Term Project Progress Report

Course Code: CNG 495

Semester-Year: Fall - 2023

NCC Campus Event Manager

Team Members:
Member 1: Berke Diler (2401503)
Member 2: Keremcan Adanur (2452902)
Member 3: Gökberk Lük (2453405)

1 Milestones Achieved

1.1 Week 5: 30 October - 3 November

1.1.1 Berke Diler

Did research on how to use AWS Cognito, decided how the user interface is going to look like, set up AWS Amplify Studio. Researched the technologies to be used and found tutorials on how to construct the user interface.

1.1.2 Keremcan Adanur

Did research on which web technology to use, comparing their benefits, drawbacks. Comapring different languages to be used in the project and came to the conclusion to use React with the team. Continued the research on the services provided by AWS, mainly, Amplify, S3 and DynamoDB.

1.1.3 Gökberk Lük

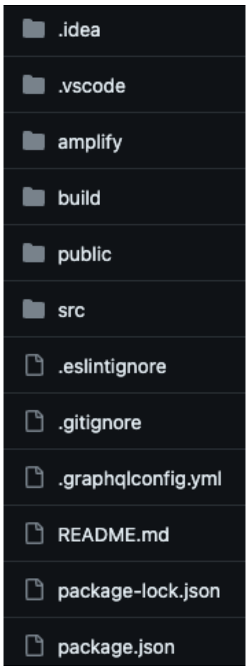
Did research on how to use AWS Amplify Studio, and how to user interface implemented on AWS Amplify Studio. Some documantation was reviewed, and the AWS systems that to be used such as Cognito were learned.

1.2 Week 6: 6 November - 10 November

1.2.1 Berke Diler

Set up the development environment (Node.js, NPM, AWS Amplify CLI, WebStorm IDE), created 'amplify' directory can be seen in the GitHub repository. Followed AWS Cognito[1] tutorials to implement the authentication system in the scope of the METU NCC Campus Event Manager and created the Admins user pool.

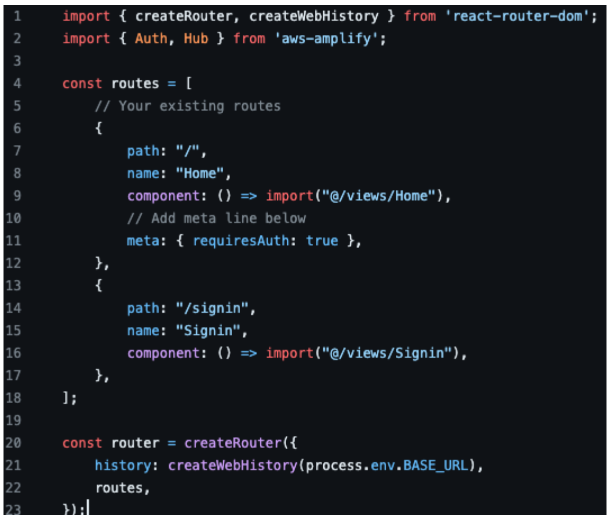
Created the project using JetBrains WebStorm and followed the initial file structure which can be seen on Figure 1.



```
.idea
.vscode
amplify
build
public
src
.eslintignore
.gitignore
.graphqlconfig.yml
README.md
package-lock.json
package.json
```

Figure 1: File Structure

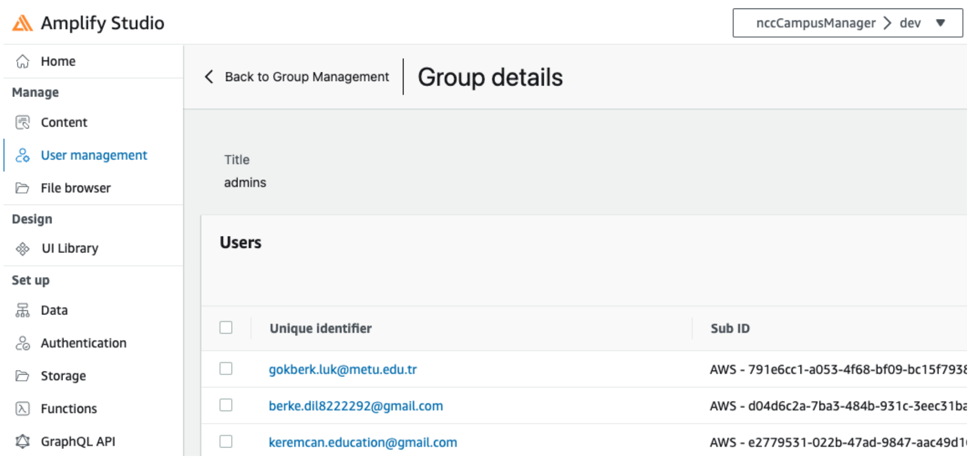
Upon creating the project and the file structure, routes of the application were created using ’react-router-dom’ as can be seen in Figure 2 and src/router/index.js inside the repository. And HTML views were created for sign-in and home pages for testing purposes which are under src/views.



```
import { createRouter, createWebHistory } from 'react-router-dom';
import { Auth, Hub } from 'aws-amplify';

const routes = [
    // Your existing routes
    {
        path: "/",
        name: "Home",
        component: () => import("@/views/Home"),
        // Add meta line below
        meta: { requiresAuth: true },
    },
    {
        path: "/signin",
        name: "Signin",
        component: () => import("@/views/Signin"),
    },
];

const router = createRouter({
    history: createWebHistory(process.env.BASE_URL),
    routes,
});
```

Figure 2: Routes

AWS Amplify was installed using **npm install -g @aws-amplify/cli**. An IAM user was created (/dev). An access key was retrieved using the IAM users panel, and the amplify project was created using the command **amplify-init**. And the AWS Cognito was deployed using the command **amplify add auth**. Lastly, user pool for admins was created on Amplify Studio and can be seen on Figure 3.


