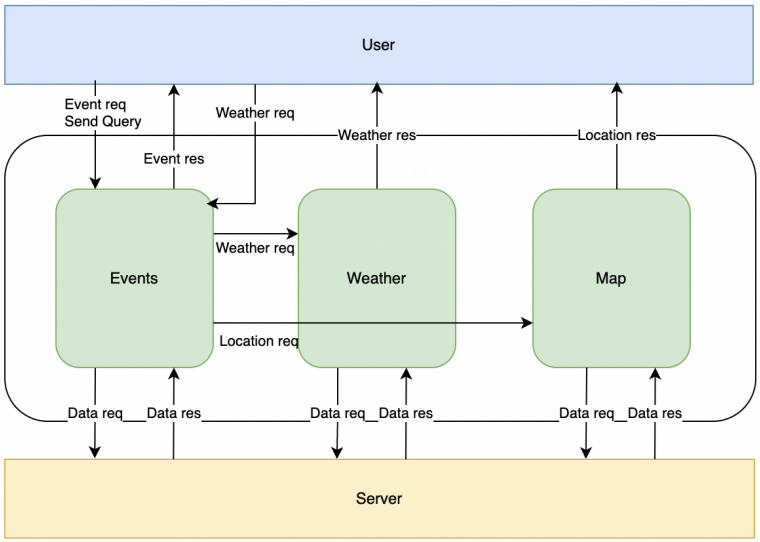
```
  if (SeakGeekAPIData !== null && SeakGeekAPIData !== undefined) {
    if (SeakGeekAPIData.length !== 0) {
```

Server-side

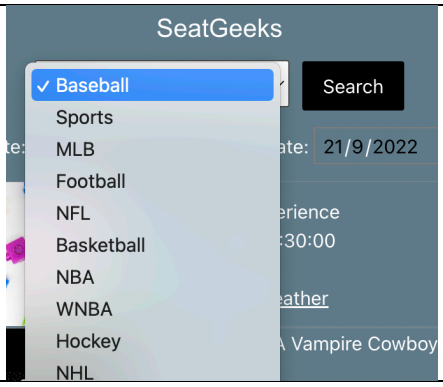
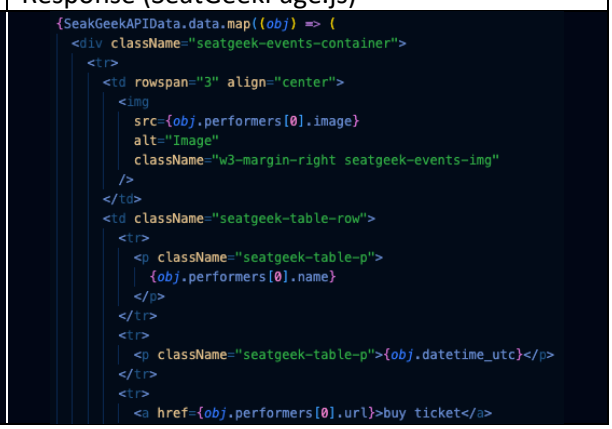
The server side requests data by request from the client side and sends it back to the client side. To get API materials, built a server with node js express and used Axios to get data from API endpoints. The object array is then delivered to the client side after the data has been converted to JSON.

```
  axios
    .get(WEATHER_ENDPOINT)
    .then((response) => {
      // console.log(response.data.list);
      res.json({ data: response.data.list });
      // res.send(express.static(path.join(__dirname,
'../../../client/build/index.html')));
    })
    .catch((error) => {
      console.log(error);
    });
```

Architecture and Data Flow



This application's overall structure is the same as that of the aforementioned architecture. First, the user requests information from Seatgeek on the screen, and Seatgeek requests data from the server. The client will display the screen if the data is returned from the server to the client. (User → Client-side →Server-side →Response) The table below shows that.

User	Client-side (SeatGeekPage.js)
	
Request/Server-side (seatGeek.js)	**Response (SeatGeekPage.js)**
	


```
fetch(url)
  .then((res) => res.json())
  .then((res) => {
    if (res.status === 500) {
      toast.error(`${res.status}`, {
        position: toast.POSITION.TOP_CENTER,
      });
    } else if (res.status === 403) {
      toast.error(`${res.status}`, {
        position: toast.POSITION.TOP_CENTER,
      });
    } else {
      // store datas and show map use state will be true
      setSeatGeekAPIData(res);
      setShowMap(true);
    }
  })
```

```
// get the data from API
axios
  .get(seatgeekEndPoint)
  .then((response) => {
    if (response.status == 200) {
      res.json({ data: response.data.events });
    }
  })
  .catch((error) => {
    if (error.response.data.status == 403) {
      next(error);
    }
  });
```

```
{SeakGeekAPIData.data.map((obj) => (
  <div className="seatgeek-events-container">
    <tr>
      <td rowspan="3" align="center">
        <img
          src={obj.performers[0].image}
          alt="Image"
          className="w3-margin-right seatgeek-events-img"
        />
      </td>
      <td className="seatgeek-table-row">
        <tr>
          <p className="seatgeek-table-p">
            {obj.performers[0].name}
          </p>
        </tr>
        <tr>
          <p className="seatgeek-table-p">{obj.datetime_utc}</p>
        </tr>
        <tr>
          <a href={obj.performers[0].url}>buy ticket</a>
```

Then, under Seatgeek, choose Weather if the user wishes to check the weather. When a user action is generated, Seatgeek provides information to the weather, and the client-side weather uses the information it has received from Seatgeek to make a server-side request for information.

