



# TEAM ROCKERS PROJECT

Submitted to Dr. Peter Bodorik  
Faculty of Computer Science  
Dalhousie University

Submitted by Team Rockers  
Date - 29 July, 2019

NAME	BANNER ID
AMAN ARYA	B00816348
ANIRUDDHA CHITLEY	B00808320
JESSICA CASTELINO	B00804805
NIKHIL TYAGI	B00809791
NITISH BHARDWAJ	B00811535
SAMARTH RAVAL	B00812673

CONTENTS

Project Overview:.....	4
Project Plan:.....	4
Resources Used:.....	6
Hardware & Software .....	6
Tools.....	6
Libraries.....	6
Languages and Framework:.....	6
CSP Services Used: .....	7
Functionalities provided over and above the requirements:.....	7
Identification of any code and Tools used:.....	7
Overview of frontend:.....	9
Sample Screenshots of Slack and Trello.....	14
Deficiencies:.....	20
Implementation: .....	20
Experimentation Results:.....	21
References: .....	33
Appendix: .....	34
Client code .....	34
Homepage.....	34
MBR Portal.....	34
Employer Portal .....	39
Real Estate Portal.....	42
Insurance Portal.....	45
Web Service Code .....	47
Employer Service .....	47
Insurance Service.....	50
MBR Broker Service .....	52
Real Estate Broker Web Service .....	54

**LIST OF FIGURES**

Figure 1: Tools used for the efficient management of the Project [1] .....	5
Figure 2: Overall Project plan [1] .....	5
Figure 3: XMLHttpRequest object for sending data to web services and fetching the response without page reload .....	8
Figure 4: Screenshot of MBR Portal .....	10
Figure 5: Screenshot to view status of MBR application .....	10
Figure 6: Screenshot of Real Estate Application Form to request an appraisal by the Employee .....	11
Figure 7: Screenshot of Real Estate appraisal form used by the broker.....	11
Figure 8: Screenshot of the Employer portal where an employee can log in.....	12
Figure 9: Screenshot of the page where the user can enter the MortID and URL of MBR to submit to the employer .....	13
Figure 10: Status check at Insurance Portal.....	14
Figure 11: Screenshot 1 for Slack.....	15
Figure 12: Screenshot 2 for Slack.....	15
Figure 13: Screenshot 3 for Slack.....	16
Figure 14: Screenshot 4 for Slack .....	16
Figure 15: Screenshot 5 for Slack.....	17
Figure 16: Screenshot 6 for Slack.....	18
Figure 17: Screenshot 1 for Trello.....	19
Figure 18: Screenshot 2 for Trello.....	19
Figure 19: Screenshot 3 for Trello.....	20
Figure 20: Screenshot of HTTP portals.....	20
Figure 12: homepage.ejs (Provides links to various portals of the projects).....	34
Figure 13: mbr-form.ejs (Form for mortgage application) .....	34
Figure 14: mbr-form.ejs (contd.).....	35
Figure 15: mbr-form.ejs (contd.).....	35
Figure 16: mbr-form.ejs (contd.).....	36
Figure 17: mbr-status.ejs (Portal to check the status of the mortgage application).....	36
Figure 18: mbr-status.ejs (contd.).....	37
Figure 19: mbr-status.ejs (contd.).....	37
Figure 20: mbr-status.ejs (contd.).....	38
Figure 21: mbr-status.ejs (contd.).....	38
Figure 22: employer-login.ejs (Employer portal) .....	39

Figure 23: employer-login.ejs (contd.).....	39
Figure 24: employer-login.ejs (contd.).....	40
Figure 25: employer-login.ejs (contd.).....	40
Figure 26: employer-login.ejs (contd.).....	41
Figure 27: employer-login.ejs (contd.).....	41
Figure 28: realestate-app.ejs (Real Estate appraisal application).....	42
Figure 29: realestate-app.ejs (contd.).....	42
Figure 30: realestate-app.ejs (contd.).....	43
Figure 31: realestate-update.ejs (Appraisal Evaluation).....	43
Figure 32: realestate-update.ejs (contd.) .....	44
Figure 33: realestate-update.ejs (contd.) .....	44
Figure 34: insinc.ejs (Insurance Portal) .....	45
Figure 35: insinc.ejs (contd.) .....	45
Figure 36: insinc.ejs (contd.) .....	46
Figure 37: insinc.ejs (contd.) .....	46
Figure 38: logging.js .....	47
Figure 39: signup.js .....	48
Figure 40: application.js .....	49
Figure 41: borrower-info.js .....	49
Figure 42: logged-in.js.....	50
Figure 43: getquote.js .....	50
Figure 44: quote.js .....	51
Figure 45: logging.js .....	51
Figure 46: logging.js .....	52
Figure 47: employment.js .....	52
Figure 48: insurance.js .....	53
Figure 49: application.js .....	53
Figure 50: status.js .....	53
Figure 51: logging.js .....	54
Figure 52: complete-info.js .....	55
Figure 53: request.js .....	55

## PROJECT OVERVIEW:

The purpose of this project is to create four different portals for a customer looking to purchase a house and is applying for a mortgage. The whole process is supposed to be automated. The four main portals that need to be developed are MBR, EMP, INSinc, and RE.

MBR is a mortgage broker portal to apply for a mortgage. EMP is the employer portal where the employee who wants to apply for the mortgage needs to log in and provide the MortID and URL of the MBR. RE portal has two pages. One to raise a request for an appraisal by the customer and another for the Broker to enter details of the appraisal. Finally, the INSinc portal is used to verify the details of the insurance. In the end, the customer can download a copy of the mortgage document from the MBR portal.

The primary requirement is to make the whole process automated. It is also ensured that once the customer raises a request for a mortgage on the MBR portal then the order of the task execution on the portal does not matter.

Sails.js app is used to build the web service present on the server and the client-side application. Then, the web service is moved to Azure Cloud. Additionally, for the purpose of testing the web service parameters and output, Apache JMeter application has been used.

## PROJECT PLAN:

For the efficient management of the project, the team brainstormed and finalized a set of tools to maintain the different deliverables. Figure 1 shows the list of tools used in the management of this assignment [1].

The project is planned to be divided into two major parts i.e. web-service and client-side code. **Group 9: Team Rockers**, having six members, were divided into two internal sub-teams consisting of three members each. The design of both parts was designed by individual teams and finalized as a group. In the next phase of development, the user portal, web services were developed and checked-in to the GitHub repository. The regular meetings and project management tool: Trello helped the group to flag the object based on their priority. It helped the team to identify and mitigate the risk associated with each object [1].

With the completion of each object, it was continuously tested by another group member who was not associated with the development of that object. This helped to identify more bugs in the objects and coherent development and testing. After the unit testing of each object, all the objects were tested i.e. the integration testing of the objects was performed. Eventually, after the reduction of bugs, the code was deployed on Azure cloud. Figure 2 depicts the overall project plan for Project [1].

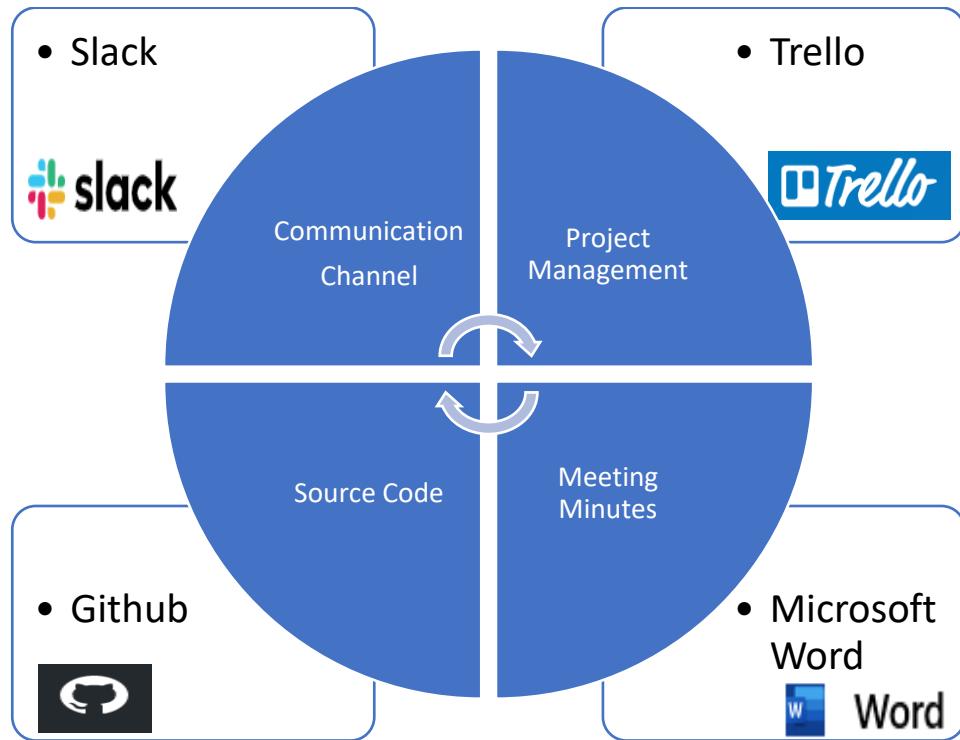


Figure 1: Tools used for the efficient management of the Project [1]

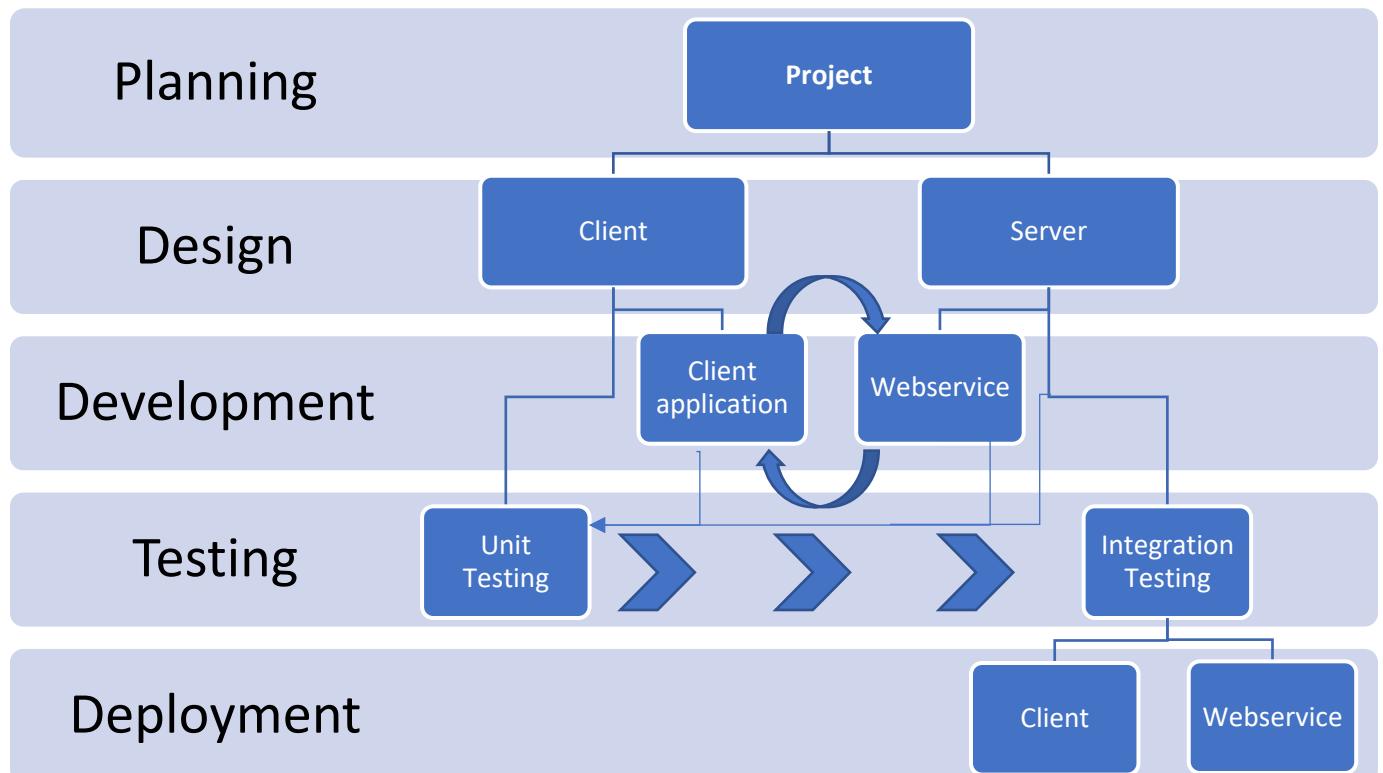


Figure 2: Overall Project plan [1]

## RESOURCES USED:

Various libraries, software, tools, languages and frameworks are used in the execution of this assignment. This section contains the list of resources used in the completion of the assignment.

### HARDWARE & SOFTWARE

All nodes must have[2] :

- Linux kernel version 3.10 or higher
- CS Docker Engine 1.13, or EE Daemon 17.03 and higher
- 8.00 GB of RAM for manager nodes or nodes running DTR
- 4.00 GB of RAM for worker nodes
- 3.00 GB of available disk space
- A static IP address

### TOOLS

Load Testing tools: JMeter

API Testing tool: Postman

Load Balancer: NGINX

### LIBRARIES

**XMLHttpRequest** – This library helps to retrieve the data from a URL without performing a full-page refresh. Thus, the web page can update a part of a page without disrupting what the user is doing [1].

**Bootstrap**: It is a free and responsive CSS framework used to create highly responsive applications. Bootstrap allows creating the application in a shorter duration of time. This ensures better usage of time and resources [1].

### LANGUAGES AND FRAMEWORK:

**JavaScript** - JavaScript® is a lightweight, interpreted, object-oriented scripting language with first-class functions. It is a prototype-based, multi-paradigm scripting language that is dynamic, and supports object-oriented, imperative, and functional programming styles [1].

**HTML** - Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. HTML elements are the building blocks of HTML pages. [2] With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. It provides a means to create structured documents [1].

**Node.js** - Node.js is an open-source, cross-platform JavaScript runtime engine to develop server-side applications. It helps the developers to create robust and distributed applications as a result of its event-driven architecture [1].

**Express.js** - Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications [1].

**Sails.js** - Sails.js is a Model-View-Controller (MVC) web application framework which is developed on top of the Node.js environment. As it is built on top of Node.js and Express.js, it enables applications to be built purely in JavaScript. This not only includes models, views, and controllers; but also configuration files and adapters (e.g., database) [1].

#### CSP SERVICES USED:

---

CSP (Cloud Service Provider): **Microsoft Azure** is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers. It provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems.

In this project, the IaaS software service is used to create the Virtual machine and deploy the backend web services (Sails.js) and load balancer module (Nginx) to provide the required functionality of scaling and load-balancing the client requests.

#### FUNCTIONALITIES PROVIDED OVER AND ABOVE THE REQUIREMENTS:

---

Token-based authentication is implemented on Employer portal. Instead of having to authenticate with username and password for each protected resource (like consent form to allow the employer to share details with mortgage broker), the user is given a token which offers access to a specific resource.

The homepage that lists URLs to various portals. This is created to help the TAs to effectively navigate through the various portals.

Additionally, we have created a sign-up API to allow the employee to sign up on the employer portal.

#### IDENTIFICATION OF ANY CODE AND TOOLS USED:

The following code snippet is used to generate logs by accessing the request and response parameters. The reference for this code was taken from StackOverflow [3].

```
// https://stackoverflow.com/a/19215370
var oldWrite = res.write,
    oldEnd = res.end;

var chunks = [];

res.write = function (chunk) {
  chunks.push(chunk);

  oldWrite.apply(res, arguments);
};

res.end = function (chunk) {
  if (chunk)
    chunks.push(chunk);

  var body = Buffer.concat(chunks).toString('utf8');
  line = '\n##### Response #####\n';
  line += res.statusCode + '\n' + res.message;
  line += '\n' + res._header;
  line += '\n' + body;
  fs.appendFile(logfile, line, () => { console.log('logged') });

  oldEnd.apply(res, arguments);
};
```

The code snippet shown in figure 3 is taken from MDN Web Docs [1]. The purpose of using this code is to be able to send an asynchronous request to the server, fetch the response and display it on frontend without page reload. XMLHttpRequest object is used in this case to help with data exchange. The readyState property monitors the status of this object, and it calls an event handler present in the onreadystatechange property which gets called when the readyState attribute changes. This code is used in the JavaScript functions of the pages used in the Employee portal after suitable modifications are made to validate the inputs and pass them as parameters for the web service call.

## Example ↗

```
1 | var xhr = new XMLHttpRequest(),
2 |   method = "GET",
3 |   url = "https://developer.mozilla.org/";
4 |
5 | xhr.open(method, url, true);
6 | xhr.onreadystatechange = function () {
7 |   if(xhr.readyState === 4 && xhr.status === 200) {
8 |     console.log(xhr.responseText);
9 |   }
10 | };
11 | xhr.send();
```

Figure 3: XMLHttpRequest object for sending data to web services and fetching the response without page reload

TRELLO BOARD: <https://trello.com/b/xYAXiGI3/project>

GITHUB: Frontend → <https://github.com/amanarya0/cloud-proj-portals>

Backend

MBR → <https://github.com/nikhilatyagi1991/mbr-portal>

EMP → <https://github.com/nikhilatyagi1991/employer-backend>

RE → <https://github.com/nikhilatyagi1991/real-estate-portal>

INSInc → <https://github.com/nikhilatyagi1991/insurance-portal>

SLACK: [macs-2018.slack.com](https://macs-2018.slack.com)

Azure: Backend

MBR → <http://rockers-mbr.eastus2.cloudapp.azure.com/>

EMP → <http://rockers-employer.centralus.cloudapp.azure.com/>

RE → <http://rockers-realestate.southcentralus.cloudapp.azure.com/>

INSInc → <http://rockers-realestate.southcentralus.cloudapp.azure.com/>

Heroku: Frontend → <https://project-portals.herokuapp.com/>

Meeting minute files:

1. [https://macs-2018.slack.com/files/UD31WAJ76/FLULZGL2Y/15th-july-2019\\_mom.pdf](https://macs-2018.slack.com/files/UD31WAJ76/FLULZGL2Y/15th-july-2019_mom.pdf)
2. [https://macs-2018.slack.com/files/UD31WAJ76/FLNQRHSGZ/22nd-july-2019\\_mom.pdf](https://macs-2018.slack.com/files/UD31WAJ76/FLNQRHSGZ/22nd-july-2019_mom.pdf)
3. [https://macs-2018.slack.com/files/UD31WAJ76/FLX147W0P/27th-july-2019\\_mom.pdf](https://macs-2018.slack.com/files/UD31WAJ76/FLX147W0P/27th-july-2019_mom.pdf)
4. [https://macs-2018.slack.com/files/UD31WAJ76/FLX9SU5TQ/28th-july-2019\\_mom.pdf](https://macs-2018.slack.com/files/UD31WAJ76/FLX9SU5TQ/28th-july-2019_mom.pdf)

## OVERVIEW OF FRONTEND:

There are four portals developed as part of this project. Each portal contains various pages. The description of each portal is given below:

### MBR Mortgage Broker portal

MBR is a portal provided by the broker to let the customer enter the details of the mortgage. This helps the broker to understand that the customer is interested in which mortgage. Customer can enter details like name, contact number, address, employee id, employer name, Mortgage Value and M1sID. If the details are correct, the customer receives a MortID that can have to be used on other portals. Customer also receives information of the portal where the status of the application can be checked. Figure 4 shows the screenshot of the MBR portal. Figure 5 shows the screenshot to view the status of the MBR application.

The screenshot shows a web browser window with the URL <https://project-portals.herokuapp.com/mbr>. The page has a light gray header bar with a search icon. Below it is a yellow banner with the text "Welcome to MBR". The main content area is titled "Mortgage Request Form" in bold black text. It contains seven input fields: "Name", "Phone number", "Address", "Employee Id", "Employer Name", "Mortgage Value", and "M1SID". A teal "Submit" button is located at the bottom of the form.

Figure 4: Screenshot of MBR Portal

The screenshot shows a web browser window with the URL <https://project-portals.herokuapp.com/mbrstatus>. The page has a light gray header bar with a search icon. Below it is a yellow banner with the text "Welcome to MBR". The main content area is titled "Retrieve Application Status" in bold black text. It contains one input field labeled "Mortgage Id" and a teal "Login" button.

Figure 5: Screenshot to view status of MBR application

#### Real Estate Broker portal

Real estate portal has two pages named application and an update appraisal page. In the real estate application page, there are three fields for the input Mortgage unique ID, Mortgage value and Name. Validation has been done on the backend, the appropriate message has been displayed if fields are blank or invalid data has been given as input. If all the validations are satisfied, the application page will display a successful message.

The update appraisal page is shown in the figure. In this page, it has three different fields named appraisal value, mortgage value and comment. All this field must have value before requesting it otherwise it will throw an exception. If all the values are valid it will submit the updated appraisal form. A successful message will also be displayed on the screen. Figure 6 shows the screenshot of the Real Estate Application form to request an appraisal. Figure 7 shows the screenshot of Real Estate appraisal form used by the broker.

A screenshot of a web browser showing a form titled "Real Estate Application Form". The URL in the address bar is <https://project-portals.herokuapp.com/re/realestate-app>. The form contains three input fields: "M1sID", "Mort ID", and "Name", followed by a blue "Submit" button. The background features a yellow header bar with the text "Welcome to RE".

**Figure 6: Screenshot of Real Estate Application Form to request an appraisal by the Employee**

A screenshot of a web browser showing a form titled "Real Estate Appraisal Form". The URL in the address bar is <https://project-portals.herokuapp.com/re/realestate-update-appraisal>. The form contains three input fields: "Appraisal Value", "Mort ID", and "Comments", followed by a blue "Submit" button. The background features a yellow header bar with the text "Welcome to RE".

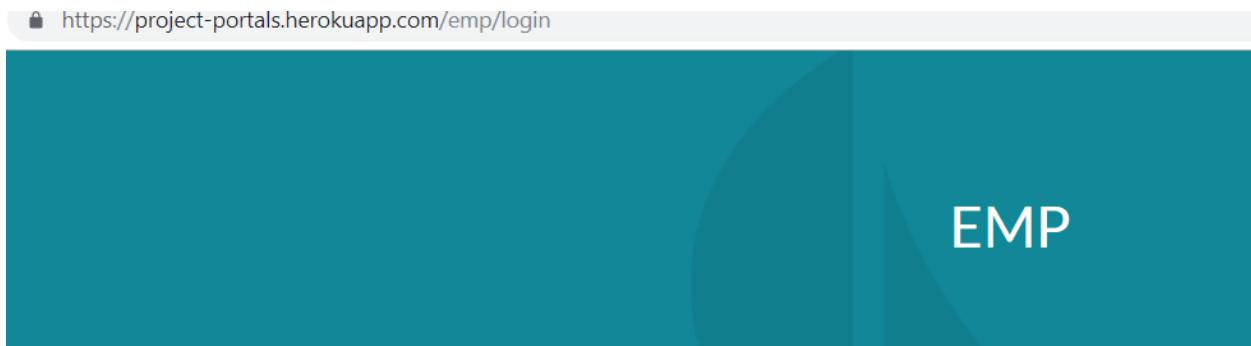
**Figure 7: Screenshot of Real Estate appraisal form used by the broker**

### Employer portal

The Employer Portal has a login page as shown in the figure which takes as input two fields – the employee ID and the password. Appropriate JavaScript validations are present to ensure that the employee ID and password are not blank. In case validations are not satisfied, appropriate error messages are displayed on the screen while blocking form submission. If all validations are satisfied, the

employee credentials are passed to a web service that verifies the credentials and redirects the employee to a consent form on successful authentication. If authentication fails, an error message is displayed on the screen.

The consent form, as shown in the figure, takes as input two fields – the mortgage application ID (as provided by the broker) and the web service endpoint URL of the Broker, along with consent in the form of a checkbox. Appropriate JavaScript validations are present to ensure that the application number and web service endpoint are not blank. It is also mandatory for the customer to give consent by clicking on the checkbox. In case validations are not satisfied, appropriate error messages are displayed on the screen while blocking form submission. If all validations are satisfied, a web service call is made to the URL for the application number provided by the customer and the required employer details are sent. On successful submission, the customer is notified through a message on the screen. Figure 8 shows the screenshot of the Employer portal where the user can log in. Figure 9 is the screenshot of the page where the user can enter the MortID and URL of MBR to submit to the employer.

A screenshot of a login form titled "LOGIN". The title is centered at the top in a bold, black, sans-serif font. Below the title, there are two input fields: one for "Employee ID" and one for "Password", both represented by empty rectangular boxes. At the bottom right of the form is a blue rectangular button labeled "Submit" in white text.

**Figure 8: Screenshot of the Employer portal where an employee can log in**

## Cloud Computing – CSCI5409

The screenshot shows a web browser window with a teal header bar. The header bar contains the text "Not secure | https://project-portals.herokuapp.com/emp/login". Below the header is a large teal rectangular area with the letters "EMP" in white. Underneath this is a white rectangular form area with a title "CONSENT FORM". The form contains fields for "MortID:" and "URL to submit the information:", each with an associated input field. There is also a checkbox labeled "I, hereby, give consent to 'EMP' to share my information with the mortgage broker." followed by a blue "Agree" button.

**Figure 9: Screenshot of the page where the user can enter the MortID and URL of MBR to submit to the employer**

### Insurance portal

The insurance portal constructs an insurance quote once it receives a request from the real estate broker. It then sends the quote to the MBR broker portal. Additional functionality has been added to

## Cloud Computing – CSCI5409

this portal that allows the user to view the quote to get the state of their application as shown in figure 10.