Amplify Studio

nccCampusManager > dev

Home

Manage

Content

User management

File browser

Design

UI Library

Set up

Data

Authentication

Storage

Functions

GraphQL API

Back to Group Management

Group details

Title

admins

Users

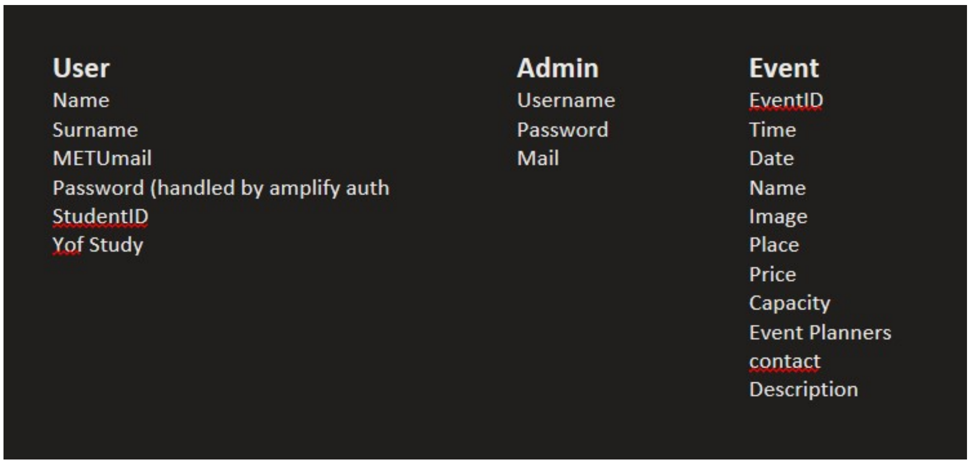
	Unique identifier	Sub ID
☐	gokberk.luk@metu.edu.tr	AWS - 791e6cc1-a053-4f68-bf09-bc15f793
☐	berke.dil8222292@gmail.com	AWS - d04d6c2a-7ba3-484b-931c-3eec31b
☐	keremcan.education@gmail.com	AWS - e2779531-022b-47ad-9847-aac49d1

Figure 3: Admins User Pool

1.2.2 Keremcan Adanur

This week we decided on the web framework and started to work on it. This week was mainly spent on getting familiar with Amplify CLI and Web Console, working on the structure of our database, and learning about the technologies that were decided. Research was mainly on React and GraphQL.

Our structure of the database can be seen in Figure 4. Thanks to the services of AWS we do not need to store any passwords in our database and worry about their security. So some of the attributes of the entities are not stored in our database and are instead handled by the services provided by AWS.


User
Name
Surname
METUmail
Password (handled by amplify auth
StudentID
Yof Study

Admin
Username
Password
Mail

Event
EventID
Time
Date
Name
Image
Place
Price
Capacity
Event Planners
contact
Description

Figure 4: Entities and Attributes

1.2.3 Gökberk Lük

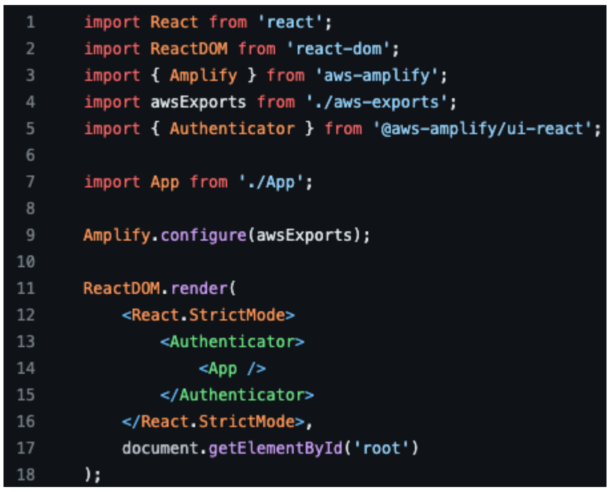
This week was mostly spent getting used to and testing the decided framework and AWS Services. First of all, AWS Amplify CLI was installed and a branch was created on Github for testing. After Amplify CLI and Amplify studio familiarization process, GraphQL and React that will be using for backend began to learn. Previous progress was reviewed on the Web by running with the npm start command on WebStorm IDE terminal on localhost.

1.3 Week 7: 13 November - 17 November

1.3.1 Berke Diler

Implemented the initial user login interface by using AWS Amplify React.js UI ‘ library and had different user pools seeing different options when logged in to the application.

Using the AWS Amplify UI library, the login and sign-up page was created which works in parallel with AWS Cognito, when the user registers, he/she recieves a verification e-mail and after verification the user is able to login. If the user is an admin, they are recognized by the application. The code can be seen on Figure 5 and the user interface can be seen on Figure 6. And how the user pool is checked (admin or normal user) can be seen on Figure 7.



```
import React from 'react';
import ReactDOM from 'react-dom';
import { Amplify } from 'aws-amplify';
import awsExports from './aws-exports';
import { Authenticator } from '@aws-amplify/ui-react';

import App from './App';

Amplify.configure(awsExports);

ReactDOM.render(
    <React.StrictMode>
        <Authenticator>
            <App />
        </Authenticator>
    </React.StrictMode>,
    document.getElementById('root')
);
```

Figure 5: Authentication using AWS Amplify UI Library | **src/index.js**

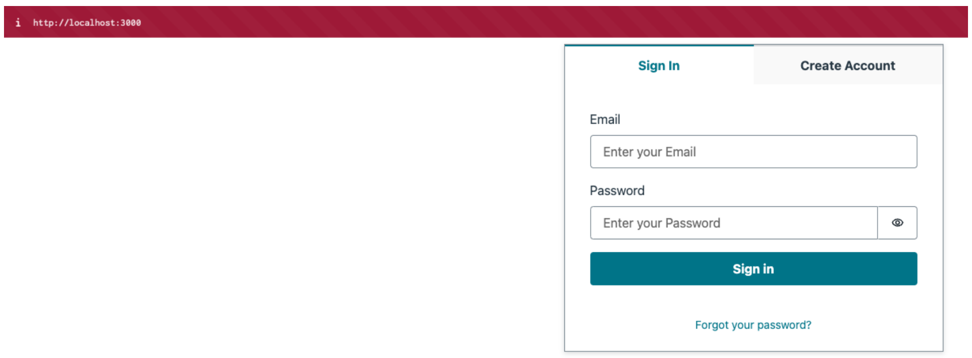
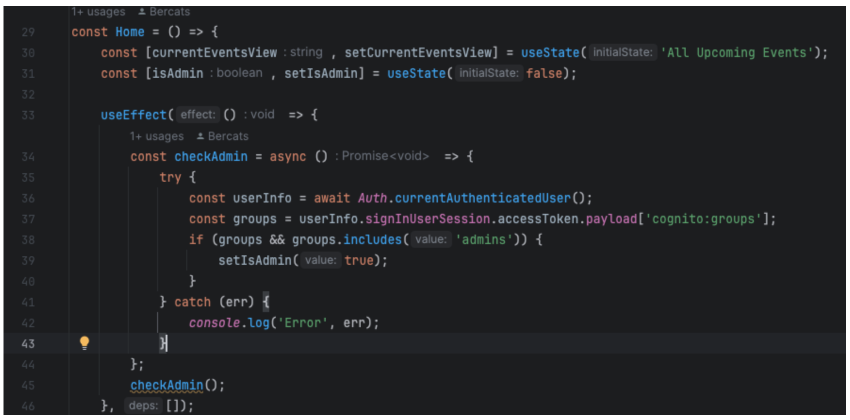