User

Client-side (SeatGeekPage.js)

```
<Link
  to="/weather"
  state={{
    locations: [
      obj.venue.location.lat,
      obj.venue.location.lon,
    ],
  }}
>
  Check 5days Weather
</Link>
```

Client-side (OpenWeatherPage.js)

```
useEffect(() => {
  url = `/weather?lat=${lat}&lon=${lon}`;

  fetch(url)
    .then((res) => res.json())
    .then((res) => {
      setOpenWeatherAPIData(res);
    });
}, [OpenWeatherAPIData]);
```

Request/Server-side (openWeather.js)

```
axios
  .get(weather_endpoint)
  .then((response) => {
    res.json({ data: response.data.list });
  })
  .catch((error) => {
    console.log(error);
  });
```

Response/Client-side (OpenWeatherPage.js)

```
<div>
  {OpenWeatherAPIData.data.map((obj) => (
    <div className="openweather-div ">
      <p className="openweather-p">{obj.dt_txt}</p>
      <p className="openweather-p">{obj.weather[0].main}</p>
    </div>
  ))}
```

The location on the map is automatically changed and displayed according to the value of SeatGeek even if there is no special request from the user.

Client-side (SeatGeekPage.js)

```
LeafletData(SeakGeekAPIData);
return (
```

LeafletPage.js

```
const rendering = () => {
  const locationArray = [];
  for (let i = 0; i < eventArray.length; i++) {
    locationArray.push(
      <Marker
        position={[
          eventArray[i].venue.location.lat,
          eventArray[i].venue.location.lon,
        ]}
      >
        <Popup>
          <p className="leaflet-p">{eventArray[i].performers[0].name}</p>
          <p className="leaflet-p">{eventArray[i].datetime_utc}</p>
          <p className="leaflet-p">
            <a href={eventArray[i].performers[0].url}>buy ticket</a>
          </p>
        </Popup>
      </Marker>
```

Deployment and the Use of Docker

This project used docker for the deployment of the project. The contents of the docker file for creating an image of the docker are as follows.
First, when node:erbium is used to deploy the image for this project, it comes above 1GB, so the dock node image is utilized with 14-alpine. The remedy was to switch to the old version in order to lower the image's storage capacity. With 14-alpine, the image's storage space was cut in half, from 1.4GB to 431MB (a drop of roughly more than 50%).

Since this project used React, it needs to build a client-side file once through npm run build. Before the npm run build, all the dependencies must be completely downloaded, so it has to be installed the dependencies before the build. By enabling the file to execute without saving the node module (325.1MB) and build (2.4MB) directories inside the folder, these commands that mentioned above reduced the file's size. When executing the docker image, the variables are set so that API key and S3 access key are not hard-coded. Finally, the server.js file was made executable using the CMD command.

```
FROM node:14-alpine
# Key set
ENV AWS_ACCESS_KEY_ID XXID
ENV AWS_SECRET_ACCESS_KEY XXSECRET
ENV AWS_SESSION_TOKEN XXTOKEN
ENV SEATGEEK_API_KEY YYKEY1
ENV SEATGEEK_SECRET_API_KEY YYKEY2
ENV OPENWEATHER_API_KEY YYKEY3
```

```
ssm-user@ip-172-31-82-183:/usr$ sudo docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/ssm-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ssm-user@ip-172-31-82-183:/usr$ sudo docker pull jasminehur238/demo
```

Test plan

Materials related to the test plan are included in appendix A.

Screenshot/s Appendix A	Task	Expected outcome	result
1	Search events based on option	Change the list of the events according to type of event	PASS
2	Search events based on calendar	Change the list of the events based on date	PASS
3	Check the event location with map	Showing some information	PASS
4	Click the buy ticket button	Going to purchase page	PASS
5	Input wrong address	Prevent to go invalid page	PASS
6	click the page number without choosing event type	Showing notification message	PASS
7	When get in to main page	Increasing visit counter with popup	PASS
8	Check the weather	Going to other page to show weather status	PASS

9	Click the page number	Change location on map based on clicked page	PASS
10	Choose end date earlier than the start date	It is prevented to select	PASS

Difficulties / Exclusions / unresolved & persistent errors /

Difficulty1: Built image with Mac silicon

When creating an image on Mac silicon, it was impossible to run even if the image was created due to a format error. Therefore, solved the problem by putting --platform=linux/amd64 when creating and running the image.

```
docker build ––platform=linux/amd64 –t jasminehur238/demo .
```

Difficulty2: Restriction of Hook

React had the restriction that hooks could not be used simultaneously or inside of conditional statements (Rules of Hooks, 2022). The weather data was intended to be displayed on the same main page at the beginning, however, because of the "Hook" conditions, it was negotiated to move it to another page.