The screenshot shows a web browser window with the title 'INSinc Insurance Portal'. The URL in the address bar is <https://project-portals.herokuapp.com/insinc>. The page has a yellow header bar with the portal's name. Below it, there is a form field labeled 'Enter M1SID to get your application status:' with the value '11' entered. A blue button labeled 'Get Application' is next to it. Below the form, several application details are listed in a table-like format:

ID:	5d3e8af8c7e01bded8a949d5
M1SID:	11
MORTID:	4f3ec715-132a-5a4a-90b0-e4fc28ca01ff
Insured Value:	8962
Appraisal Value:	9000
Deductible:	896.2

Figure 10: Status check at Insurance Portal

### SAMPLE SCREENSHOTS OF SLACK AND TRELLO

Figure 11 to 16 shows the screenshot of the Slack channel. Please note that as Slack is used as the communication channel, so only the sample screenshots are provided in this report. However, complete communication can be viewed at Slack.

## Cloud Computing – CSCI5409

**#cloudcomputing-s2019**

☆ | 6 | 2 | Add a topic

Saturday, July 27th

**Nik** 5:02 PM  
<https://github.com/nikhilatyagi1991/real-estate-portal>  
<https://github.com/nikhilatyagi1991/mbr-portal>  
<https://github.com/nikhilatyagi1991/insurance-portal>  
<https://github.com/nikhilatyagi1991/employer-backend>

All backend repos

**GitHub**  
**nikhilatyagi1991/real-estate-portal**  
Contribute to nikhilatyagi1991/real-estate-portal development by creating an account on GitHub.

**GitHub**  
**nikhilatyagi1991/mbr-portal**  
Contribute to nikhilatyagi1991/mbr-portal development by creating an account on GitHub.

**GitHub**  
**nikhilatyagi1991/insurance-portal**  
Contribute to nikhilatyagi1991/insurance-portal development by creating an account on GitHub.

**GitHub**  
**nikhilatyagi1991/employer-backend**  
Employer Backend for cloud assignment 5. Contribute to nikhilatyagi1991/employer-backend development by creating an account on GitHub.

**Nitish** 5:02 PM  
Awesome

**Nik** 5:04 PM  
pushed new changes.....all portals are working as per requirement



Figure 11: Screenshot 1 for Slack

**#cloudcomputing-s2019**

☆ | 6 | 2 | Add a topic

Saturday, July 27th

**Aniruddha Chitley** 5:04 PM  
<https://github.com/amanarya0/cloud-proj-portals>

**GitHub**  
**amanarya0/cloud-proj-portals**  
Contribute to amanarya0/cloud-proj-portals development by creating an account on GitHub.

**Nitish** 5:05 PM  
<http://localhost:1331/insurance/quote?m1sid=823743>

**Nik** 5:05 PM  
<https://www.getpostman.com/collections/53b0fa4b09ea440b2888>

Import this collection in postman.....it contains all requests info.

**Nitish** 5:05 PM  
There are just 3 apis in the collection! For create employee, we don't need a page! And for the other two APIs the pages exist!

**Aman Arya** 5:06 PM  
login page I think is their already.  
you guys created it last time  
application page is new.  
What happens on application page? They enter the mortgage number and URL and submit it, right?

**Nitish** 5:07 PM  
Agreed

But @Jessica Castelino and I have one question. What happens on application page? They enter the mortgage number and URL and submit it, right?

**Aman Arya** 5:08 PM  
That is already present!

**Nitish** 5:09 PM  
@Samarth Raval has shared the following link:



Figure 12: Screenshot 2 for Slack

## Cloud Computing – CSCI5409

**#cloudcomputing-s2019**

☆ | 8 6 | 2 | Add a topic

Nitish 5:09 PM Saturday, July 27th

@Samarth Raval has shared the following link:  
<https://project-portals.herokuapp.com/login>

this is the frontend link which is deployed

github code link for this is <https://github.com/amanarya0/cloud-proj-portals>

Aniruddha Chitley 5:11 PM I'm not able to sign up for Employee

Nitish 5:11 PM This is weird!

Aniruddha Chitley 5:11 PM This is the end point I used: <https://project-portals.herokuapp.com/account/signup>

Nitish 5:11 PM Please share the url

Great!

Aniruddha Chitley 5:11 PM Body:

```
{  
    "employeeId": 1234,  
    "name": "abcd",  
    "password": "1234",  
    "confirmPwd": "1234",  
    "address": "abcdefg"  
}
```

Nitish 5:11 PM Let me try!

Yes, it's an issue!!!

Good catch @Aniruddha Chitley

Figure 13: Screenshot 3 for Slack

**#cloudcomputing-s2019**

☆ | 8 6 | 2 | Add a topic

Saturday, July 27th

Mortgage Unique Id

Submit

1 reply 22 hours ago

Nitish 8:11 PM

**Observation 5:** Real Estate Page - The first two fields highlighted on the front end screenshot have wrong title. Verified the fields required for this page in the requirement document and postman. The first field should be M1sID and second should be Mortgage Unique Id (Mort ID). (edited)

2 files ▾

Realestate Application Form

Mortgage Unique Id

Mortgage Value

Nitish Bhardwaj

Submit

43,  
ee125b-19ac-549c-9f5c  
\*\*

1 reply 22 hours ago

Nitish 8:34 PM

**Observation 6:** Insurance Portal -Mortgage ID name is confusing. It should be M1sID or the one which is used consistently in the previous pages. (edited)

image.png ▾

INSinc Insurance Portal

Enter Mortgage ID to get your application status

Figure 14: Screenshot 4 for Slack

## Cloud Computing – CSCI5409

The screenshot shows a Slack channel interface. At the top, it says "cloudcomputing-s2019" with 6 members and 2 guests. There are buttons for "Add a topic", a phone icon, a gear icon, and a settings icon. The date "Saturday, July 27th" is at the top right.

**Pinned by you:** Nitish 9:44 PM  
**Observation 13:** This requirement is missing: "Finally, once the MBR receives all information, it prepares a mortgage document and lets Bob know that it is ready".

**Nitish 9:56 PM:** Status of Observation on 27th June, 10 PM:  
Observation 1, 2: Fixed  
Observation 3-13: Pending

**Priority Distribution of Observations:**  
High/Show Stoppers: Observation 12 and 13.  
Please co-ordinate with [@Aman Arya](#) if you're working on any of the observation.  
If an observation has been addressed then please make sure to comment on that observation that it is fixed. Few observations also have comments/details within their own thread.

**Asana APP 10:05 PM:** was added to this conversation by Nitish.

**Asana APP 10:11 PM:** was removed from this conversation by Nitish.

**Jessica Castelino 12:28 AM:** @Nitish You have done an amazing job with testing! **💯**  
I will make the changes soon..

**Jessica Castelino 1:09 AM:** Observation 9 and 10 is fixed! Code is pushed on Git!

**Nitish 5:07 PM:**

Figure 15: Screenshot 5 for Slack

The screenshot shows a Slack channel named "cloudcomputing-s2019". At the top, there are icons for notifications (star), messages (person), files (document), and a gear for settings. Below the header, there are two messages:

- Nitish** 11:14 PM: Testing of observations:  
Passed Observations: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 --All Passed (edited)
- Aman Arya** 12:20 PM: Can someone send the meeting minutes again here so we dont have to search for the during presentation?

Below the messages, there is a file sharing interface. It shows four PDF files:

- 15th-July-2019\_MOM.pdf (41 kB PDF)
- 22nd-July-2019\_MOM.pdf (40 kB PDF)
- 27th-July-2019\_MOM.pdf (39 kB PDF)
- 28th-July-2019\_MOM.pdf (38 kB PDF)

Below the files, there is a preview of a PDF titled "DemoCases.pdf" (731 kB PDF). The preview shows a blue and white abstract graphic design at the top, followed by the text "PROJECT DEMO CASES".

Figure 16: Screenshot 6 for Slack

Figure 17 to 19 shows the screenshot of the Trello board for project management.

## Cloud Computing – CSCI5409

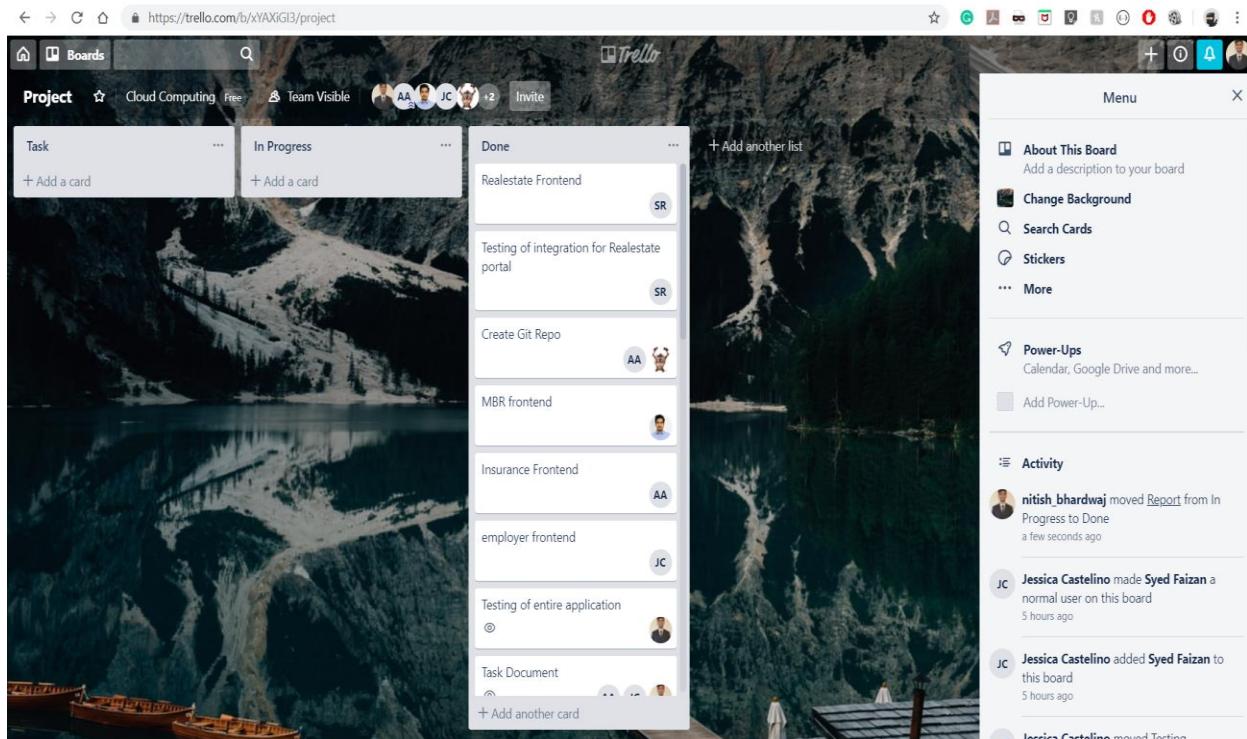


Figure 17: Screenshot 1 for Trello

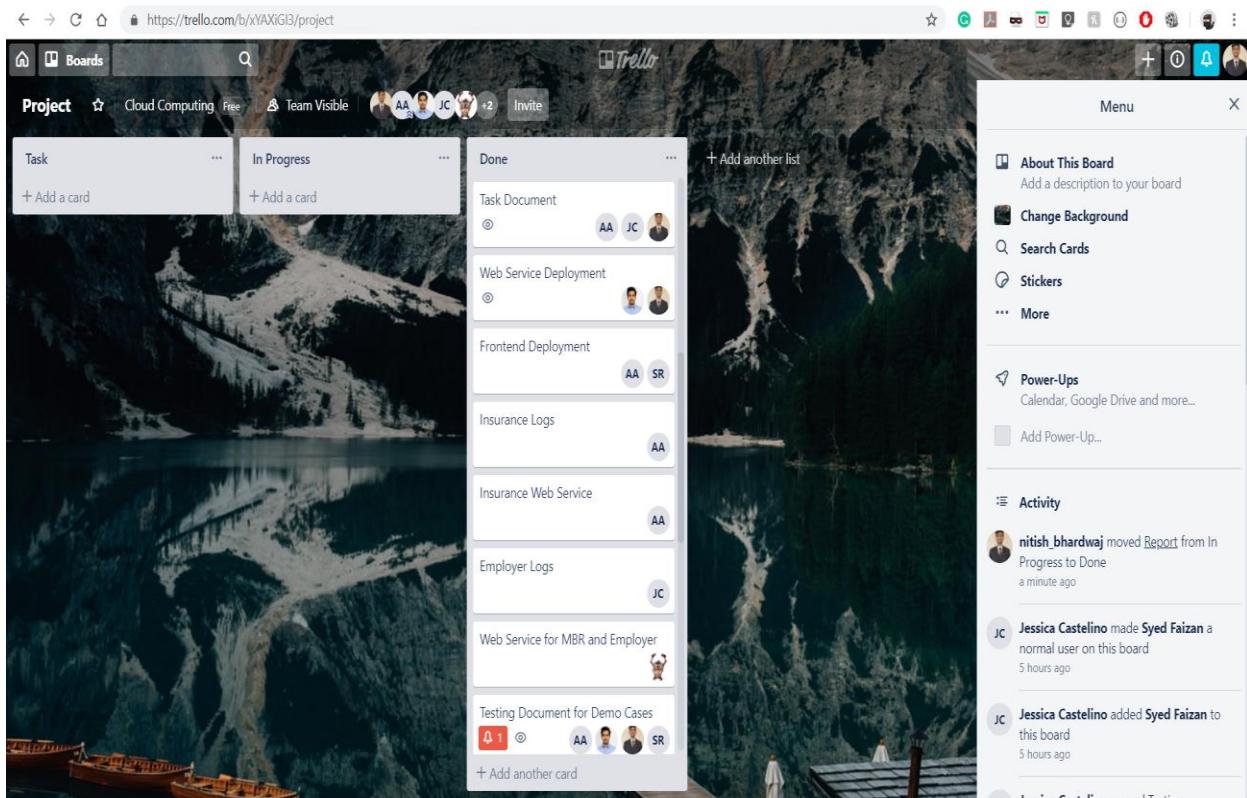


Figure 18: Screenshot 2 for Trello

## Cloud Computing – CSCI5409

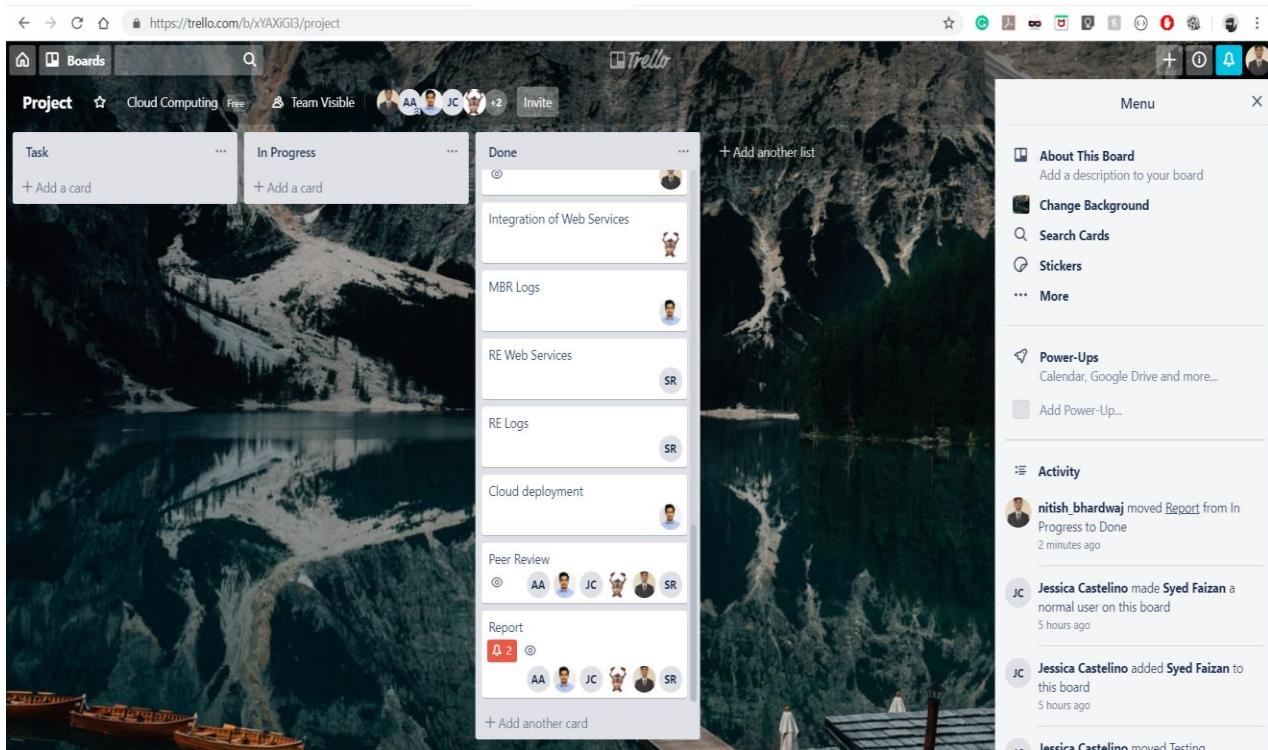


Figure 19: Screenshot 3 for Trello

### DEFICIENCIES:

The application is developed as specified in the assignment. The classwork and lab material was sufficient to complete the assignment without any major deficiencies.

The project portal runs on HTTP port instead of HTTPS. Hence, even for the testing purpose, it is requested to run the system on HTTP. As shown in figure 20, please select the “Load unsafe scripts” to perform testing on all the portals.

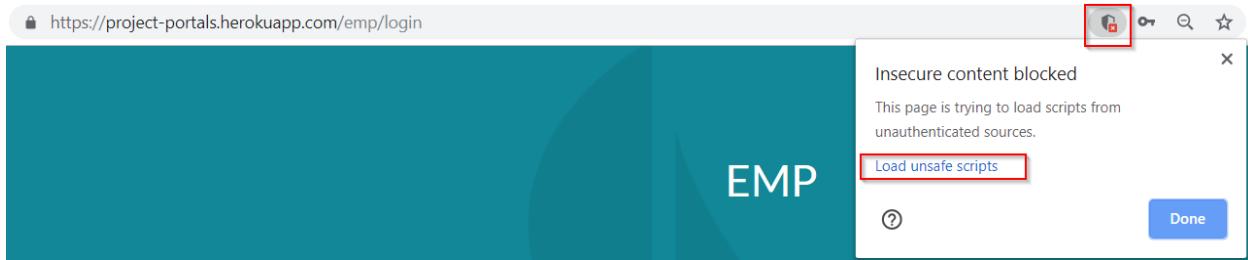


Figure 20: Screenshot of HTTP portals

However, secure communication using the authentication token has been implemented in the Employer (EMP) portal.

### IMPLEMENTATION:

To handle the bulk incoming requests from the cloud service consumer, a third-party reverse proxy server Nginx is used. It is deployed on Virtual machine and acts as the load balancer. It distributes the

incoming requests among the various web services instances running on Virtual Machine behind the Nginx module in round-robin manner so that no server is overloaded with requests.

In order to achieve the scalability, a new Virtual Machine is manually spun up and added to Nginx config, now the subsequent requests are distributed among all the initial web service instances and newly added Virtual Machine.

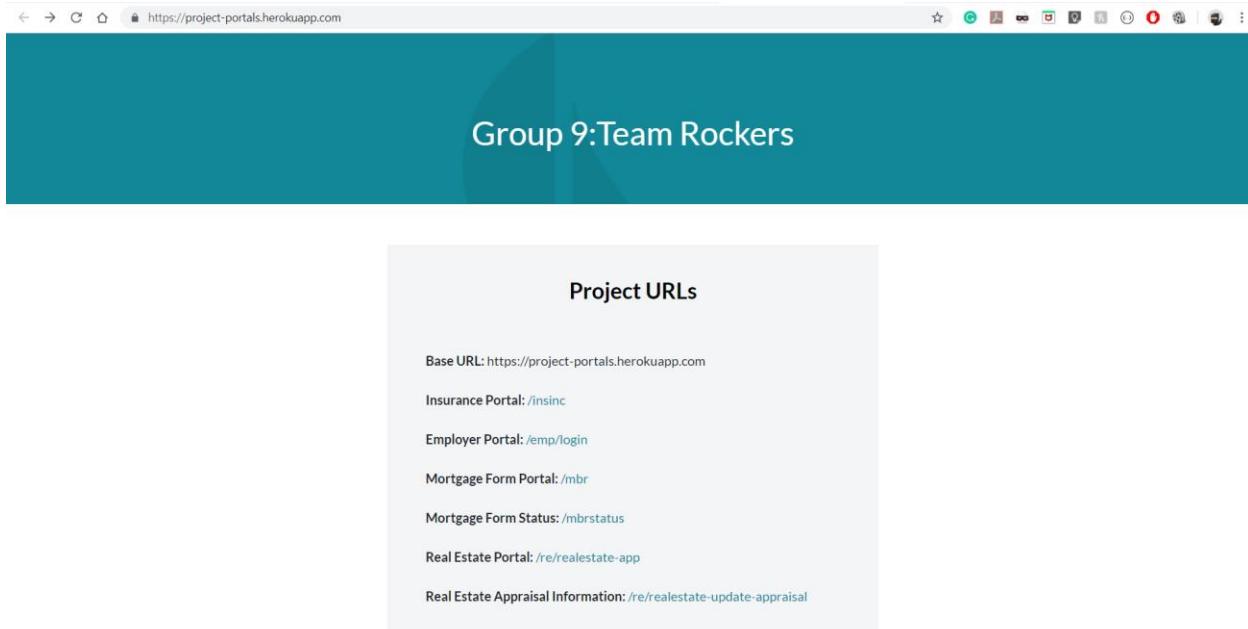
### EXPERIMENTATION RESULTS:

This section contains the result of our experiments based on the specifications mentioned the assignment requirement document.

#### Description of the experiment

**Test Case 1:** Verify the main page for the project urls.

**Tested Output:** Success



**Test Case 2:** Verify if the Mortgage Form is being redirected from the main page.

**Tested Output:** Success

https://project-portals.herokuapp.com/mbr

# Welcome to MBR

## Mortgage Request Form

Name

Phone number

Address

Employee Id

Employer Name

Mortgage Value

M1SID

Submit

The screenshot shows a web-based mortgage application form titled "Welcome to MBR". The form consists of several input fields: Name, Phone number, Address (with a text area for address details), Employee Id, Employer Name, Mortgage Value, and M1SID. A prominent blue "Submit" button is located at the bottom of the form. The URL https://project-portals.herokuapp.com/mbr is visible in the browser's address bar.

**Test Case 3:** Fill the details of the MBR and submit the form

**Tested Output:** Success

http://project-portals.herokuapp.com/mbr

## Welcome to MBR

### Mortgage Request Form

Nitish Bhardwaj

9024128919

Testing 

102

RBC

10000

112

**Submit**

**Test Case 4:** Verify that the MBR page has shown a success message and the Mortgage id:

**Tested Output:** Success

## Cloud Computing – CSCI5409

http://project-portals.herokuapp.com/mbr

Nitish Bhardwaj	
9024128919	
Testing	G
102	
RBC	
10000	
112	
<b>Submit</b>	

**Success!**  
Sucess : Application sucessfully created.  
Use the below ID to log into the Broker Portal at <https://project-portals.herokuapp.com/mbrstatus>  
Mortgage Id : 04f721e9-7c6f-57eb-92ac-4156b1feab7a

**Test Case 5:** Check the status of the application. It should be “Pending”.

**Tested Output:** Success

## Cloud Computing – CSCI5409

⚠ Not secure | <https://project-portals.herokuapp.com/mbrstatus>

# Welcome to MBR

## Your Application

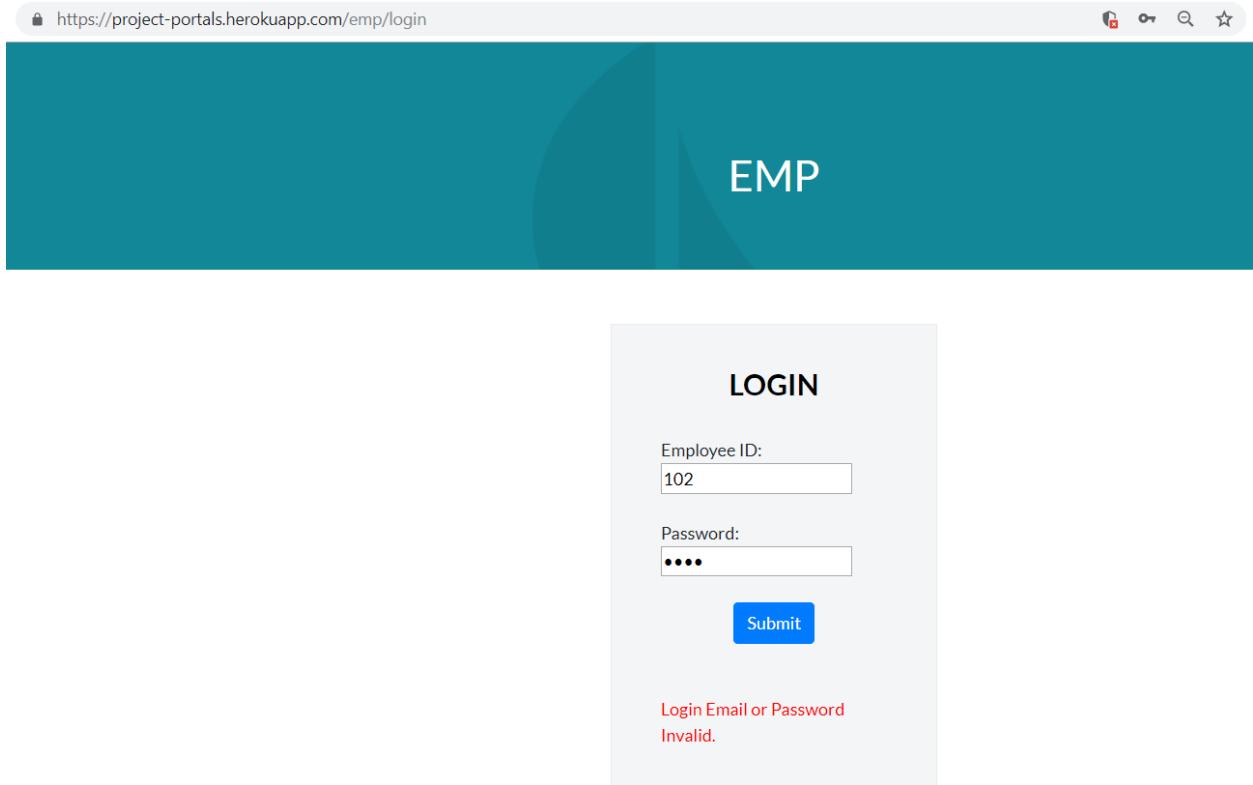
Name	Nitish Bhardwaj
Phone	9024128919
Address	Testing
Id	102
Employer Name	RBC
Mortgage Value	10000
M1Sid	112
Employee Salary	
Year of Employment	
Appraisal Value	
Insured Value	
Deductible	
Status	Pending

[Back](#)

**Test Case 6:** Login to the Employer portal. Give wrong credentials and verify that a message is shown to the user for the wrong credentials entered.

