



CIS 5669: Communication Network Management

Spring 2024

A Project on

Kansas Camp Adventure

Sayali Prakash Kadam (#700751777)
Pavankumar Reddy Kunduru (#700760406)
Prashanthi Repakula (#700754388)

Submission Date: 04/20/2024

Abstract:

The camping project is a website designed to simplify the planning and booking process for camping trips. In today's dynamic landscape, where outdoor activities and recreational experiences are in high demand, camp owners require efficient tools to streamline operations and enhance customer satisfaction. The KC Adventure Camp Scheduling System addresses these needs by providing a comprehensive solution for managing campgrounds, scheduling bookings, and facilitating communication between camp owners and attendees.

At the core of the Camp Scheduling System is a powerful backend server built on Python and Flask, which handles data processing, business logic, and communication with external services. The frontend application, developed using HTML, CSS, and JavaScript, offers an intuitive and responsive interface for camp owners and attendees to interact with the system. Leveraging modern web technologies, the frontend application provides seamless navigation, real-time updates, and support for various devices and screen sizes.

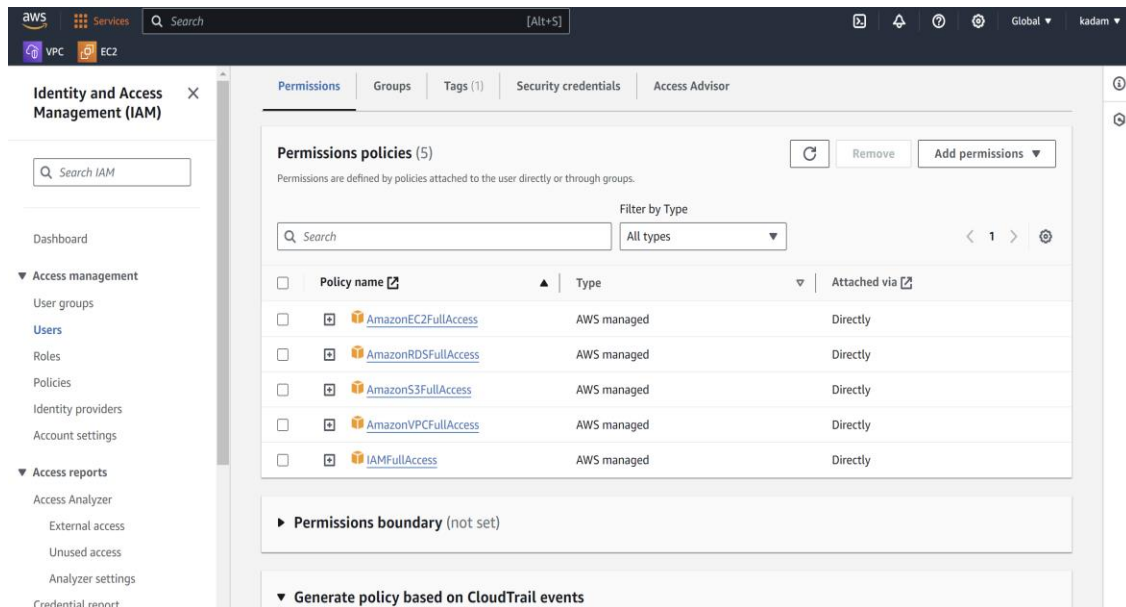
Key features of the system include:

- 1. Campground Management:** Camp owners can easily list their campgrounds, including detailed descriptions, availabilities, and pricing information. They can update availability calendars, set booking restrictions, and manage campground profiles to attract more visitors.
- 2. Booking Scheduling:** Attendees can search for campgrounds based on location, availability, and their budget. The system provides a streamlined booking process, allowing users to reserve campsites and select preferred dates according to their budget.
- 3. User Authentication and Authorization:** Secure user authentication ensures that only authorized individuals can access the system. Camp owners and attendees have personalized accounts with unique login credentials, allowing them to manage bookings, view past reservations, and update profile information.
- 4. Administrative Capabilities:** Camp owners have access to administrative tools for managing user accounts, reviewing booking requests, and generating reports. They have access to add properties according to their management and manage pricings.

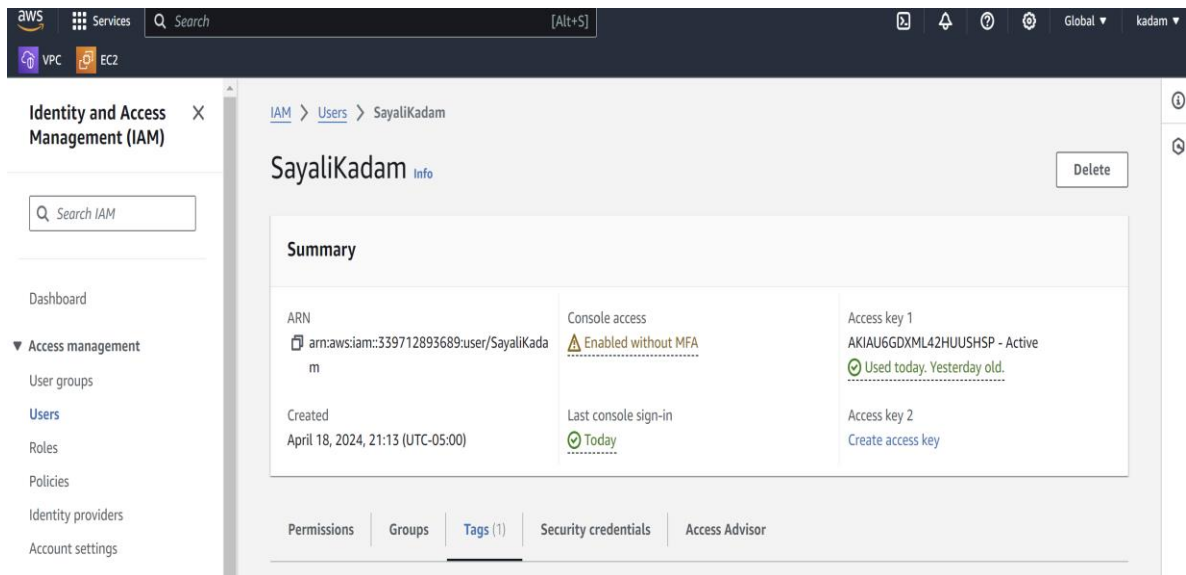
In summary, the Camp Scheduling System offers a comprehensive solution for camp owners and attendees to efficiently manage campgrounds, schedule bookings, and enhance the overall camping experience. Our project touches all the components of functional application, including the VPC, internet gateway, route table, subnets, security group, key pair, EC2 instance, S3 bucket, and RDS instance.