Difficulty3: Visit counter

It needed to use the useState to connect to the server and store server data in order to display the visit counter. But even if the client did nothing, employing useState made the number increase because the page kept hitting the server. The pop-up window was therefore used to display the data as soon as it was received in order to avoid this.

Difficulty4: Cannot find file directory on docker image

An HTML file for error handling was written, but there was a problem that the operation was good locally but it did not work in the docker image.

```
Error: ENOENT: no such file or directory, stat '/src/pageError.html'
```

Therefore, HTML was written as a string and replaced by showing it.

```
// send the error page
app.use((err, req, res, next) => {
  res.writeHead(404, { "content-type": "text/html" });
  res.write(errorPage);
  res.end();
});
```

User guide

The user manual is located in Appendix B.

Analysis

Question 1: Vendor Lock-in

Looking at your mashup as it stands, how dependent are you on the people who provide the services that you use? In a commercial context, the APIs, the data and the cloud services all matter to you. How hard would it be to change?

In your response, you should consider the following prompts:

- *How hard would it be to replace the APIs that you are working with? Could you easily replace them if they were shut down suddenly or their terms of service changed? Consider each of them in turn and explore the domain of the API and tell us about obvious competitors or their absence.*

  For any program, it is challenging to replace the used API. First, changing the API means that the format of the data must be checked again. Second, it is necessary to check how much the API service supports the data, and finally, if you cannot find an API that provides the same data, you may have to change the client side. Luckily, however, there are several APIs to replace SeatGeek, Leaflet, and open weather map APIs.

SeatGeek	Leaflet	OpenWeather Map
• Gametime • Eventbrite • Ticketmaster	• Google Maps • Mapbox • OpenLayers	• Meteomatics • Weatherbit • AccuWeather

  (4 marks)

- *How hard would it be to change the persistence service you have used to one supplied by another vendor? Identify equivalent alternatives and discuss briefly how that might affect your approach.*

  Changing persistent services is especially difficult if you change them to other companies. This is due to the fact that syntax must also be adjusted, even if performance, resiliency, and network security are all taken into account.

  (2 marks)

Question 2: Container Deployment

The mashup you have created as part of this assignment is a very limited application by commercial standards. Without getting too carried away, I want you now to think of a more substantial application which has similar characteristics – drawing mainly on external services, perhaps extending to include some user accounts and security, adding in some additional persistence services like those discussed in the weeks leading up to the assignment submission.

Working with this more substantial application, what are the advantages and disadvantages of the container-based deployment? This new context will in practice involve scaling and load balancing. You don't yet have direct experience of these services, so I want you to focus on the deployment of the application through software containers. You should refer to earlier notes about the trade-offs between containers and 'full' VMs and consider which might apply here, and which might not.

In your response, you should consider the following prompts:

- *Assuming that we have access to a service to manage container deployment and communication, are there any disadvantages to a container-based deployment for this application? Would we consider deployment of the application directly to a virtual machine i.e. one instance of the application for each instance of the EC2 or Azure Linux VM?*

  First, containers are still vulnerable to performance overhead owing to overlay networking, interface between containers and the host system and so on. Second, despite the fact that the fundamental Docker platform is open source, certain container products do not integrate with others, typically as a result of rivalry between the businesses that support them. Last, Storage of persistent data is challenging. By default, when a container goes down, all of the data within vanishes completely unless you save it somewhere else first.

  Deploying directly to a virtual machine requires consideration. First, heavy weight with limited performance. Second, each VM runs in its own OS and can't run more VMs on an average server, and third, Virtual machines take new minutes to start, so this should be considered carefully.

  (2 marks)

- *Drawing upon the discussions of cloud architectures in the early lectures and the material on persistence from week 5 onwards, what persistence options would you consider if the application were to be deployed at scale using a collection of software containers? You may consider more than one level.*

  I would like to consider AWS Amplify and MS Azure as persistence options.
  AWS Amplify provides a programming model that allows developers to leverage shared and distributed data without writing additional code for offline and online scenarios. This allows developers to create a rich app experience by making it as easy as working with distributed cross-user data as locally dedicated data. AWS Amplify is being used by Capstone for this semester, which supports both the front and back ends at the same time, providing a convenient experience for full-stack developers.
  MS Azure, on the other hand, provides high availability, massive scale, durability, and secure storage for various data objects in the cloud. Azure Storage data objects can be accessed from anywhere in the world via HTTP or HTTPS via the REST API.