Figure 6: Login Interface Running on localhost

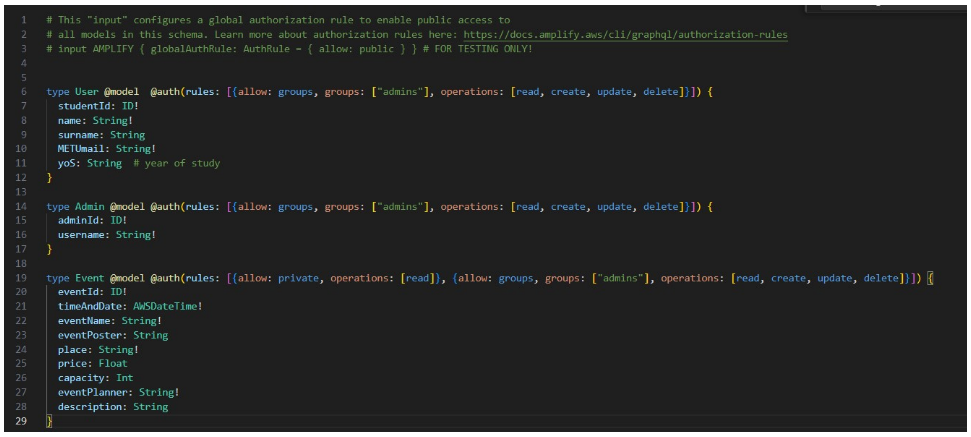


```
const Home = () => {
    const [currentEventsView, setCurrentEventsView] = useState('All Upcoming Events');
    const [isAdmin, setIsAdmin] = useState(false);

    useEffect(() => {
        const checkAdmin = async () => {
            try {
                const userInfo = await Auth.currentAuthenticatedUser();
                const groups = userInfo.signInUserSession.accessToken.payload['cognito:groups'];
                if (groups && groups.includes('admins')) {
                    setIsAdmin(true);
                }
            } catch (err) {
                console.log('Error', err);
            }
        };
        checkAdmin();
    }, []);
```

Figure 7: User Pool Checking | **src/views/home.js**

1.3.2 Keremcan Adanur

This week I worked on the implementation of the database. To have your databases in AWS Amplify you need to provide them as a GraphQL schema. I implemented the tables that I have created in the previous weeks in the correct syntax required by GraphQL as can be seen in Figure 8.



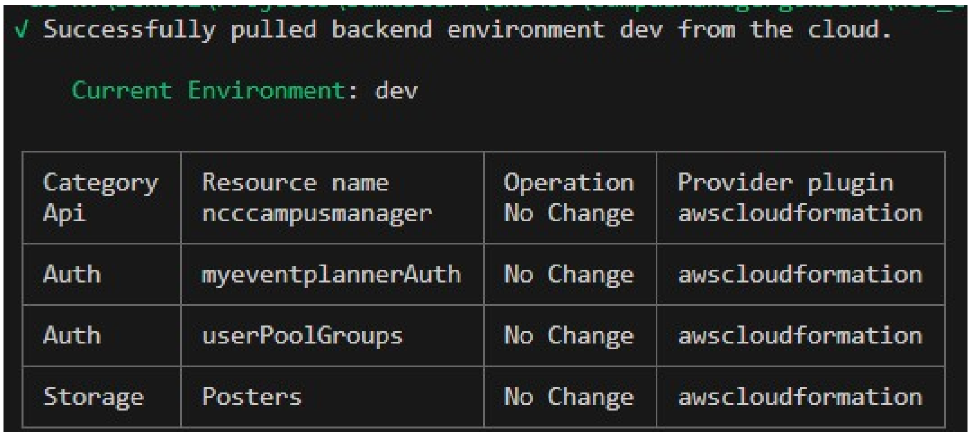
```graphql
# This "input" configures a global authorization rule to enable public access to
# all models in this schema. Learn more about authorization rules here: https://docs.amplify.aws/cli/graphql/authorization-rules
# input AMPLIFY { globalAuthRule: AuthRule = { allow: public } } # FOR TESTING ONLY!


type User @model  @auth(rules: [{allow: groups, groups: ["admins"], operations: [read, create, update, delete]}]) {
  studentId: ID!
  name: String!
  surname: String
  METUmail: String!
  yoS: String  # year of study
}

type Admin @model @auth(rules: [{allow: groups, groups: ["admins"], operations: [read, create, update, delete]}]) {
  adminId: ID!
  username: String!
}

type Event @model @auth(rules: [{allow: private, operations: [read]}, {allow: groups, groups: ["admins"], operations: [read, create, update, delete]}]) {
  eventId: ID!
  timeAndDate: AWSDateTime!
  eventName: String!
  eventPoster: String
  place: String!
  price: Float
  capacity: Int
  eventPlanner: String!
  description: String
}
```

Figure 8: GraphQL Schema

To be able to use this structure and edit backend of your application like I did int Figure 8, after you have cloned the repository, while connecting to amplify you need to select specifically that you want to work on the backend. To add graphQL support to your app you need to use the **amplify add api** command.

Figure 8 also specifies the permissions given on each table with the auth tag. Where we need to define which of our users can read, write and delete the data on a given table.