**Tested Output:** Success

## Cloud Computing – CSCI5409



**Test Case 7:** Login to the Employer portal. Give correct credentials and verify a user is redirected to a page to enter MortID and URL.

**Tested Output:** Success

## Cloud Computing – CSCI5409

⚠ Not secure | <https://project-portals.herokuapp.com/emp/login>

The screenshot shows a web browser window with a teal header bar containing the text "EMP". Below the header is a "CONSENT FORM" page. The page has fields for "MortID:" and "URL to submit the information:", both represented by empty input boxes. There is also a checkbox labeled "I, hereby, give consent to 'EMP' to share my information with the mortgage broker." followed by an "Agree" button.

**CONSENT FORM**

MortID:

URL to submit the information:

I, hereby, give consent to "EMP" to share my information with the mortgage broker.

Agree

**Test Case 8:** Login to the Employer portal. Provide correct MortID and URL to submit the information. Verify that a success message is displayed to the user.

**Tested Output:** Success

⚠ Not secure | <https://project-portals.herokuapp.com/emp/login>



**CONSENT FORM**

MortID:  
04f721e9-7c6f-57eb-92

URL to submit the information:  
<http://rockers-mbr.eastus.cloudapp.azure.com>

I, hereby, give consent to "EMP" to share my information with the mortgage broker.

**Agree**

Request sent.

**Test Case 9:** Check the status of the application. It should be “Pending”. The employee salary and year of employment should be displayed to the user.

**Tested Output:** Success

**Test Case 10:** Check the status of the application with a wrong MortID. User should receive an error that the application wasn’t found.

**Tested Output:** Success

A screenshot of a web browser showing the URL <https://project-portals.herokuapp.com/mbrstatus>. The page title is "Welcome to MBR". The main content area has a heading "Retrive Application Status" and a text input field containing "1563504293129". Below the input field is a teal "Login" button. A pink error message box at the bottom contains the text "Error! Error : Unable to find application. Please check the mortgage id".

**Test Case 11:** Go to the Real Estate portal and request for the appraisal of the property.

**Tested Output:** Success

A screenshot of a web browser showing the URL <https://project-portals.herokuapp.com/re/realestate-app>. The page title is "Welcome to RE". The main content area has a heading "Real Estate Application Form" and three input fields: the first contains "112", the second contains "04f721e9-7c6f-57eb-92ac-4156b1feab7a", and the third contains "Nitish Bhardwaj". Below the input fields is a teal "Submit" button. A green success message box at the bottom contains the text "Success! Success : Appraisal Request Successful".

**Test Case 12:** Go to the Real Estate Appraisal Information page and submit the details including the M1sID and MortID.

**Tested Output:** Success

## Cloud Computing – CSCI5409

⚠ Not secure | <https://project-portals.herokuapp.com/re/realestate-update-appraisal>

### Welcome to RE

Real Estate Appraisal Form

9000
04f721e9-7c6f-57eb-92ac-4156b1feab7a
Approved.

**Submit**

**Success!**  
Success : Appraisal Successful

**Test Case 13:** Check the status of the application. It should be “Completed”.

**Tested Output:** Success

⚠ Not secure | <https://project-portals.herokuapp.com/mbrstatus>

### Welcome to MBR

Your Application

Name	Nitish Bhardwaj
Phone	9024128919
Address	Testing
Id	102
Employer Name	RBC
Mortgage Value	10000
M1Sid	112
Employee Salary	0
Year of Employment	-1564367750088
Appraisal Value	undefined
Insured Value	8970
Deductible	897
Status	Completed

**Back**

## Cloud Computing – CSCI5409

**Test Case 14:** Go to the Insurance Portal and give the wrong M1SID. Verify that the system shows an error that the application not found.

**Tested Output:** Success

The screenshot shows a web browser with the URL <https://project-portals.herokuapp.com/insinc>. The page title is "INSinc Insurance Portal". A search bar contains the text "Enter M1SID to get your application status: 111". Below the search bar is a blue button labeled "Get Application". Underneath the button, a red error message reads "Application not found in the database!".

**Test Case 15:** Go to the Insurance Portal and give correct M1SID. Verify that the system shows an error that the application not found.

**Tested Output:** Success

The screenshot shows a web browser with the URL <https://project-portals.herokuapp.com/insinc>. The page title is "INSinc Insurance Portal". A search bar contains the text "Enter M1SID to get your application status: 112". Below the search bar is a blue button labeled "Get Application". Underneath the button, the following application details are listed:  
ID: 5d3e60610cf6a1adea28c35f  
M1SID: 112  
MORTID: 04f721e9-7c6f-57eb-92ac-4156b1feab7a  
Insured Value: 8970  
Appraisal Value: 9000  
Deductible: 897

**Test Case 16:** Check the status of the application. It should be “Completed”. An option to download the mortgage document should be shown to the user.

**Tested Output:** Success

## Cloud Computing – CSCI5409

The screenshot shows a web application titled "Welcome to MBR". The main content area is labeled "Your Application" and displays a table of user information:

Name	Nitish Bhardwaj
Phone	9024128919
Address	Testing
Id	102
Employer Name	RBC
Mortgage Value	10000
M1Sid	112
Employee Salary	0
Year of Employment	-1564367750088
Appraisal Value	undefined
Insured Value	8970
Deductible	897
Status	Completed

Below the table are two buttons: a green "Print" button and a red "Back" button.

**Test Case 18:** Based on Test Case 17, click on Print button and verify that the print option of document is displayed.

**Tested Output:** Success

The screenshot shows a "Print" dialog box on the left and the application's main content on the right. The print dialog includes fields for Destination (Save as PDF), Pages (All), Pages per sheet (1), Margins (Default), and Options (Headers and footers, Background graphics). The "Save as PDF" option is highlighted with a red box. The main content area shows the same user profile table as the previous screenshot.

**REFERENCES:**

---

- [1] N. Bhardwaj, "Assignment 6\_Group-Rockers", Halifax, 2019.
- [2] "UCP System requirements", Docker Documentation, 2019. [Online]. Available: <https://docs.docker.com/v17.09/datacenter/ucp/2.1/guides/admin/install/system-requirements/#hardware-and-software-requirements>. [Accessed: 27- Jul- 2019].
- [3] L. Perrin, "express logging response body", Stack Overflow, 2013. [Online]. Available: <https://stackoverflow.com/a/19215370>. [Accessed: 29- Jul- 2019].

## APPENDIX:

### CLIENT CODE

#### Homepage

```

File Edit Selection View Go Debug Terminal Help homepage.ejs - cloud-proj-portals-master - Visual Studio Code


# Group 9:Team Rockers



<4>Project URLs</4><br /><br />



<b>Base URL:</b> https://project-portals.herokuapp.com



<b>Insurance Portal:</b> /insinc



<b>Employer Portal:</b> /emp/login



<b>Mortgage Form Portal:</b> /mbr



<b>Mortgage Form Status:</b> /mbrstatus



<b>Real Estate Portal:</b> /re/realestate-app



<b>Real Estate Appraisal Information:</b> /re/realestate-update-appraisal


```

Figure 21: homepage.ejs (Provides links to various portals of the projects)

#### MBR PORTAL

```

File Edit Selection View Go Debug Terminal Help mbr-form.ejs - cloud-proj-portals-master - Visual Studio Code
EXPLORER employer_login.ejs mbr-form.ejs
OPEN EDITORS
CLOUD-PROJ-PORTALS-MA...
tmp
api
assets
config
node_modules
tasks
views
layouts
pages
employer_login.ejs
homepage.ejs
insinc.ejs
mbr-form.ejs
mbr-status.ejs
realestate-app.ejs
realestate-update...
.eslintrc
404.ejs
500.ejs
.editorconfig
.eslintignore
.eslintrc
OUTLINE
<script>
function applyBMR() {
    var endpoint = 'http://rockers-mbr.eastus2.cloudapp.azure.com/apply';

    var name = document.getElementById("nam").value;
    var phone = document.getElementById("phn").value;
    var address = document.getElementById("add").value;
    var employee_id = document.getElementById("emp_id").value;
    var employer_name = document.getElementById("emp_name").value;
    var mortgageValue = document.getElementById("mortVal").value;
    var msid = document.getElementById("msid").value;

    var params = { "name": name, "addr": address, "phone_no": phone, "employee_id": employee_id, "employer_na
    var xhr = new XMLHttpRequest();
    xhr.open("POST", endpoint, true);

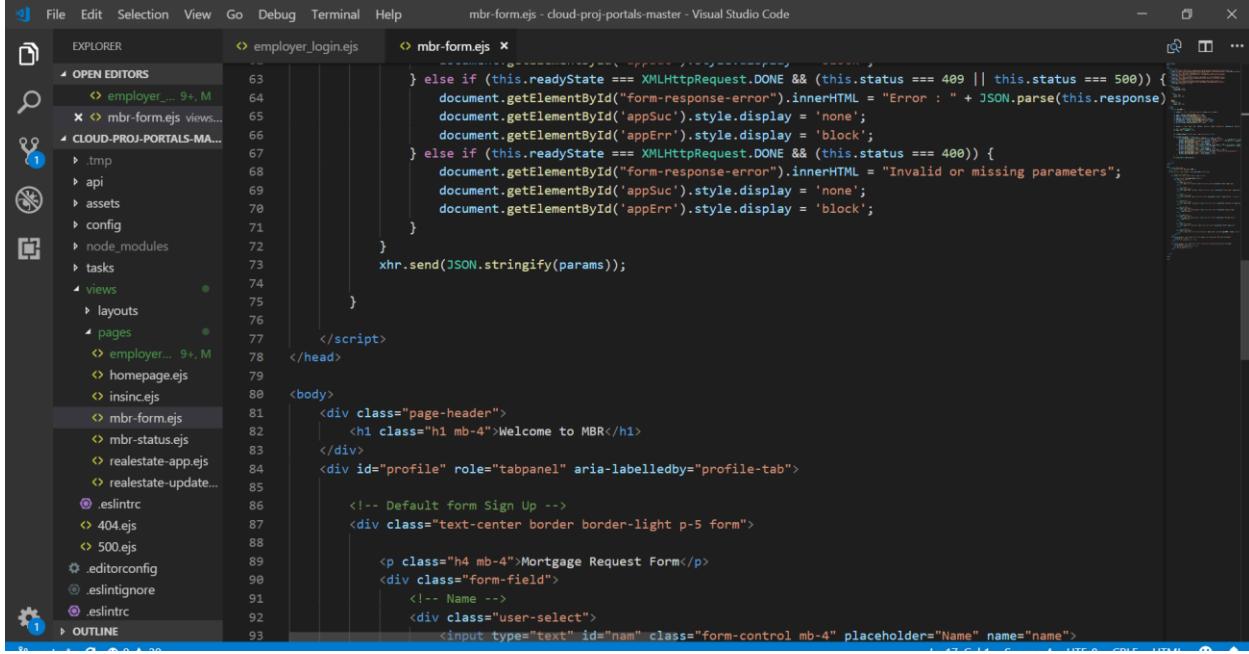
    xhr.setRequestHeader("Content-Type", "application/json; utf-8");

    xhr.onreadystatechange = function () {
        if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
            document.getElementById("form-response-success").innerHTML = "Success : " + JSON.parse(this.respon
            document.getElementById("mortgage-id").innerHTML = "Mortgage Id : " + JSON.parse(this.response).a
            document.getElementById("appErr").style.display = 'none';
            document.getElementById('appSuc').style.display = 'block';
        } else if (this.readyState === XMLHttpRequest.DONE && (this.status === 409 || this.status === 500)) {
            document.getElementById("form-response-error").innerHTML = "Error : " + JSON.parse(this.response)
            document.getElementById('appSuc').style.display = 'none';
        }
    }
}

```

Figure 22: mbr-form.ejs (Form for mortgage application)

## Cloud Computing – CSCI5409



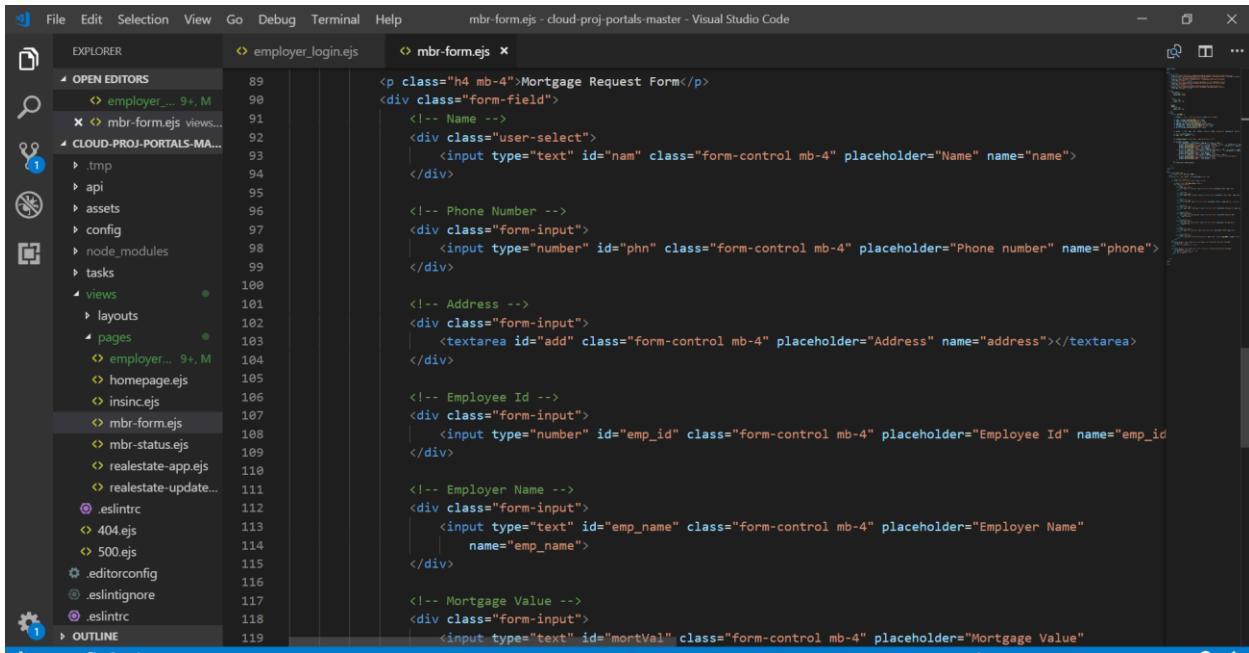
The screenshot shows the Visual Studio Code interface with the file `mbr-form.ejs` open in the editor. The code is an EJS template for a mortgage request form. It includes logic for handling XMLHttpRequest responses and rendering HTML for the form fields. The code is well-structured with comments and proper indentation.

```

File Edit Selection View Go Debug Terminal Help mbr-form.ejs - cloud-proj-portals-master - Visual Studio Code
EXPLORER employer_login.ejs mbr-form.ejs
OPEN EDITORS
CLOUD-PROJ-PORTALS-MA...
tmp
api
assets
config
node_modules
tasks
views
layouts
pages
employer... 9+, M
homepage.ejs
insinc.ejs
mbr-form.ejs
mbr-status.ejs
realestate-app.ejs
realestate-update...
.eslintrc
404.ejs
500.ejs
.editorconfig
.eslintignore
.eslintrc
OUTLINE
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
} else if (this.readyState === XMLHttpRequest.DONE && (this.status === 409 || this.status === 500)) {
    document.getElementById("form-response-error").innerHTML = "Error : " + JSON.parse(this.response);
    document.getElementById('appSuc').style.display = 'none';
    document.getElementById('appErr').style.display = 'block';
} else if (this.readyState === XMLHttpRequest.DONE && (this.status === 400)) {
    document.getElementById("form-response-error").innerHTML = "Invalid or missing parameters";
    document.getElementById('appSuc').style.display = 'none';
    document.getElementById('appErr').style.display = 'block';
}
xhr.send(JSON.stringify(params));
}
</script>
</head>
<body>
<div class="page-header">
    <h1 class="h1 mb-4">Welcome to MBR</h1>
</div>
<div id="profile" role="tabpanel" aria-labelledby="profile-tab">
    <!-- Default form Sign Up -->
    <div class="text-center border border-light p-5 form">
        <p class="h4 mb-4">Mortgage Request Form</p>
        <div class="form-field">
            <!-- Name -->
            <div class="user-select">
                <input type="text" id="nam" class="form-control mb-4" placeholder="Name" name="name">
            </div>
            <!-- Phone Number -->
            <div class="form-input">
                <input type="number" id="phn" class="form-control mb-4" placeholder="Phone number" name="phone">
            </div>
            <!-- Address -->
            <div class="form-input">
                <textarea id="add" class="form-control mb-4" placeholder="Address" name="address"></textarea>
            </div>
            <!-- Employee Id -->
            <div class="form-input">
                <input type="number" id="emp_id" class="form-control mb-4" placeholder="Employee Id" name="emp_id">
            </div>
            <!-- Employer Name -->
            <div class="form-input">
                <input type="text" id="emp_name" class="form-control mb-4" placeholder="Employer Name" name="emp_name">
            </div>
            <!-- Mortgage Value -->
            <div class="form-input">
                <input type="text" id="mortVal" class="form-control mb-4" placeholder="Mortgage Value" name="mortVal">
            </div>
        </div>
    </div>
</div>

```

Figure 23: `mbr-form.ejs` (contd.)



This screenshot continues the `mbr-form.ejs` file from Figure 23. It shows the remaining part of the form template, including fields for address, employee ID, employer name, and mortgage value. The code uses EJS syntax to conditionally render fields based on the user's role (Employer vs. Employee).

```

File Edit Selection View Go Debug Terminal Help mbr-form.ejs - cloud-proj-portals-master - Visual Studio Code
EXPLORER employer_login.ejs mbr-form.ejs
CLOUD-PROJ-PORTALS-MA...
tmp
api
assets
config
node_modules
tasks
views
layouts
pages
employer... 9+, M
homepage.ejs
insinc.ejs
mbr-form.ejs
mbr-status.ejs
realestate-app.ejs
realestate-update...
.eslintrc
404.ejs
500.ejs
.editorconfig
.eslintignore
.eslintrc
OUTLINE
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
<p class="h4 mb-4">Mortgage Request Form</p>
<div class="form-field">
    <!-- Name -->
    <div class="user-select">
        <input type="text" id="nam" class="form-control mb-4" placeholder="Name" name="name">
    </div>
    <!-- Phone Number -->
    <div class="form-input">
        <input type="number" id="phn" class="form-control mb-4" placeholder="Phone number" name="phone">
    </div>
    <!-- Address -->
    <div class="form-input">
        <textarea id="add" class="form-control mb-4" placeholder="Address" name="address"></textarea>
    </div>
    <!-- Employee Id -->
    <div class="form-input">
        <input type="number" id="emp_id" class="form-control mb-4" placeholder="Employee Id" name="emp_id">
    </div>
    <!-- Employer Name -->
    <div class="form-input">
        <input type="text" id="emp_name" class="form-control mb-4" placeholder="Employer Name" name="emp_name">
    </div>
    <!-- Mortgage Value -->
    <div class="form-input">
        <input type="text" id="mortVal" class="form-control mb-4" placeholder="Mortgage Value" name="mortVal">
    </div>
</div>

```

Figure 24: `mbr-form.ejs` (contd.)

## Cloud Computing – CSCI5409

```

File Edit Selection View Go Debug Terminal Help mbr-form.ejs - cloud-proj-portals-master - Visual Studio Code
EXPLORER employer_login.ejs mbr-form.ejs
OPEN EDITORS
CLOUD-PROJ-PORTALS-MA...
tmp
api
assets
config
node_modules
tasks
views
layouts
pages
employer... 9+, M
homepage.ejs
inscje.js
mbr-form.ejs
mbr-status.ejs
realestate-app.ejs
realestate-update...
.eslintrc
404.ejs
500.ejs
.editorconfig
.eslintignore
.eslintrc
OUTLINE
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
<!-- Mortgage Value -->
<div class="form-input">
    <input type="text" id="mortVal" class="form-control mb-4" placeholder="Mortgage Value"
           name="mortVal">
</div>
<!-- MSID -->
<div class="form-input">
    <input type="text" id="msid" class="form-control mb-4" placeholder="MSID" name="msid">
</div>
<!-- Sign up button -->
<div class="form-input">
    <button class="btn btn-info btn-block my-4" type="submit" onclick="applyBMR()">Submit</button>
</div>
<div style='display: none' class="alert alert-danger alert-dismissible fade show" id="appErr">
    <strong>Error!</strong>
    <div id="form-response-error"></div>
</div>
<div style='display: none' class="alert alert-success alert-dismissible fade show" id="appSuc">
    <strong>Success!</strong>
    <div id="form-response-success"></div>
    <div id="mortgage-id"></div>
</div>

```

Figure 25: mbr-form.ejs (contd.)

```

File Edit Selection View Go Debug Terminal Help mbr-status.ejs - cloud-proj-portals-master - Visual Studio Code
mbr-status.ejs
<script>
    function loginBMR() {
        var mbrid = document.getElementById("mbrid").value;
        var endpoint = "http://rockers-mbr.eastus2.cloudapp.azure.com/application/status?mortId=" + mbrid;

        var xhr = new XMLHttpRequest();
        xhr.open("GET", endpoint);

        xhr.setRequestHeader("Content-Type", "application/json; utf-8");

        xhr.onreadystatechange = function() {
            if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {

                var info = JSON.parse(this.response).status;

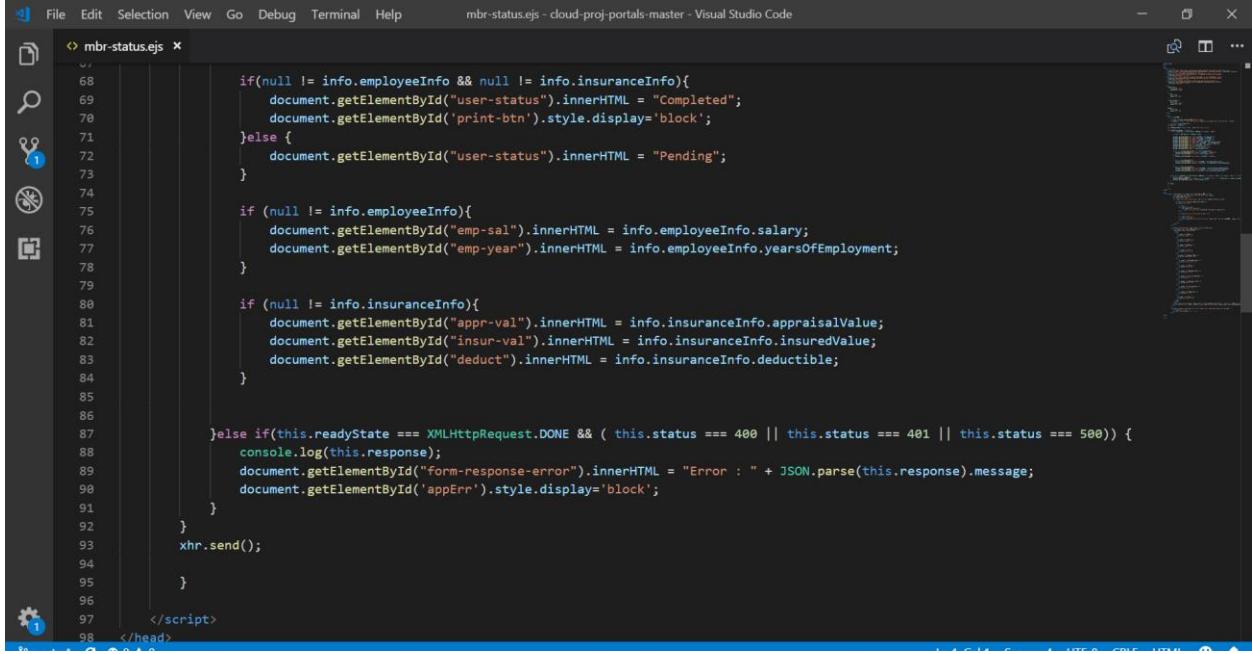
                document.getElementById("user-id").innerHTML = info.employee_id;
                document.getElementById("user-name").innerHTML = info.name;
                document.getElementById("user-phone").innerHTML = info.phone_no;
                document.getElementById("user-add").innerHTML = info.addr;
                document.getElementById("user-empl-name").innerHTML = info.employer_name;
                document.getElementById("user-mort-val").innerHTML = info.mortgageValue;
                document.getElementById("mort-id").innerHTML = info.msid;
                document.getElementById('home').style.display='none';
                document.getElementById('app-table').style.display='block';
                document.getElementById('appErr').style.display='none';

                if(null != info.employeeInfo && null != info.insuranceInfo){
                    document.getElementById("user-status").innerHTML = "Completed";
                    document.getElementById('print-btn').style.display='block';
                }else {

```

Figure 26: mbr-status.ejs (Portal to check the status of the mortgage application)

## Cloud Computing – CSCI5409

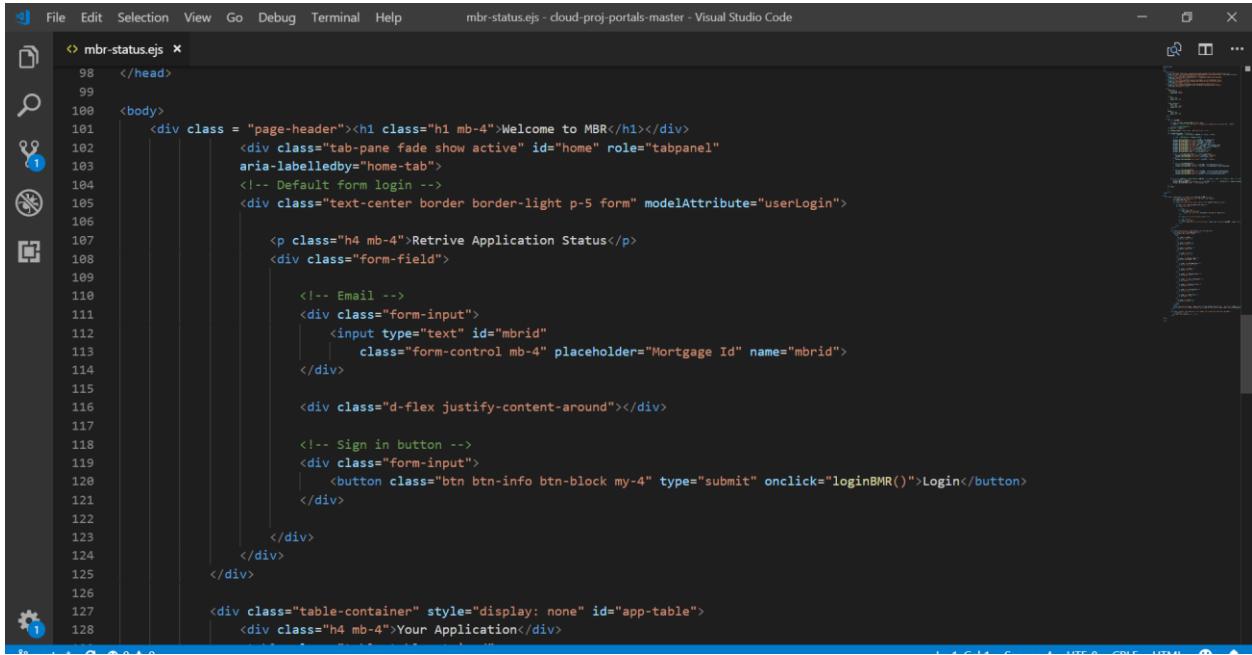


The screenshot shows the Visual Studio Code interface with the file 'mbr-status.ejs' open. The code is a combination of JavaScript and HTML. It includes logic to handle employee and insurance information, and an XMLHttpRequest to fetch data from a server. If the request fails, it displays an error message.

```
File Edit Selection View Go Debug Terminal Help mbr-status.ejs - cloud-proj-portals-master - Visual Studio Code

1<head>
2    <script>
3        ...
4        if(null != info.employeeInfo && null != info.insuranceInfo){
5            document.getElementById("user-status").innerHTML = "Completed";
6            document.getElementById('print-btn').style.display='block';
7        }else {
8            document.getElementById("user-status").innerHTML = "Pending";
9        }
10
11        if (null != info.employeeInfo){
12            document.getElementById("emp-sal").innerHTML = info.employeeInfo.salary;
13            document.getElementById("emp-year").innerHTML = info.employeeInfo.yearsOfEmployment;
14        }
15
16        if (null != info.insuranceInfo{
17            document.getElementById("appr-val").innerHTML = info.insuranceInfo.appraisalValue;
18            document.getElementById("insur-val").innerHTML = info.insuranceInfo.insuredValue;
19            document.getElementById("deduct").innerHTML = info.insuranceInfo.deductible;
20        }
21
22        }else if(this.readyState === XMLHttpRequest.DONE && ( this.status === 400 || this.status === 401 || this.status === 500)) {
23            console.log(this.response);
24            document.getElementById("form-response-error").innerHTML = "Error : " + JSON.parse(this.response).message;
25            document.getElementById('appErr').style.display='block';
26        }
27    }
28    xhr.send();
29
30
31    </script>
32</head>
```

Figure 27: mbr-status.ejs (contd.)