  (2 marks)

References

Bk, C. (2019, February 19). *Application Deployment Strategy – Virtual Machine vs. Containers*. CodeProject. Retrieved September 21, 2022, from https://www.codeproject.com/Articles/1277634/Application-Deployment-Strategy-Virtual-Machine-vs

Deploy an Ubuntu VM with Docker Engine. (n.d.). Microsoft Azure. Retrieved September 21, 2022, from https://azure.microsoft.com/en-gb/resources/templates/docker-simple-on-ubuntu/

How to use built-in persistent storage in Azure Spring Apps. (2022, August 2). Microsoft Learn. Retrieved September 19, 2022, from https://learn.microsoft.com/en-us/azure/spring-apps/how-to-built-in-persistent-storage?tabs=azure-portal

Introducing the Amplify DataStore, a persistent storage engine that synchronizes data between apps and the cloud. (n.d.). Amazon Web Services, Inc. Retrieved September 21, 2022, from https://aws.amazon.com/about-aws/whats-new/2019/12/introducing-amplify-datastore/

Introduction to Azure Storage - Cloud storage on Azure. (2022, March 17). Microsoft Learn. Retrieved September 19, 2022, from https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction

Migrate data from Amazon S3 to Azure Storage - Azure Data Factory. (2022, August 6). Microsoft Learn. Retrieved September 19, 2022, from https://learn.microsoft.com/en-us/azure/data-factory/data-migration-guidance-s3-azure-storage

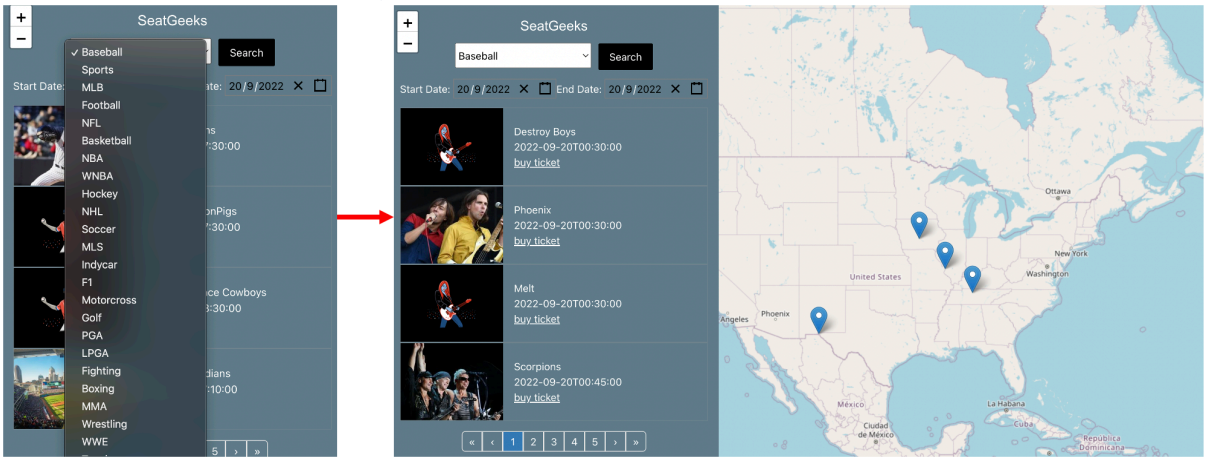
Rules of Hooks. (2022). React. Retrieved September 19, 2022, from https://reactjs.org/docs/hooks-rules.html

Tozzi, C. (2017, May 26). *Docker Downsides: Container Cons to Consider before Adopting Docker*. Channel Futures. Retrieved September 21, 2022, from https://www.channelfutures.com/open-source/docker-downsides-container-cons-to-consider-before-adopting-docker

