0.1 Website

a. Login Screen: index.html

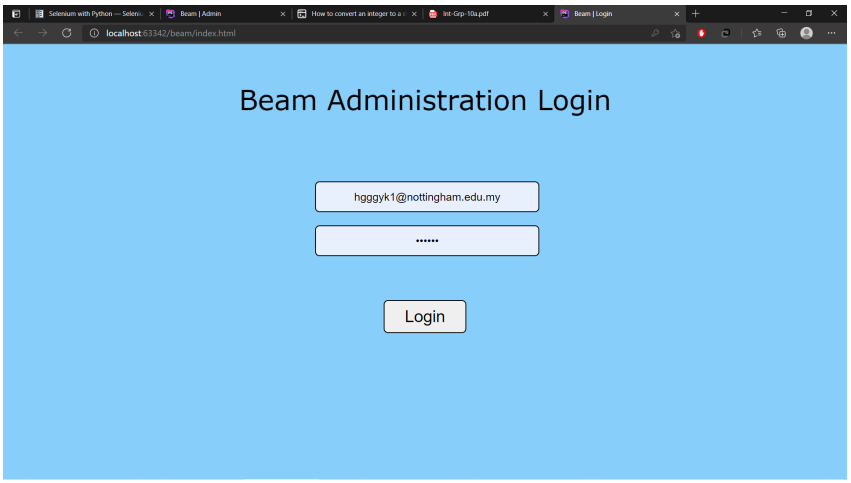

Figure 0.1.1: *Login interface of the website*

The login interface comprises of a title – Beam Administration Login, an input field for email, and an input field for password. Upon clicking to the login button, the *login()* function will be executed, as shown below.

```
function login(){
    var email = document.getElementById("email");
    var password = document.getElementById("password");
    const promise = auth.signInWithEmailAndPassword(email.value, password.value
    );
    promise.catch(e => alert(e.message));
}

auth.onAuthStateChanged(user => {
    if(user) {
        window.location.href = 'home.html';
    }
});
auth.setPersistence(firebase.auth.Auth.Persistence.SESSION).then(function() {
    return auth.signInWithEmailAndPassword(email, password);
});
```

The *login* function takes the email and password entered by the user and sends a promise to Firebase Authentication to check the login credentials. If the credentials are valid, the authentication state will be changed because of a successful user account login and the user will be redirected to *home.html*. The user will remain logged in unless the account is logged out or the user switched to a new tab or window. The state of being logged in will only persist in the current tab or session, and it will be cleared of the tab or window is closed.

```
function logOut(){
    firebase.auth().signOut();
    window.location.href = 'index.html';
}
var user = firebase.auth().currentUser;
firebase.auth().onAuthStateChanged(function(user) {
    if (user) {
        // User is signed in.
    } else {
        window.location.href = 'index.html';
```

```
    }
});
```

The *logOut* function logs the user out from the account and redirects the user back to the login page. If the user accesses the home page without going through the login page, the system will check if the user has logged in on Firebase Authentication. If not, the user will be redirected to the login page. The user will remain on the home page if Firebase Authentication detects that the user has performed a successful login.

