Testing:

Username: test

Password: test

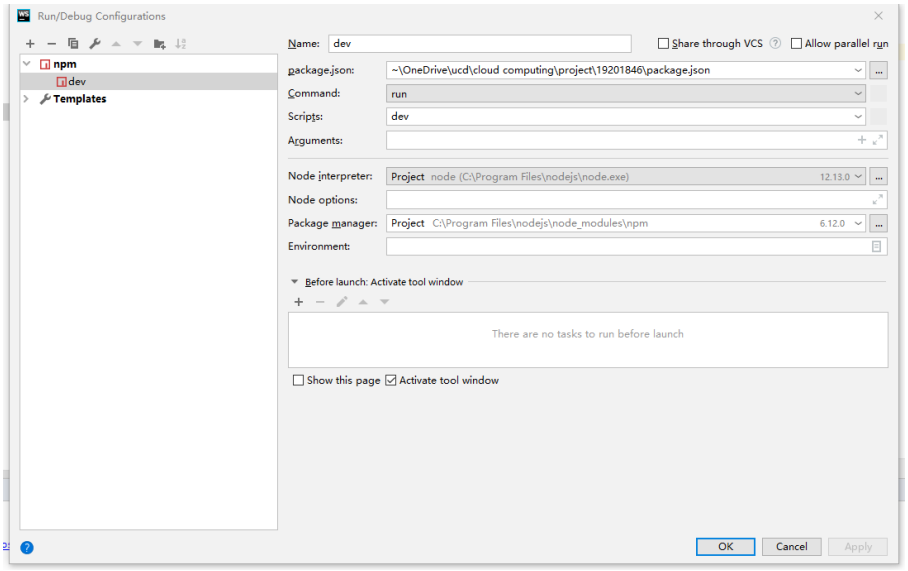
How To Run:

1. Front end project name: 19201846.

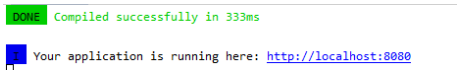
   Installing Node.js: Download the node.js from nodejs.org and install it.

   To run the project: npm run dev

   The project will be run in http://localhost:8080. Or you can simply open it in Webstorm and configure to run it:



   It will look like this:



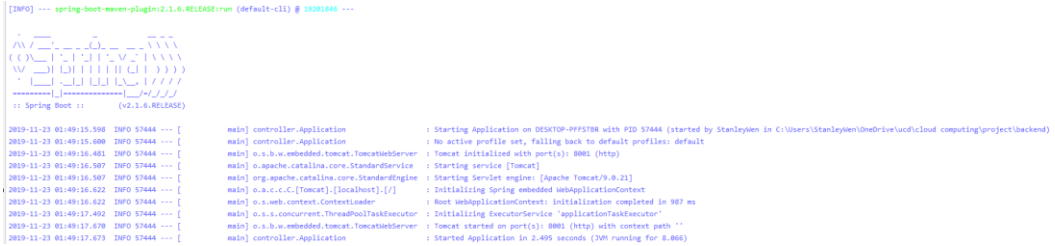
2. Back end project name: backend.

   Install the Maven following the instructions: https://maven.apache.org/install.html.

   To run the project:

   1. mvn install
   2. mvn compile spring-boot:run

   It will look like this:



Key Class:

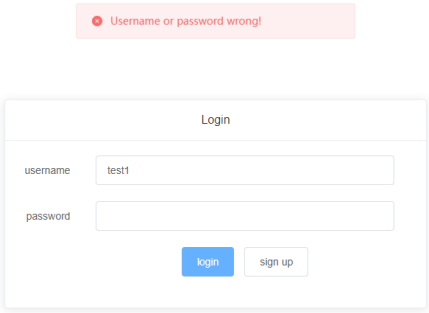
"AmazonController" Class: It includes 2 parts: initialize Amazon S3 client, and initialize Amazon DynamoDB mapper.

```
public AmazonController(){
    AWSCredentials credentials = new BasicAWSCredentials(accessKeyID, secretKey);
    ClientConfiguration clientConfig = new ClientConfiguration();
    clientConfig.setSignerOverride("S3SignerType");//凭证验证方式
    clientConfig.setProtocol(Protocol.HTTP);//访问协议
    s3client = AmazonS3ClientBuilder.standard()
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .withRegion(Regions.EU_WEST_1)
            .build();

    amazonDynamoDBClient =  AmazonDynamoDBClientBuilder.standard()
                                .withCredentials(awsCredentialsProvider)
                                .withRegion(Regions.EU_WEST_1)
                                .build();
    dbMapper = new DynamoDBMapper(amazonDynamoDBClient);
    table = new DynamoDB(amazonDynamoDBClient).getTable(TABLE_NAME);
}
```