Python Boto3 Automation:

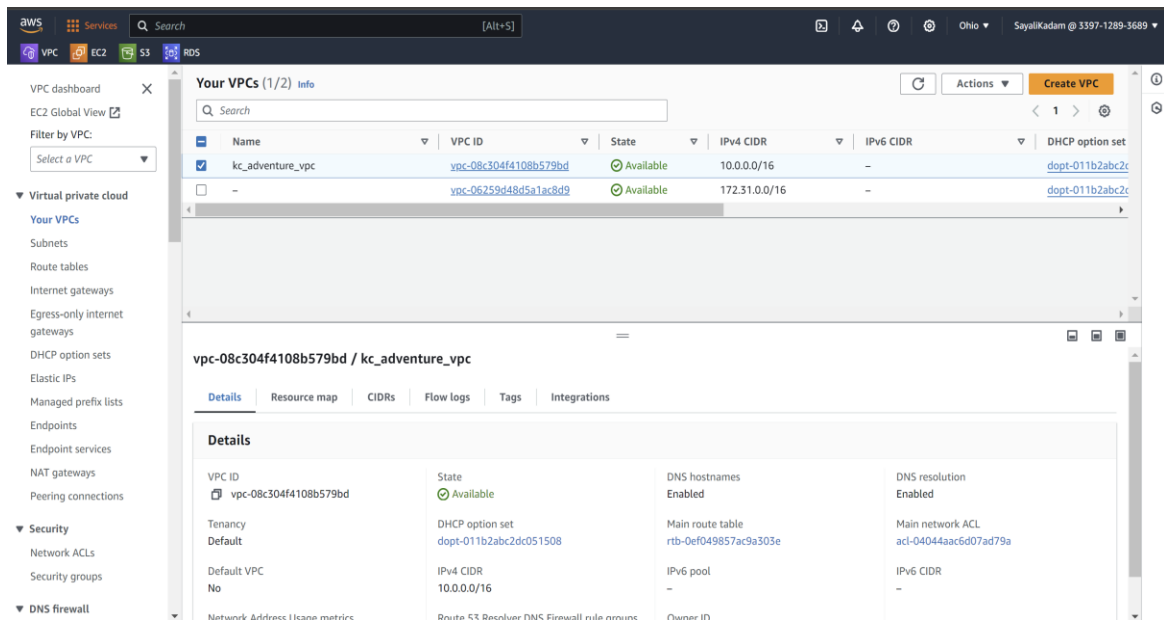
1. We have created a new user in IAM and have added policies such as AmazonRDSFullAccess, AmazonEC2FullAccess, AmazonS3FullAccess, AmazonVPCFullAccess, and IAMFullAccess.



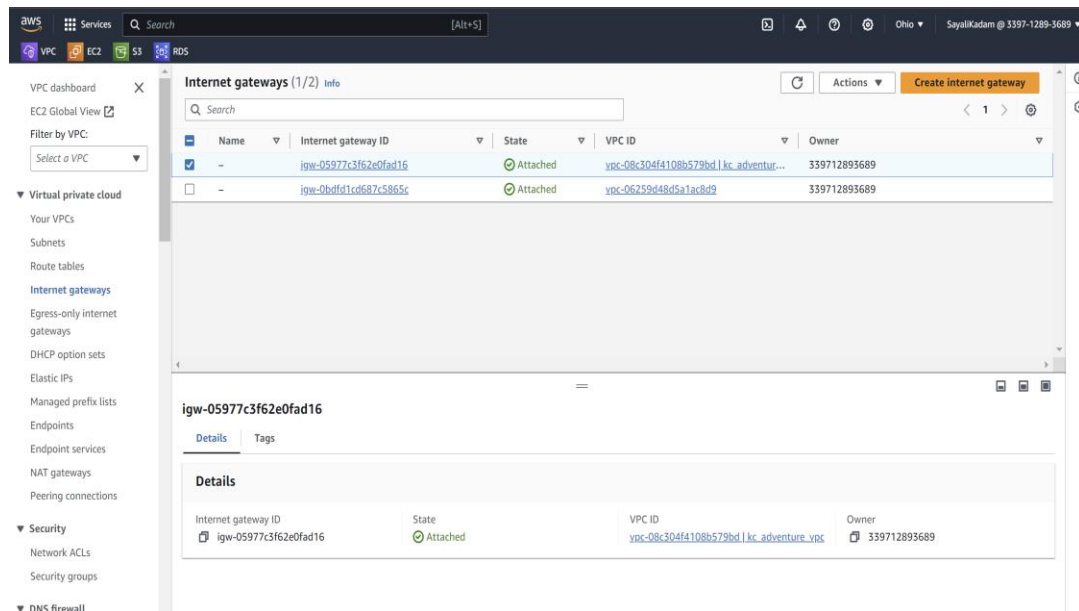
2. We've generated an access key within the security credentials section of the IAM user interface. These credentials will enable us to utilize Boto3 services within our local Python code.



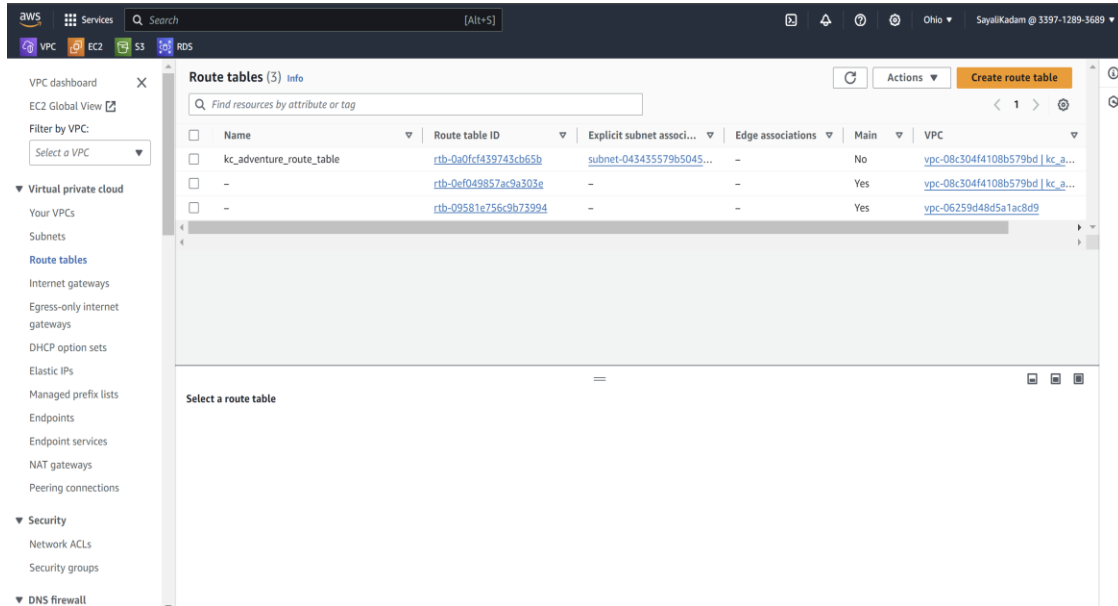
3. We have automated the process of creating VPC by writing code to create VPC as provided in zipped file (kc_adventure_script.py). We have used `create_vpc` function, setting its CIDR block to "10.0.0.0/16".