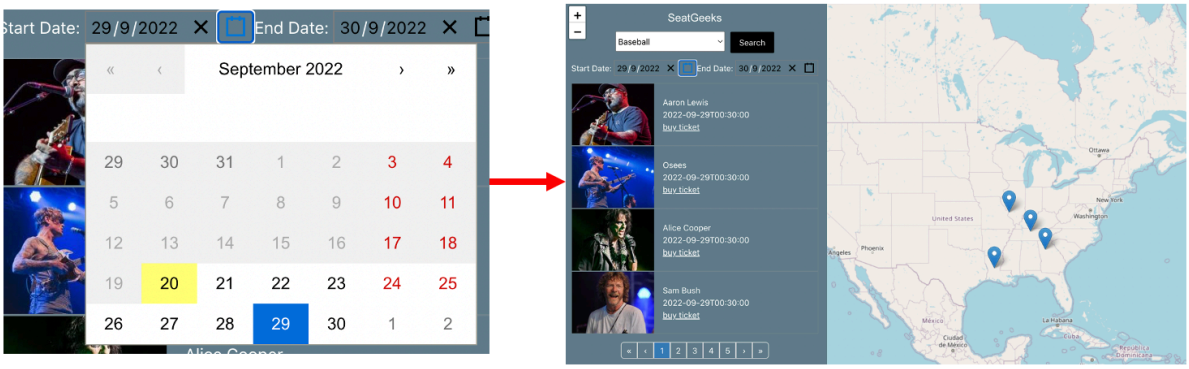
Appendices

Appendix A. Test case

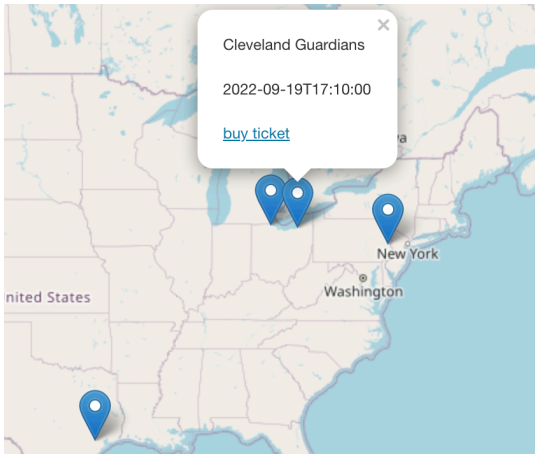
Case1: search events based on option



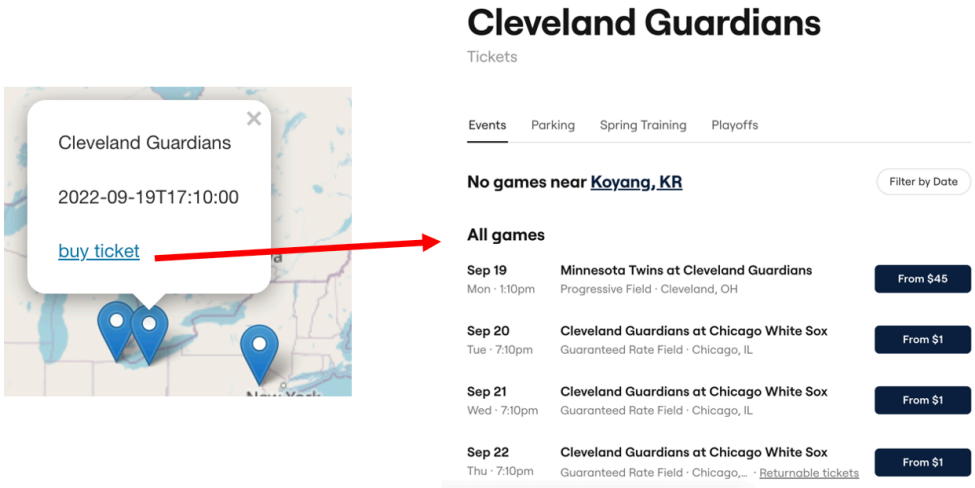
Case2: search events based on calendar



Case3: Check the event location with map

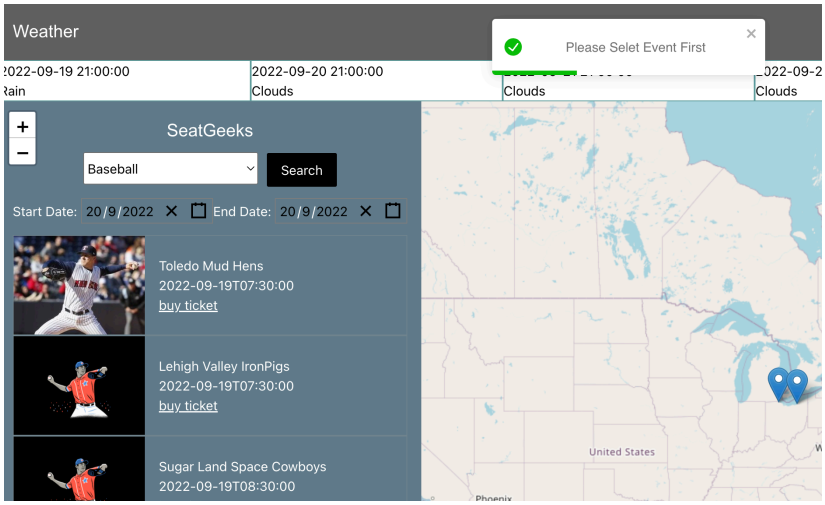