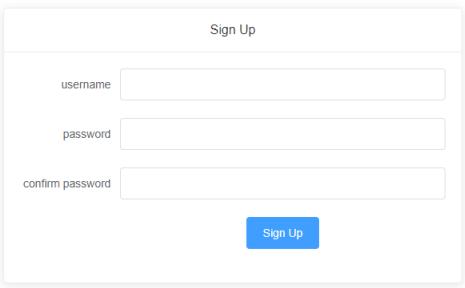
The other classes will be introduced with the following function's introduction.

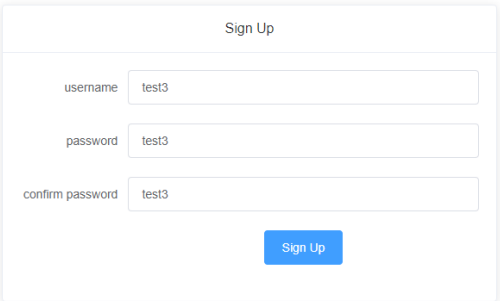
Function Introduction:

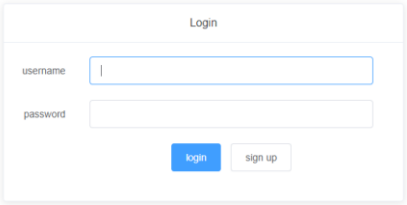
1. Login with unsigned username:
   Example: Username: test1



2. Sign up: Clicking the "sign up" button and will go to the following "/signUp" page:
   After typing in the username and password and clicking the "Sign Up" bottom, it will send a "post" request to backend ("http://localhost:8001/signUp"), including the data of username and password. After finishing the sign up process, it will be automatically redirected to the login page.
   Example: Username: test3 Password: test3
   Web page:







Front end:

```
this.$http({
    method: 'post',
    url: 'http://localhost:8001/signUp',
    data: {
        "username": this.signUpInfo.username,
        "password": this.signUpInfo.password
    }
}
).then(result => {
    console.log(result);
    this.fullscreenLoading = false;
    if (result.status === 200){
        this.$message.success("Sign Up Success!")
        this.$router.push('/');
    }
    else {
        this.fullscreenLoading = false;
        this.$message.error(result.data.message);
    }
```

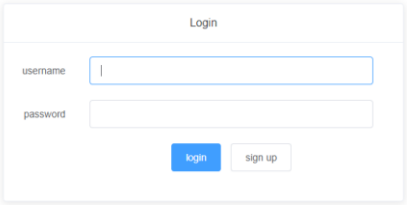
Back end:

```
@RequestMapping(value = "/signUp", method = RequestMethod.POST)
public static void addOneItem(@RequestBody AwsUser user) {
    dbMapper.save(user);
}
```

The Class AwsUser is defined by the following code:

```
@DynamoDBTable(tableName = "awsUser")
public class AwsUser {
    private String username = null;
    private String password = null;
    public AwsUser(String username, String password) {
        super();
        this.username = username;
        this.password = password;
    }
    //主键
    @DynamoDBHashKey(attributeName = "username")
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    public AwsUser() {}

    //配有索引 userName-index
    @DynamoDBAttribute(attributeName = "password")
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
}
```

3. Login: Using the testing username and password, we will go to the following page:



The “login” bottom will send a post request including the username and password to backend (“http://localhost:8001/login“), and backend will see if the DynamoDB has this

user and whether the password is correct.