b. Home Screen: home.html

Home Component

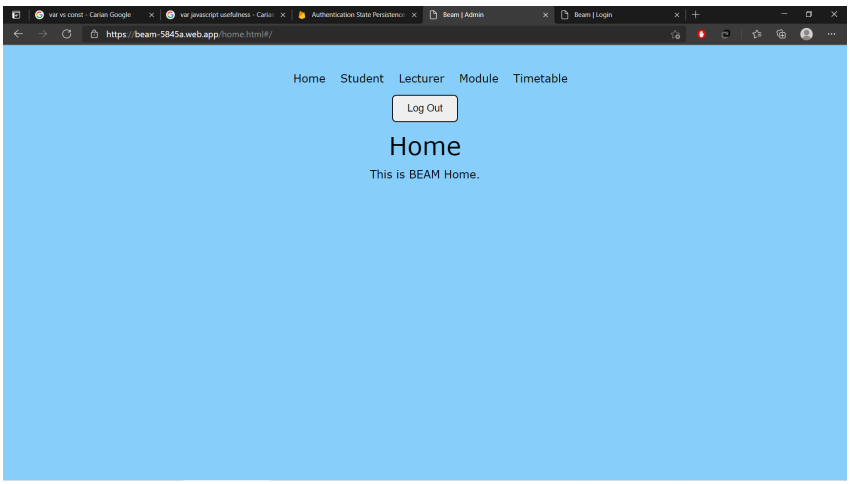

Figure 0.1.2: *This is the Home interface of the website. Note that it changed to any important announcement from the university*

```
var routes = [
    { path: '/', component: Home},
    { path: '/student', component: Student },
    { path: '/lecturer', component: Lecturer },
    { path: '/modules', component: Modules },
    { path: '/plan', component: Plan },
    { path: '/timetable', component: Timetable },
    { path: '', redirect: '/' },
];

var router = new VueRouter({
    routes: routes,
    mode: 'hash',
    base: '/'
});

var app = new Vue({
    el: '#app',
    router: router,
})
```

This home page is a *single-page application (SPA)* that contains all the features needed for the admin to enter data into the database. Vue.js functions are imported using CDN and used to enable us to build a *SPA*. This Vue.js *SPA* is created in a * * with an id of *app* with a *router* to link each tab of the navigation with a path. The router will load the contents of each tab in *hash* mode, which means # is used before the navigation path is passed internally. Since

this section of the URL is not sent to the server, no special treatment is required on the server level.

The code above directs each path to a component. Every component is an x-template, where the html elements (front-end) of a template in stored in a *<script>* tag in a HTML file. The router will redirect the tabs to their respective template.

Student Component

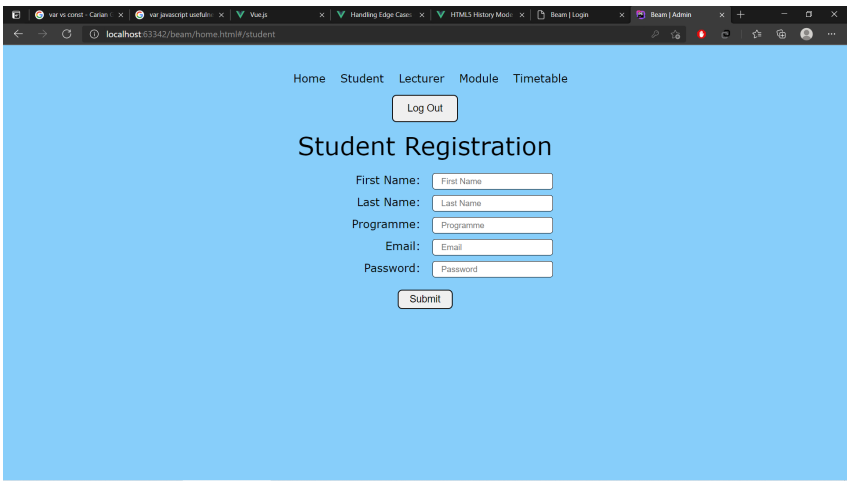

Figure 0.1.3: *This is the Student Registration interface of the website*

```
firebase.auth().createUserWithEmailAndPassword(email, password).then((
userCredential) => {
    var userUid = userCredential.user.uid;
    var student = {
        FirstName: fName,
        LastName: lName,
        Programme: programme,
        Email: email
    };
    firebase.database().ref("student/" + userUid).set(student).catch(e =>
alert(e.message));
    var userRole = {
        Programme: programme,
        Role: "Student"
    }
    firebase.database().ref("users/" + userUid).set(userRole).catch(e =>
alert(e.message));
    var key = firebase.database().ref("plan/" + userRole.Programme);
    var children = key.child("Core");
    children.once('value', (snapshot) => {
        snapshot.forEach((child) => {
            firebase.database().ref("users/" + userUid + "/modules").push(
child.key).catch(e => alert(e.message));
            firebase.database().ref("modules/" + child.key + "/students").
push(userUid).catch(e => alert(e.message));
        });
    });
}).catch(e => alert(e.message));
```

When the *email* and *password* are sent to Firebase Authentication to create a new account for the student, the *first name*, *last name*, *programme*, and *email*, are entered under the *student* node and grouped by the student account's *user uid*. Under the *users* node, the role of the account (Student) and the *programme* are entered, grouped by *user uid*. With the programme

name, the system will fetch all the *module ids* of the core modules of the *programme* and store them in the *modules* node under *user uid* node, which is also under *users* node. The system will also register the *user uid* of the students under *students* node of each *modules* node.