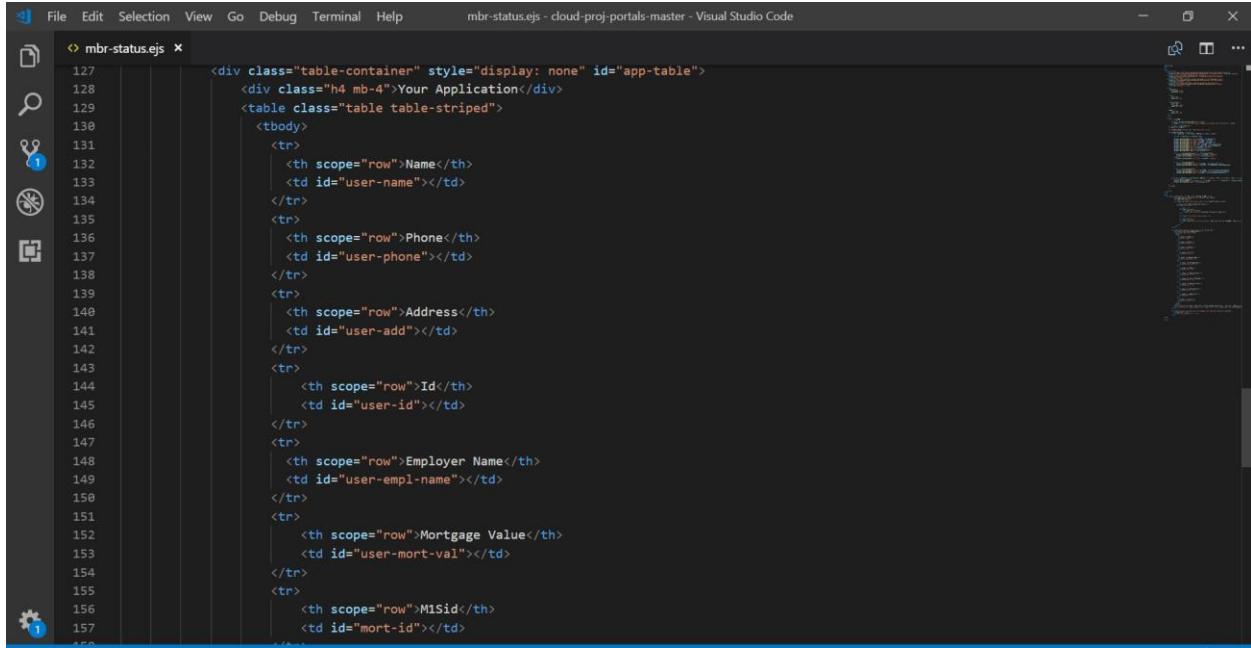
The screenshot shows the Visual Studio Code interface with the file 'mbr-status.ejs' open. The code continues from where Figure 27 left off, showing the body of the HTML page. It includes a welcome header, a tabbed navigation bar, and a form for logging in. The form uses Bootstrap classes like 'text-center', 'border-light', and 'p-5'.

```
File Edit Selection View Go Debug Terminal Help mbr-status.ejs - cloud-proj-portals-master - Visual Studio Code

98</head>
99
100<body>
101    <div class = "page-header"><h1 class="h1 mb-4">Welcome to MBR</h1></div>
102    <div class="tab-pane fade show active" id="home" role="tabpanel"
103        aria-labelledby="home-tab">
104        <!-- Default form login -->
105        <div class="text-center border border-light p-5 form" modelAttribute="userLogin">
106
107            <p class="h4 mb-4">Retrieve Application Status</p>
108            <div class="form-field">
109
110                <!-- Email -->
111                <div class="form-input">
112                    <input type="text" id="mbrid"
113                        class="form-control mb-4" placeholder="Mortgage Id" name="mbrid">
114                </div>
115
116                <div class="d-flex justify-content-around"></div>
117
118                <!-- Sign in button -->
119                <div class="form-input">
120                    <button class="btn btn-info btn-block my-4" type="submit" onclick="loginBMR()">Login</button>
121                </div>
122
123            </div>
124        </div>
125
126        <div class="table-container" style="display: none" id="app-table">
127            <div class="h4 mb-4">Your Application</div>
128        </div>
129
130    </div>
131
132</body>
```

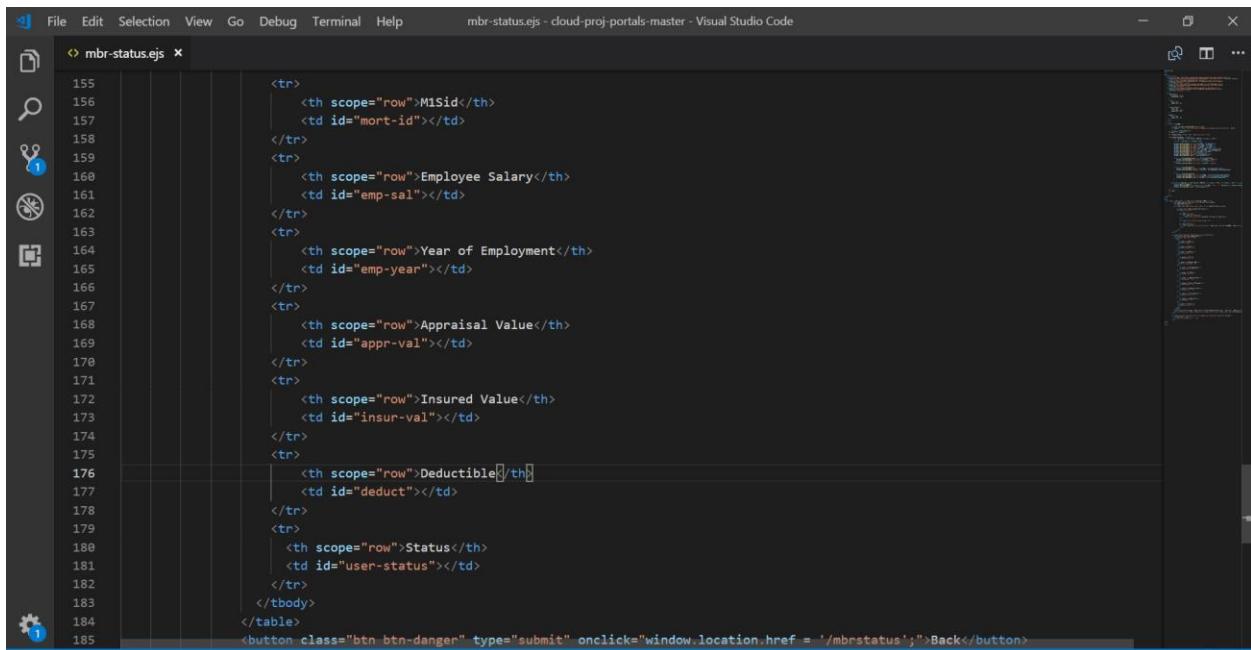
Figure 28: mbr-status.ejs (contd.)

## Cloud Computing – CSCI5409



```
<div class="table-container" style="display: none" id="app-table">
  <div class="mb-4">Your Application</div>
  <table class="table table-striped">
    <tbody>
      <tr>
        <th scope="row">Name</th>
        <td id="user-name"></td>
      </tr>
      <tr>
        <th scope="row">Phone</th>
        <td id="user-phone"></td>
      </tr>
      <tr>
        <th scope="row">Address</th>
        <td id="user-add"></td>
      </tr>
      <tr>
        <th scope="row">Id</th>
        <td id="user-id"></td>
      </tr>
      <tr>
        <th scope="row">Employer Name</th>
        <td id="user-empl-name"></td>
      </tr>
      <tr>
        <th scope="row">Mortgage Value</th>
        <td id="user-mort-val"></td>
      </tr>
      <tr>
        <th scope="row">MSId</th>
        <td id="mort-id"></td>
      </tr>
    </tbody>
  </table>
</div>
```

Figure 29: mbr-status.ejs (contd.)



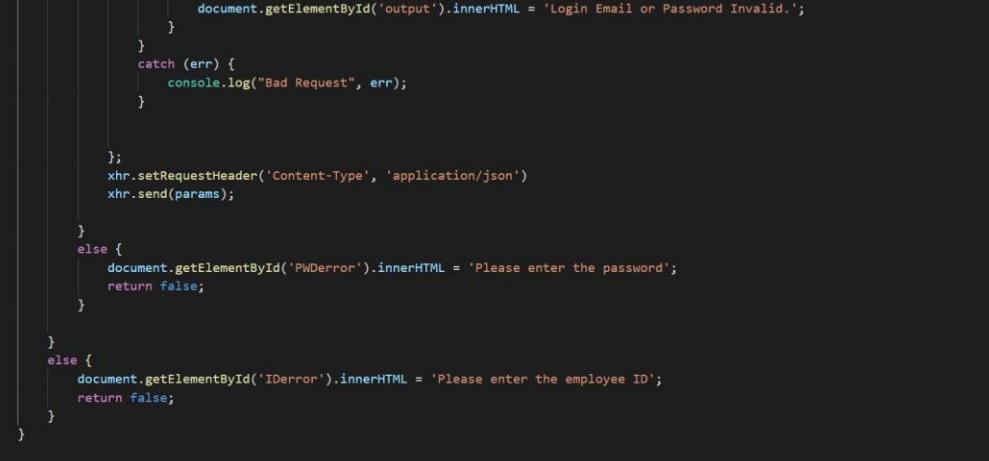
```
<tr>
  <th scope="row">MSId</th>
  <td id="mort-id"></td>
</tr>
<tr>
  <th scope="row">Employee Salary</th>
  <td id="emp-sal"></td>
</tr>
<tr>
  <th scope="row">Year of Employment</th>
  <td id="emp-year"></td>
</tr>
<tr>
  <th scope="row">Appraisal Value</th>
  <td id="appr-val"></td>
</tr>
<tr>
  <th scope="row">Insured Value</th>
  <td id="insur-val"></td>
</tr>
<tr>
  <th scope="row">Deductible</th>
  <td id="deduct"></td>
</tr>
<tr>
  <th scope="row">Status</th>
  <td id="user-status"></td>
</tr>
</tbody>
</table>
<button class="btn btn-danger" type="submit" onclick="window.location.href = '/mbrstatus';">Back</button>
```

Figure 30: mbr-status.ejs (contd.)

Cloud Computing – CSCI5409

## EMPLOYER PORTAL

**Figure 31: employer-login.ejs (Employer portal)**



The screenshot shows the Visual Studio Code interface with the file `employer_login.ejs` open. The code is written in EJS (Embedded JavaScript) and handles user input validation for login credentials. It uses document.getElementById to get element by ID and innerHTML to set or update the content of those elements. It also uses XMLHttpRequest (xhr) to send JSON data to a server. The code includes error handling for invalid email/password and missing employee ID.

```
File Edit Selection View Go Debug Terminal Help employer_login.ejs - cloud-proj-portals-master - Visual Studio Code

367     }
368     else {
369         document.getElementById('output').innerHTML = 'Login Email or Password Invalid.';
370     }
371 }
372 catch (err) {
373     console.log("Bad Request", err);
374 }

375
376
377 };
378 xhr.setRequestHeader('Content-Type', 'application/json')
379 xhr.send(params);
380
381 }
382 else {
383     document.getElementById('PWError').innerHTML = 'Please enter the password';
384     return false;
385 }
386
387 }
388 else {
389     document.getElementById('IDerror').innerHTML = 'Please enter the employee ID';
390     return false;
391 }
392
393
394
395
396 function consent() {
397     var urlMBR = document.getElementById("url").value;
```

**Figure 32: employer-login.ejs (contd.)**

Cloud Computing – CSCI5409

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** employer\_login.ejs - cloud-proj-portals-master - Visual Studio Code
- File Explorer (Left):** Shows a tree view of files and folders.
- Search Bar (Top Right):** A magnifying glass icon.
- Code Editor (Main Area):** Displays the content of the employer\_login.ejs file. The code is a JavaScript function named consent() that handles form submission and sends a POST request to a server endpoint. It uses XMLHttpRequest to make the call and parses the response using JSON.parse(). The code includes comments explaining the regex used for URLs and the structure of the URL.
- Bottom Status Bar:** Shows the current file path: employer\_login.ejs.

```
function consent() {
    var urlMBR = document.getElementById("url").value;
    var mortId = document.getElementById("appNo").value;
    var employeeId = document.getElementById("employeeID").value;
    var consentCheck = document.getElementById("checkbox").checked;
    var applicationNo = parseInt(appNo, 10);
    var consent = JSON.parse(consentCheck);
    //REFERENCE STARTS----- REGEX TAKEN FROM http://urlregex.com/
    var regexp = /(([A-Za-z]{3,9}:(?:\/\/)?)(?:[~-;:&=+\$,\\w]+@[A-Za-z0-9\.~-]+|(?::www\.|[~-;:&=+\$,\\w]+@[A-Za-z0-9\.~-]+))(?:\/|[/~?#])*/;
    //REFERENCE ENDS
    document.getElementById('URLerror').innerHTML = '';
    document.getElementById('appNumberError').innerHTML = '';
    document.getElementById('consentError').innerHTML = '';
    if (mortId != '') {
        if (regexp.test(urlMBR) && urlMBR != '') {

            if (consent) {
                var xhr = new XMLHttpRequest(),
                    method = "POST",
                    url = "http://rockers-employer.centralus.cloudapp.azure.com/share/apply";

                let params = JSON.stringify({ mortId, urlMBR, employeeId });
                console.log(params);
                xhr.open(method, url, true);
                xhr.onreadystatechange = function () {
                    if (xhr.readyState === 4 && xhr.status === 200) {
                        console.log(JSON.parse(xhr.responseText).message);
                        document.getElementById('output1').innerHTML = JSON.parse(xhr.responseText).message;
                    }
                else if (xhr.readyState === 4 && xhr.status === 409) {
                    document.getElementById('output1').innerHTML = JSON.parse(xhr.responseText).message;
                }
            }
        }
    }
}
```

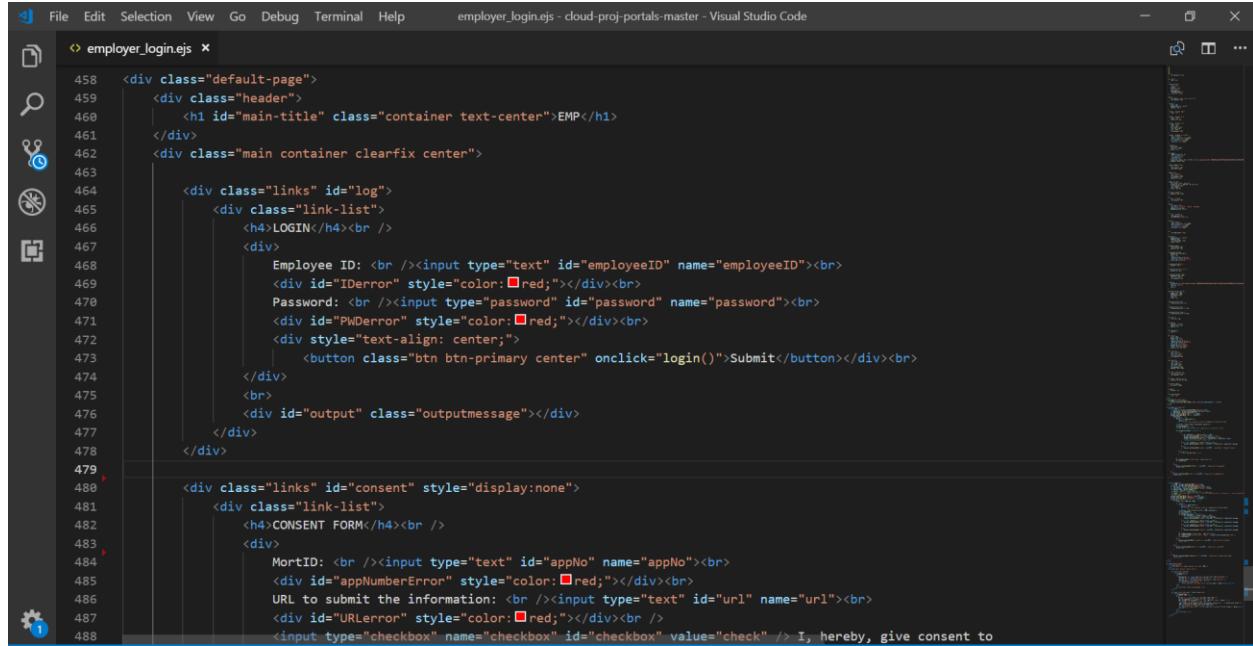
**Figure 33: employer-login.ejs (contd.)**

The screenshot shows the Visual Studio Code interface with the file `employer_login.ejs` open. The code is written in EJS (Embedded JavaScript) and handles various user inputs and validation logic. The code includes conditional statements for different HTTP status codes (403, 400, 409), setting request headers for JSON and Bearer token, and displaying error messages for URL and application number fields.

```
425     else if (xhr.readyState === 4 && xhr.status === 409) {
426         document.getElementById('output1').innerHTML = JSON.parse(xhr.responseText).message;
427     }
428     else if (xhr.readyState === 4 && xhr.status === 400) {
429         document.getElementById('output1').innerHTML = JSON.parse(xhr.responseText).message;
430     }
431     else {
432         document.getElementById('output1').innerHTML = JSON.parse(xhr.responseText).message;
433     }
434 };
435 xhr.setRequestHeader('Content-Type', 'application/json');
436 xhr.setRequestHeader('Authorization', 'Bearer ' + window.localStorage.getItem('token'));
437 xhr.send(params);
438 }
439 else {
440     document.getElementById('consentError').innerHTML = 'Please check the checkbox';
441     return false;
442 }
443 }
444 else {
445     document.getElementById('URLerror').innerHTML = 'Please enter a valid URL';
446     return false;
447 }
448 }
449 }
450
451 else {
452     document.getElementById('appNumberError').innerHTML = 'Please enter the application number';
453     return false;
454 }
// }
```

**Figure 34: employer-login.ejs (contd.)**

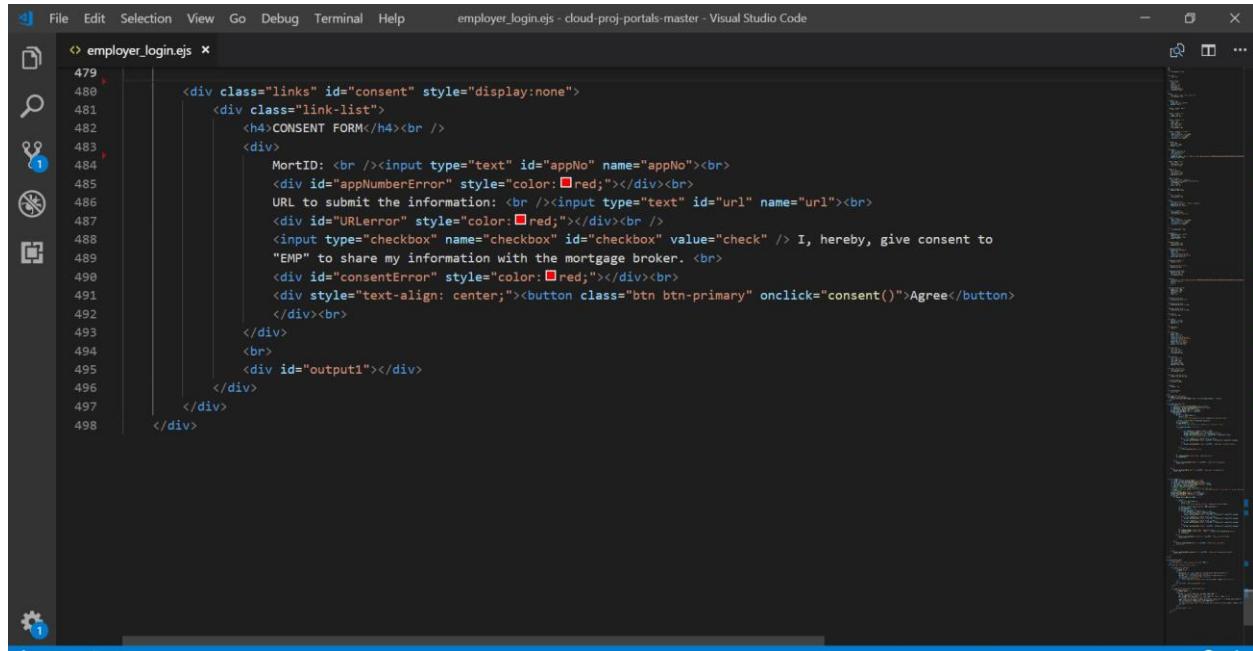
## Cloud Computing – CSCI5409



The screenshot shows the Visual Studio Code interface with the file 'employer\_login.ejs' open. The code is an EJS template for a login page. It includes sections for employee login and consent form submission. Error messages are displayed in red for fields like Employee ID and URL.

```
<div class="default-page">
  <div class="header">
    <h1 id="main-title" class="container text-center">EMP</h1>
  </div>
  <div class="main container clearfix center">
    <div class="links" id="log">
      <div class="link-list">
        <h4>LOGIN</h4><br />
        <div>
          Employee ID: <br /><input type="text" id="employeeID" name="employeeID"><br />
          <div id="IDerror" style="color: red;"></div><br />
          Password: <br /><input type="password" id="password" name="password"><br />
          <div id="PWDerror" style="color: red;"></div><br />
          <div style="text-align: center;">
            <button class="btn btn-primary center" onclick="login()">Submit</button><br />
          </div>
          <br>
          <div id="output" class="outputmessage"></div>
        </div>
      </div>
    </div>
    <div class="links" id="consent" style="display:none">
      <div class="link-list">
        <h4>CONSENT FORM</h4><br />
        <div>
          MortID: <br /><input type="text" id="appNo" name="appNo"><br />
          <div id="appNumberError" style="color: red;"></div><br />
          URL to submit the information: <br /><input type="text" id="url" name="url"><br />
          <div id="URLerror" style="color: red;"></div><br />
          <input type="checkbox" name="checkbox" id="checkbox" value="check" /> I, hereby, give consent to
        </div>
      </div>
    </div>
  </div>
</div>
```

Figure 35: employer-login.ejs (contd.)