Front end:

```
this.$http({
    method: 'post',
    data: {
        "username": this.loginInfo.username,
        "password": this.loginInfo.password
    },
    url: 'http://localhost:8001/login'
}).then(result => {
    this.fullscreenLoading = false;
    if (result.status === 200){
        this.$store.commit('LOGIN', this.loginInfo.username);
        this.$message({
            message: 'Login Success!',
            type: 'success',
        });
        this.$router.push('/event_organiser');
    }
    else {
        this.$message.error('Username or password wrong!');
        this.password = '';
    }
```

Backend:

```
@RequestMapping(value = "/login", method = RequestMethod.POST)
private ResponseEntity<Object> login(@RequestBody User user) {
    AwsUser awsUser = getItemByUsername(user.getUsername());
    if (awsUser == null){
        return new ResponseEntity<Object>( headers: null, HttpStatus.UNAUTHORIZED);
    }
    if (awsUser.getPassword().equals(user.getPassword())){
        return new ResponseEntity<Object>( headers: null, HttpStatus.OK);
    }
    else {
        return new ResponseEntity<Object>( headers: null, HttpStatus.UNAUTHORIZED);
    }
}
```

The Class User is defined by the following code:

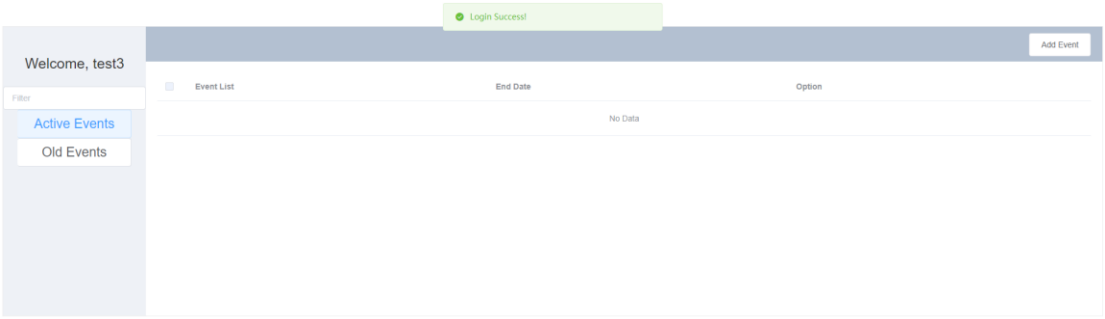
```
public class User {
    private String username;
    private String password;
```

The method “getItemByUsername” is a self-implemented fuction:

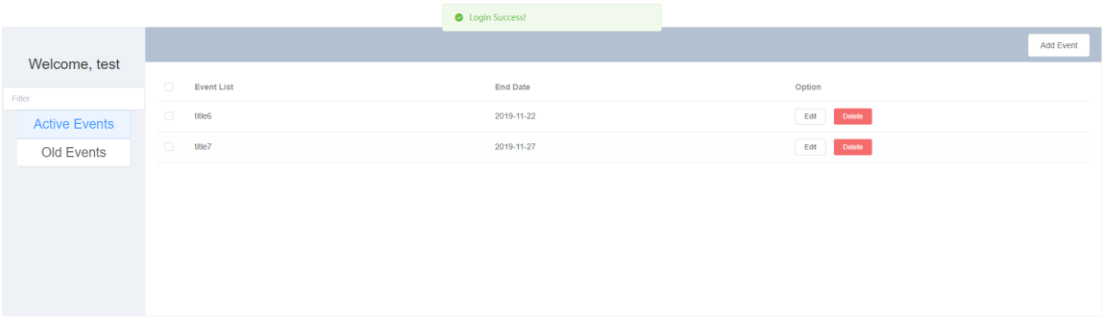
```
private static AwsUser getItemByUsername(String username) { return dbMapper.load(AwsUser.class, username); }
```

It will send the username to the DynamoDB to get the AwsUser instance, and by comparing whether the typed in password matches the AwsUser password, we can judge whether the user typed in the correct password of its username.

After logging in, it will display the username in the page. If we use the username “test3” that we signed up in 2., it will show the following page(because we just sign it up, it includes 0 item):



If we use the username “test”, we can get the following event list.



It is done by sending the username to the backend to get the exact event list of a specific user. After getting the list, the front end will judge whether every event is active or old by its end date.

Front end: The method “judgeStatus” is a self-implemented fuction, by passing the date, it will compare the date to the current date and return whether it’s an active event or an old one. The active ones will be pushed into the “activeEvent” Array, and the old ones will be pushed into the “oldEvent” Array. Because I set the default display list to be active events, I assign the active events list to the “tableData” variable to be displayed.