Lecturer Component

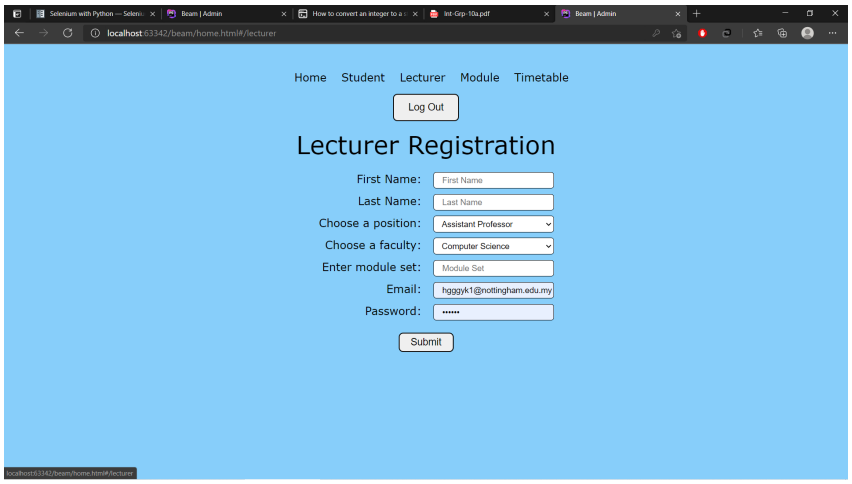

Figure 0.1.4: *This is the Lecturer Registration interface of the website*

```
1   firebase.auth().createUserWithEmailAndPassword(email, password).then((
    userCredential) => {
2       var userUid = userCredential.user.uid;
3       var lecturer = {
4           FirstName: fName,
5           LastName: lName,
6           Position: position,
7           Faculty: faculty,
8           Email: email
9       };
10      firebase.database().ref("lecturer/" + userUid).set(lecturer).catch(e =>
     alert(e.message));
11      var userRole = {
12          Role: "Lecturer",
13          set: set,
14          Programme: faculty
15      }
16      firebase.database().ref("users/" + userUid).set(userRole).catch(e =>
    alert(e.message));
17      var key = firebase.database().ref("plan/" + userRole.Programme);
18      var children = key.child(userRole.set);
19      children.once('value', (snapshot) => {
20          snapshot.forEach((child) => {
21              firebase.database().ref("users/" + userUid + "/modules").push(
    child.key).catch(e => alert(e.message));
22              firebase.database().ref("modules/" + child.key + "/lecturer").
    push(userUid).catch(e => alert(e.message));
23          });
24      });
25  }).catch(e => alert(e.message));
```

When the *email* and *password* are sent to Firebase Authentication to create a new account for the lecturer, the *first name*, *last name*, *position*, *faculty* and *email*, are entered under the *lecturer* node and grouped by the lecturer account's *user uid*. Under the *users* node, the role of the account (Lecturer), module set and the *programme* are entered, grouped by *user uid*. With the programme name, the system will fetch all the *module ids* of the modules under the module

set selected and store them in the *modules* node under *user uid* node, which is also under *users* node. The system will also register the user uid of the lecturers under *lecturers* node of each *modules* node.