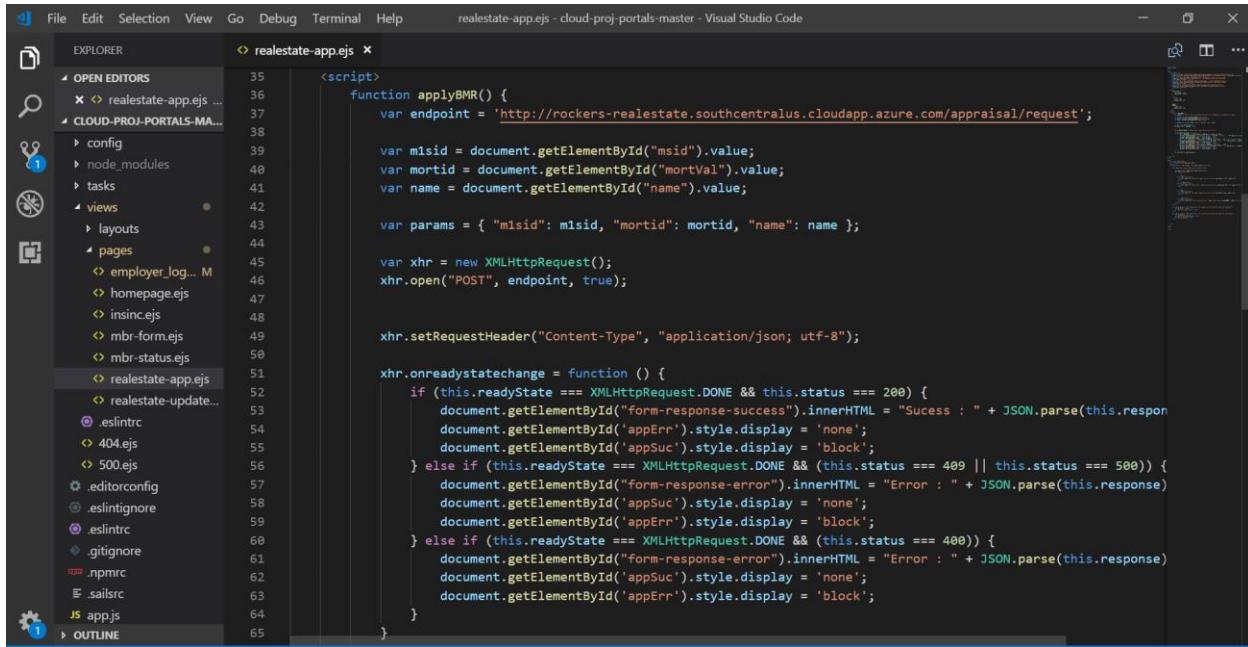


The screenshot shows the Visual Studio Code interface with the file 'employer\_login.ejs' open. The code is identical to Figure 35, but the 'consent' section has been modified. A new line has been added after the checkbox input, and the text 'I, hereby, give consent to "EMP" to share my information with the mortgage broker.' is now enclosed in quotes.

```
<div class="links" id="consent" style="display:none">
  <div class="link-list">
    <h4>CONSENT FORM</h4><br />
    <div>
      MortID: <br /><input type="text" id="appNo" name="appNo"><br />
      <div id="appNumberError" style="color: red;"></div><br />
      URL to submit the information: <br /><input type="text" id="url" name="url"><br />
      <div id="URLerror" style="color: red;"></div><br />
      <input type="checkbox" name="checkbox" id="checkbox" value="check" /> I, hereby, give consent to
      "EMP" to share my information with the mortgage broker. <br />
      <div id="consentError" style="color: red;"></div><br />
      <div style="text-align: center;"><button class="btn btn-primary" onclick="consent()">Agree</button>
    </div>
  </div>
</div>
```

Figure 36: employer-login.ejs (contd.)

### REAL ESTATE PORTAL



The screenshot shows the Visual Studio Code interface with the file "realstate-app.ejs" open in the editor. The code is a JavaScript file containing logic for sending an appraisal request via XMLHttpRequest. It includes functions for applying Business Method Refactoring (BMR) and handling the response from the server. The code uses JSON.parse() to parse responses and sets up event listeners for XMLHttpRequest.onreadystatechange.

```

<script>
    function applyBMR() {
        var endpoint = 'http://rockers-realestate.southcentralus.cloudapp.azure.com/appraisal/request';

        var msid = document.getElementById("msid").value;
        var mortid = document.getElementById("mortval").value;
        var name = document.getElementById("name").value;

        var params = { "msid": msid, "mortid": mortid, "name": name };

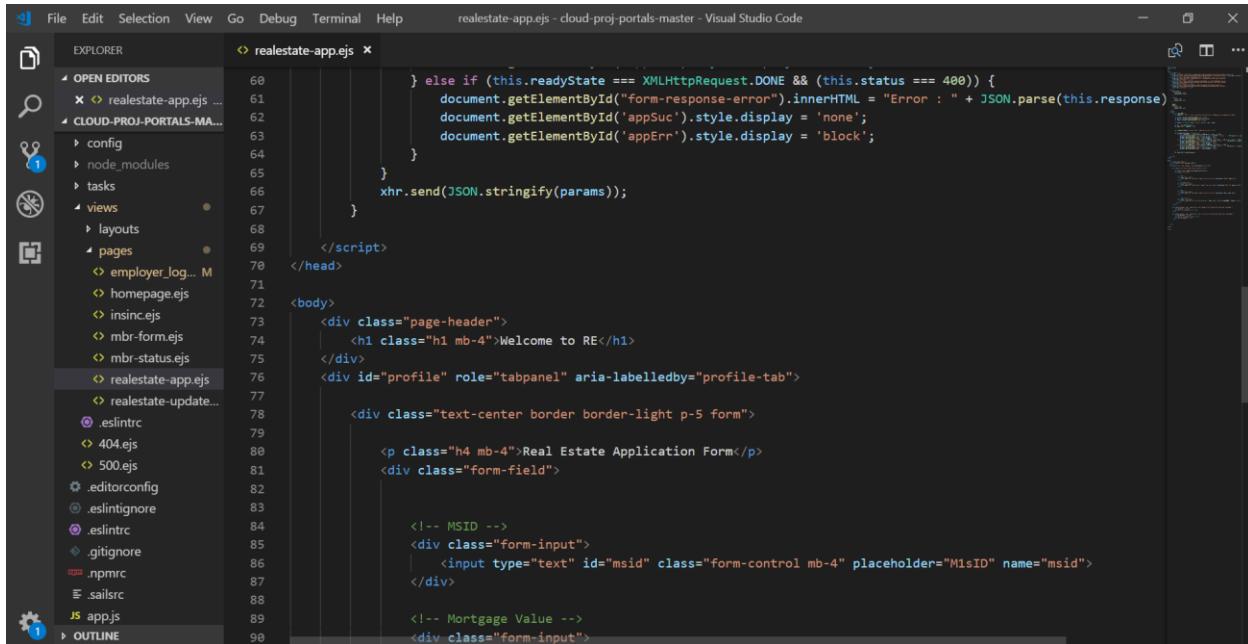
        var xhr = new XMLHttpRequest();
        xhr.open("POST", endpoint, true);

        xhr.setRequestHeader("Content-Type", "application/json; utf-8");

        xhr.onreadystatechange = function () {
            if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
                document.getElementById("form-response-success").innerHTML = "Sucess : " + JSON.parse(this.response);
                document.getElementById('appErr').style.display = 'none';
                document.getElementById('appSuc').style.display = 'block';
            } else if (this.readyState === XMLHttpRequest.DONE && (this.status === 409 || this.status === 500)) {
                document.getElementById("form-response-error").innerHTML = "Error : " + JSON.parse(this.response);
                document.getElementById('appSuc').style.display = 'none';
                document.getElementById('appErr').style.display = 'block';
            } else if (this.readyState === XMLHttpRequest.DONE && (this.status === 400)) {
                document.getElementById("form-response-error").innerHTML = "Error : " + JSON.parse(this.response);
                document.getElementById('appSuc').style.display = 'none';
                document.getElementById('appErr').style.display = 'block';
            }
        }
    }
}

```

Figure 37: realstate-app.ejs (Real Estate appraisal application)



This screenshot continues the Visual Studio Code session, showing the same file "realstate-app.ejs". The code now includes the XMLHttpRequest.send() method call after the JSON.stringify(params) line. Below the script, the HTML structure of the page is visible, featuring a header with "Welcome to RE" and a main content area with a tab panel for "profile". The code also includes comments for "MSID" and "Mortgage Value" fields.

```

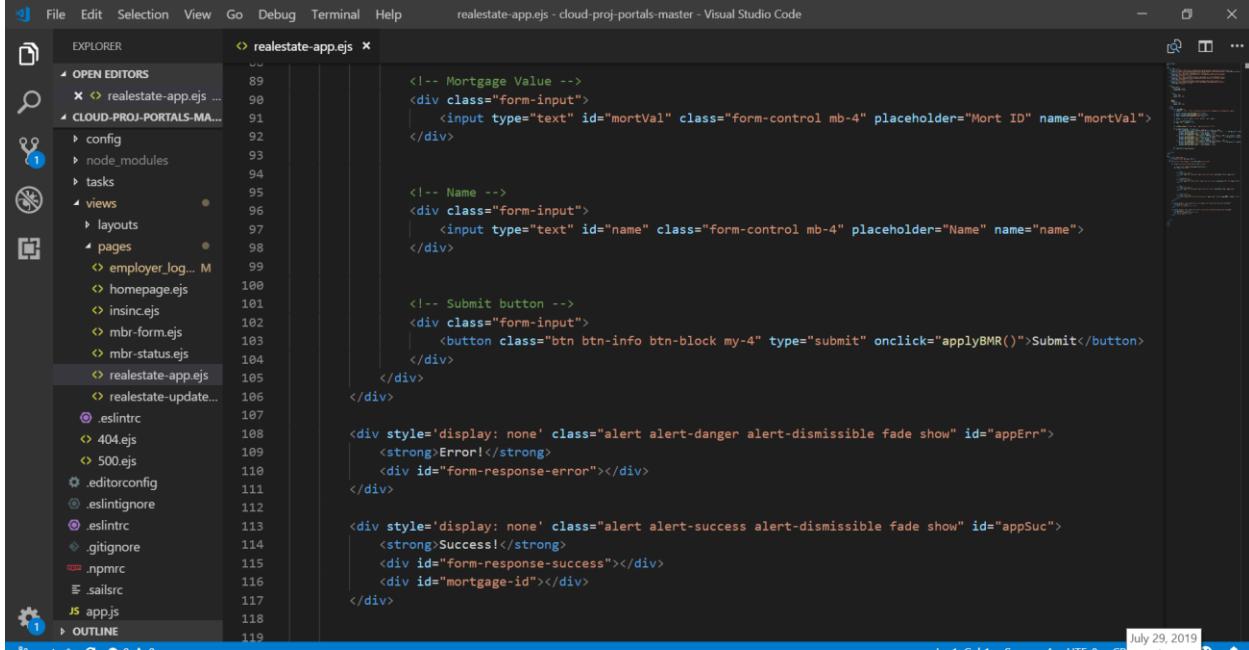
        } else if (this.readyState === XMLHttpRequest.DONE && (this.status === 400)) {
            document.getElementById("form-response-error").innerHTML = "Error : " + JSON.parse(this.response);
            document.getElementById('appSuc').style.display = 'none';
            document.getElementById('appErr').style.display = 'block';
        }
    }
    xhr.send(JSON.stringify(params));
}

</script>
</head>
<body>
    <div class="page-header">
        <h1 class="h1 mb-4">Welcome to RE</h1>
    </div>
    <div id="profile" role="tabpanel" aria-labelledby="profile-tab">
        <div class="text-center border border-light p-5 form">
            <p class="h4 mb-4">Real Estate Application Form</p>
            <div class="form-field">
                <!-- MSID -->
                <div class="form-input">
                    <input type="text" id="msid" class="form-control mb-4" placeholder="MlsID" name="msid">
                </div>
                <!-- Mortgage Value -->
                <div class="form-input">

```

Figure 38: realstate-app.ejs (contd.)

## Cloud Computing – CSCI5409



This screenshot shows the Visual Studio Code interface with the file `realestate-app.ejs` open in the editor. The code is an Express.js route handler for a real estate application. It includes logic for handling form inputs, displaying error messages, and showing success messages. The code uses ES6 syntax and includes comments explaining the structure.

```

    <!-- Mortgage Value -->
<div class="form-input">
  <input type="text" id="mortVal" class="form-control mb-4" placeholder="Mort ID" name="mortVal">
</div>

<!-- Name -->
<div class="form-input">
  <input type="text" id="name" class="form-control mb-4" placeholder="Name" name="name">
</div>

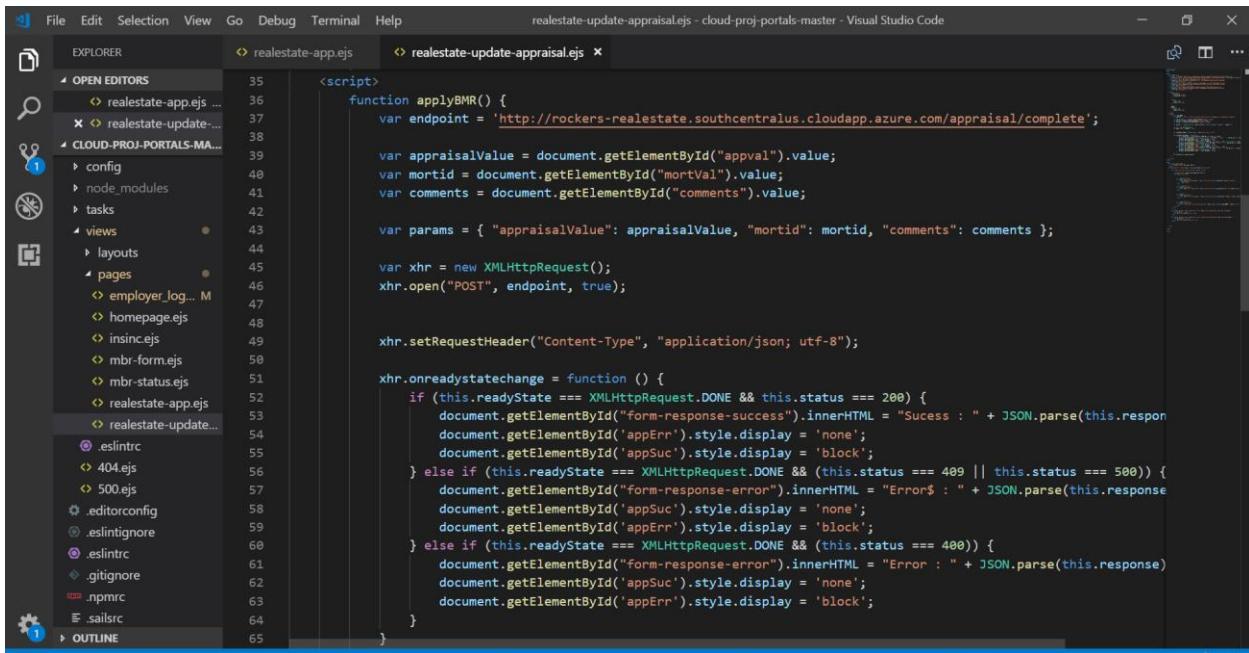
<!-- Submit button -->
<div class="form-input">
  <button class="btn btn-info btn-block my-4" type="submit" onclick="applyBMR()>Submit</button>
</div>
</div>

<div style='display: none' class="alert alert-danger alert-dismissible fade show" id="appErr">
  <strong>Error!</strong>
  <div id="form-response-error"></div>
</div>

<div style='display: none' class="alert alert-success alert-dismissible fade show" id="appSuc">
  <strong>Success!</strong>
  <div id="form-response-success"></div>
  <div id="mortgage-id"></div>
</div>

```

Figure 39: `realestate-app.ejs` (contd.)



This screenshot shows the Visual Studio Code interface with the file `realestate-update.appraisal.ejs` open in the editor. The code is a JavaScript function named `applyBMR()` located in the `realestate-app.ejs` file. It performs an asynchronous POST request to an external endpoint to evaluate an appraisal. The function handles the response and updates UI elements based on the status code.

```

function applyBMR() {
  var endpoint = 'http://rockers-realestate.southcentralus.cloudapp.azure.com/appraisal/complete';

  var appraisalValue = document.getElementById("appval").value;
  var mortid = document.getElementById("mortVal").value;
  var comments = document.getElementById("comments").value;

  var params = { "appraisalValue": appraisalValue, "mortid": mortid, "comments": comments };

  var xhr = new XMLHttpRequest();
  xhr.open("POST", endpoint, true);

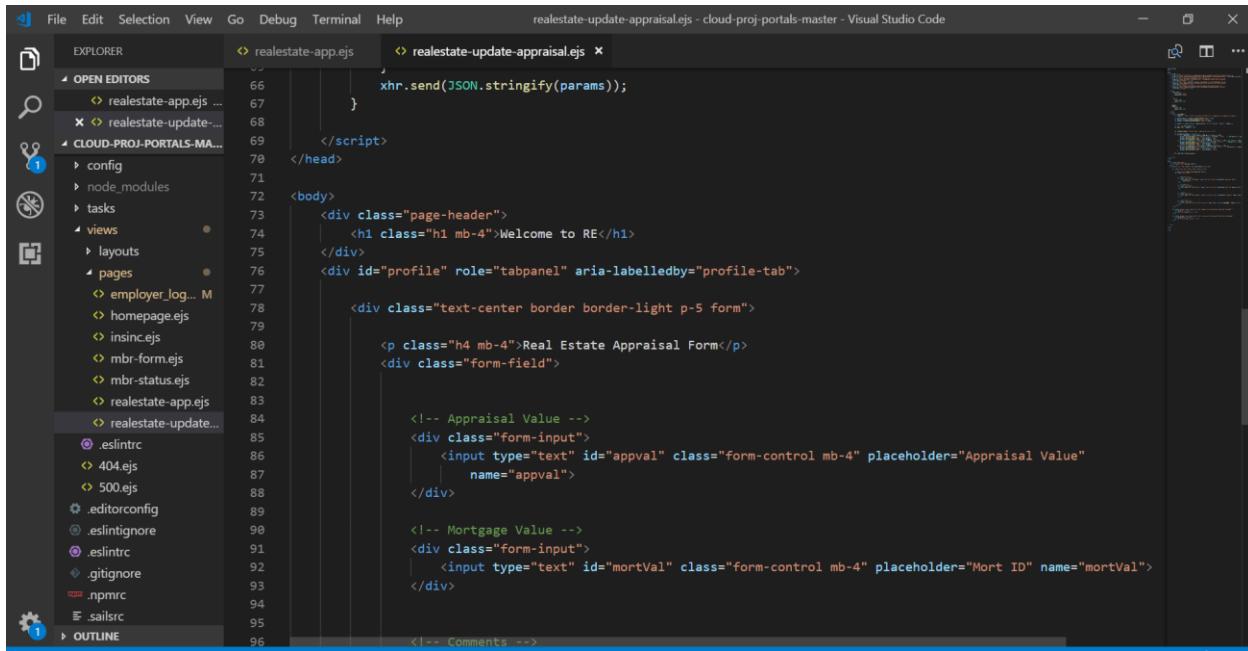
  xhr.setRequestHeader("Content-Type", "application/json; utf-8");

  xhr.onreadystatechange = function () {
    if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
      document.getElementById("form-response-success").innerHTML = "Success : " + JSON.parse(this.responseText);
      document.getElementById('appErr').style.display = 'none';
      document.getElementById('appSuc').style.display = 'block';
    } else if (this.readyState === XMLHttpRequest.DONE && (this.status === 400 || this.status === 500)) {
      document.getElementById("form-response-error").innerHTML = "Error$ : " + JSON.parse(this.responseText);
      document.getElementById('appSuc').style.display = 'none';
      document.getElementById('appErr').style.display = 'block';
    } else if (this.readyState === XMLHttpRequest.DONE && (this.status === 400)) {
      document.getElementById("form-response-error").innerHTML = "Error : " + JSON.parse(this.responseText);
      document.getElementById('appSuc').style.display = 'none';
      document.getElementById('appErr').style.display = 'block';
    }
  }
}

```

Figure 40: `realestate-update.ejs` (Appraisal Evaluation)

## Cloud Computing – CSCI5409



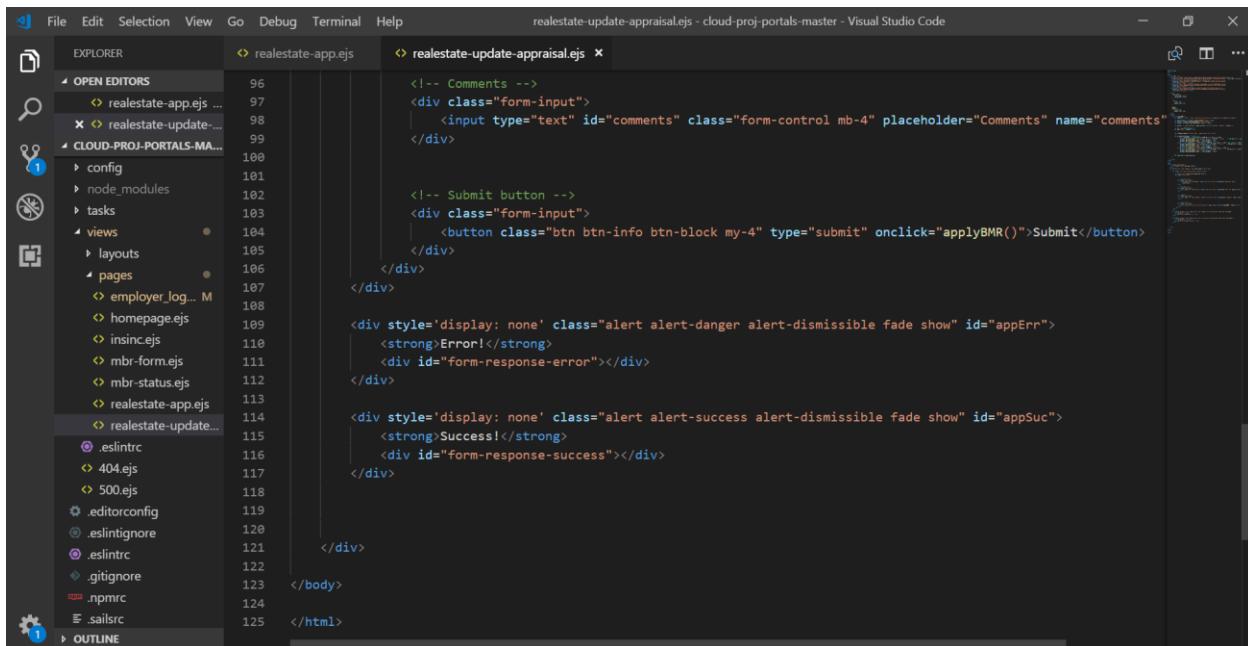
The screenshot shows the Visual Studio Code interface with the file "realstate-update.ejs" open in the editor. The code is an EJS template for a real estate appraisal form. It includes sections for Appraisal Value, Mortgage Value, and Comments, along with a submit button and error/success alerts.

```
<!-- Appraisal Value -->
<div class="form-input">
  <input type="text" id="appval" class="form-control mb-4" placeholder="Appraisal Value" name="appval">
</div>

<!-- Mortgage Value -->
<div class="form-input">
  <input type="text" id="mortVal" class="form-control mb-4" placeholder="Mort ID" name="mortVal">
</div>

<!-- Comments -->
<div class="form-input">
  <input type="text" id="comments" class="form-control mb-4" placeholder="Comments" name="comments">
</div>
```

Figure 41: realstate-update.ejs (contd.)



The screenshot shows the Visual Studio Code interface with the file "realstate-update.ejs" open in the editor. The code continues from the previous snippet, showing the rest of the EJS template. It includes sections for Comments, Submit button, and alerts for errors and success messages.

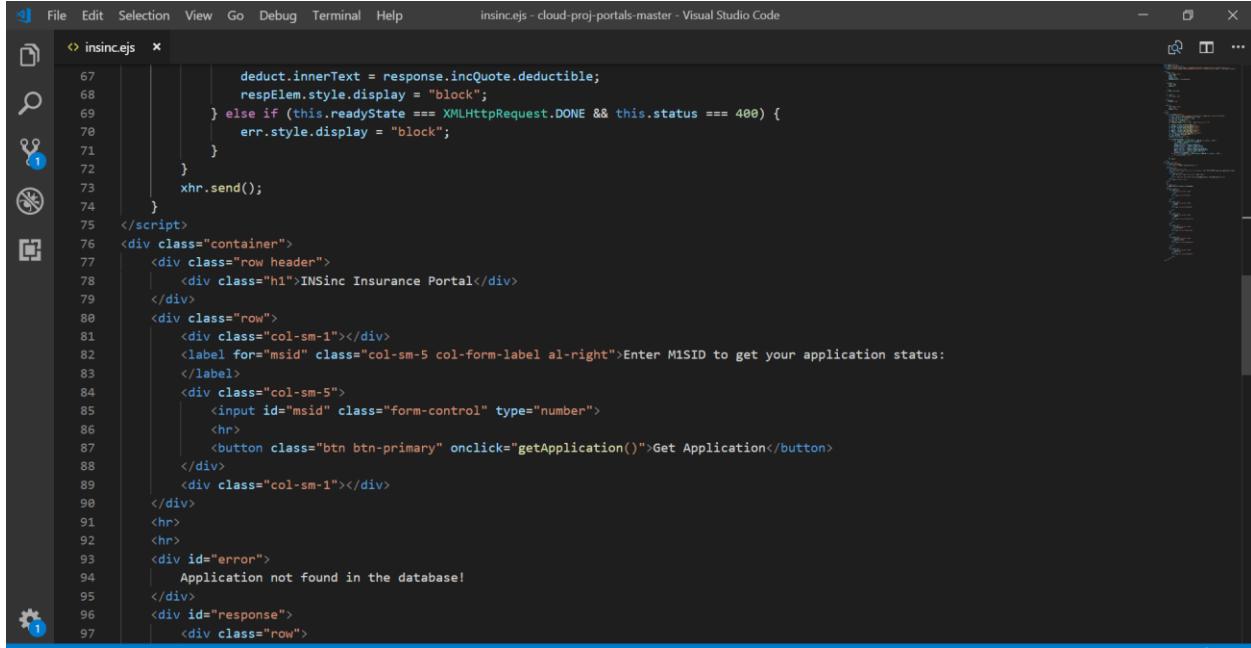
```
<!-- Submit button -->
<div class="form-input">
  <button class="btn btn-info btn-block my-4" type="submit" onclick="applyBMR()>Submit</button>
</div>
</div>

<div style='display: none' class="alert alert-danger alert-dismissible fade show" id="appErr">
  <strong>Error!</strong>
  <div id="form-response-error"></div>
</div>

<div style='display: none' class="alert alert-success alert-dismissible fade show" id="appSuc">
  <strong>Success!</strong>
  <div id="form-response-success"></div>
</div>
```

Figure 42: realstate-update.ejs (contd.)