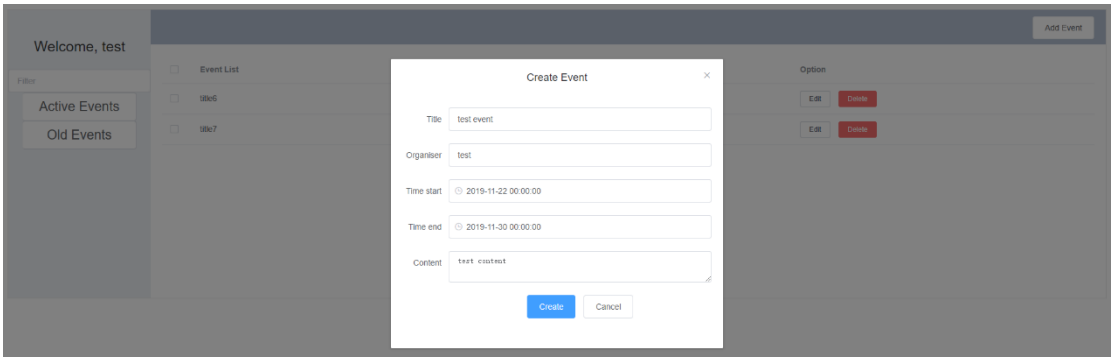
```
this.$http({
    method: 'post',
    data: {
        "username": this.username,
    },
    // url: 'https://5c39574f-0591-4486-abbc-aafa89d4dbec.mock.pstmn.io/amazonS3/list'
    url: 'http://localhost:8001/amazonS3/list'
}).then(result => {
    //this.tableData = result.data;
    var resultData;
        for (resultData in result.data){
            if (this.judgeStatus(result.data[resultData].date) === "Active"){
            this.activeEvent.push(result.data[resultData]);
        }
        else {
            this.oldEvent.push(result.data[resultData]);
        }
        this.tableData = this.activeEvent;
    }
```

Back end: On receiving the Class “Username”, the backend will extract the username and send the bucket name and username(folder name) to the S3. Because the return list contains the item that only include the folder name, I add a condition statement to skip this item.

```
@RequestMapping(value = "/amazonS3/list", method = RequestMethod.POST)
private ResponseEntity<Object> listObjects(@RequestBody Username username){
    ArrayList<VueTable> arrayList= new ArrayList<>();
    ObjectListing objectListing = s3client.listObjects(bucket, username.getUsername());
    for(S3ObjectSummary os : objectListing.getObjectSummaries()) {
        String tmp = os.getKey();
        if (!tmp.contains(username.getUsername() + "/")){
            continue;
        }
        if (tmp.equals(username.getUsername()+"/")){
            continue;
        }
        tmp = tmp.replace( target: username.getUsername()+"/",  replacement: "");
        //System.out.println(tmp);
        int index = tmp.indexOf("@");
        arrayList.add(new VueTable(username.getUsername(), tmp.substring(0, index), tmp.substring(index+1)));
    }
    return new ResponseEntity<Object>(arrayList, HttpStatus.OK);
}
```

4. Create new event: by clicking the “Add Event” button, user can add new event in the following dialog:

   Web page:



After clicking the “Create” bottom, it will trigger the “onSubmit” function to submit the event details to the backend(http://localhost:8001/amazonS3/addEvent).

Front end:

```
onSubmit(){
    this.fullscreenLoading = true;
    this.$http({
        method: 'post',
        data: {
            "user": this.username,
            "title": this.form.title,
            "organiser": this.form.organiser,
            "startTime": this.form.startTime,
            "endTime": this.form.endTime,
            "content": this.form.content
        },
        url: 'http://localhost:8001/amazonS3/addEvent'
    }).then(result => {
        this.fullscreenLoading = false;
        console.log(result);
        if (result.status === 200){
            window.location.reload();
        }
    }).catch(err => {
            this.fullscreenLoading = false;
            this.$message.error('Login Fail! ' + err.response.data.message);
        }
    );
},
```