Module Component

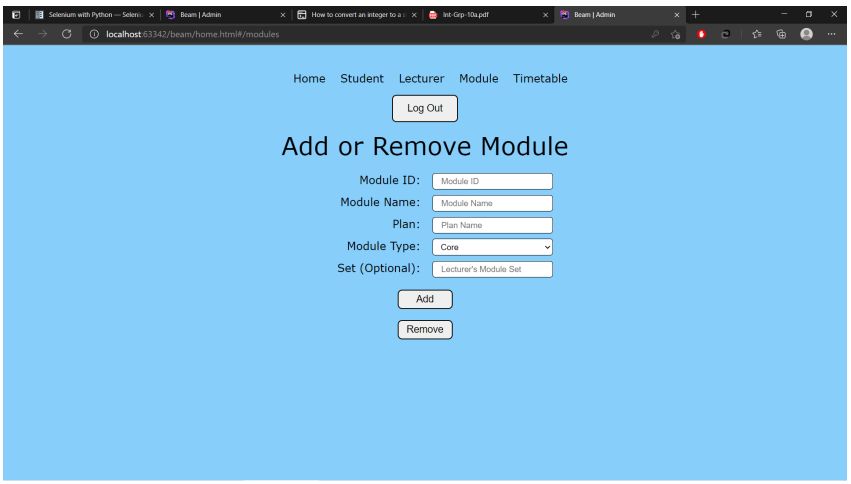

Figure 0.1.5: *This is the Add or Remove Module interface of the website*

```
function writeModulesData() {
    var mID = document.getElementById("mID").value;
    var mName = document.getElementById("mName").value;
    var mPlan = document.getElementById("mPlan").value;
    var moduleType = document.getElementById("moduleType").value;
    var mSet = document.getElementById("mSet").value;
    if (mID === "" || mName === "" || mPlan === "" || moduleType === "") {
        alert("Please fill in all fields");
        return false;
    }
    var modules = {
        name: mName
    }
    firebase.database().ref("modules/" + mID).set(modules).catch(e => alert(e.
    message));
    firebase.database().ref("plan/" + mPlan + "/" + moduleType + "/" + mID).set
    (modules).catch(e => alert(e.message));
    if (!(mSet ===""")){
        firebase.database().ref("plan/" + mPlan + "/" + mSet + "/" + mID).set(
    modules).catch(e => alert(e.message));
    }
    document.getElementById("mID").value = '';
    document.getElementById("mName").value = '';
    document.getElementById("mPlan").value = '';
    document.getElementById("moduleType").value = '';
    document.getElementById("mSet").value = '';
};
```

To add module details to the database, the system will enter the *module names* under the *module id*, which is under the *modules* node. By the names of the *academic plan* which the *modules* fall under, the *modules ids* of these *modules*, grouped by *module type*, will be placed under the *plan name*, which is under the *plan* node. For the modules taught by a lecturer, the *module ids* will be stored under *module set*, under the modules' *plan name*, which is also under the *plan* node.

```
function removeModulesData() {
```

```
    var mID = document.getElementById("mID").value;
    var mPlan = document.getElementById("mPlan").value;
    var moduleType = document.getElementById("moduleType").value;
    firebase.database().ref("modules/" + mID).remove().catch(e => alert(e.
    message));
    firebase.database().ref("plan/" + mPlan + "/" + moduleType + "/" + mID).
    remove().catch(e => alert(e.message));
    document.getElementById("mID").value = '';
    document.getElementById("mName").value = '';
    document.getElementById("mPlan").value = '';
    document.getElementById("moduleType").value = '';
    document.getElementById("mSet").value = '';
};
```

To delete module details from the database, fetch the *module id* from the website and query the modules and remove the data under the matched *module id*. The system can remove the same module ids from the database directory: *plan/plan name/module type*.