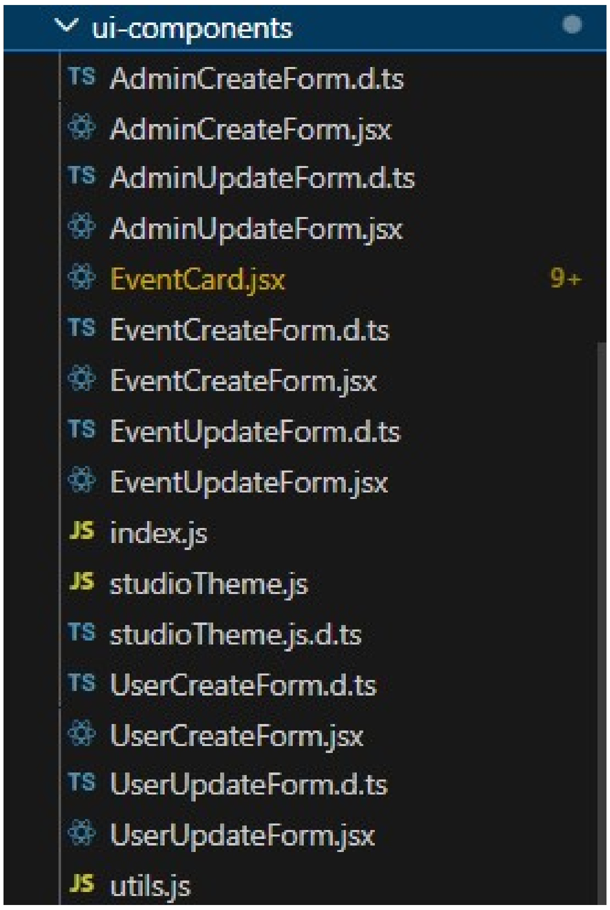
After updating and pushing changes to amplify using **amplify api update** and **amplify push –y**. We can check the status of our backend using **amplify status**, result of which can be seen in Figure 9.



```
√ Successfully pulled backend environment dev from the cloud.

    Current Environment: dev

┌──────────┬───────────────────┬───────────┬───────────────────┐
│ Category │ Resource name     │ Operation │ Provider plugin   │
│ Api      │ ncccampusmanager  │ No Change │ awscloudformation │
├──────────┼───────────────────┼───────────┼───────────────────┤
│ Auth     │ myeventplannerAuth│ No Change │ awscloudformation │
├──────────┼───────────────────┼───────────┼───────────────────┤
│ Auth     │ userPoolGroups    │ No Change │ awscloudformation │
├──────────┼───────────────────┼───────────┼───────────────────┤
│ Storage  │ Posters           │ No Change │ awscloudformation │
└──────────┴───────────────────┴───────────┴───────────────────┘
```

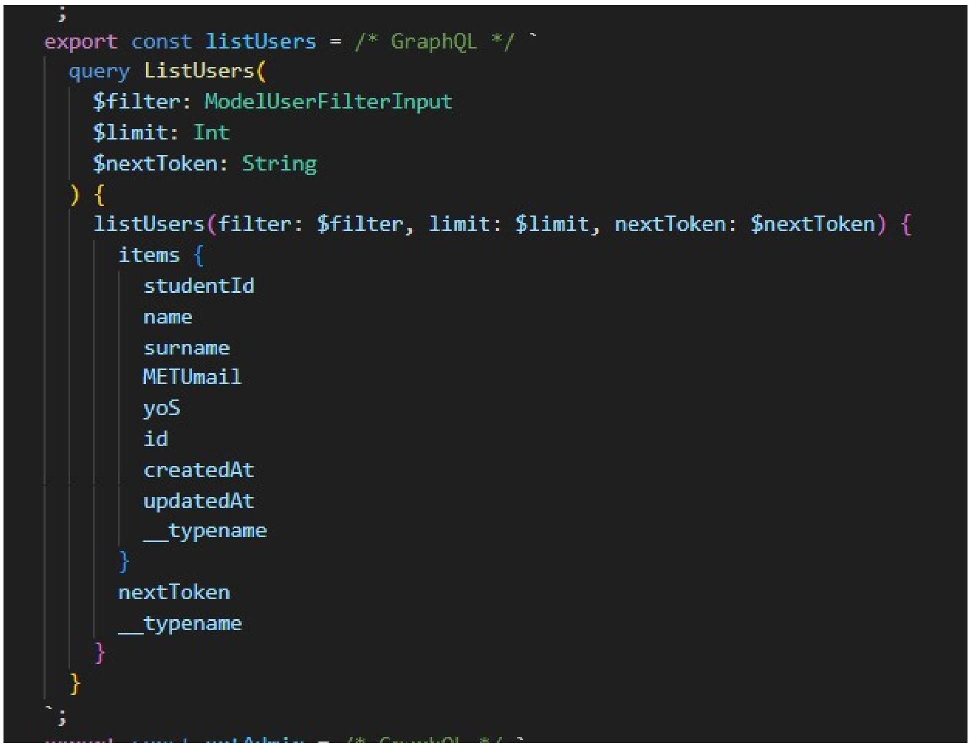
Figure 9: Amplify Status

In the end Amplify and GraphQL automatically creates the CRUD operations to be used on these tables along with some other functionalities such as queries which can be directly referenced and used in your program.



```
ui-components
AdminCreateForm.d.ts
AdminCreateForm.jsx
AdminUpdateForm.d.ts
AdminUpdateForm.jsx
EventCard.jsx                9+
EventCreateForm.d.ts
EventCreateForm.jsx
EventUpdateForm.d.ts
EventUpdateForm.jsx
index.js
studioTheme.js
studioTheme.js.d.ts
UserCreateForm.d.ts
UserCreateForm.jsx
UserUpdateForm.d.ts
UserUpdateForm.jsx
utils.js
```

Figure 10: UI Components



```
;
export const listUsers = /* GraphQL */ `
  query ListUsers(
    $filter: ModelUserFilterInput
    $limit: Int
    $nextToken: String
  ) {
    listUsers(filter: $filter, limit: $limit, nextToken: $nextToken) {
      items {
        studentId
        name
        surname
        METUmail
        yoS
        id
        createdAt
        updatedAt
        __typename
      }
      nextToken
      __typename
    }
  }
`;
```

Figure 11: ListUsers function created by graphQL

How these forms will look can be checked out through Amplify Console.