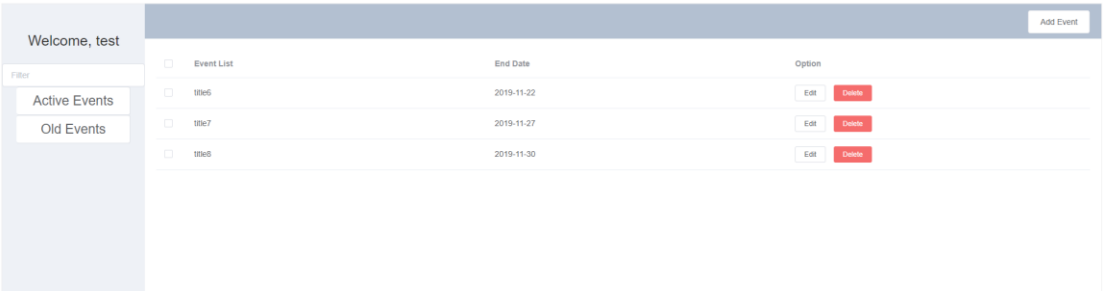
Back end: I use a file to storage the event item. After receiving a request, it will create a template file to store the event details, and upload the file to Amazon S3. I separate the files by creating a folder for every single user, the name of the folders are the registered username.

```
@RequestMapping(value = "/amazonS3/addEvent", method = RequestMethod.POST)
private boolean addEvent(@RequestBody Event event){
    try {
        System.out.println(event.getUser() + "/" + event.getTitle()+"@"+event.getEndTime().substring(0, 10));
        File temp = new File( pathname: event.getTitle()+"@"+event.getEndTime().substring(0, 10));
        temp.createNewFile();
        BufferedWriter out = new BufferedWriter(new FileWriter(temp));
        out.write(event.getTitle());
        out.newLine();
        out.write(event.getOrganiser());
        out.newLine();
        out.write(event.getStartTime());
        out.newLine();
        out.write(event.getEndTime());
        out.newLine();
        out.write(event.getContent());
        out.close();
        FileInputStream fileInputStream = new FileInputStream(temp);
        MultipartFile file = new MockMultipartFile(temp.getName(), temp.getName(),
                ContentType.APPLICATION_OCTET_STREAM.toString(), fileInputStream);
        ObjectMetadata metadata = new ObjectMetadata();
        //System.out.println(file.isEmpty());
        metadata.setContentType(file.getContentType());
        metadata.setContentLength(file.getSize());
        metadata.addUserMetadata("x-amz-meta-title", "aws file");
        PutObjectResult result = s3client.putObject(new PutObjectRequest(bucket,  key: event.getUser() + "/" + file.getOriginalFilename(), file.getInputStream(), metadata));
        temp.delete();
        return true;
```

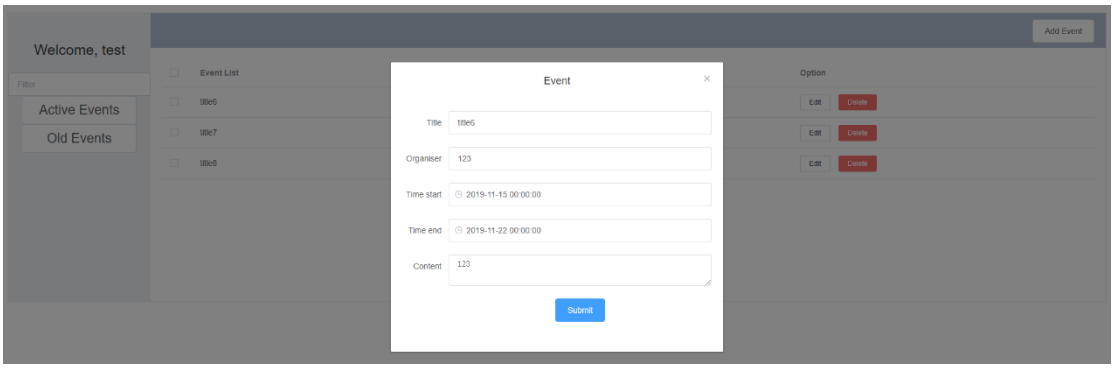
The “Event” class is defined by the following code:

```
public class Event {
    private String user;
    private String title;
    private String organiser;
    private String startTime;
    private String endTime;
    private String content;
```

After the backend return the 200 status, the front end will refresh the page automatically, and the added event will show up in the web page.



5. Edit active event: by clicking the "Edit" button in every row, user can edit the specific event:



It will automatically fill in the details by calling a "handleEdit" function, which get the parameter of the username, title and date, and send these three data to the backend (http://localhost:8001/getObject).

Front end:

```
handleEdit(row){
    this.editEvent = true;
    this.$http({
        method: 'post',
        data: {
            "user": this.username,
            "title": row.title,
            "date": row.date,
        },
        url: 'http://localhost:8001/getObject'
    }).then(result => {
        //console.log(result.data);
        this.event = result.data;
    }).catch(err => {
            this.$message.error('Login Fail! ' + err.response.data.message);
        }
    );
    //console.log(row);
},
```

Back end: It is quite complex because it need to firstly download the file according to its title and end date. After downloading the file, I get every string object by getting a char each time and stop until getting a "\n". After getting all the details, it construct an Event object and return it to the front end. On receiving the data, the front end will pass them to the "event" variable to be displayed.