Timetable Component

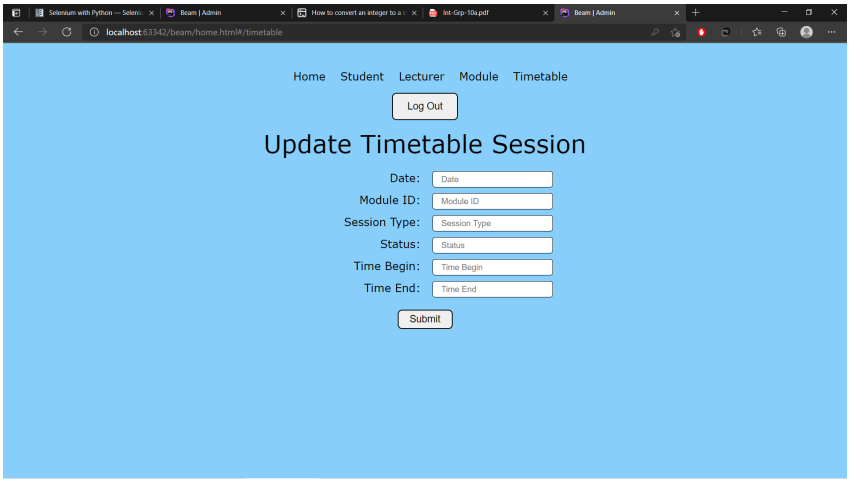

Figure 0.1.6: *This is the Update Timetable Session interface of the website*

```
function writeTimetableData() {
    var date = document.getElementById("date").value;
    var module_id = document.getElementById("module_id").value;
    var sessionType = document.getElementById("sessionType").value;
    var status = document.getElementById("status").value;
    var time_begin = document.getElementById("time_begin").value;
    var time_end = document.getElementById("time_end").value;
    if (date === "" || module_id === "" || sessionType === "" || status === ""
    || time_begin === "" || time_end === "") {
        alert("Please fill in all fields");
        return false;
    }
    var timetable = {
        module_id: module_id,
        session_id: "temp",
        sessionType: sessionType,
        status: status,
        time_begin: time_begin,
        time_end: time_end
    };
    var newRef = firebase.database().ref("timetable/" + date + "/" + timetable.
    module_id ).push();
```

```
21      timetable.session_id = newRef.key;
22      newRef.set(timetable).catch(e => alert(e.message));
23      firebase.database().ref("module_session/" + timetable.module_id + "/" +
        newRef.key).set(timetable).catch(e => alert(e.message));
```

The system sends the *date*, *module_id*, *session type*, *session status* (Opened/Closed), *time_begin*, and *time_end* to the database. Under the *timetable* node, the module details will be entered, grouped by the *date*. Under the *modules_session* node, the module details will be entered, grouped by the *module_id*. This groups the sessions of a module under a *module_id*.

c. Test Results

The changes reflected in the database due to the Python script will be shown below in screenshots