Case4: Redirecting to ticket purchase page when click the buy ticket button

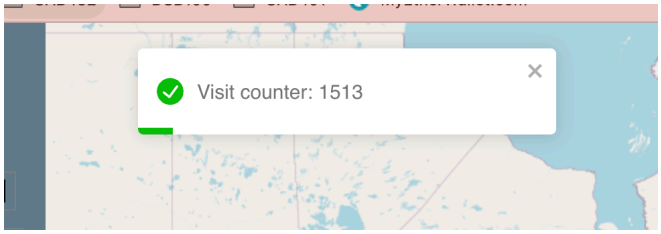


Case5: Prevent to go invalid route page

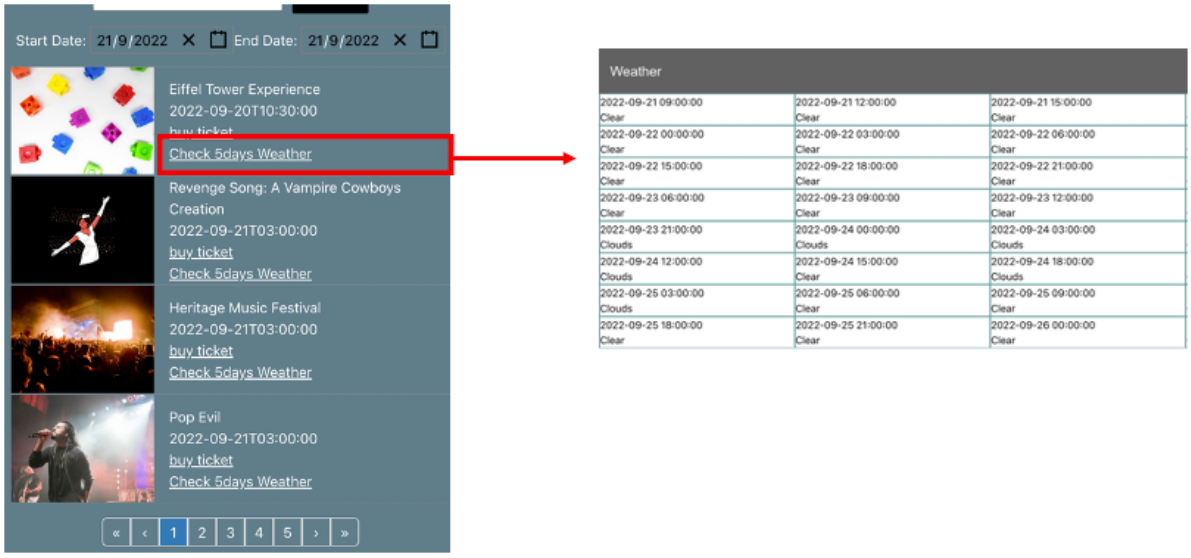
Case6: Showing message when click the page number without choosing event type



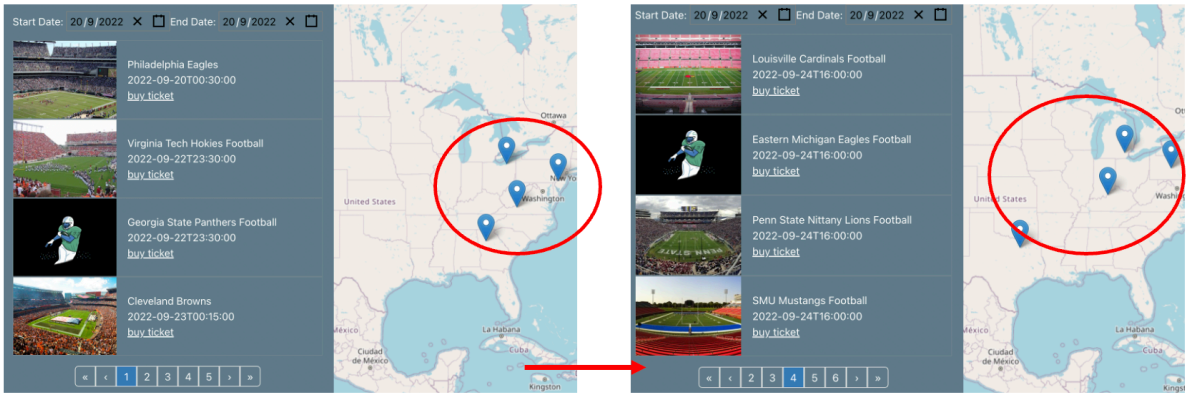
Case7: Increasing visit counter with popup