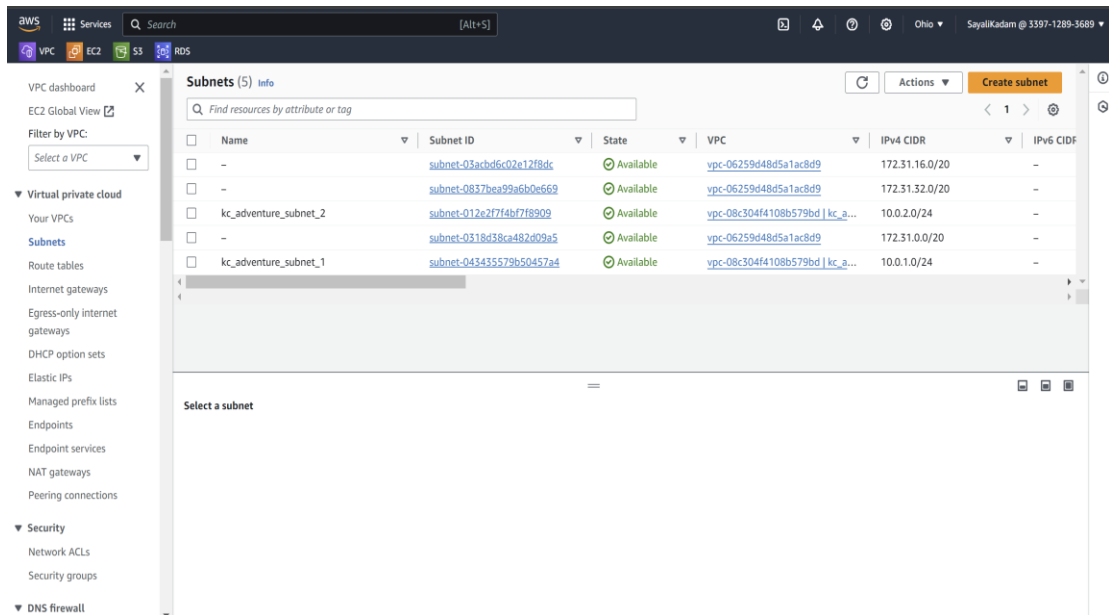
4. To create internet gateway for VPC we have used `create_internet_gateway` function and to attach it to VPC we have used `attach_internet_gateway` function.



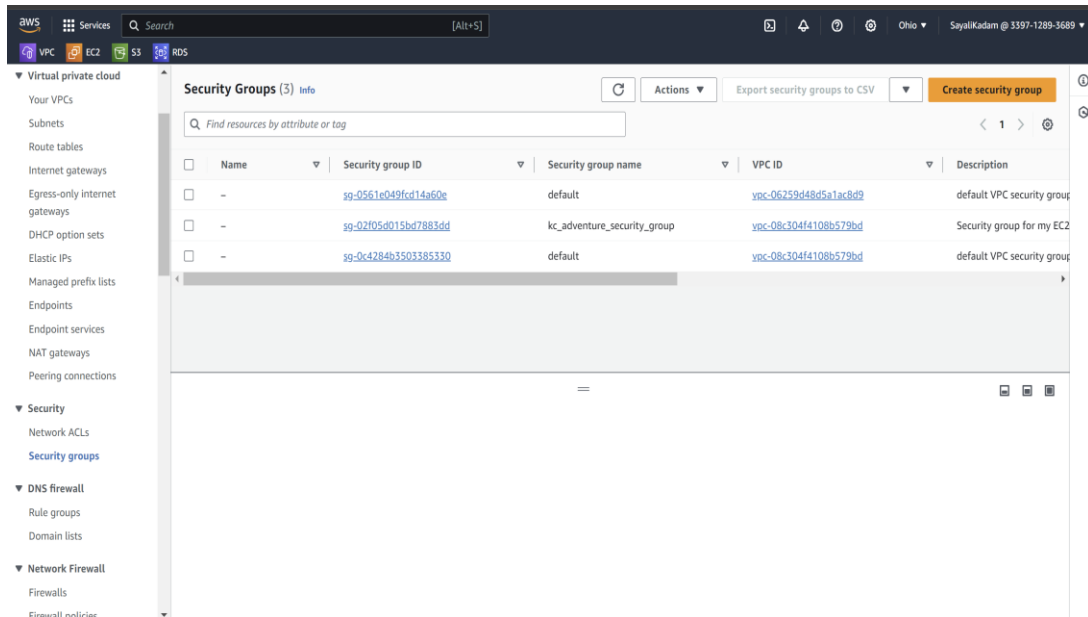
- For creating routing table for VPC we have used `create_route_table` function. This `create_route` function is used point at internet gateway by adding default route allowing instances in VPC to establish communication with internet.



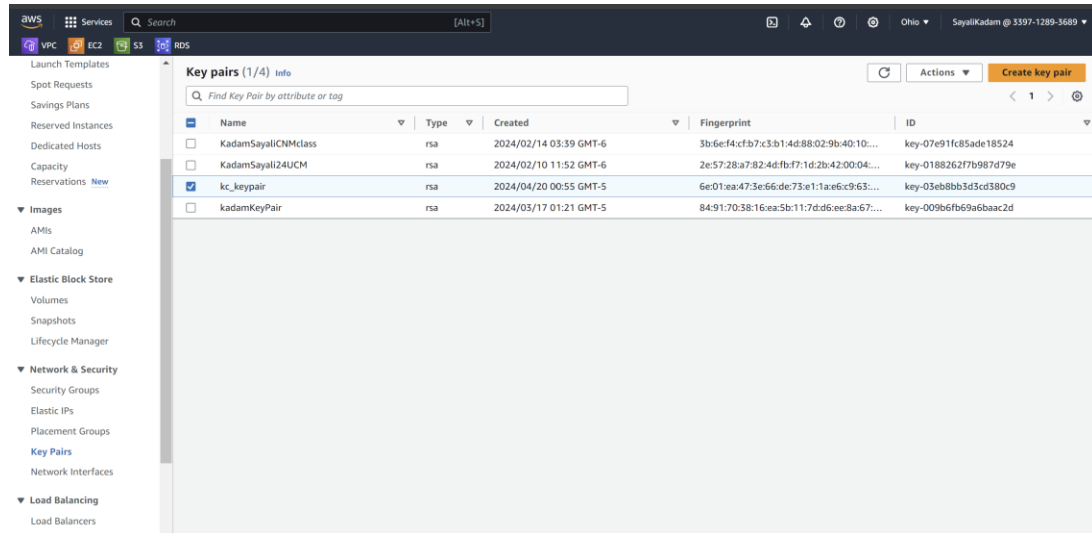
- The `create_subnet` function is creating 2 subnets, with CIDR blocks designated as "10.0.1.0/24" and "10.0.2.0/24" respectively. Using `associate_route_table` function, route tables are associated with these subnets.



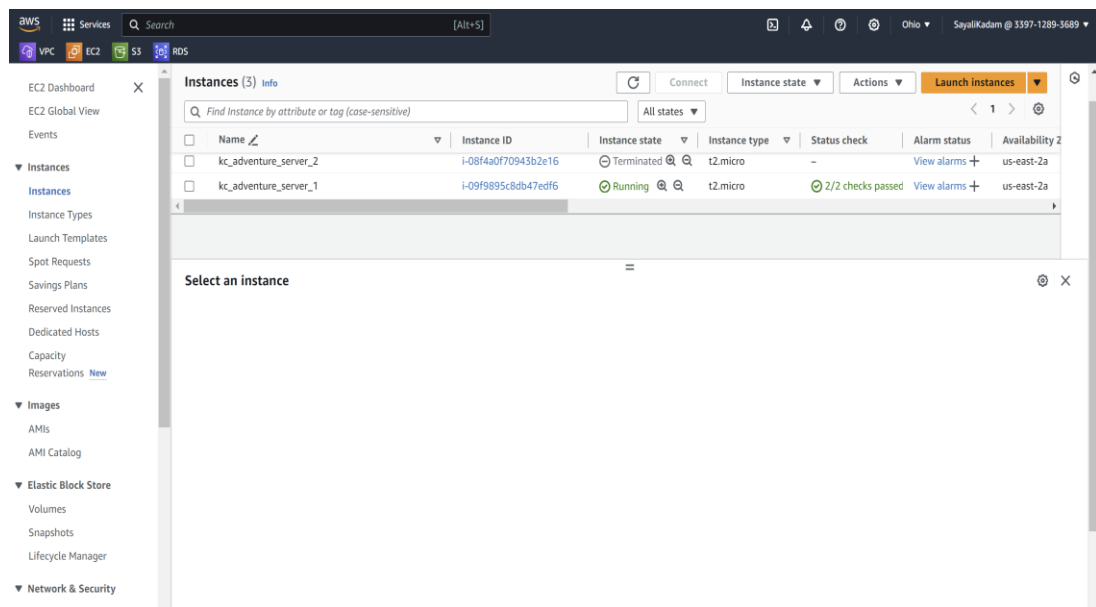
- Using `authorize_security_group_ingress` we have created security group allowing inbound traffic for SSH, HTTP, and HTTPS protocols.