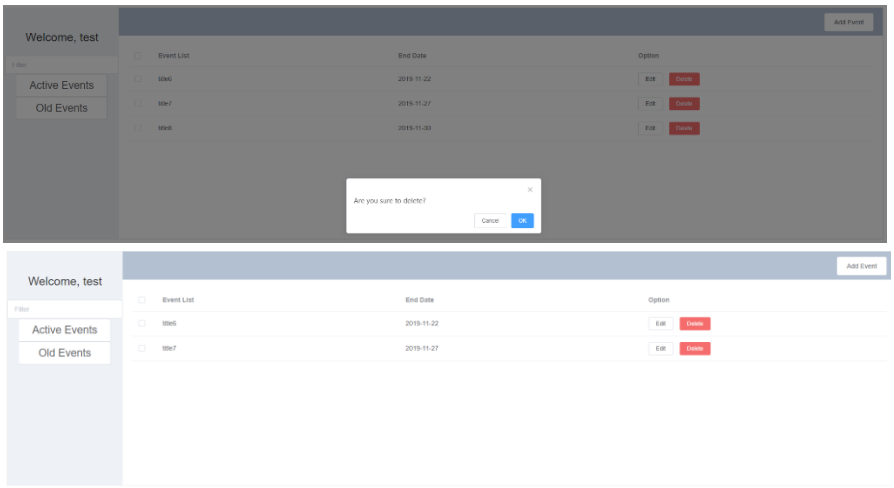
```
@RequestMapping(value = "/getObject", method = RequestMethod.POST)
private ResponseEntity<Object> download(@RequestBody VueTable object) throws IOException {
    S3Object s3object = s3client.getObject(bucket,  s1: object.getUser() + "/" + object.getTitle()+"@"+object.getDate());
    S3ObjectInputStream inputStream = s3object.getObjectContent();
    int i;
    String title = "";
    while((i = inputStream.read()) != -1){
        char c=(char)i;
        if (c == '\n')
            break;
        title = title + c;
    }
    String organiser = "";
    while((i = inputStream.read()) != -1){
        char c=(char)i;
        if (c == '\n')
            break;
        organiser = organiser + c;
    }
    String startTime = "";
    while((i = inputStream.read()) != -1){
        char c=(char)i;
        if (c == '\n')
            break;
        startTime = startTime + c;
    }
    startTime = startTime.replace( target: "T",  replacement: " ");
    startTime = startTime.replace( target: "Z",  replacement: "");
    String endTime = "";
    while((i = inputStream.read()) != -1){
        char c=(char)i;
        if (c == '\n')
            break;
        endTime = endTime + c;
    }
    endTime = endTime.replace( target: "T",  replacement: " ");
    endTime = endTime.replace( target: "Z",  replacement: "");
    String content = "";
    while((i = inputStream.read()) != -1){
        char c=(char)i;
        content = content + c;
    }
    Event event = new Event(object.getUser(), title, organiser, startTime, endTime, content);
    return new ResponseEntity<Object>(event, HttpStatus.OK);
```

If the user want to modify the data, after editing data, user can click the “Submit” bottom, whcih will call the OnEditSubmit function in the front end:

```
onEditSubmit(){
    this.$http({
        method: 'post',
        data: {
            "user": this.username,
            "title": this.event.title,
            "organiser": this.event.organiser,
            "startTime": this.event.startTime,
            "endTime": this.event.endTime,
            "content": this.event.content
        },
        url: 'http://localhost:8001/amazonS3/editEvent'
    }).then(result => {
        //console.log(result);
        if (result.status === 200){
            window.location.reload();
        }
    }).catch(err => {
            this.$message.error('Login Fail! ' + err.response.data.message);
        }
    );
},
```

It will send a post request to http://localhost:8001/amazonS3/editEvent. It’s similar to the “addEvent” function in 4., so I’m not going to describe it. Because the Amazon S3’s mechanism is that if a user upload two files that have the same name, the later uploaded one will replace the earlier one, so the editing can be simply done by uploading the modified file with the same name.