## INSURANCE PORTAL



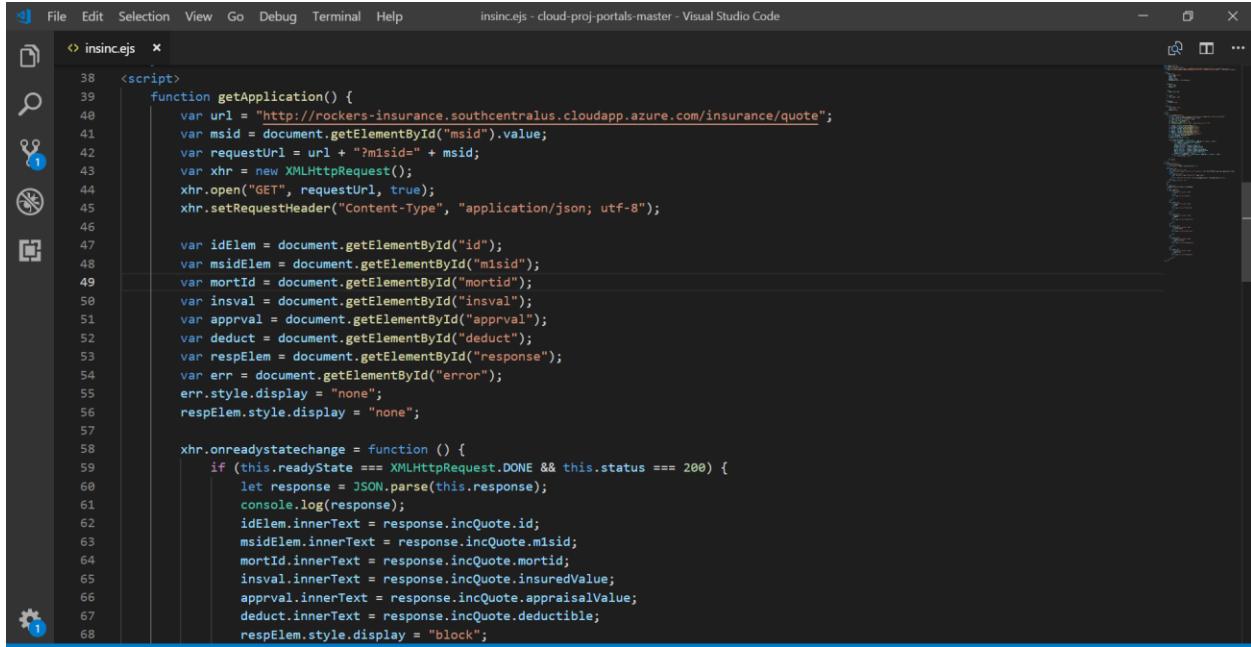
The screenshot shows the Visual Studio Code interface with the file 'insinc.ejs' open. The code is an EJS template for an insurance portal. It includes a script block with a function 'getApplication()' that sends an XMLHttpRequest to a server to retrieve insurance quote information based on a MISID. The template uses Bootstrap CSS classes for layout.

```

    deduct.innerText = response.incQuote.deductible;
    respElem.style.display = "block";
} else if (this.readyState === XMLHttpRequest.DONE && this.status === 400) {
    err.style.display = "block";
}
}
xhr.send();
</script>
<div class="container">
    <div class="row header">
        <div class="col-sm-12">INSinc Insurance Portal</div>
    </div>
    <div class="row">
        <div class="col-sm-1"></div>
        <label for="msid" class="col-sm-5 col-form-label al-right">Enter MISID to get your application status:</label>
        <div class="col-sm-5">
            <input id="msid" class="form-control" type="number">
            <hr>
            <button class="btn btn-primary" onclick="getApplication()">Get Application</button>
        </div>
        <div class="col-sm-1"></div>
    </div>
    <hr>
    <hr>
    <div id="error">
        Application not found in the database!
    </div>
    <div id="response">
        <div class="row">

```

Figure 43: insinc.ejs (Insurance Portal)



This screenshot continues the code from Figure 43. It shows the implementation of the 'getApplication()' function. The function constructs a URL with a query parameter 'msid', creates an XMLHttpRequest, and sets its headers to 'Content-Type: application/json; utf-8'. It then defines several variables to store elements from the DOM and the JSON response. The 'onreadystatechange' event listener checks if the request is successful (status 200). If so, it parses the JSON response, logs it to the console, and updates various DOM elements with the retrieved data.

```

    <script>
        function getApplication() {
            var url = "http://rockers-insurance.southcentralus.cloudapp.azure.com/insurance/quote";
            var msid = document.getElementById("msid").value;
            var requestUrl = url + "?msid=" + msid;
            var xhr = new XMLHttpRequest();
            xhr.open("GET", requestUrl, true);
            xhr.setRequestHeader("Content-Type", "application/json; utf-8");

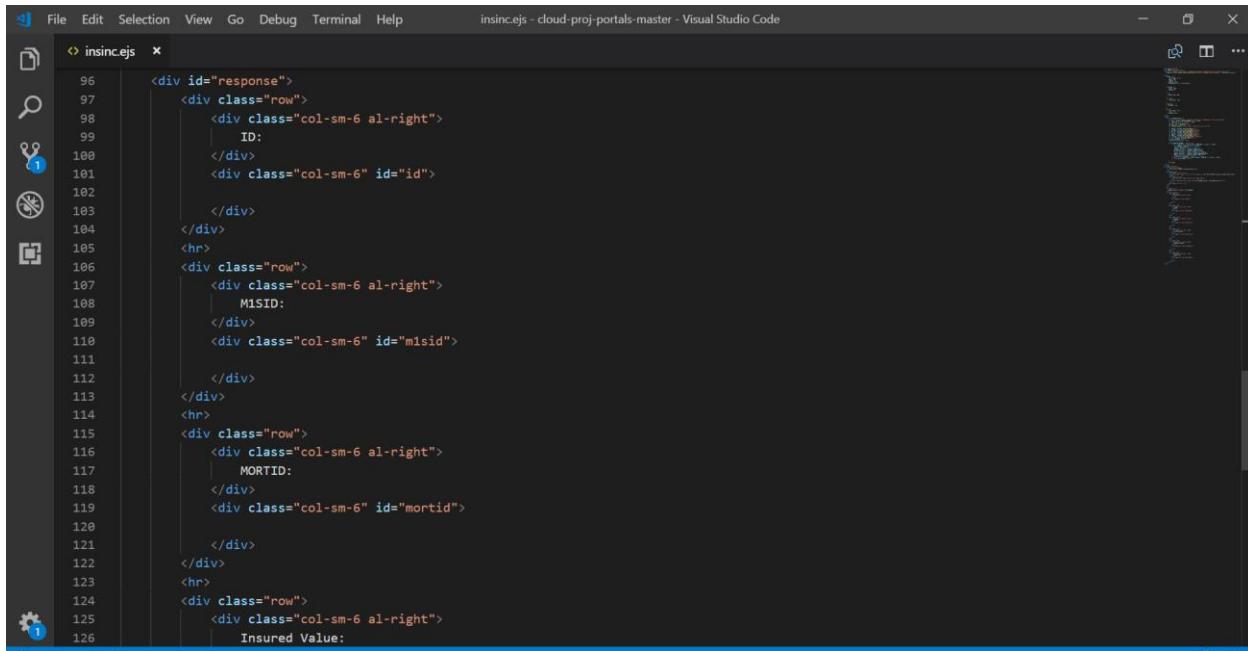
            var idElem = document.getElementById("id");
            var msidElem = document.getElementById("msid");
            var mortId = document.getElementById("mortid");
            var insval = document.getElementById("insval");
            var apprval = document.getElementById("apprval");
            var deduct = document.getElementById("deduct");
            var respElem = document.getElementById("response");
            var err = document.getElementById("error");
            err.style.display = "none";
            respElem.style.display = "none";

            xhr.onreadystatechange = function () {
                if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
                    let response = JSON.parse(this.response);
                    console.log(response);
                    idElem.innerText = response.incQuote.id;
                    msidElem.innerText = response.incQuote.msid;
                    mortId.innerText = response.incQuote.mortid;
                    insval.innerText = response.incQuote.insuredValue;
                    apprval.innerText = response.incQuote.appraisalValue;
                    deduct.innerText = response.incQuote.deductible;
                    respElem.style.display = "block";
                }
            }
        }
    </script>

```

Figure 44: insinc.ejs (contd.)

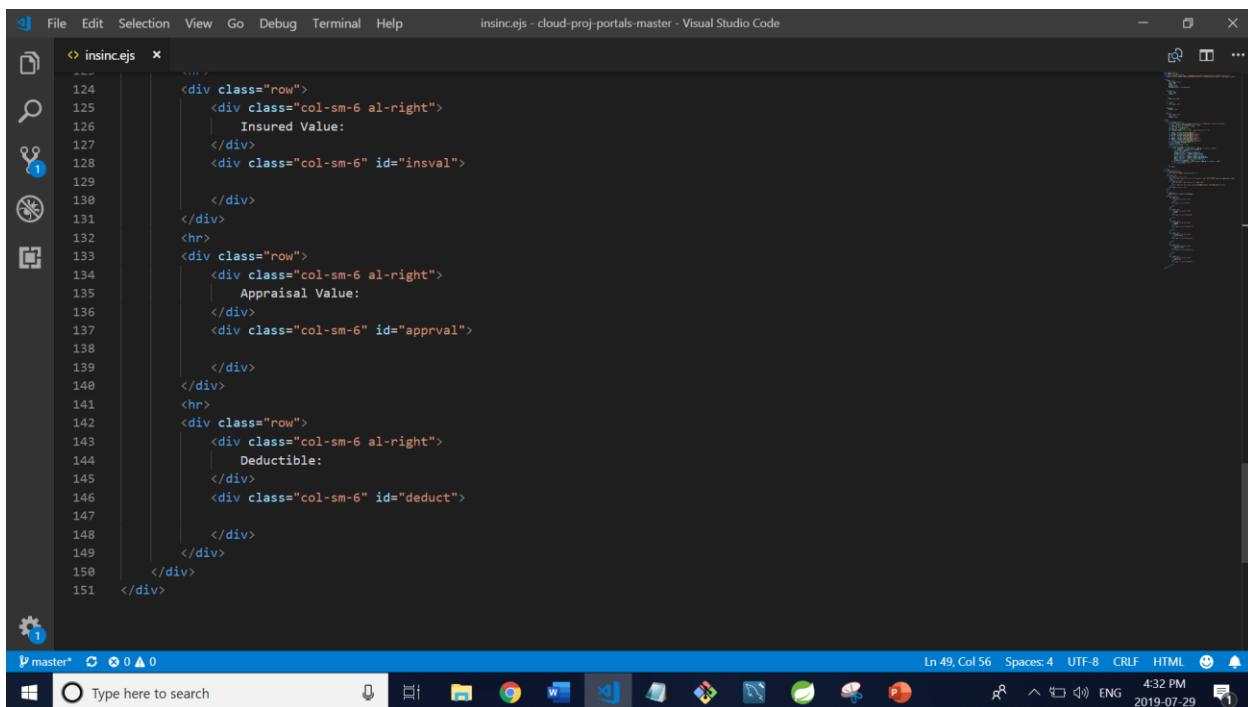
## Cloud Computing – CSCI5409



A screenshot of the Visual Studio Code interface showing the file 'insinc.ejs'. The code is an EJS template with several rows of HTML-like syntax. The code includes div elements with classes like 'row' and 'col-sm-6', and IDs like 'response', 'id', 'm1sid', 'mortid', 'insval', 'apprval', and 'deduct'. The code is heavily indented, and the line numbers from 96 to 126 are visible on the left.

```
<div id="response">
  <div class="row">
    <div class="col-sm-6 al-right">
      | ID:
      |</div>
    <div class="col-sm-6" id="id">
      |</div>
    </div>
  </div>
  <hr>
  <div class="row">
    <div class="col-sm-6 al-right">
      | M1SID:
      |</div>
    <div class="col-sm-6" id="m1sid">
      |</div>
    </div>
  </div>
  <hr>
  <div class="row">
    <div class="col-sm-6 al-right">
      | MORTID:
      |</div>
    <div class="col-sm-6" id="mortid">
      |</div>
    </div>
  </div>
  <hr>
  <div class="row">
    <div class="col-sm-6 al-right">
      | Insured Value:
      |</div>
    <div class="col-sm-6" id="insval">
      |</div>
    </div>
  </div>
  <hr>
  <div class="row">
    <div class="col-sm-6 al-right">
      | Appraisal Value:
      |</div>
    <div class="col-sm-6" id="apprval">
      |</div>
    </div>
  </div>
  <hr>
  <div class="row">
    <div class="col-sm-6 al-right">
      | Deductible:
      |</div>
    <div class="col-sm-6" id="deduct">
      |</div>
    </div>
  </div>
</div>
```

Figure 45: insinc.ejs (contd.)



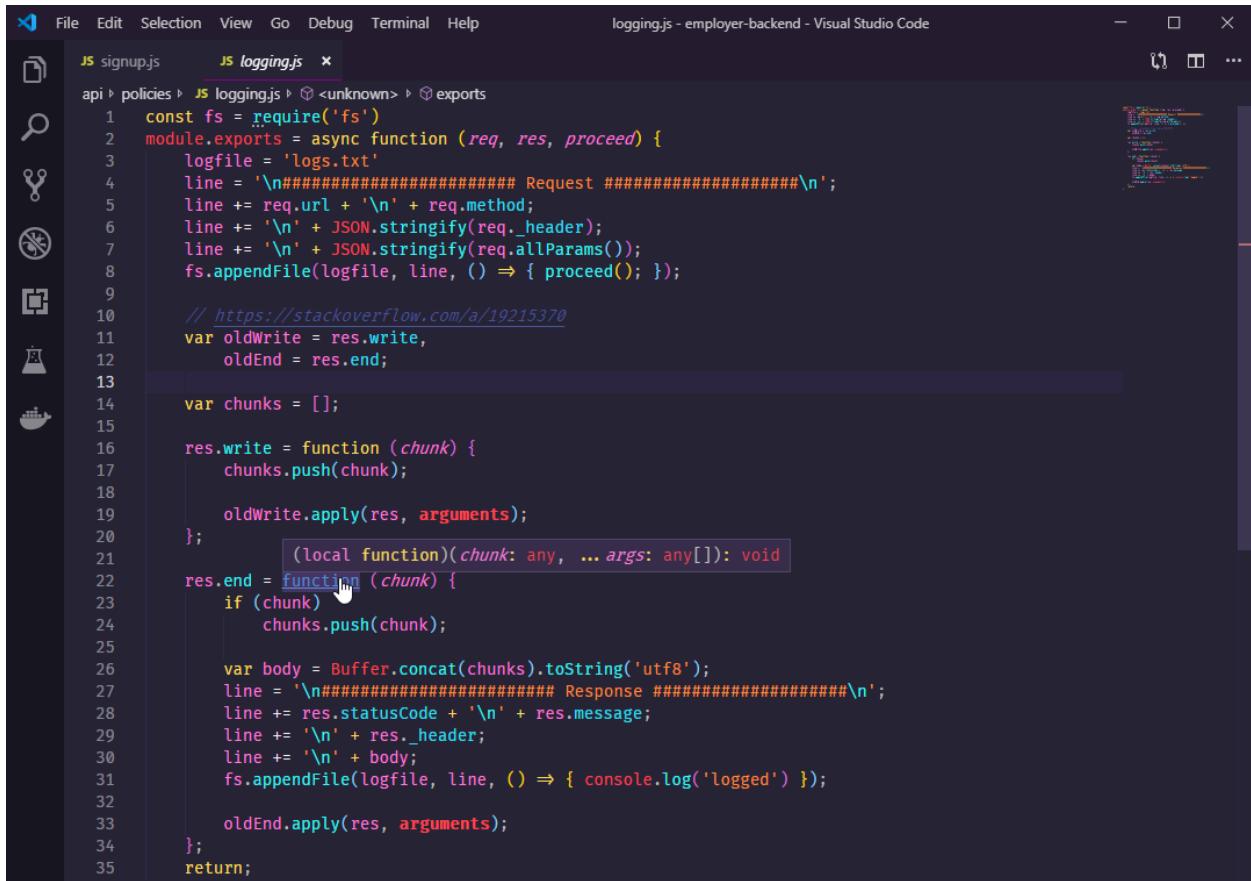
A screenshot of the Visual Studio Code interface showing the file 'insinc.ejs'. The code is identical to Figure 45. At the bottom of the screen, the 'Terminal' tab is active, showing the command line interface with the following information: master\*, 0, 0, Type here to search, and a toolbar with various icons. The status bar at the bottom right shows 'Ln 49, Col 56 Spaces: 4 UTF-8 CRLF HTML' and the date and time '2019-07-29 4:32 PM'.

```
<div class="row">
  <div class="col-sm-6 al-right">
    | Insured Value:
    |</div>
  <div class="col-sm-6" id="insval">
    |</div>
  </div>
  <hr>
  <div class="row">
    <div class="col-sm-6 al-right">
      | Appraisal Value:
      |</div>
    <div class="col-sm-6" id="apprval">
      |</div>
    </div>
  </div>
  <hr>
  <div class="row">
    <div class="col-sm-6 al-right">
      | Deductible:
      |</div>
    <div class="col-sm-6" id="deduct">
      |</div>
    </div>
  </div>
</div>
```

Figure 46: insinc.ejs (contd.)

## WEB SERVICE CODE

## EMPLOYER SERVICE

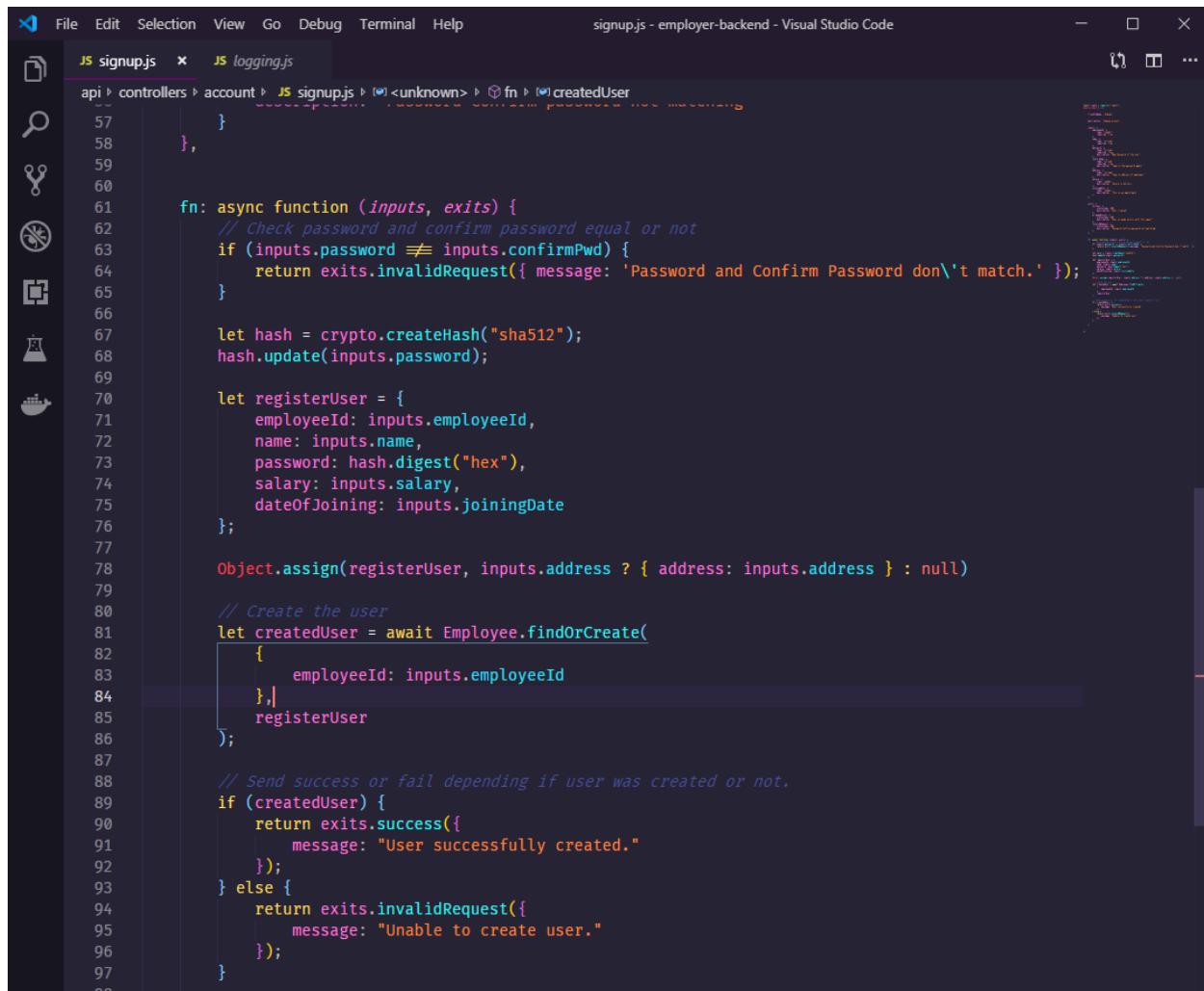


The screenshot shows a Visual Studio Code interface with a dark theme. The title bar reads "logging.js - employer-backend - Visual Studio Code". The left sidebar has icons for file operations like Open, Save, Find, and Refresh. The main editor area contains the following code:

```
JS signup.js      JS logging.js x
api > policies > JS logging.js > <unknown> > exports
1  const fs = require('fs')
2  module.exports = async function (req, res, proceed) {
3    logfile = 'logs.txt'
4    line = '\n##### Request #####\n';
5    line += req.url + '\n' + req.method;
6    line += '\n' + JSON.stringify(req._header);
7    line += '\n' + JSON.stringify(req.allParams());
8    fs.appendFile(logfile, line, () => { proceed(); });
9
10   // https://stackoverflow.com/a/19215370
11   var oldWrite = res.write,
12       oldEnd = res.end;
13
14   var chunks = [];
15
16   res.write = function (chunk) {
17     chunks.push(chunk);
18
19     oldWrite.apply(res, arguments);
20   };
21   res.end = function (chunk) {
22     if (chunk)
23       chunks.push(chunk);
24
25     var body = Buffer.concat(chunks).toString('utf8');
26     line = '\n##### Response #####\n';
27     line += res.statusCode + '\n' + res.message;
28     line += '\n' + res._header;
29     line += '\n' + body;
30     fs.appendFile(logfile, line, () => { console.log('logged') });
31
32     oldEnd.apply(res, arguments);
33   };
34
35   return;
```

Figure 47: logging.js

## Cloud Computing – CSCI5409

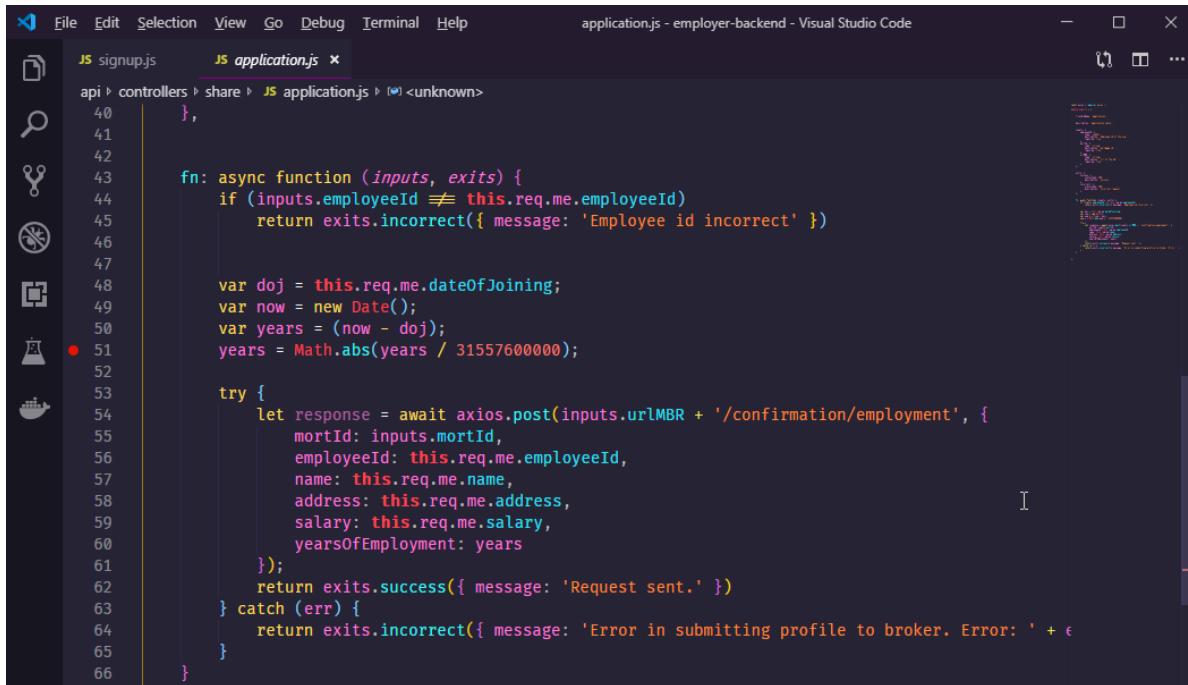


A screenshot of the Visual Studio Code interface showing the `signup.js` file. The code is written in JavaScript and defines a function `fn` that checks if the password and confirm password are equal. It then creates a hash of the password using SHA512 and constructs a user object with employee ID, name, password, salary, and joining date. The user object is then assigned to the `registerUser` variable. The code then attempts to find or create a user with the same employee ID. If successful, it returns a success message; if not, it returns an invalid request message.

```
File Edit Selection View Go Debug Terminal Help
signup.js - employer-backend - Visual Studio Code
JS signup.js x JS logging.js
api > controllers > account > JS signup.js > <unknown> > fn > fn.createdUser
57     },
58 },
59 },
60 },
61 fn: async function (inputs, exits) {
62     // Check password and confirm password equal or not
63     if (inputs.password !== inputs.confirmPwd) {
64         return exits.invalidRequest({ message: 'Password and Confirm Password don\'t match.' });
65     }
66
67     let hash = crypto.createHash("sha512");
68     hash.update(inputs.password);
69
70     let registerUser = {
71         employeeId: inputs.employeeId,
72         name: inputs.name,
73         password: hash.digest("hex"),
74         salary: inputs.salary,
75         dateOfJoining: inputs.joiningDate
76     };
77
78     Object.assign(registerUser, inputs.address ? { address: inputs.address } : null)
79
80     // Create the user
81     let createdUser = await Employee.findOrCreate(
82         {
83             employeeId: inputs.employeeId
84         },
85         registerUser
86     );
87
88     // Send success or fail depending if user was created or not.
89     if (createdUser) {
90         return exits.success({
91             message: "User successfully created."
92         });
93     } else {
94         return exits.invalidRequest({
95             message: "Unable to create user."
96         });
97     }
98 }
```