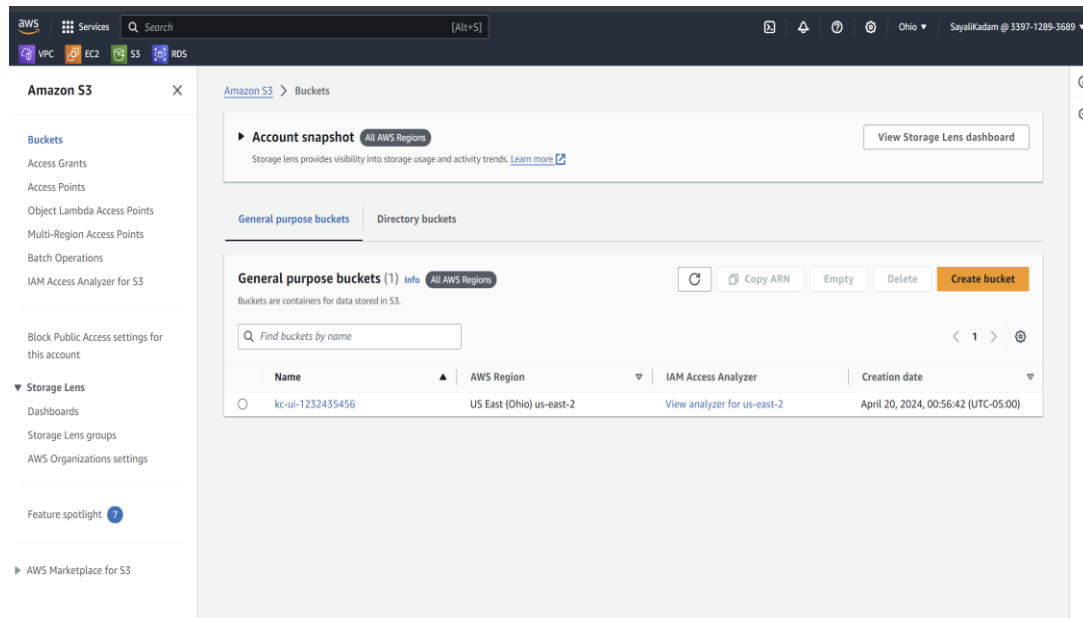
8. To authenticate access to EC2 instance we need key pair. For this a key_pair is assigned a name for key pair. Then through code we try to fetch the key pair by using `describe_key_pairs()` method. If key exist we use same key but if key doesn't match then code proceeds with new `create_key_pair()` method and stores it in a local file.



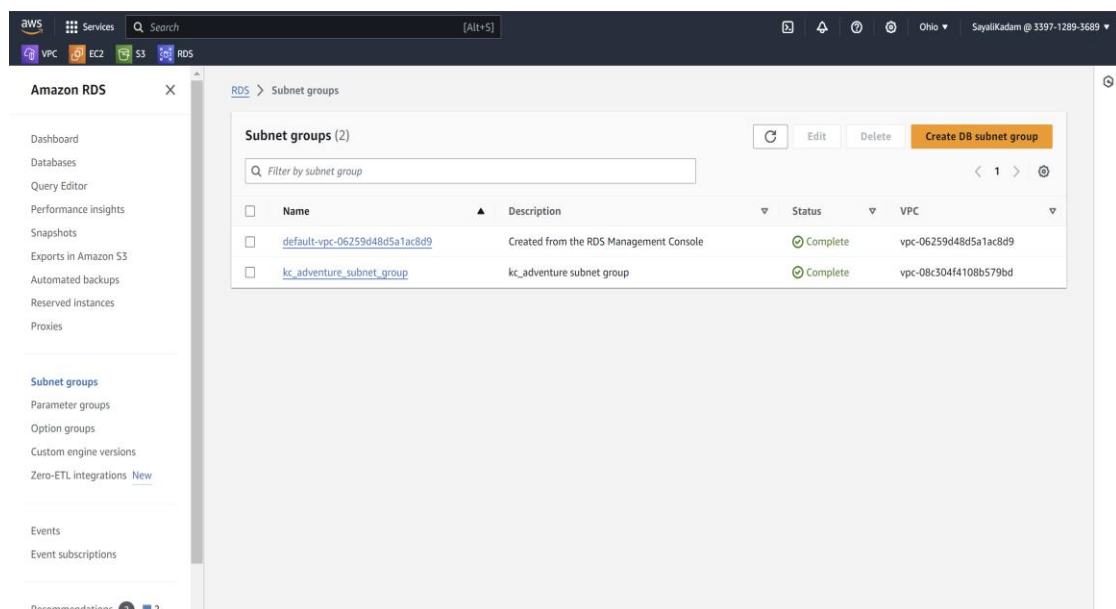
9. The instance is launched with specified AMI ID and it is associated with network interface within designated subnet and security group. The `server_name` variable is assigned a name for the EC2 instance. The code then verifies if an instance with the same name already exists by utilizing the `describe_instances()` method. If an instance is found, it retrieves the instance ID. Otherwise, it proceeds to create a new instance using the `run_instances()` method.



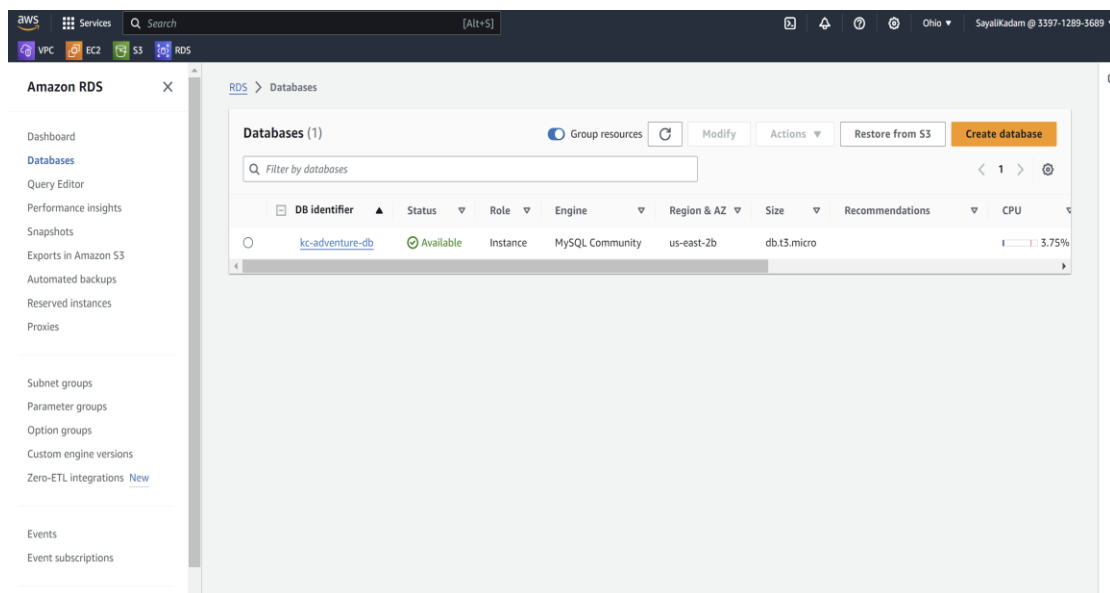
10. This code creates a S3 bucket and sets bucket policies to make object public. It also enable static website hosting for bucket by calling `put_bucket_website()`. The code first checks if the S3 bucket already exist and if it doesn't exist then the code creates S3 bucket using the specified bucket name. Then the code prints URL for static website hosted in S3 bucket.