6. Delete an event: By clicking the “delete” bottom, the event will be deleted.



Front end: After confirming that the user want to delete the event, it will send a delete request to the backend (http://localhost:8001/delete).

```
handleDelete(row){
    this.$confirm('Are you sure to delete?')
        .then(_ => {
            this.$http({
                method: 'delete',
                data: {
                    "user": this.username,
                    "title": row.title,
                    "date": row.date,
                },
                url: 'http://localhost:8001/delete'
            }).then(result => {
                //console.log(result.data);
                window.location.reload();
            }).catch(err => {
                    this.$message.error('Login Fail! ' + err.response.data.message);
                }
            );
            done();
        })
        .catch(_ => {});
},
```

Backend: On receiving the request, it will directly delete the file in the Amazon S3.

```
@RequestMapping(value = "/delete", method = RequestMethod.DELETE)
private ResponseEntity<Object> deleteObjects(@RequestBody VueTable table){
    s3client.deleteObject(bucket, s1: table.getUser() + "/" + table.getTitle()+"@"+table.getDate());
    return new ResponseEntity<Object>( headers: null, HttpStatus.OK);
}
```

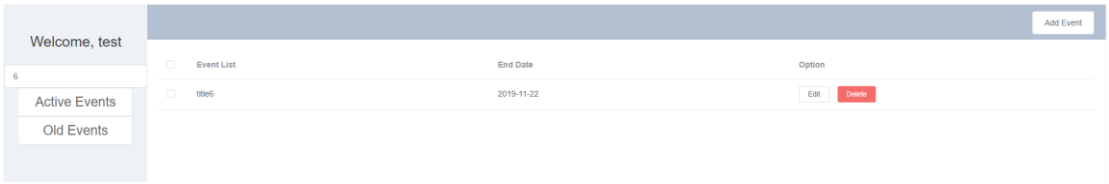
7. Archive old events: If the user click the “Old Events” bottom, it will change to display the old events.
   Web page:

Front end: I implement a simple “judgeStatus” function to define whether an event is an active one or an old one. This is done on logging in, when the received whole list will be separate by this function to two Array: activeEvent and oldEvent. The two bottoms ”Active

Event" and "Old Event" are used to switch between these two event list to be shown.

```
judgeStatus: function(oldTime){
    var time = new Date();
    // let oldTime = row.row.date;
    if (oldTime.substr(0,4) < time.getFullYear()){
        return "Old";
    }
    else if (oldTime.substr(0,4) > time.getFullYear()){
        return "Active";
    }
    else if (oldTime.substr(5,2) < time.getMonth()+1){
        return "Old";
    }
    else if (oldTime.substr(5,2) > time.getMonth()+1){
        return "Active";
    }
    else if (oldTime.substr(8,2) < time.getDate()){
        return "Old";
    }
    else if (oldTime.substr(8,2) >= time.getDate()){
        return "Active";
    }
},
```

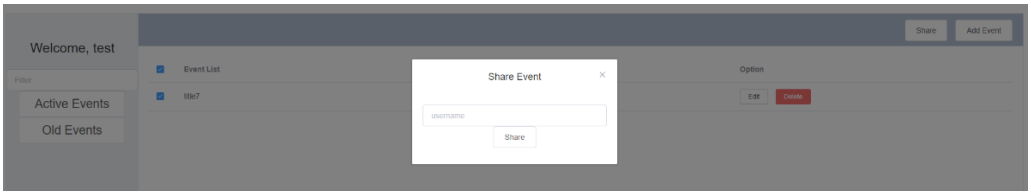
8. Search for events: After typing in part of title's name of an event and press "Enter", it will change the table list to display only the search result.
   Web page: After I typed in "6" and press "Enter", it changes to show only the event with title "title6".



   Front end: Pressing the "Enter" will trigger "search" function, which go through the "activeEvent" list and "oldEvent" list to find whether the keyword exists in these two lists, and if it exists, the event will be pushed into a list. In the end, this list will be assigned to the "tableData" variable to be displayed.

```
search(){
    var keyWord = this.filterText;
    var filterResult = [];
    var index;
    for (index in this.activeEvent){
        if (this.activeEvent[index].title.indexOf(keyWord)!==-1){
            //console.log(this.activeEvent[index]);
            filterResult.push(this.activeEvent[index]);
        }
        if (this.oldEvent[index].title.indexOf(keyWord)!==-1){
            //console.log(this.oldEvent[index]);
            filterResult.push(this.activeEvent[index]);
        }
        //console.log(filterResult);
        this.tableData = filterResult;
    }
},
```

9. Share an event or a ser of events: After clicking the check box one the left side of each row, the "Share" bottom will appear, and after typing in the user that you want to share event to, the event will be shared to that user.