Figure 48: `signup.js`

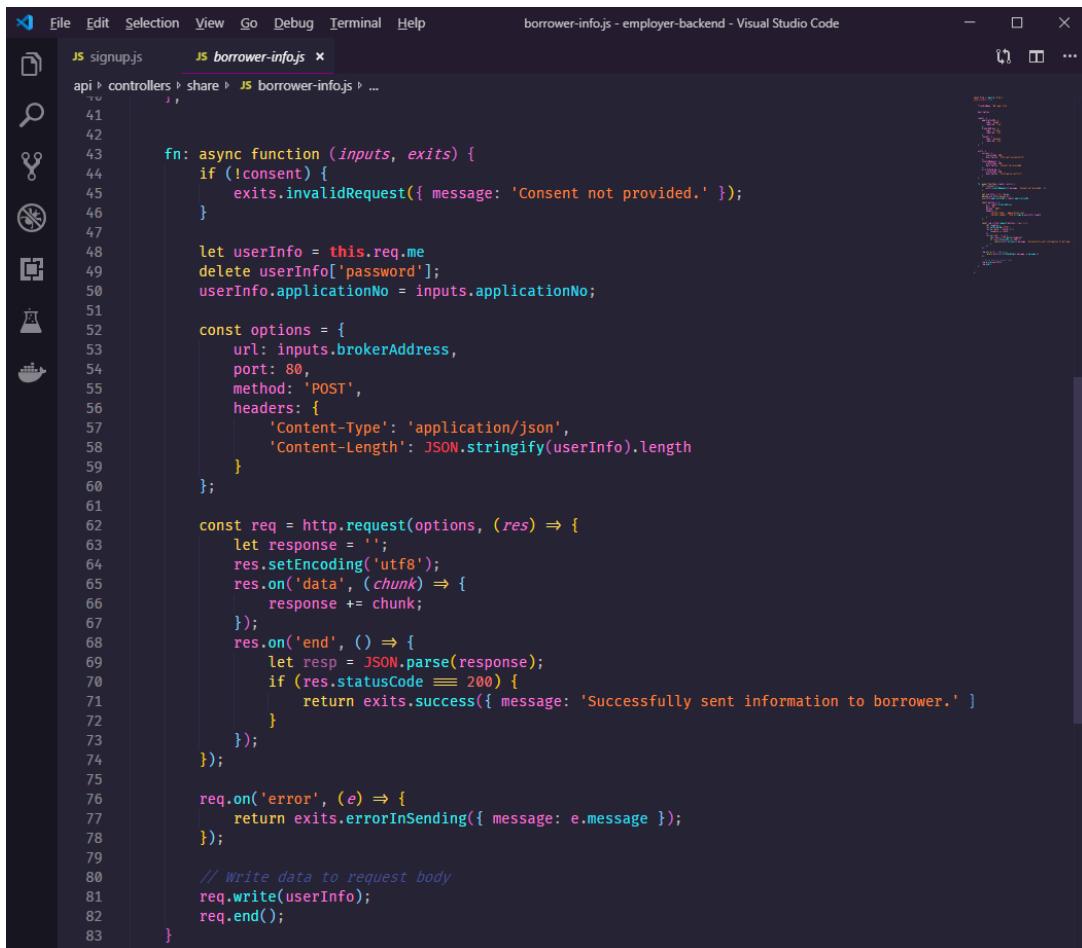
## Cloud Computing – CSCI5409



A screenshot of the Visual Studio Code interface showing the `application.js` file. The code is written in JavaScript and defines an asynchronous function `fn` that takes `inputs` and `exits` as parameters. It checks if the employee ID in `inputs` matches the one in `this.req.me.employeeId`. If not, it returns an error message. It then calculates the years of employment by subtracting the date of joining from the current date. A try-catch block is used to send a POST request to a confirmation endpoint with the calculated years and other user information. If successful, it returns a success message; otherwise, it returns an error message.

```
JS signup.js JS application.js
api > controllers > share > JS application.js > <unknown>
40     },
41
42
43     fn: async function (inputs, exits) {
44         if (inputs.employeeId !== this.req.me.employeeId)
45             return exits.incorrect({ message: 'Employee id incorrect' })
46
47         var doj = this.req.me.dateOfJoining;
48         var now = new Date();
49         var years = (now - doj);
50         years = Math.abs(years / 31557600000);
51
52         try {
53             let response = await axios.post(inputs.urlMBR + '/confirmation/employment', {
54                 mortId: inputs.mortId,
55                 employeeId: this.req.me.employeeId,
56                 name: this.req.me.name,
57                 address: this.req.me.address,
58                 salary: this.req.me.salary,
59                 yearsOfEmployment: years
60             });
61             return exits.success({ message: 'Request sent.' })
62         } catch (err) {
63             return exits.incorrect({ message: 'Error in submitting profile to broker. Error: ' + err.message })
64         }
65     }
66 }
```

Figure 49: application.js

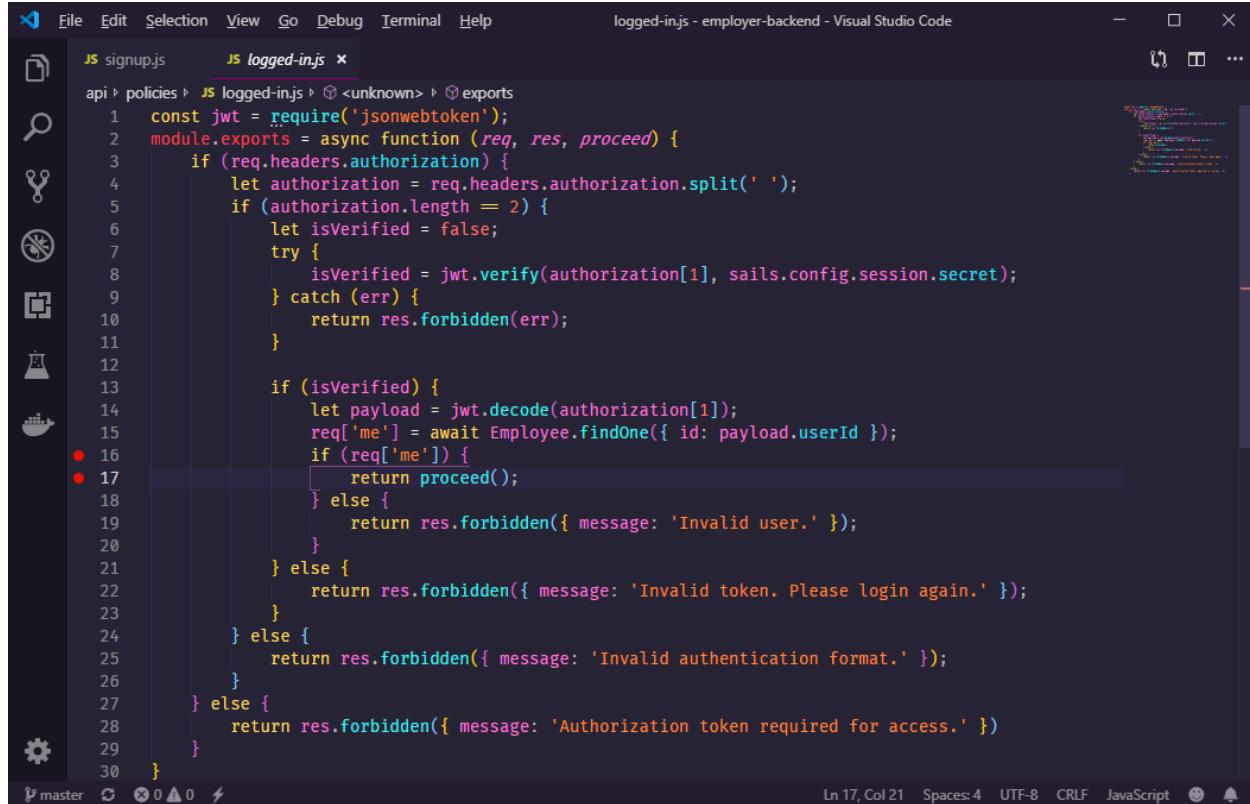


A screenshot of the Visual Studio Code interface showing the `borrower-info.js` file. The code is written in JavaScript and defines an asynchronous function `fn` that takes `inputs` and `exits` as parameters. It first checks if consent is provided. If not, it returns an invalid request error. It then creates a user info object by deleting the password from the req.me object and setting the application number. It constructs an options object for an HTTP POST request to a broker address. The request body is the JSON stringified user info. The response is read chunk by chunk until the end, and if successful (status code 200), it returns a success message. If there's an error during the request, it returns an error message.

```
JS signup.js JS borrower-info.js
api > controllers > share > JS borrower-info.js > ...
41
42
43     fn: async function (inputs, exits) {
44         if (!consent) {
45             exits.invalidRequest({ message: 'Consent not provided.' });
46         }
47
48         let userInfo = this.req.me;
49         delete userInfo['password'];
50         userInfo.applicationNo = inputs.applicationNo;
51
52         const options = {
53             url: inputs.brokerAddress,
54             port: 80,
55             method: 'POST',
56             headers: {
57                 'Content-Type': 'application/json',
58                 'Content-Length': JSON.stringify(userInfo).length
59             }
60         };
61
62         const req = http.request(options, (res) => {
63             let response = '';
64             res.setEncoding('utf8');
65             res.on('data', (chunk) => {
66                 response += chunk;
67             });
68             res.on('end', () => {
69                 let resp = JSON.parse(response);
70                 if (res.statusCode === 200) {
71                     return exits.success({ message: 'Successfully sent information to borrower.' })
72                 }
73             });
74         });
75
76         req.on('error', (e) => {
77             return exits.errorInSending({ message: e.message });
78         });
79
80         // Write data to request body
81         req.write(userInfo);
82         req.end();
83     }
84 }
```

Figure 50: borrower-info.js

## Cloud Computing – CSCI5409

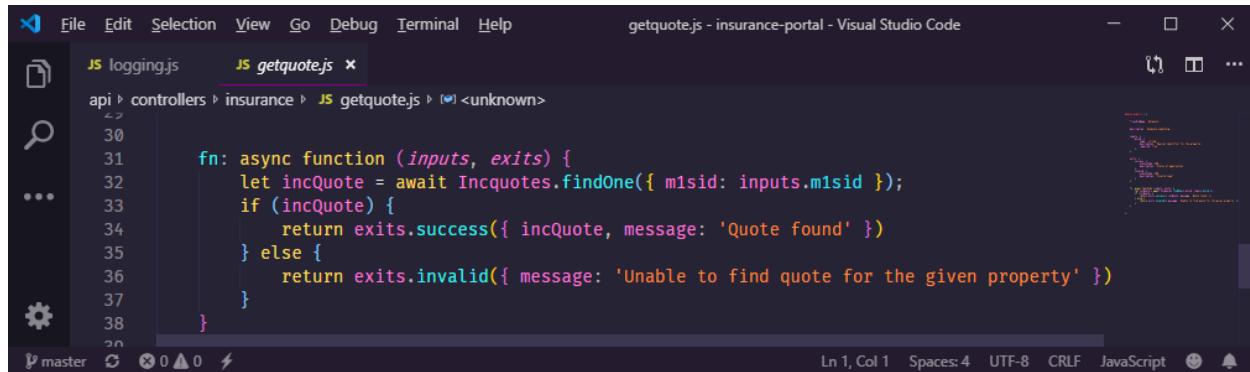


A screenshot of Visual Studio Code showing the `logged-in.js` file. The code is a Node.js module for a Sails.js application. It contains logic to verify an authorization token and check if the user is authenticated. The code uses the `jsonwebtoken` library to decode the token and the `sails.config.session.secret` to verify it. If the token is valid, it checks if the user exists in the database. If the user exists, it proceeds with the request; otherwise, it returns a forbidden response. If the token is invalid or does not exist, it returns a forbidden response with a message. If the authentication format is invalid, it also returns a forbidden response.

```
const jwt = require('jsonwebtoken');
module.exports = async function (req, res, proceed) {
  if (req.headers.authorization) {
    let authorization = req.headers.authorization.split(' ');
    if (authorization.length == 2) {
      let isVerified = false;
      try {
        isVerified = jwt.verify(authorization[1], sails.config.session.secret);
      } catch (err) {
        return res.forbidden(err);
      }
    }
    if (isVerified) {
      let payload = jwt.decode(authorization[1]);
      req['me'] = await Employee.findOne({ id: payload.userId });
      if (req['me']) {
        return proceed();
      } else {
        return res.forbidden({ message: 'Invalid user.' });
      }
    } else {
      return res.forbidden({ message: 'Invalid token. Please login again.' });
    }
  } else {
    return res.forbidden({ message: 'Invalid authentication format.' });
  }
} else {
  return res.forbidden({ message: 'Authorization token required for access.' })
}
```

Figure 51: logged-in.js

### INSURANCE SERVICE

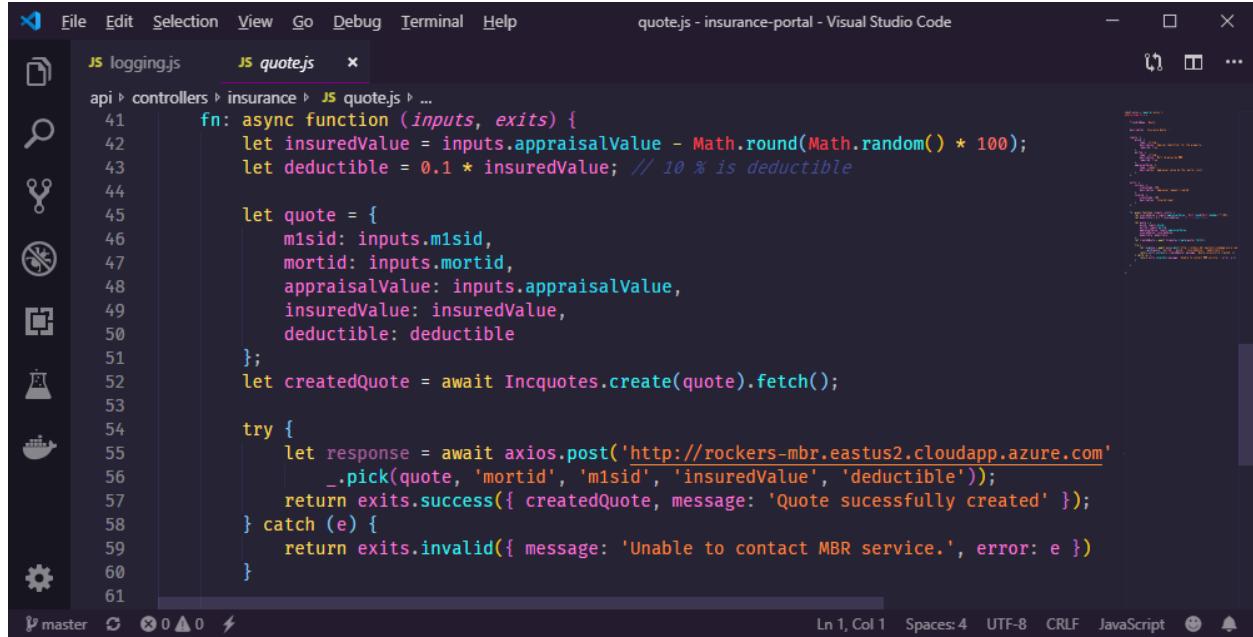


A screenshot of Visual Studio Code showing the `getquote.js` file. The code is a Node.js module for an insurance portal. It defines a function `fn` that takes `inputs` and `exits` as parameters. It uses the `Inquotes` model to find a quote by `m1sid`. If a quote is found, it returns a success response with the quote details. If no quote is found, it returns an invalid response with a message indicating that the quote could not be found for the given property.

```
fn: async function (inputs, exits) {
  let incQuote = await Inquotes.findOne({ m1sid: inputs.m1sid });
  if (incQuote) {
    return exits.success({ incQuote, message: 'Quote found' })
  } else {
    return exits.invalid({ message: 'Unable to find quote for the given property' })
  }
}
```

Figure 52: getquote.js

## Cloud Computing – CSCI5409



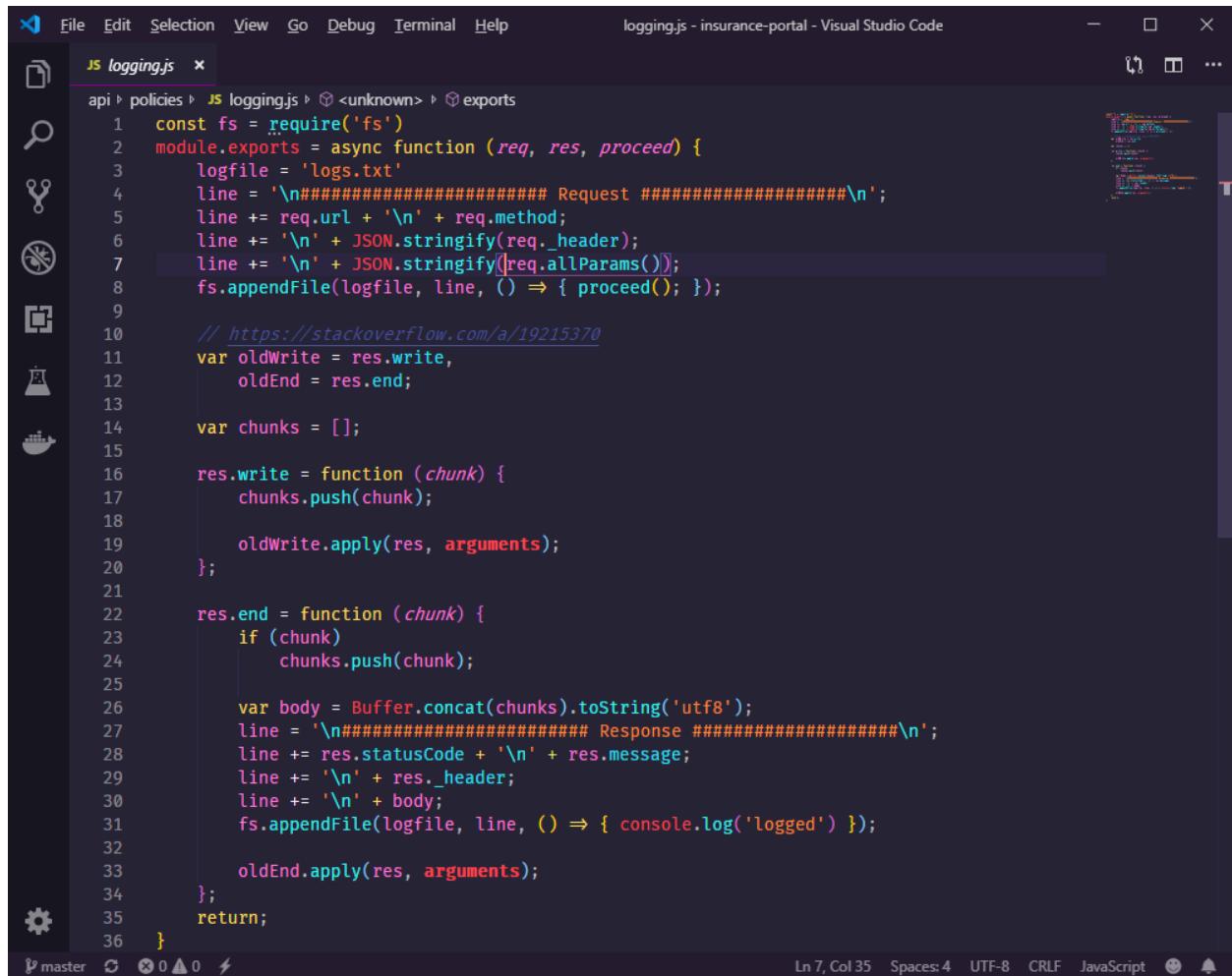
```

File Edit Selection View Go Debug Terminal Help
quote.js - insurance-portal - Visual Studio Code
JS logging.js JS quote.js x
api > controllers > insurance > JS quote.js > ...
41     fn: async function (inputs, exits) {
42         let insuredValue = inputs.appraisalValue - Math.round(Math.random() * 100);
43         let deductible = 0.1 * insuredValue; // 10 % is deductible
44
45         let quote = {
46             m1sid: inputs.m1sid,
47             mortid: inputs.mortid,
48             appraisalValue: inputs.appraisalValue,
49             insuredValue: insuredValue,
50             deductible: deductible
51         };
52         let createdQuote = await Incquotes.create(quote).fetch();
53
54         try {
55             let response = await axios.post('http://rockers-mbr.eastus2.cloudapp.azure.com'
56                 .pick, 'mortid', 'm1sid', 'insuredValue', 'deductible'));
57             return exits.success({ createdQuote, message: 'Quote sucessfully created' });
58         } catch (e) {
59             return exits.invalid({ message: 'Unable to contact MBR service.', error: e })
60         }
61     }

```

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF JavaScript

Figure 53: quote.js



```

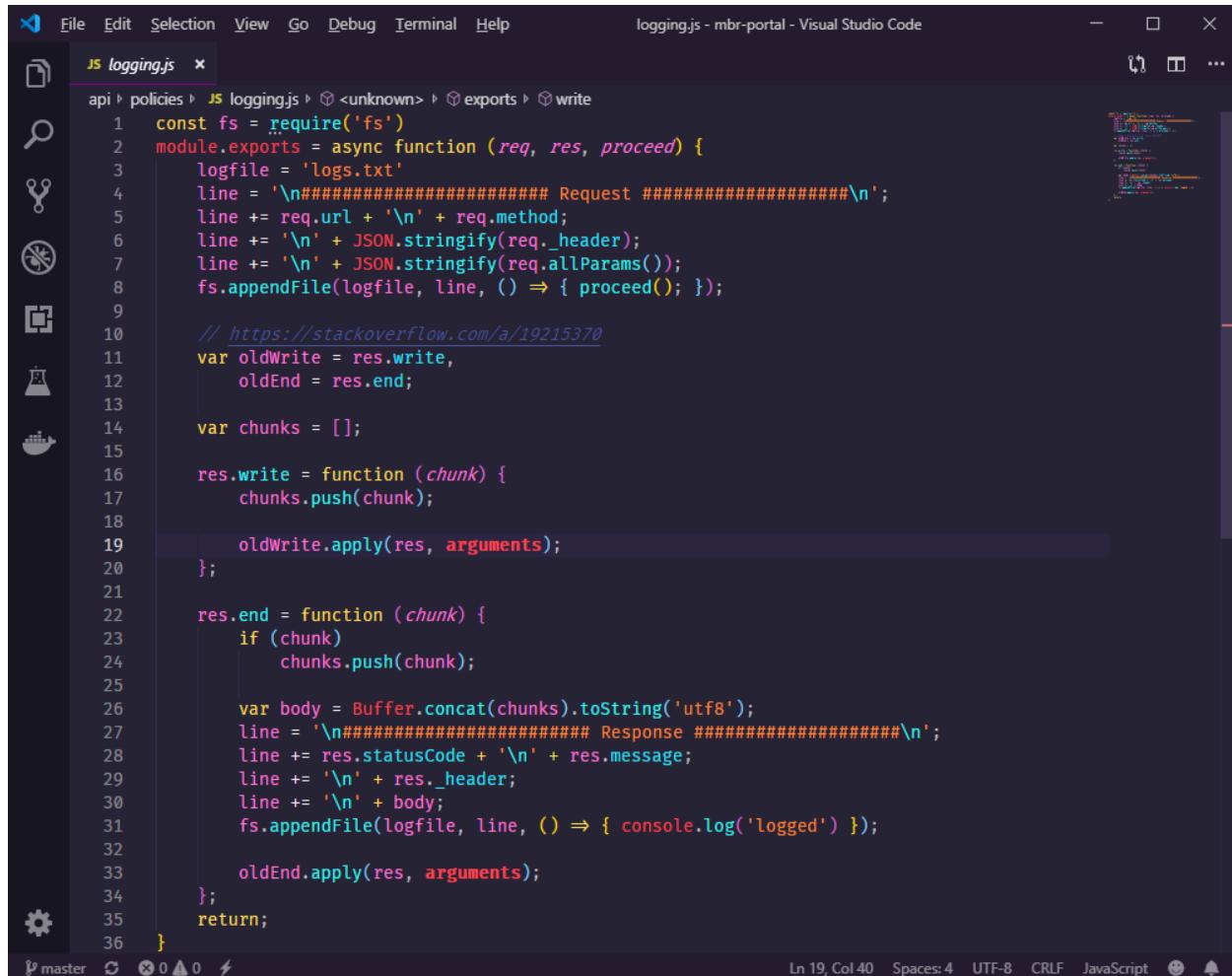
File Edit Selection View Go Debug Terminal Help
logging.js - insurance-portal - Visual Studio Code
JS logging.js x
api > policies > JS logging.js > <unknown> > exports
1 const fs = require('fs')
2 module.exports = async function (req, res, proceed) {
3     logfile = 'logs.txt'
4     line = '\n##### Request #####\n';
5     line += req.url + '\n' + req.method;
6     line += '\n' + JSON.stringify(req._header);
7     line += '\n' + JSON.stringify(req.allParams());
8     fs.appendFile(logfile, line, () => { proceed(); });
9
10    // https://stackoverflow.com/a/19215370
11    var oldWrite = res.write,
12        oldEnd = res.end;
13
14    var chunks = [];
15
16    res.write = function (chunk) {
17        chunks.push(chunk);
18
19        oldWrite.apply(res, arguments);
20    };
21
22    res.end = function (chunk) {
23        if (chunk)
24            chunks.push(chunk);
25
26        var body = Buffer.concat(chunks).toString('utf8');
27        line = '\n##### Response #####\n';
28        line += res.statusCode + '\n' + res.message;
29        line += '\n' + res._header;
30        line += '\n' + body;
31        fs.appendFile(logfile, line, () => { console.log('logged') });
32
33        oldEnd.apply(res, arguments);
34    };
35
36}