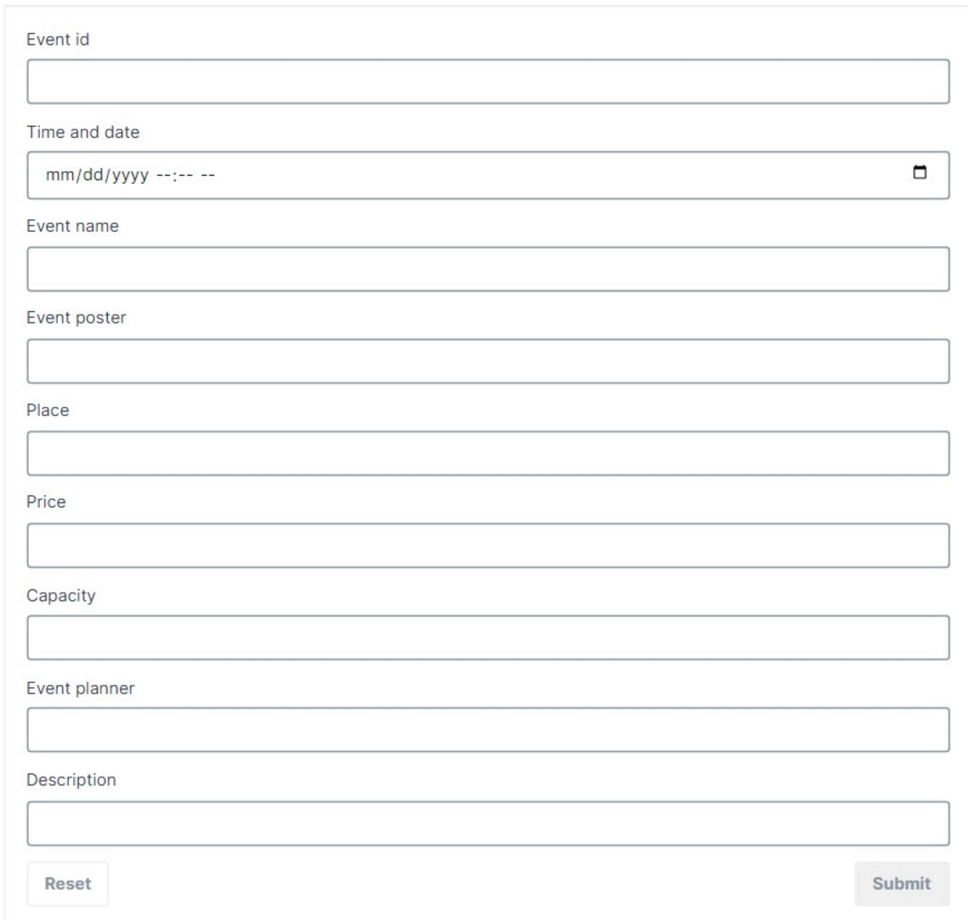
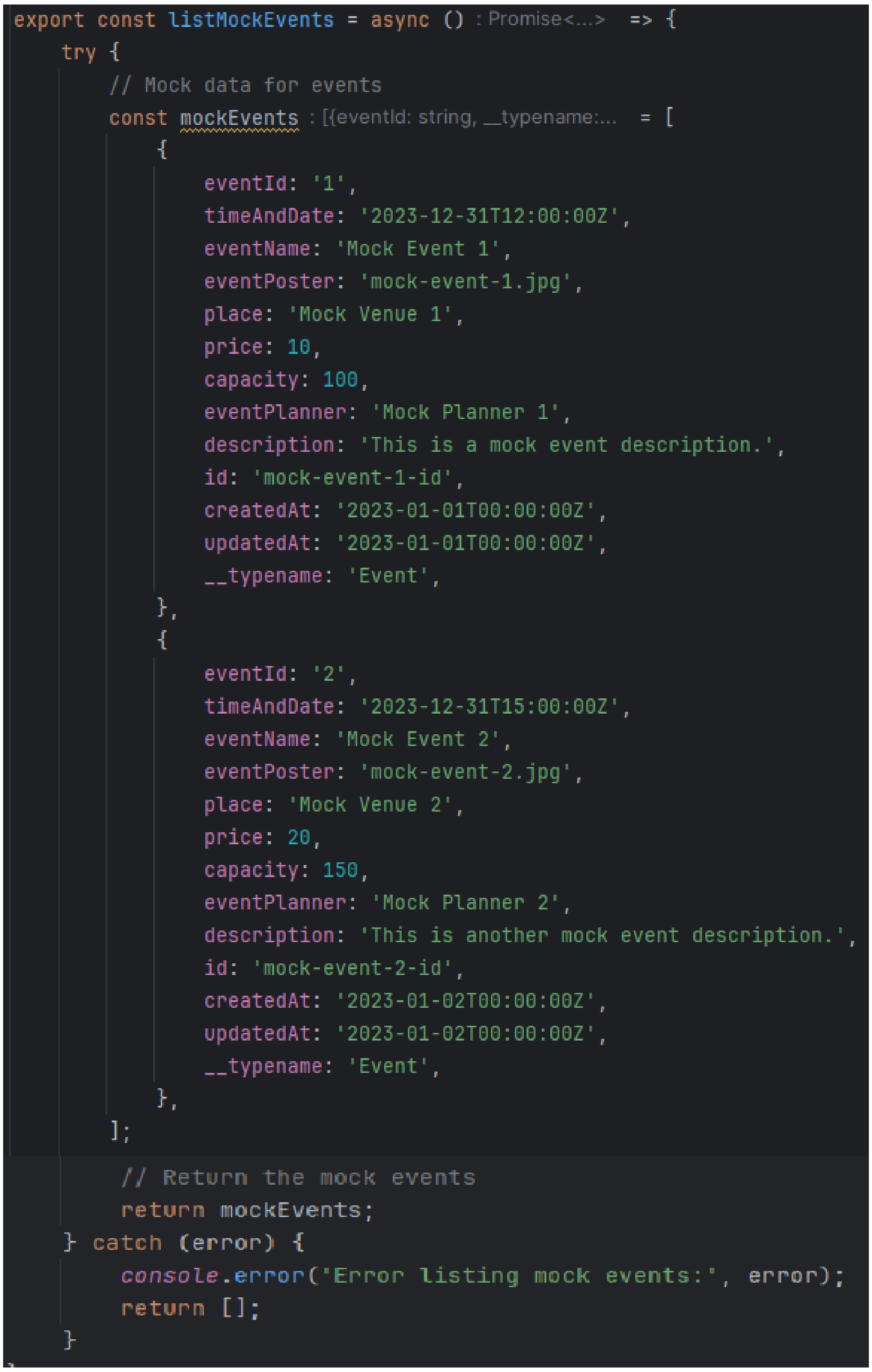


Figure 12: Event Create Form

1.3.3 Gökberk Lük

The bug in the Admin user that occurred in the system we created was fixed. At first, it was not visible in amplify studio even though it was created from Admin GraphQL. The admin user was recreated in both Amplify studio and GraphQL and this problem was resolved. During this process, Amplify CLI commands were learned in more depth.