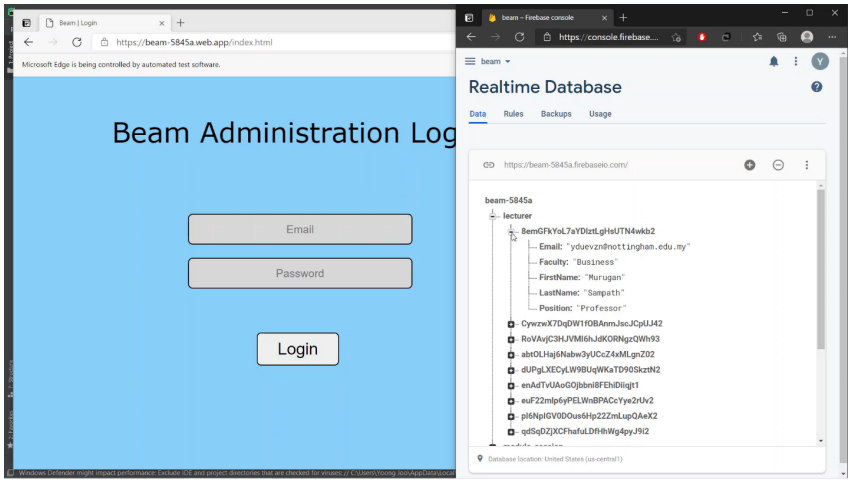

Figure 0.1.7: *Changes made to the lecturer node*

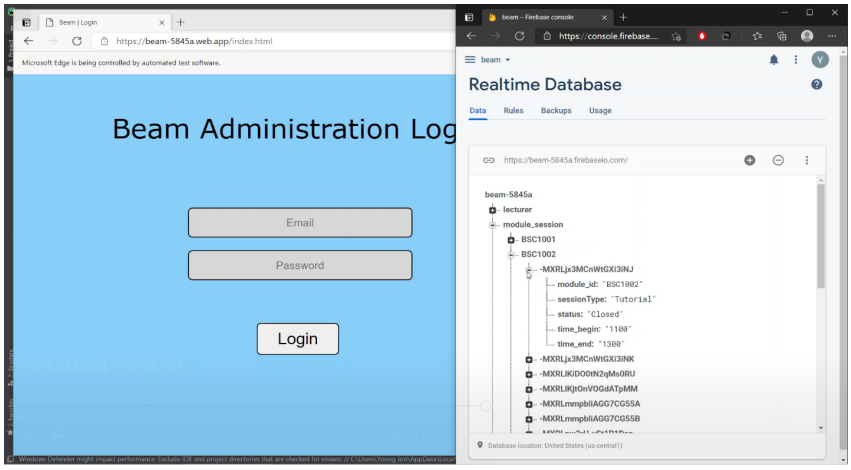

Figure 0.1.8: *Changes made to the module_session node*

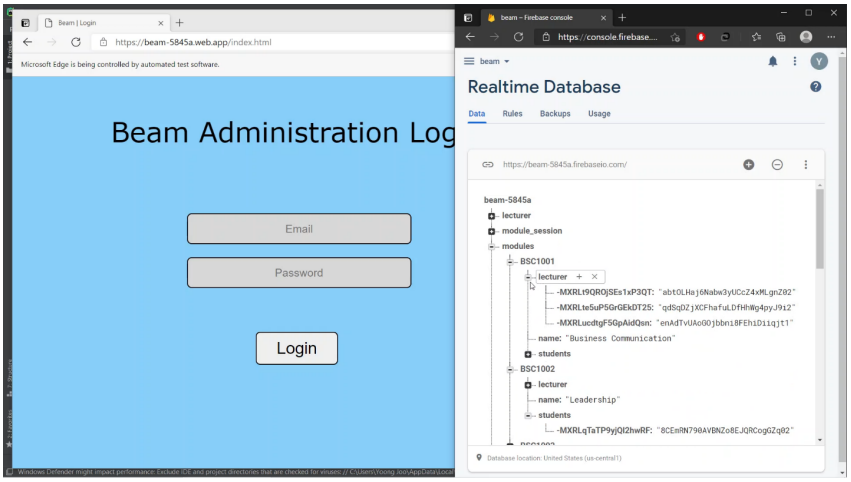

Figure 0.1.9: *Changes made to the modules node*

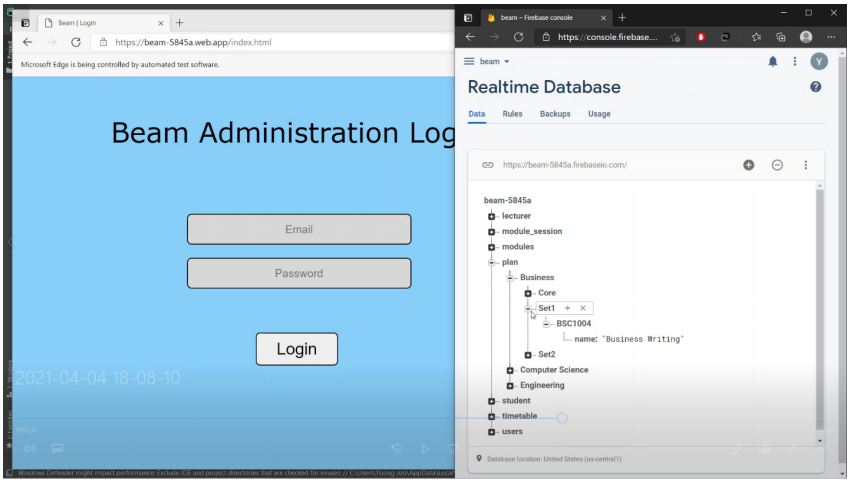