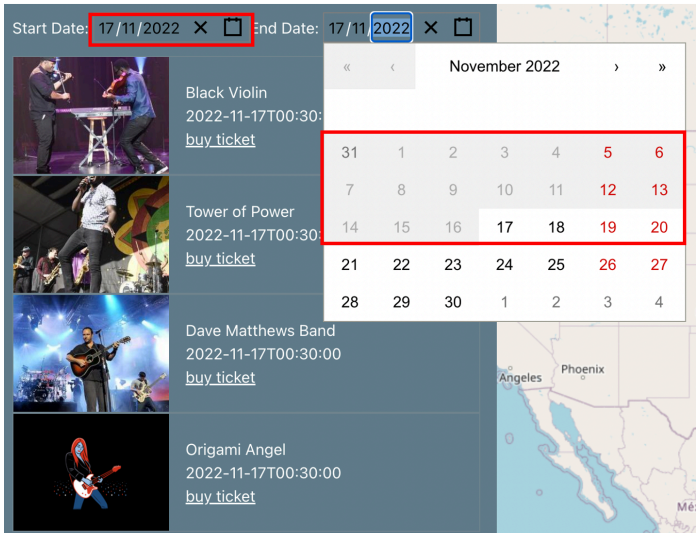
Case8: Check the weather



Case9: Change location on map based on clicked page

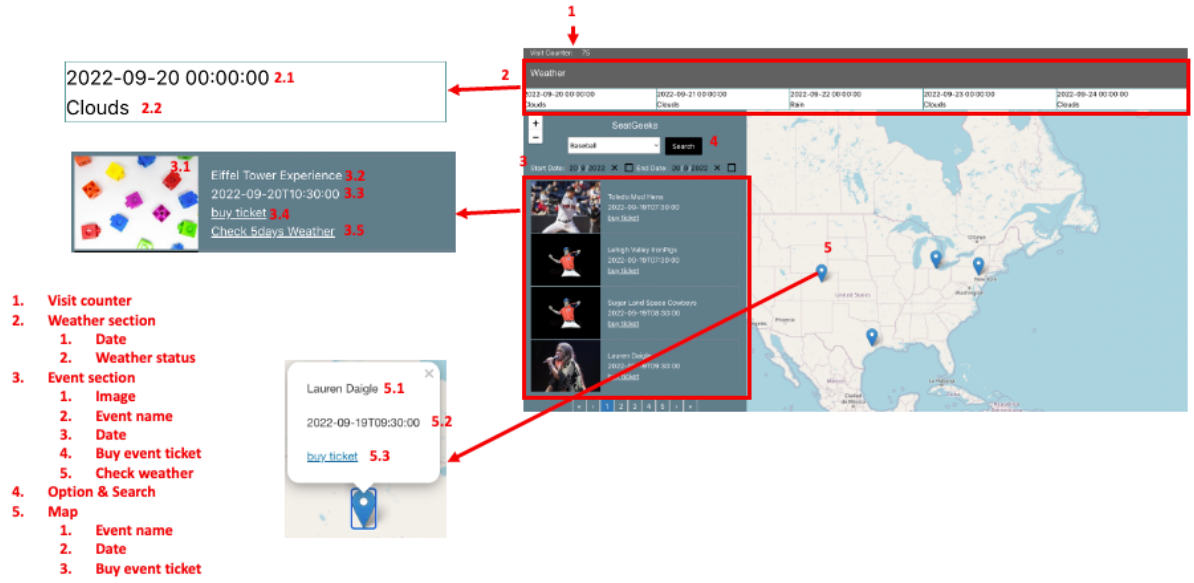


Case10: Choose end date earlier than start date

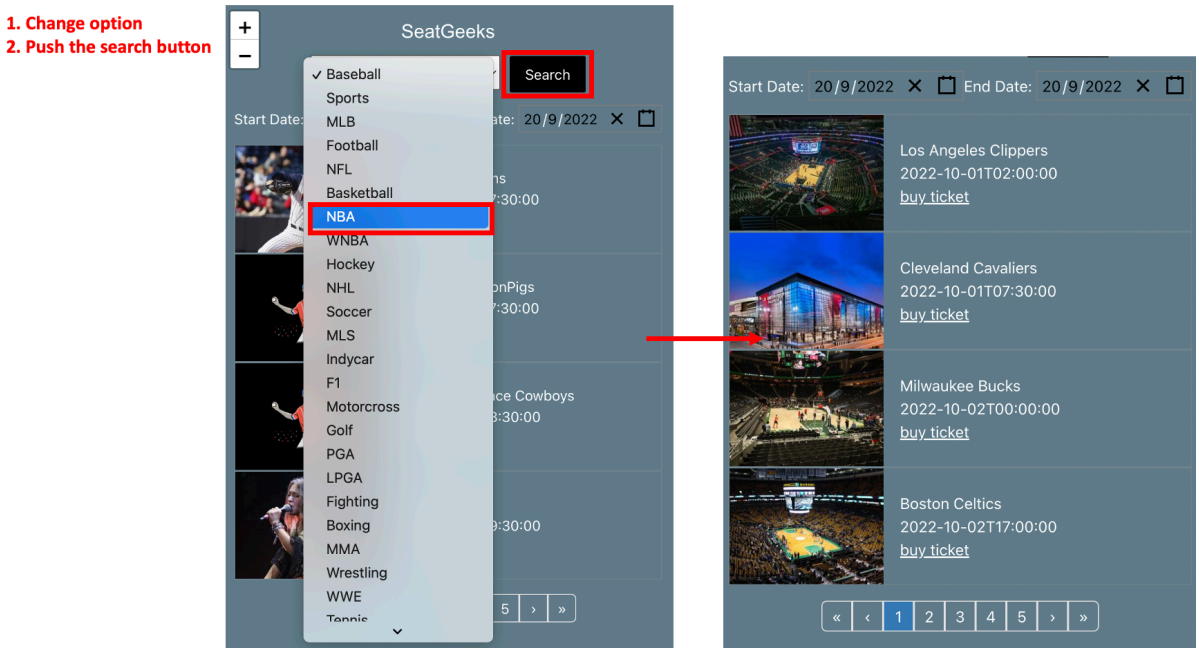


Appendix B. User Guide