Mock data has started to be created so that it can be shown at Upcoming events. For this, React and Javascript were researched. As seen in Figure 13, mock data was created for testing.



```
export const listMockEvents = async () : Promise<...>  => {
    try {
        // Mock data for events
        const mockEvents : [{eventId: string, __typename:...  = [
            {
                eventId: '1',
                timeAndDate: '2023-12-31T12:00:00Z',
                eventName: 'Mock Event 1',
                eventPoster: 'mock-event-1.jpg',
                place: 'Mock Venue 1',
                price: 10,
                capacity: 100,
                eventPlanner: 'Mock Planner 1',
                description: 'This is a mock event description.',
                id: 'mock-event-1-id',
                createdAt: '2023-01-01T00:00:00Z',
                updatedAt: '2023-01-01T00:00:00Z',
                __typename: 'Event',
            },
            {
                eventId: '2',
                timeAndDate: '2023-12-31T15:00:00Z',
                eventName: 'Mock Event 2',
                eventPoster: 'mock-event-2.jpg',
                place: 'Mock Venue 2',
                price: 20,
                capacity: 150,
                eventPlanner: 'Mock Planner 2',
                description: 'This is another mock event description.',
                id: 'mock-event-2-id',
                createdAt: '2023-01-02T00:00:00Z',
                updatedAt: '2023-01-02T00:00:00Z',
                __typename: 'Event',
            },
        ];
        // Return the mock events
        return mockEvents;
    } catch (error) {
        console.error('Error listing mock events:', error);
        return [];
    }
```

Figure 13: Mock.js for testing All Upcoming Events

1.4 Week 8: 20 November - 24 November

1.4.1 Berke Diler

No work was done due to the Midterm schedule.

1.4.2 Keremcan Adanur

No work was done due to the Midterm schedule.

1.4.3 Gökberk Lük

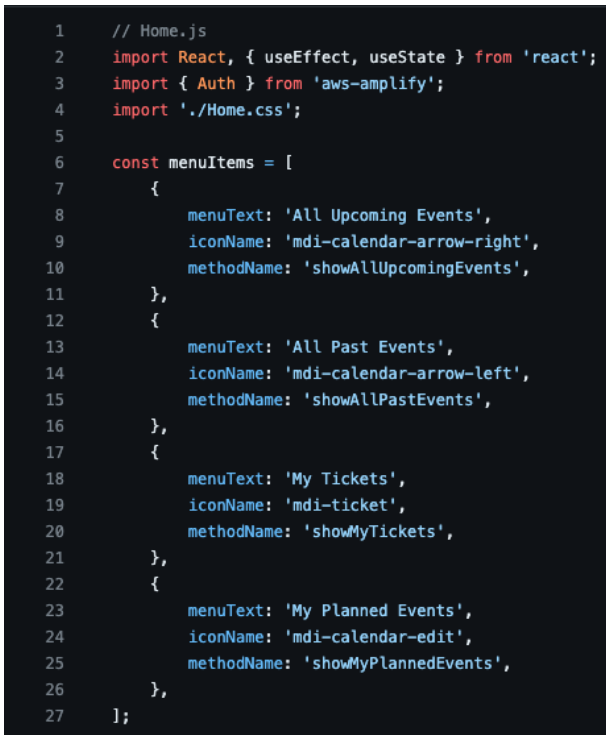
No work was done due to the Midterm schedule.

1.5 Week 9: 27 November - 1 December

1.5.1 Berke Diler

Built the user interface of the landing page of the application, where the events are shown when a user logs in. Created different tabs to show to the user, and the navigation system between these tabs. Also put buttons to sign out from the page and for admin functionalities.

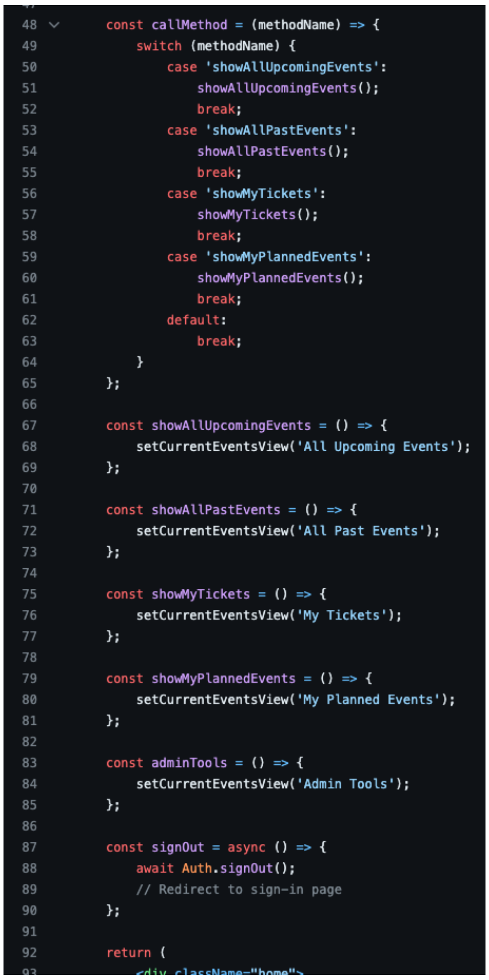
Menu items were created inside **src/views/Home.js** as can be seen on Figure 7.



```javascript
// Home.js
import React, { useEffect, useState } from 'react';
import { Auth } from 'aws-amplify';
import './Home.css';

const menuItems = [
    {
        menuText: 'All Upcoming Events',
        iconName: 'mdi-calendar-arrow-right',
        methodName: 'showAllUpcomingEvents',
    },
    {
        menuText: 'All Past Events',
        iconName: 'mdi-calendar-arrow-left',
        methodName: 'showAllPastEvents',
    },
    {
        menuText: 'My Tickets',
        iconName: 'mdi-ticket',
        methodName: 'showMyTickets',
    },
    {
        menuText: 'My Planned Events',
        iconName: 'mdi-calendar-edit',
        methodName: 'showMyPlannedEvents',
    },
];
```

Figure 14: Menu Items

Continuing from **src/views/Home.js**, the functionality to navigate between menu item tabs can be seen on Figure 8.



```
    const callMethod = (methodName) => {
        switch (methodName) {
            case 'showAllUpcomingEvents':
                showAllUpcomingEvents();
                break;
            case 'showAllPastEvents':
                showAllPastEvents();
                break;
            case 'showMyTickets':
                showMyTickets();
                break;
            case 'showMyPlannedEvents':
                showMyPlannedEvents();
                break;
            default:
                break;
        }
    };

    const showAllUpcomingEvents = () => {
        setCurrentEventsView('All Upcoming Events');
    };

    const showAllPastEvents = () => {
        setCurrentEventsView('All Past Events');
    };

    const showMyTickets = () => {
        setCurrentEventsView('My Tickets');
    };

    const showMyPlannedEvents = () => {
        setCurrentEventsView('My Planned Events');
    };

    const adminTools = () => {
        setCurrentEventsView('Admin Tools');
    };

    const signOut = async () => {
        await Auth.signOut();
        // Redirect to sign-in page
    };

    return (
        <div className="home">
```

Figure 15: Navigation

Finally the files **src/views/Home.css** and **src/views/Home.js** working together can be seen on Figure 9 as the home screen user interface logged as an Admin.