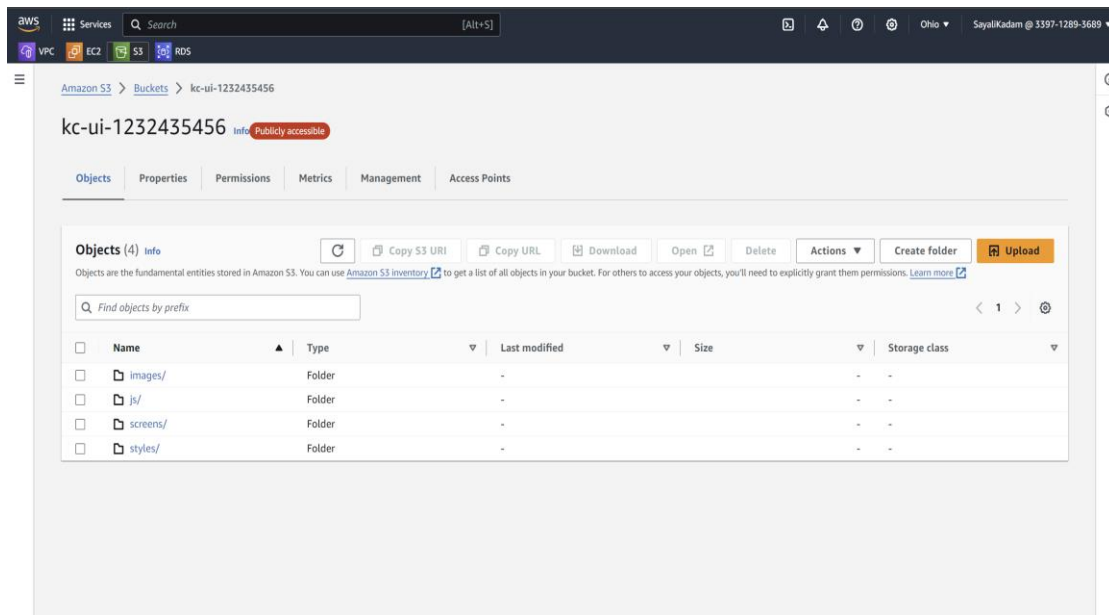
11. Using `create_db_subnet_group` method within the RDS client a new subnet group is created.



12. Now using `create_db_instance` method we create new RDS instance. This method encompasses various parameters configuring the instance, including the database engine (Engine), instance class (DBInstanceClass), and database name (DBName). The `VpcSecurityGroupIds` parameter designates the security group to employ, while the `DBSubnetGroupName` parameter specifies the subnet group. The `PubliclyAccessible` parameter is set to `True`, enabling access from the Internet.



13. The final phase of the kc adventure involves connecting to the EC2 instance, setting up `nginx`, and installing the required packages for the backend server. The EC2 instance's IP address is then utilized within the frontend code to establish connection with the server. This frontend code is subsequently deployed within the S3 bucket previously created.



Technologies used.

- JavaScript, CSS, HTML
- Python Boto3
- AWS Services IAM, VPC, EC2, S3 and RDS.

Functionalities

- Camp owners can log in and manage their Camping location availability, Camping activities, pricing, and other details.
- Customers can search for available camping location and schedule bookings based on their preferences.
- Customers can view location details, including pricing, availability, and activities.
- Camp owners(admin) can manage their bookings and approve or reject requests from customers.

KC_Adventure App

1. Signup Screen

In this Signup Screen, we have the fields like full name, username, and password. Once these fields are filled, we need to hit the signup button then those details are stored into the database only if the details are not stored in the database previously. otherwise, we will get a suggestion like “Already have an account? Please Login”.

Welcome to KC Adventure

Sign up

Sign up

[Already have an account? Login](#)



Code Snippet:

```
File Edit Selection View Go Run ... Search
js signup.js
C:\Users> PAVAN > AppData > Local > Temp > 273e4bcd-2733-457d-bd8e-d702026a7143_KC_Adventure[1].zip.143 > KC_Adventure > frontend > js > js signup.js > signupAPI > res
1 function signupAPI(name, username, password) {
2   const res = fetch(`${BASE_URL}register`, {
3     method: 'POST',
4     body: JSON.stringify({
5       "name": name,
6       "username": username,
7       "password": password
8     })
9   })
10  .then(async (response) => {
11    const res = await response.json();
12    const resStatus = {
13      status: response.status,
14      data: res
15    }
16    console.log(resStatus);
17    return resStatus;
18  })
19  .catch(error => console.error('Error:', error));
20
21  return res;
22
23 }
24
25 async function signup() {
26   let name = document.getElementById('name').value;
27   let username = document.getElementById('username').value;
28   let password = document.getElementById('password').value;
29   console.log(name, username, password);
30 }
```

```
JS signup.js x
C:\Users\PAVAN> AppData\Local\Temp\273e4bcd-2733-457d-bd8e-d702026a7143_KC_Adventure[1].zip.143 > KC_Adventure > frontend > js > JS signup.js > signupAPI > [0] res
1 function signupAPI(name, username, password) {
2   const res = fetch(`${BASE_URL}register`, {
18   })
19   .catch(error => console.error('Error:', error));
20
21   return res;
22
23 }
24
25 async function signup() {
26   let name = document.getElementById('name').value;
27   let username = document.getElementById('username').value;
28   let password = document.getElementById('password').value;
29   console.log(name, username, password);
30
31   blockUI();
32   if (name.length > 0 && username.length > 0 && password.length > 0) {
33
34     const res = await signupAPI(name, username, password);
35     if (res?.status === 200) {
36       showToast(res?.data?.response);
37     } else {
38       showToast(res?.data?.response);
39     }
40   } else {
41     showToast("Fill all the fields");
42   }
43   unblockUI();
44 }
```

2. Login Screen

When we enter the username and password and clicks the login button, then the app checks if any user exist with those credentials (username and password) or not. If exists, then the user will be logged in to the Camp sites application or else, we have the suggestion like “Don’t have an account? Please Signup”. In this screen, we also have the toggle button to log in as a user or an owner.