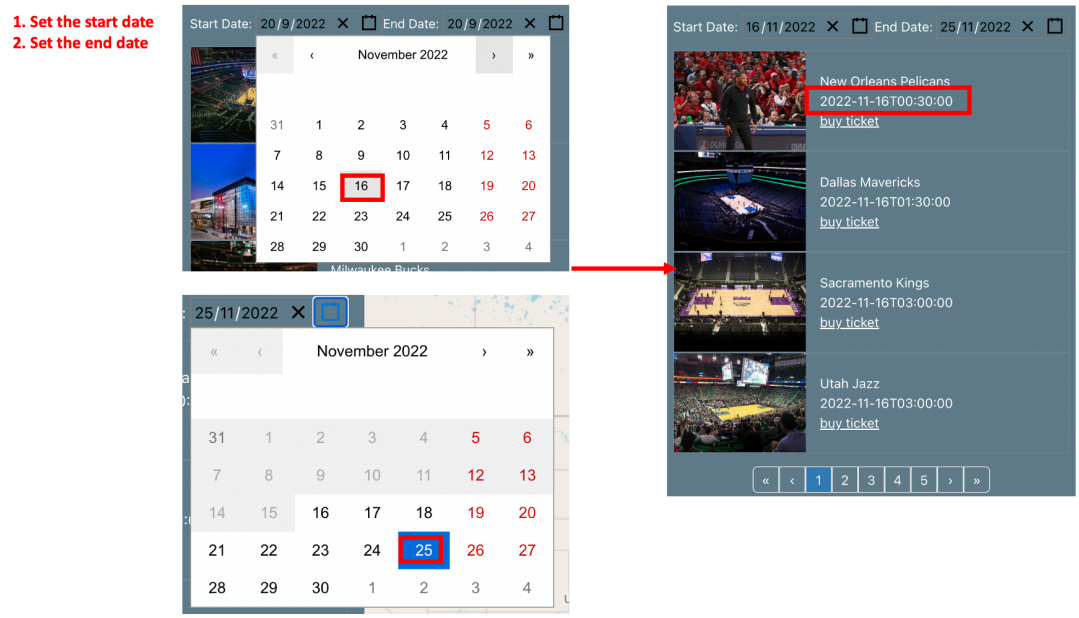
Main page contents



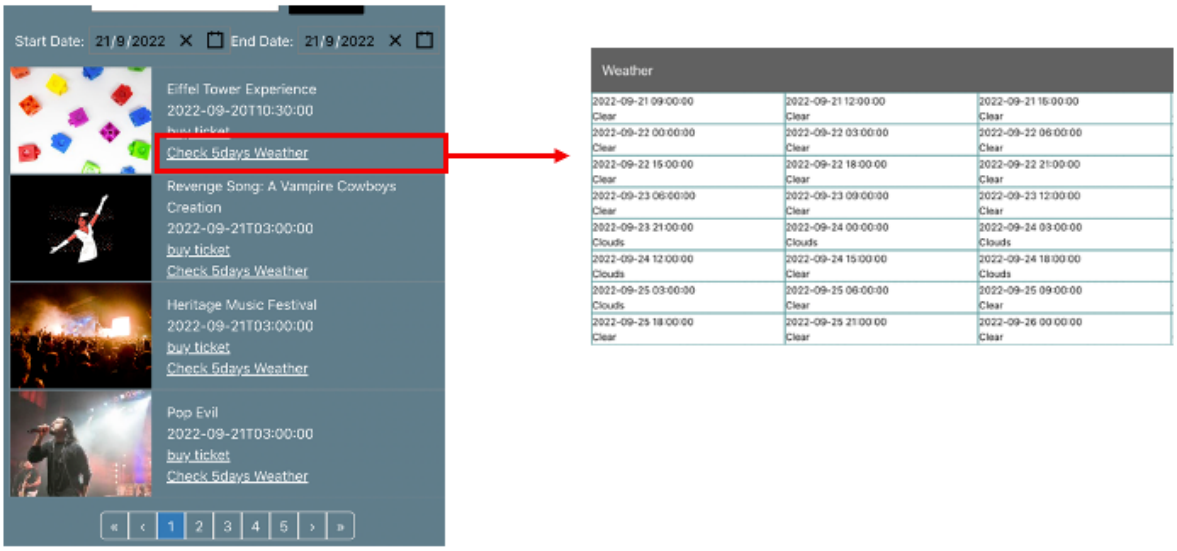
1. Change option and click search then you will see the result



2. Set the start and end date then you can check the event



3. Check the weather status



4. Check the date and event location