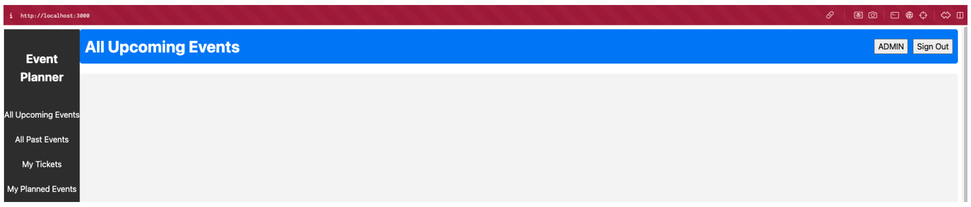
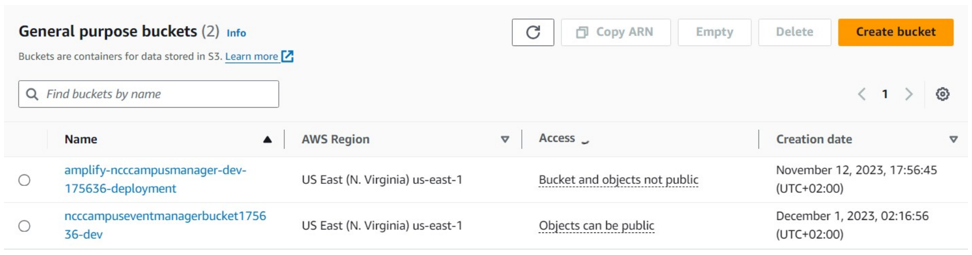


Figure 16: Enter Caption

1.5.2 Keremcan Adanur

This week I researched about how to add images to our application. I did research on S3 buckets how they are configured and be used for our projects purpose. Amplify already supports S3 buckets so I used **amplify add storage** and configured it to our needs.


General purpose buckets (2) Info

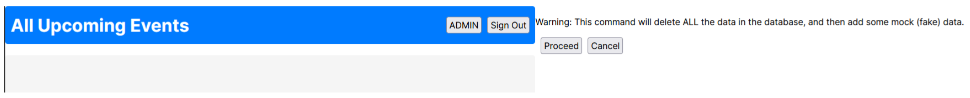
Buckets are containers for data stored in S3. Learn more

	Name	AWS Region	Access	Creation date
○	amplify-ncccampusmanager-dev-175636-deployment	US East (N. Virginia) us-east-1	Bucket and objects not public	November 12, 2023, 17:56:45 (UTC+02:00)
○	ncccampuseventmanagerbucket175636-dev	US East (N. Virginia) us-east-1	Objects can be public	December 1, 2023, 02:16:56 (UTC+02:00)

Figure 17: S3 Buckets

1.5.3 Gökberk Lük

This week, mock.js containing mocking data was created for testing. Additionally, as seen in the figure 18, when you click on the admin panel, an admin panel has been placed that pops up on the left side. Normally, mock data should be visible when you click the proceed button. However, this did not happen due to some communication problems between the pages. That's why another milestone has emerged that needs to be done.



Warning: This command will delete ALL the data in the database, and then add some mock (fake) data.

Figure 18: Admin Panel

Figure 19 shows the method created to display mock data on the upcoming event page.