Welcome to KC Adventure

Login

Login

User ☐ Owner

[Don't have an account? Sign up](#)

Code Snippet

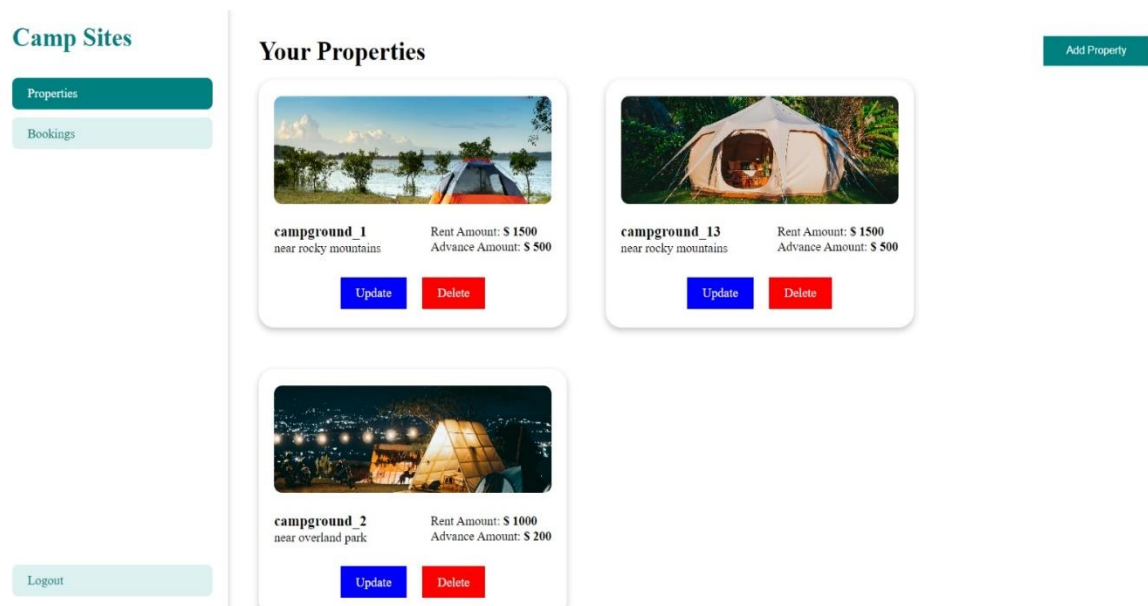
```
1 function loginAPI(username, password, userType) {
2   const res = fetch(`${BASE_URL}login/${userType}`, {
3     method: 'POST',
4     body: JSON.stringify({
5       "username": username,
6       "password": password
7     })
8   })
9   .then(async (response) => {
10     const res = await response.json();
11     const resStatus = {
12       status: response.status,
13       data: res
14     }
15     console.log(resStatus);
16     return resStatus;
17   })
18   .catch(error => console.error('Error:', error));
19
20 console.log(res);
21 return res;
22 }
23
24 function changeUserType(evt) {
25   console.log(evt.target.value)
26 }
27
28
29 async function login() {
```

```
27
28
29 async function login() {
30   let username = document.getElementById('username').value;
31   let password = document.getElementById('password').value;
32   let userType = document.getElementById('user-type').checked;
33   console.log(username, password, userType);
34   blockUI();
35   if (username.length > 0 && password.length > 0) {
36
37     const res = await loginAPI(username, password, userType ? "owner" : "user");
38     if (res?.status === 200) {
39       localStorage.setItem("username", username);
40       localStorage.setItem("user_id", res?.data?.id);
41       localStorage.setItem("isOwner", userType);
42       if (userType) {
43         window.location.href = "./admin-property.html";
44       } else {
45         window.location.href = "./home.html";
46       }
47     } else {
48       showToast(res?.data?.response);
49     }
50   } else {
51     showToast("Fill all the fields");
52   }
53   unblockUI();
54 }
55 }
```

3. Owner Home Screen

Once we log in as an Owner into the Camp sites app, the owner will have access to update the existing properties details in camp properties page and camp bookings page. In the camp properties page, all the properties of the owner will be shown. Besides that, the

owner has one more option like add new properties into the app or delete them. All these changes can be done anytime by the owner.

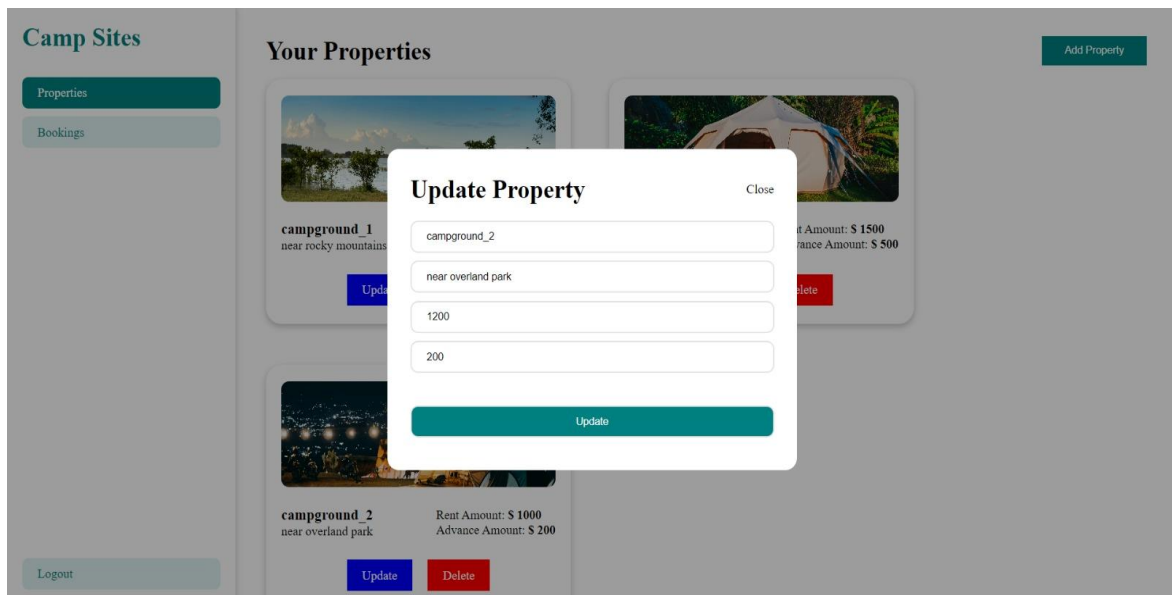


Code Snippet:

```
File Edit Selection View Go Run ... Search
JS admin-property.js JS bookings.js JS utils.js JS schedule.js JS admin-bookings.js JS home.js
C:\Users> PAVAN > AppData > Local > Temp > 1ccb90f8-fae9-4add-9303-c15b75c9388d_KC_Adventure[1].zip.88d > KC_Adventure > frontend > js > JS admin-property.js > ...
1 let user_id = localStorage.getItem("user_id");
2 let userType = localStorage.getItem("isOwner");
3
4 async function getProperties() {
5   isLoggedIn();
6   if (userType == 'false') {
7     window.location.href = "../home.html";
8   }
9   blockUI();
10  await fetch(`${BASE_URL}properties?owner_id=${user_id}`)
11    .then(async response => {
12      const res = await response.json();
13      properties = res.data;
14      getPropertyCards();
15    })
16    .catch(error => console.log('error', error));
17  unblockUI();
18 }
19
20 function getPropertyCards() {
21   let cards = "";
22   let main = document.getElementById("property-main");
23   const images = ["../images/property_1.jpg", "../images/property_2.jpg", "../images/property_3.jpg"];
24   for (let i = 0; i < properties.length; i++) {
25     cards += `<div class="rental-card">
26       
28       <div class="card-body">
29         `
30   }
31 }
```

4. Property Update Screen

This update screen page is one of the forward pages for the home screen page for owner. As we mentioned earlier in this update property page, the owner can update the existing camp properties details like property name, its location, rent amount, and advance amount.