Front end: Firstly, it will send a post request to backend ("http://localhost:8001/find") to check that whether the username that the user want to share event to exists, and if it exists, it will send a post request to backend (http://localhost:8001/share) with the destination and events list data.

```
shareTablesToUser(){
    this.$http({
        method: 'post',
        data: {
            "username": this.shareTo,
        },
        url: 'http://localhost:8001/find'
    }).then(result => {
        if (result.status === 200){
            this.$http({
                method: 'post',
                data: {
                    "from": this.username,
                    "to": this.shareTo,
                    "tables": this.shareTables
                },
                url: 'http://localhost:8001/share'
            }).then(result => {
                //console.log(result);
            }).catch(err => {
                    this.$message.error('Fail! ' + err.response);
                }
            );
        }
```

Backend: To find whether the user exists, it will call the "getItemByUsername" function to see whether the return value is a null pointer or not. If it isn't a null pointer, the user exists.

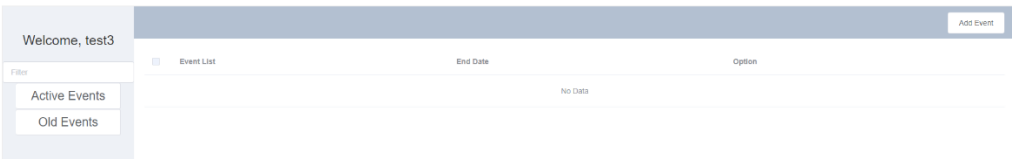
```
@RequestMapping(value = "/find", method = RequestMethod.POST)
private ResponseEntity<Object> findUser(@RequestBody Username username) {
    AwsUser awsUser = getItemByUsername(username.getUsername());
    if (awsUser != null){
        return new ResponseEntity<Object>( headers: null, HttpStatus.OK);
    }
    else {
        return new ResponseEntity<Object>( headers: null, HttpStatus.UNAUTHORIZED);
    }
}
```

Then, to share the event to another user, I use the copy method to copy the event file stored in Amazon S3 from its origin folder to the destine user's folder. Because it may contains several events, so I use an Array to store the events.

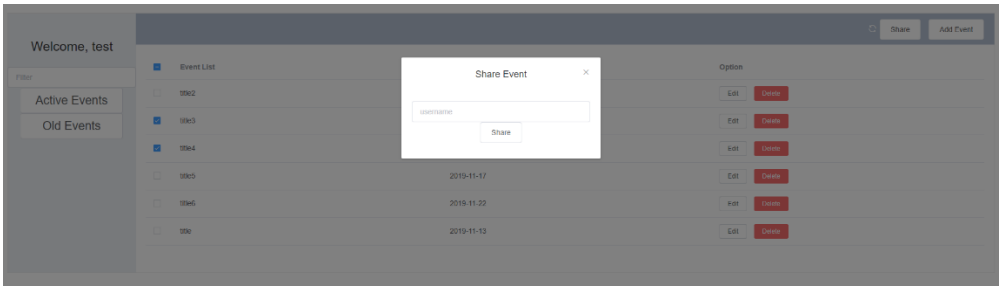
```
@RequestMapping(value = "/share", method = RequestMethod.POST)
private ResponseEntity<Object> share(@RequestBody ShareEvent shareEvent){
    for (VueTable vueTable : shareEvent.getTables()){
        s3client.copyObject(bucket, s1: shareEvent.getFrom()+"/" + vueTable.getTitle() + "@" +
                vueTable.getDate(), bucket, s3: shareEvent.getTo()+"/" + vueTable.getTitle() + "@" + vueTable.getDate());
    }

    return new ResponseEntity<Object>( headers: null, HttpStatus.OK);
}
```

```
public class ShareEvent {
    private String from;
    private String to;
    private ArrayList<VueTable> tables;
```

To show its effect, I use the account that we just signed up: test3. At first, it has no data.



After I share two events from "test" to "test3",



These two events will be added to “test3”’s events lists. Clicking the refresh bottom on the header, and we can find that the two events appear.