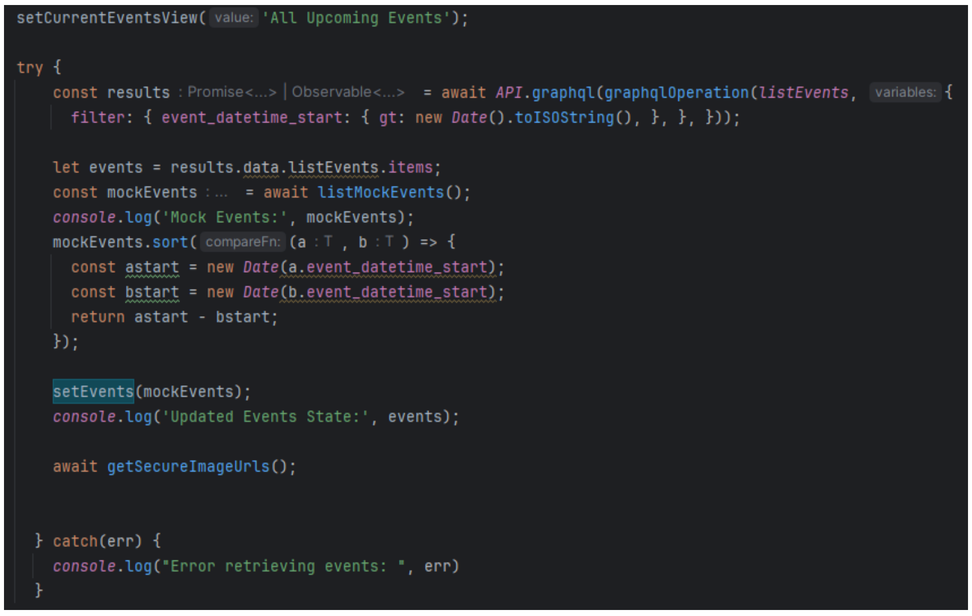
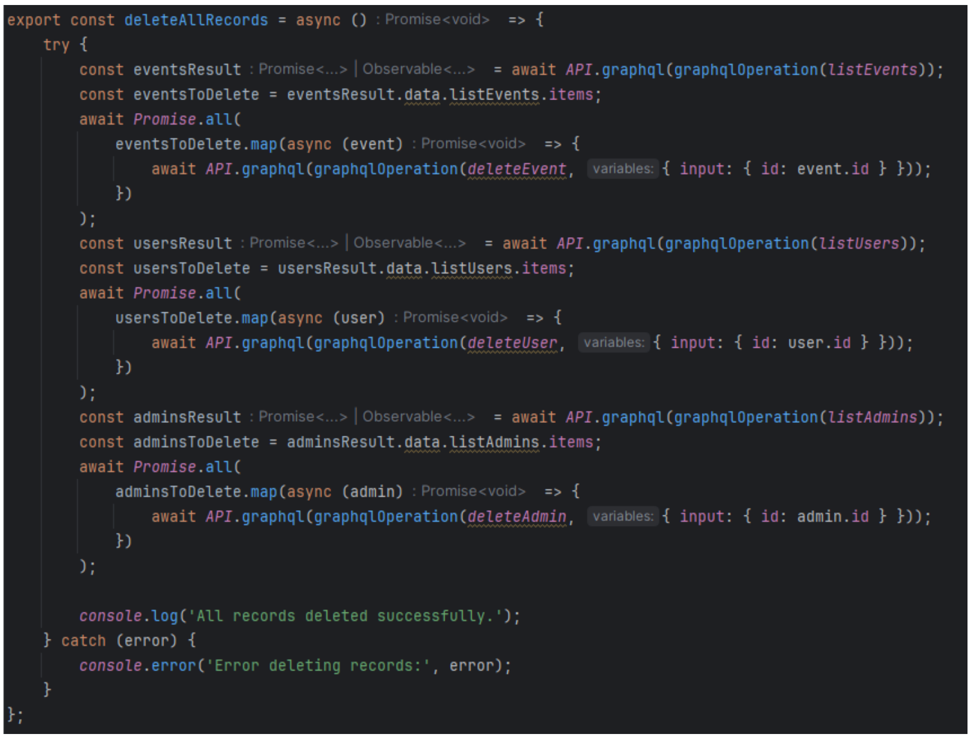

```
setCurrentEventsView( value: 'All Upcoming Events');

try {
    const results : Promise<...> | Observable<...>  = await API.graphql(graphqlOperation(listEvents,  variables: {
      filter: { event_datetime_start: { gt: new Date().toISOString(), }, }, }));

    let events = results.data.listEvents.items;
    const mockEvents : ...  = await listMockEvents();
    console.log('Mock Events:', mockEvents);
    mockEvents.sort( compareFn: (a : T , b : T ) => {
      const astart = new Date(a.event_datetime_start);
      const bstart = new Date(b.event_datetime_start);
      return astart - bstart;
    });

    setEvents(mockEvents);
    console.log('Updated Events State:', events);

    await getSecureImageUrls();


  } catch(err) {
    console.log("Error retrieving events: ", err)
  }
```

Figure 19: Upcoming Event View with Mocking

Figure 20 shows the method created to delete existing data.



```
export const deleteAllRecords = async () : Promise<void>  => {
    try {
        const eventsResult : Promise<...> | Observable<...>  = await API.graphql(graphqlOperation(listEvents));
        const eventsToDelete = eventsResult.data.listEvents.items;
        await Promise.all(
            eventsToDelete.map(async (event) : Promise<void>  => {
                await API.graphql(graphqlOperation(deleteEvent,  variables: { input: { id: event.id } }));
            })
        );
        const usersResult : Promise<...> | Observable<...>  = await API.graphql(graphqlOperation(listUsers));
        const usersToDelete = usersResult.data.listUsers.items;
        await Promise.all(
            usersToDelete.map(async (user) : Promise<void>  => {
                await API.graphql(graphqlOperation(deleteUser,  variables: { input: { id: user.id } }));
            })
        );
        const adminsResult : Promise<...> | Observable<...>  = await API.graphql(graphqlOperation(listAdmins));
        const adminsToDelete = adminsResult.data.listAdmins.items;
        await Promise.all(
            adminsToDelete.map(async (admin) : Promise<void>  => {
                await API.graphql(graphqlOperation(deleteAdmin,  variables: { input: { id: admin.id } }));
            })
        );

        console.log('All records deleted successfully.');
    } catch (error) {
        console.error('Error deleting records:', error);
    }
};
```

Figure 20: Dele All Data for Mocking

In addition, console logs have been added to follow the progress from the console over the Web. This will make things easier during the testing phase.

1.6 Week 9: 4 December - 8 December

1.6.1 Berke Diler

No work was done due to the Midterm schedule.

1.6.2 Keremcan Adanur

This week I worked on trying to display events on the main page. I wasn't able to finish it but it is one of the main functionalities of our system, to show the events correctly so it stays as one of the milestones. Also added the functionality to retrieve images from the s3 storage that I have configured the previous week but it still needs testing.

1.6.3 Gökberk Lük

No work was done due to I was out of the Cyprus.

2 Milestones Remained

Team Member	Milestones Remained
Berke Diler	**Milestone 1:**Implement the AWS Lambda function (maybe to be solved by configuring AWS Cognito) that checks if the user is registering with a metu mail, otherwise display an error. **Milestone 2:** Work on enhancing the UI and what other technologies to use to make the interface more appealing.
Keremcan Adanur	**Milestone 1:** Continue to implement and test retrieving images from S3 bucket. **Milestone 2:** Create event details cards that the users and admins will be able to interact with in order to book, create, edit or delete an event.
Gökberk Lük	**Milestone 1:** The problem causing the mockdata not to work will be found and the mock data will be implemented. **Milestone 2:** UI will be developed for admins and users to add and remove events.

3 Deliverables

As a team, we will be delivering the following components for the web app:

- A fully functional GitHub repository with the web application code, including mock data and a comprehensive readme file.
- Implementation of a fully functional non-relational database to support the application's data storage needs.
- Implementation of an authentication system using AWS Cognito to ensure secure user access.
- Implementation of backend Lambda functions to handle serverless computing tasks.
- Storage of relevant elements, such as images, inside AWS S3 buckets.

4 GitHub Repository

https://github.com/bercats/ncc_campus_event_manager.git

5 References

[1]Amazon Web Services. (2002). Create a React app by using AWS Amplify and add authentication with Amazon Cognito. Amazon. https://catalog.us-east-1.prod.workshops.aws/workshops/1665a9b6-

958b-4b70-ba52-14127b8fa99f/en-US/introduction/scenario

Amazon Web Services Console. https://console.aws.amazon.com/