Code Snippet:

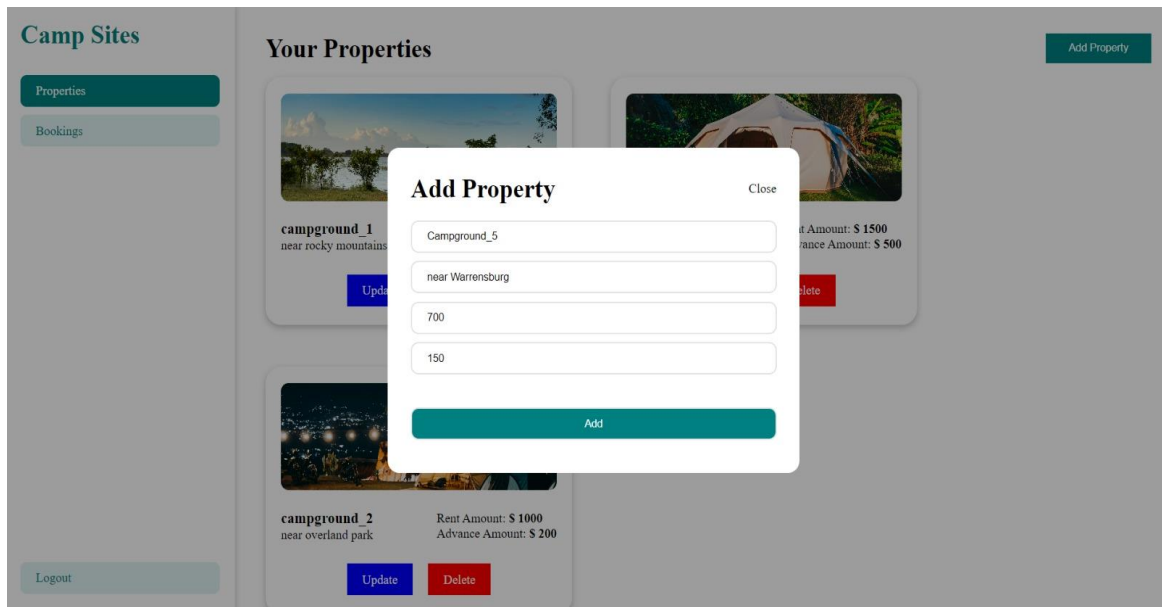
```
File Edit Selection View Go Run ... Search
JS admin-property.js JS bookings.js JS utils.js JS schedule.js JS admin-bookings.js JS home.js
C:\Users> PAVAN > AppData > Local > Temp > 1ccb90f8-fae9-4add-9303-c15b75c9388d_KC_Adventure[1].zip.88d > KC_Adventure > frontend > js > JS admin-property.js > ...

87 async function updateProperty() {
88
89   let name = document.getElementById("uname").value;
90   let place = document.getElementById("uplace").value;
91   let rent = document.getElementById("urent").value;
92   let advance = document.getElementById("uadvance").value;
93
94   blockUI();
95   var requestOptions = {
96     method: 'PUT',
97     body: JSON.stringify({
98       "name": name,
99       "location": place,
100      "rent_amount": rent,
101      "advance_amount": advance
102    }),
103    redirect: 'follow'
104  };
105
106  await fetch(`${BASE_URL}properties/${update_id}`, requestOptions)
107    .then(async response => {
108      const res = await response.json();
109      showToast(res.response);
110      if (response.status == 200) {
111        closeModal();
112        getProperties();
113      }
114    })
115    .catch(error => console.log('error', error));

```

5. Property Add Screen

This add property page is also one of the forward pages for the home screen page for owner. The owner can add any camp property with details like property name, location, rent amount, and advance amount by clicking the update button.



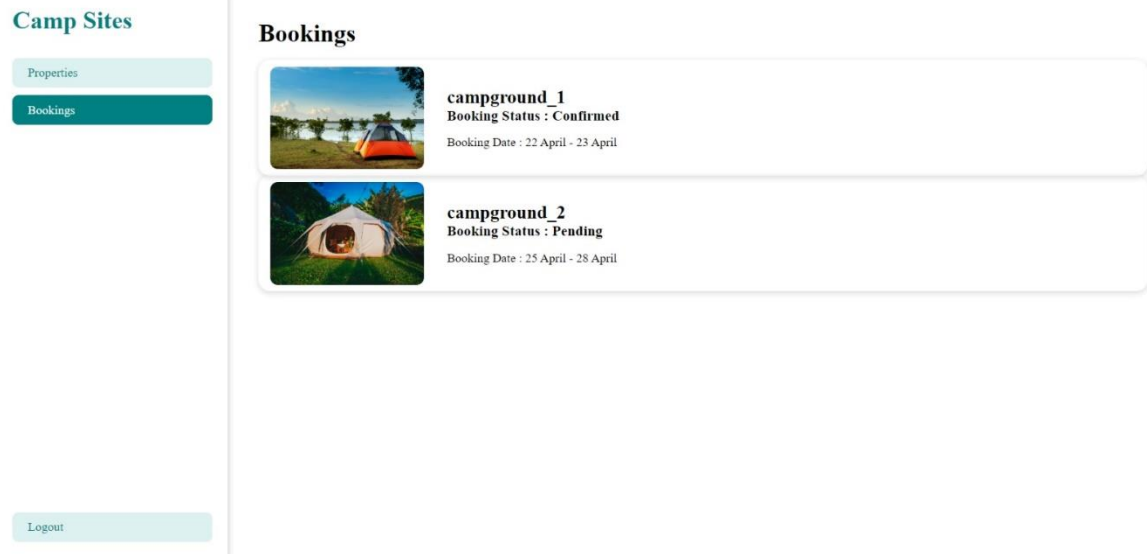
Code Snippet:

```
File Edit Selection View Go Run ... Search
C: > Users > PAVAN > AppData > Local > Temp > 1ccb90f8-fae9-4add-9303-c15b75c9388d_KC_Adventure[1].zip.88d > KC_Adventure > frontend > js > JS admin-property.js > ...

54 async function addProperty() {
55   let name = document.getElementById("name").value;
56   let place = document.getElementById("place").value;
57   let rent = document.getElementById("rent").value;
58   let advance = document.getElementById("advance").value;
59   // let availability = document.getElementById("available").value;
60   blockUI();
61   var requestOptions = {
62     method: 'POST',
63     body: JSON.stringify({
64       "name": name,
65       "location": place,
66       "rent_amount": rent,
67       "advance_amount": advance
68     }),
69   };
70
71   await fetch(`${BASE_URL}properties/${user_id}`, requestOptions)
72     .then(async response => {
73       const res = await response.json();
74       console.log(res.response);
75       showToast(res.response);
76       if (response.status == 200) {
77         closeModal();
78         getProperties();
79       }
80     })
81     .catch(error => { console.log('error', error); showToast(res.data.response); });
82 }
```


6. Bookings Screen

In this camp bookings page, the owner can check for all the booking requests from the customers for their camp properties. Owner can both the confirm requests and reject.