Figure 0.1.10: *Changes made to the plan node*

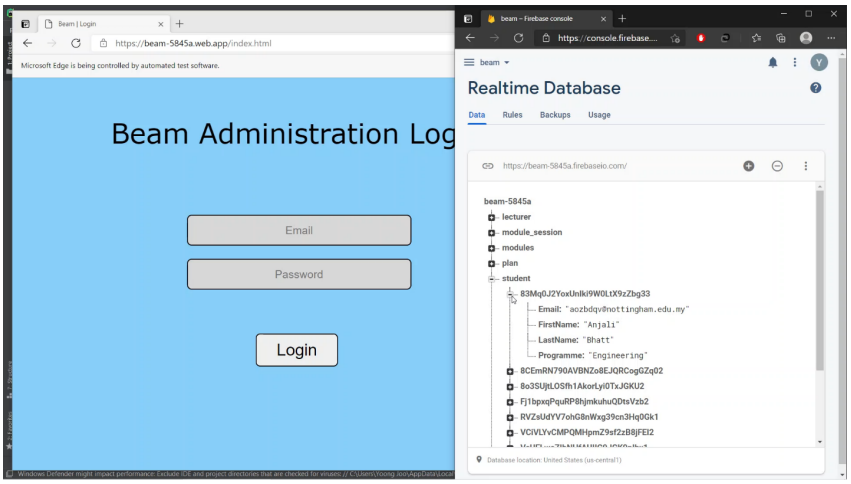

Figure 0.1.11: *Changes made to the student node*

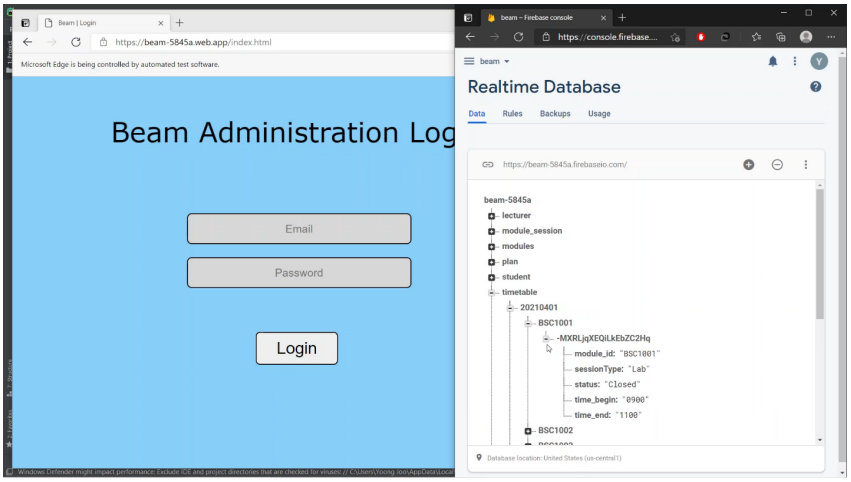

Figure 0.1.12: *Changes made to the timetable node*

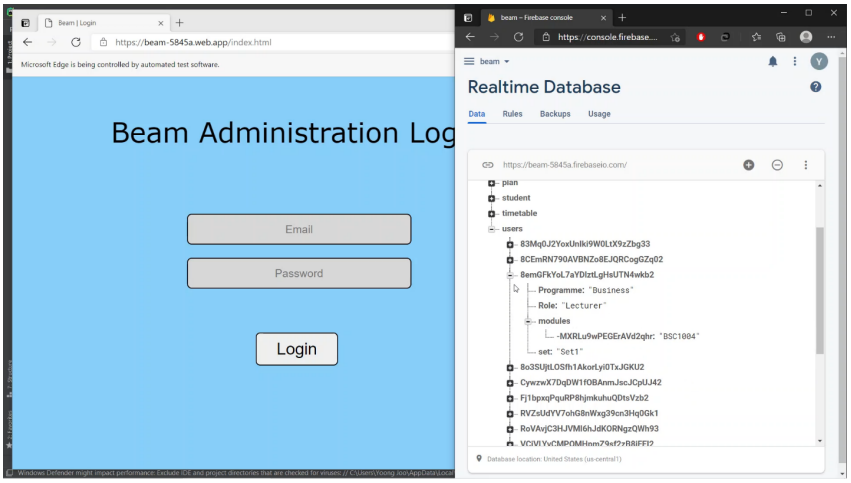

Figure 0.1.13: *Changes made to the users node*

0.2 Application