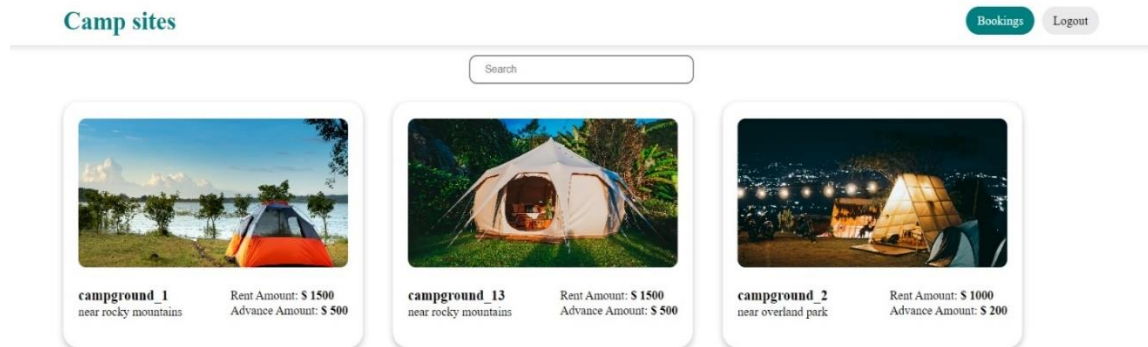


Code Snippet:

```
File Edit Selection View Go Run ... Search
JS admin-property.js JS bookings.js JS utils.js JS schedule.js JS admin-bookings.js JS home.js
C:\Users> PAVAN > AppData > Local > Temp > f5c40b22-54a8-43d7-bc39-fc12043a0c8e_KC_Adventure[1].zip.c8e > KC_Adventure > frontend > js > JS admin-bookings.js > ...
26 function renderScheduleCards() {
27   </div>
28 }
29
30 main.innerHTML = cards;
31 }
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60 async function updateBookings(id) {
61   blockUI();
62   var requestOptions = {
63     method: 'PUT',
64     body: JSON.stringify({
65       "id": id,
66       "is_approved": true
67     })
68   };
69
70   await fetch(`${BASE_URL}property/book`, requestOptions)
71     .then(async response => {
72       const res = await response.json();
73       showToast(res.response);
74       if (response.status == 200) {
75         getSchedulingsById();
76       }
77     })
78     .catch(error => console.log('error', error));
79   unblockUI();
80 }
```

7. User Home Screen

In this home page like when we login as a user, A user can check all the camp properties listed by all the owners and can search for properties based on name, location, and cost.



Code Snippet:

```
JS homejs x
C: > Users > PAVAN > OneDrive > Desktop > JS homejs > getProperties
1 let properties = [];
2
3 async function getProperties() {
4   isLoggedIn();
5   blockUI();
6   await fetch(`${BASE_URL}properties`)
7     .then(async response => {
8       const res = await response.json();
9       console.log(res);
10      properties = res.data;
11      getPropertyCards();
12    });
13   .catch(error => console.log('error', error));
14   unblockUI();
15 }
16
17 function getPropertyCards() {
18   let cards = "";
19   let main = document.getElementById("home-main");
20   const images = ["../images/property_1.jpg", "../images/property_2.jpg", "../images/property_3.jpg"];
21   for (let i = 0; i < properties.length; i++) {
22     cards += `<div class="rental-card" onclick="goToSchedule(${properties[i].id}, '${properties[i].name}', '${properties[i].location}',
23               ${properties[i].rental_amount}, ${properties[i].advance_amount}, ${properties[i].availability})">
24       <img
25         src= "${images[i]}" />
26       <div class="card-body">
27         <div>
28           <h3>${properties[i].name}</h3>
29         </div>
30       </div>
31     `;
32   }
33   main.innerHTML = cards;
34 }
```

```
File Edit Selection View Go Run ... Search
JS home.js x
C:\Users\PAVAN\OneDrive\Desktop\JS home.js > getProperties
43
44 function searchProperties() {
45   let pname = document.getElementById('property-input').value;
46   let main = document.getElementById('home-main');
47
48   console.log(pname);
49   let cards = "";
50   const images = ["../images/property_1.jpg", "../images/property_2.jpg", "../images/property_3.jpg"];
51   for (let i = 0; i < properties.length; i++) {
52     if (properties[i].name.toLowerCase().includes(pname) || properties[i].location.toLowerCase().includes(pname)) {
53       cards += `<div class="rental-card" onclick="goToSchedule(${properties[i].id}, '${properties[i].name}', '${properties[i].location}', $
54                 {properties[i].rental_amount}, ${properties[i].advance_amount}, ${properties[i].availability})">
55                 <img
56                   src= "${images[i]}" />
57                 <div class="card-body">
58                   <div>
59                     <h3>${properties[i].name}</h3>
60                     <p>${properties[i].location}</p>
61                   </div>
62                   <div>
63                     <p>Rent Amount: <b>$ ${properties[i].rental_amount}</b></p>
64                     <p>Advance Amount: <b>$ ${properties[i].advance_amount}</b></p>
65                   </div>
66                 </div>
67               </div>`
68     }
69   }
70 }
```


8. Booking Screen

In this bookings page, when the user satisfies with the camp properties details like location, rent amount, and advance amount then the user can make booking request based on their convenience.

Camp Sites

BookingsLogout

Book now



campground_1

near rocky mountains

Available

Rent Amount : \$500

Advance Amount : \$500

04/04/2024 09:06 PM

Book now

Code Snippet:

```
File Edit Selection View Go Run ... Search
JS admin-property.js JS bookings.js JS schedule.js X JS admin-bookings.js JS home.js
C:\Users> PAVAN > AppData > Local > Temp > 0256d9d0-c452-4c8a-b07e-664aa22264c3_KC_Adventure[1].zip.4c3 > KC_Adventure > frontend > js > JS schedule.js > ...
1 let id = localStorage.getItem('prop_id');
2 let prop_name = localStorage.getItem('prop_name');
3 let place = localStorage.getItem('prop_place');
4 let rent = localStorage.getItem('prop_rent');
5 let advance = localStorage.getItem('prop_advance');
6 let availability = localStorage.getItem('prop_availability');
7
8
9 document.getElementById('name').innerHTML = prop_name;
10 document.getElementById('place').innerHTML = place;
11 document.getElementById('available').innerHTML = availability == 1 ? "Available" : "Not Available";
12 document.getElementById('rent').innerHTML = 'Rent Amount : $ ${rent}';
13 document.getElementById('advance').innerHTML = 'Advance Amount : $ ${advance}';
14
15
16
17 async function schedule() {
18   isLoggedIn();
19   let dateTime = document.getElementById('date-input').value;
20   if (dateTime.length > 0) {
21
22     const date = new Date(dateTime);
23
24     const formattedDate = date.toISOString().slice(0, 19).replace('T', ' ');
25     console.log(dateTime, formattedDate);
26
27     blockUI();
28     let user_id = localStorage.getItem('user_id');
29     var raw = JSON.stringify({
30       // ...
31     });
32   }
33 }
```

9. Booking History Screen

In this booking history page, An user can check for his/her booking status and check for all their booking history from the past bookings to the future upcoming bookings.



Code Snippet:

```
File Edit Selection View Go Run ... Search
JS admin-property.js JS bookings.js X JS utils.js JS schedule.js JS admin-bookings.js JS home.js
C:\Users> PAVAN > AppData > Local > Temp > d08d5a8b-d4e3-41c1-a5d8-0c7c2050eaf0_KC_Adventure[1].zip.af0 > KC_Adventure > frontend > js > JS bookings.js > ...

1 let schedulings = [];
2
3 async function getSchedulingsById() {
4   isLoggedInOut();
5   let userType = localStorage.getItem('isOwner') == 'true' ? 'owner' : 'user';
6   let user_id = localStorage.getItem("user_id");
7   blockUI();
8   await fetch(`${BASE_URL}property/book/${userType}/${user_id}`)
9     .then(async response => {
10       const res = await response.json();
11       if (response.status === 200) {
12         console.log(res.data);
13         schedulings = res.data;
14         renderScheduleCards()
15       }
16     })
17   .catch(error => console.log('error', error));
18   unblockUI();
19 }
20
21
22 function renderScheduleCards() {
23   let cards = "";
24   let main = document.getElementById("schedule-main");
25   const images = ["../images/property_1.jpg", "../images/property_2.jpg"];
26   for (let i = 0; i < schedulings.length; i++) {
27     cards += `<div class="booking-card">
28       <div style="display: flex; align-items: center;">
29         <img`
```