```

Ln 7, Col 35 Spaces: 4 UTF-8 CRLF JavaScript

Figure 54: logging.js

## MBR BROKER SERVICE



The screenshot shows the Visual Studio Code interface with the file `logging.js` open. The code is a middleware function for a Node.js application. It uses the `fs.appendFile` method to log requests to a file named `logs.txt`. The logs include the URL, method, headers, and query parameters. It also captures the response body and logs it at the end. The code is annotated with comments explaining its purpose.

```

const fs = require('fs')
module.exports = async function (req, res, proceed) {
    const logfile = 'logs.txt'
    let line = '\n##### Request #####\n';
    line += req.url + '\n' + req.method;
    line += '\n' + JSON.stringify(req._header);
    line += '\n' + JSON.stringify(req.allParams());
    fs.appendFile(logfile, line, () => { proceed(); });

    // https://stackoverflow.com/a/19215370
    var oldwrite = res.write,
        oldEnd = res.end;

    var chunks = [];

    res.write = function (chunk) {
        chunks.push(chunk);
    };

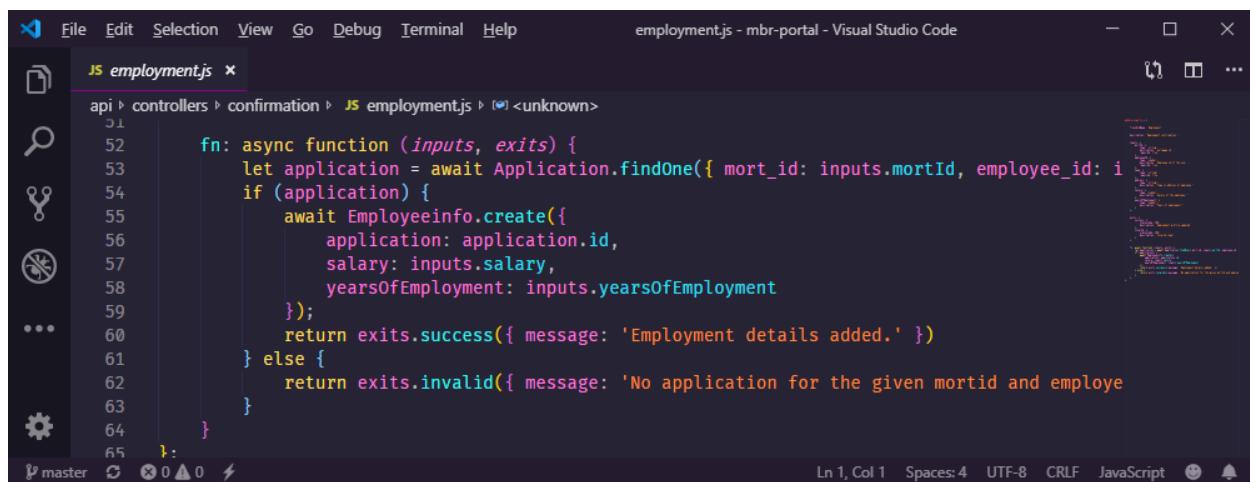
    oldWrite.apply(res, arguments);

    res.end = function (chunk) {
        if (chunk)
            chunks.push(chunk);

        let body = Buffer.concat(chunks).toString('utf8');
        line = '\n##### Response #####\n';
        line += res.statusCode + '\n' + res.message;
        line += '\n' + res._header;
        line += '\n' + body;
        fs.appendFile(logfile, line, () => { console.log('logged') });
        oldEnd.apply(res, arguments);
    };
    return;
}

```

Figure 55: logging.js



The screenshot shows the Visual Studio Code interface with the file `employment.js` open. The code is an asynchronous function that adds employment details to a database. It finds an application by `mort_id` and `employee_id`, creates a new `Employeeinfo` record with `application.id`, `salary`, and `yearsOfEmployment`, and returns a success message. If no application is found, it returns an invalid message.

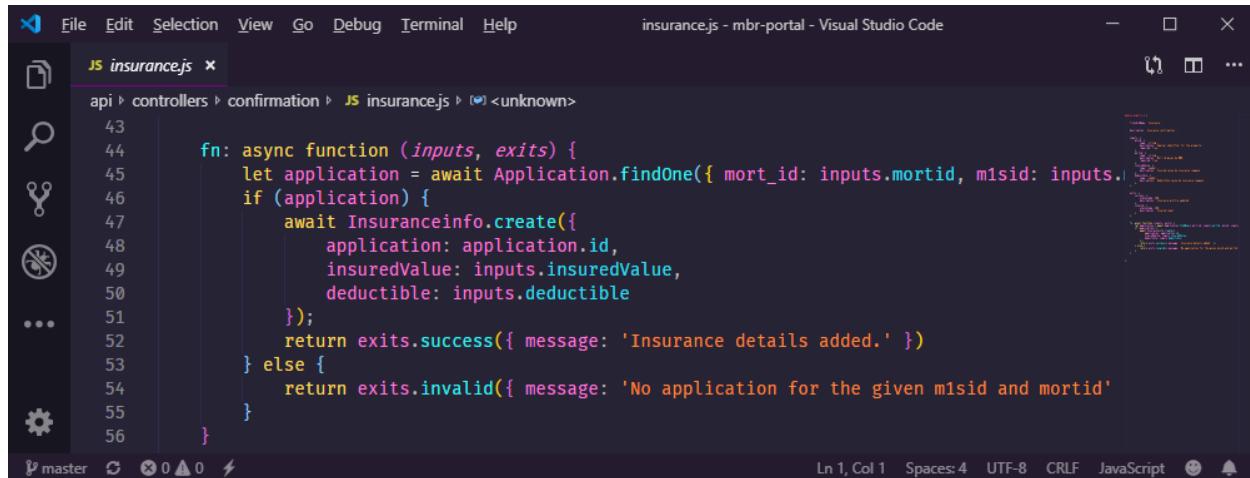
```

fn: async function (inputs, exits) {
    let application = await Application.findOne({ mort_id: inputs.mortId, employee_id: inputs.employeeId });
    if (application) {
        await Employeeinfo.create({
            application: application.id,
            salary: inputs.salary,
            yearsOfEmployment: inputs.yearsOfEmployment
        });
        return exits.success({ message: 'Employment details added.' });
    } else {
        return exits.invalid({ message: 'No application for the given mortid and employee id' });
    }
}

```

Figure 56: employment.js

## Cloud Computing – CSCI5409

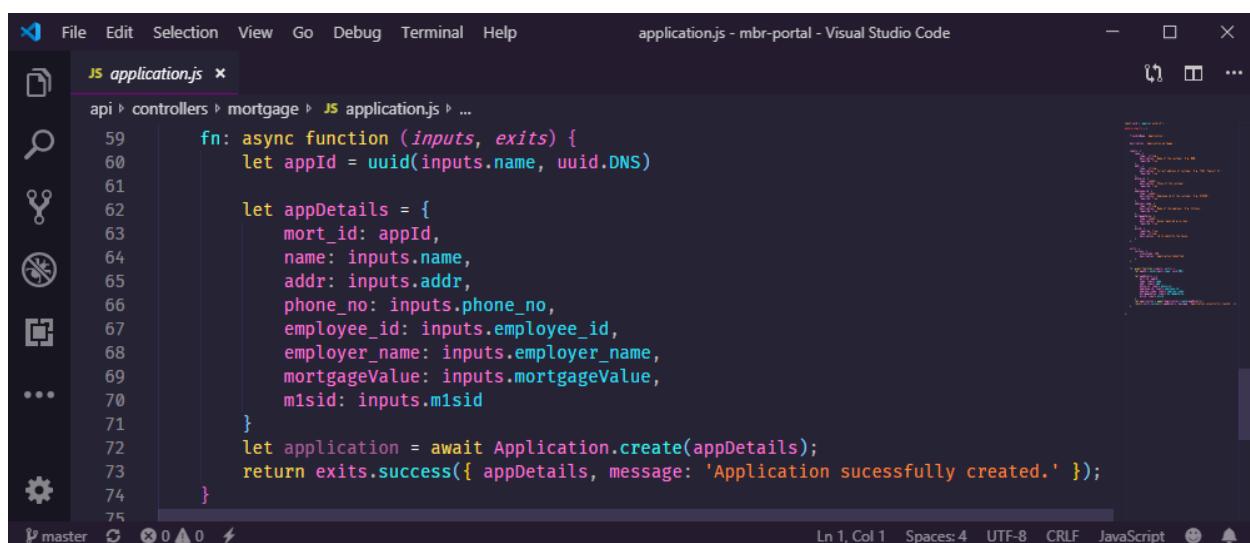


A screenshot of Visual Studio Code showing the file `insurance.js`. The code is part of a function `fn` that handles insurance details. It uses `await Application.findOne` to find an application by `mort_id` and `m1sid`. If found, it creates a new `Insuranceinfo` object with `application.id`, `insuredValue`, and `deductible`. If no application is found, it returns an invalid response. The code is written in JavaScript.

```
43
44     fn: async function (inputs, exits) {
45         let application = await Application.findOne({ mort_id: inputs.mortid, m1sid: inputs.m1sid });
46         if (application) {
47             await Insuranceinfo.create({
48                 application: application.id,
49                 insuredValue: inputs.insuredValue,
50                 deductible: inputs.deductible
51             });
52             return exits.success({ message: 'Insurance details added.' })
53         } else {
54             return exits.invalid({ message: 'No application for the given m1sid and mortid' })
55         }
56     }

```

Figure 57: insurance.js

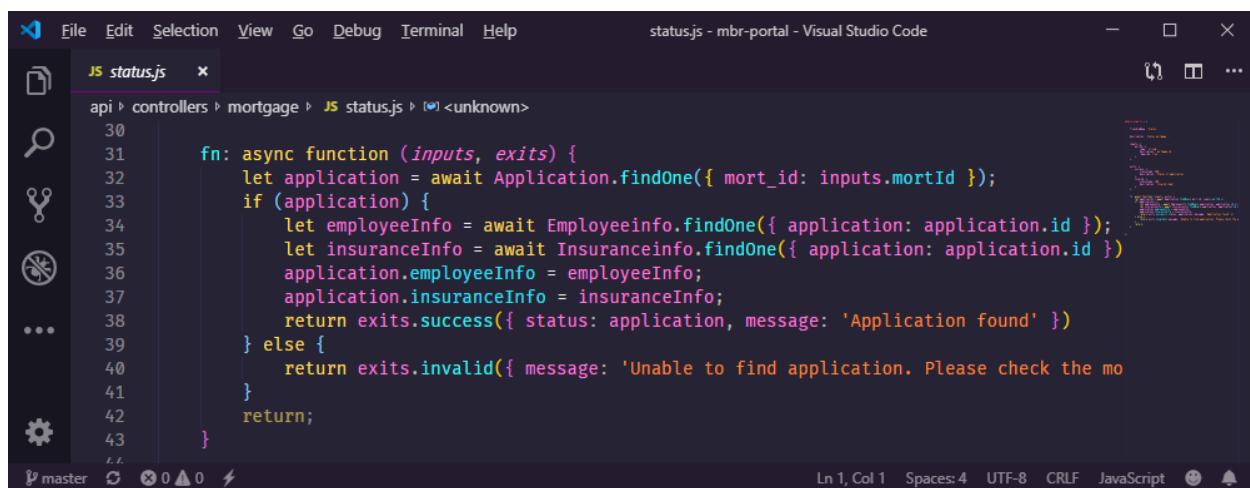


A screenshot of Visual Studio Code showing the file `application.js`. The code is part of a function `fn` that creates a new application. It generates an `appId` using `uuid` and creates an `appDetails` object with fields like `mort_id`, `name`, `addr`, `phone_no`, `employee_id`, `employer_name`, `mortgageValue`, and `m1sid`. It then calls `Application.create` with `appDetails` and returns a success response. The code is written in JavaScript.

```
59
60     fn: async function (inputs, exits) {
61         let appId = uuid(inputs.name, uuid.DNS)
62
63         let appDetails = {
64             mort_id: appId,
65             name: inputs.name,
66             addr: inputs.addr,
67             phone_no: inputs.phone_no,
68             employee_id: inputs.employee_id,
69             employer_name: inputs.employer_name,
70             mortgageValue: inputs.mortgageValue,
71             m1sid: inputs.m1sid
72         }
73         let application = await Application.create(appDetails);
74         return exits.success({ appDetails, message: 'Application sucessfully created.' });
75     }

```

Figure 58: application.js



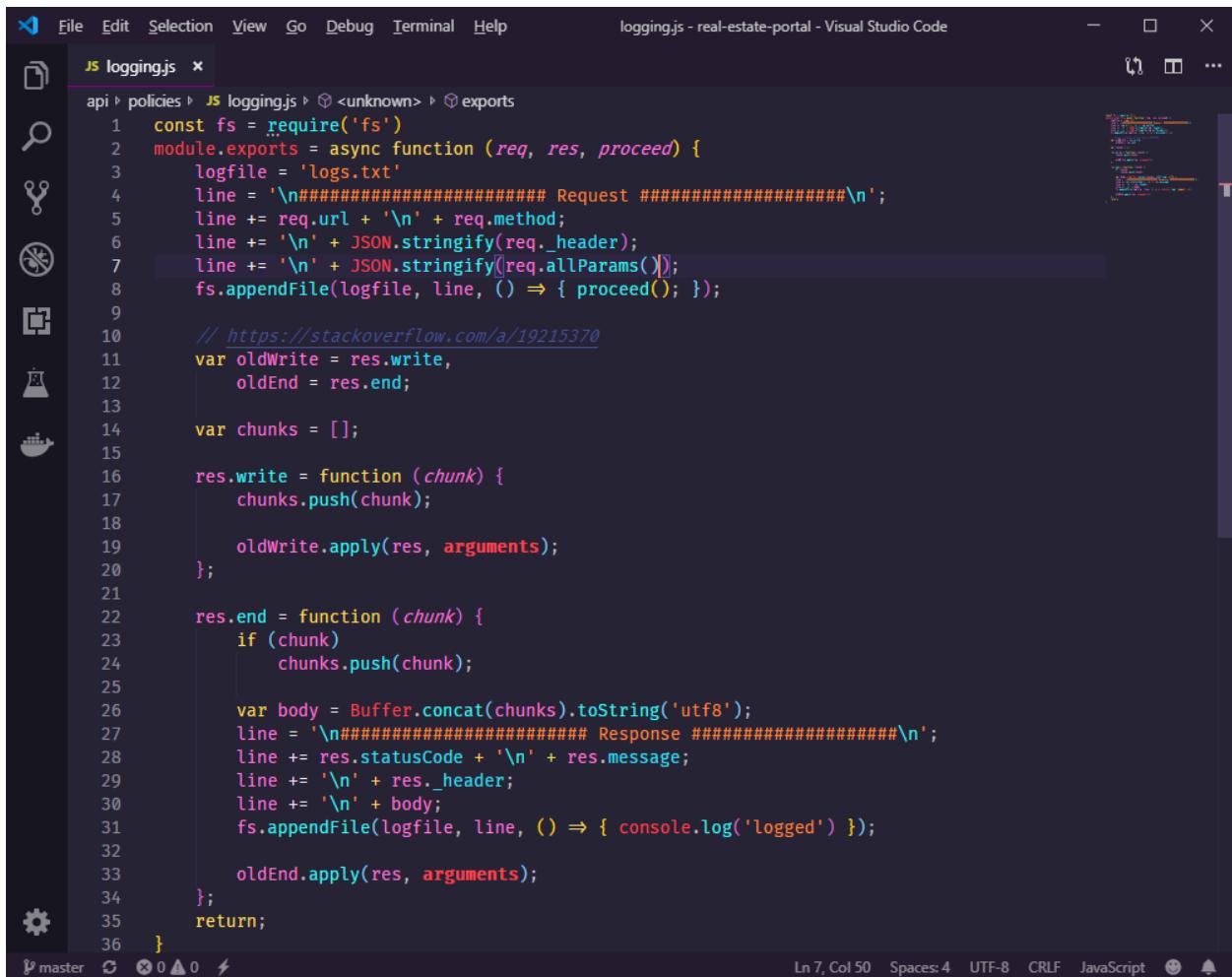
A screenshot of Visual Studio Code showing the file `status.js`. The code is part of a function `fn` that finds an application by `mort_id`. It retrieves the `employeeInfo` and `insuranceInfo` for the application. If found, it returns a success response with the application details. If no application is found, it returns an invalid response. The code is written in JavaScript.

```
30
31     fn: async function (inputs, exits) {
32         let application = await Application.findOne({ mort_id: inputs.mortId });
33         if (application) {
34             let employeeInfo = await Employeeinfo.findOne({ application: application.id });
35             let insuranceInfo = await Insuranceinfo.findOne({ application: application.id });
36             application.employeeInfo = employeeInfo;
37             application.insuranceInfo = insuranceInfo;
38             return exits.success({ status: application, message: 'Application found' })
39         } else {
40             return exits.invalid({ message: 'Unable to find application. Please check the mo
41         }
42         return;
43     }

```

Figure 59: status.js

### REAL ESTATE BROKER WEB SERVICE



The screenshot shows the Visual Studio Code interface with the file `logging.js` open. The code is a middleware function for a Node.js application. It uses the `fs` module to append log entries to a file named `logs.txt`. The logs include the request URL, method, headers, and parameters. It also captures the response body and logs it along with the status code and message. The code is annotated with comments explaining its purpose.

```
const fs = require('fs');
module.exports = async function (req, res, proceed) {
  const logfile = 'logs.txt';
  let line = '\n##### Request #####\n';
  line += req.url + '\n' + req.method;
  line += '\n' + JSON.stringify(req._header);
  line += '\n' + JSON.stringify(req.allParams());
  fs.appendFile(logfile, line, () => { proceed(); });

  // https://stackoverflow.com/a/19215370
  var oldwrite = res.write,
      oldEnd = res.end;
  var chunks = [];

  res.write = function (chunk) {
    chunks.push(chunk);
  };

  oldWrite.apply(res, arguments);
};

res.end = function (chunk) {
  if (chunk)
    chunks.push(chunk);

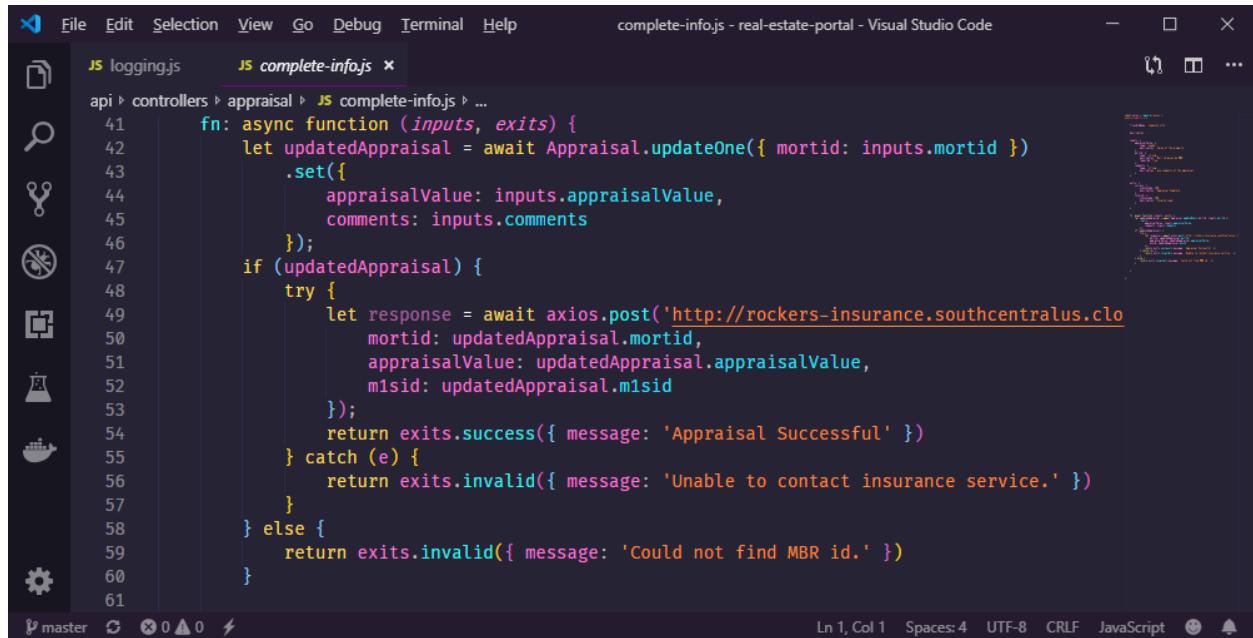
  var body = Buffer.concat(chunks).toString('utf8');
  line = '\n##### Response #####\n';
  line += res.statusCode + '\n' + res.message;
  line += '\n' + res._header;
  line += '\n' + body;
  fs.appendFile(logfile, line, () => { console.log('logged') });
};

oldEnd.apply(res, arguments);
};

return;
}
```

Figure 60: logging.js

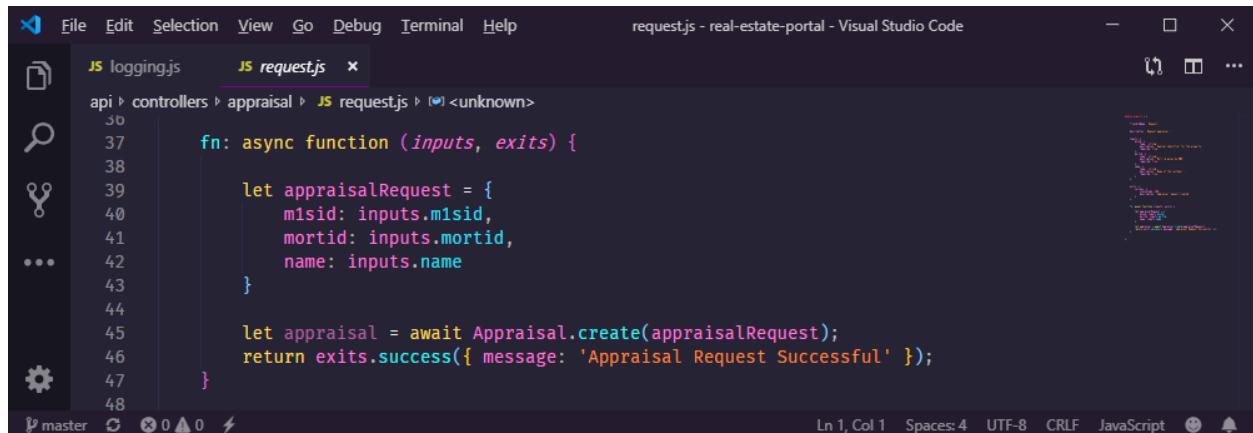
## Cloud Computing – CSCI5409



A screenshot of the Visual Studio Code interface showing the `complete-info.js` file. The code is a JavaScript function within a module named `complete-info.js`. It performs an asynchronous update on an `Appraisal` object and then sends a POST request to an external insurance service using `axios`. If successful, it returns a success message; if there's a connection error, it returns an invalid message; if no MBR ID is found, it returns another invalid message.

```
api > controllers > appraisal > JS complete-info.js > ...
41     fn: async function (inputs, exits) {
42         let updatedAppraisal = await Appraisal.updateOne({ mortid: inputs.mortid })
43         .set({
44             appraisalValue: inputs.appraisalValue,
45             comments: inputs.comments
46         });
47         if (updatedAppraisal) {
48             try {
49                 let response = await axios.post('http://rockers-insurance.southcentralus.cloudapp.azure.com/api/Appraisals')
50                     .set('Content-Type', 'application/json')
51                     .set('Accept', 'application/json')
52                     .set('mortid', updatedAppraisal.mortid,
53                         appraisalValue: updatedAppraisal.appraisalValue,
54                         m1sid: updatedAppraisal.m1sid
55                     );
56                 return exits.success({ message: 'Appraisal Successful' })
57             } catch (e) {
58                 return exits.invalid({ message: 'Unable to contact insurance service.' })
59             }
60         } else {
61             return exits.invalid({ message: 'Could not find MBR id.' })
62         }
63     }
64 }
```

Figure 61: `complete-info.js`



A screenshot of the Visual Studio Code interface showing the `request.js` file. The code is a JavaScript function within a module named `request.js`. It creates a new `AppraisalRequest` object with `m1sid`, `mortid`, and `name` properties. Then it calls the `create` method on the `Appraisal` model to save the request. Finally, it returns a success message.

```
api > controllers > appraisal > JS request.js > <unknown>
36
37     fn: async function (inputs, exits) {
38
39         let appraisalRequest = {
40             m1sid: inputs.m1sid,
41             mortid: inputs.mortid,
42             name: inputs.name
43         }
44
45         let appraisal = await Appraisal.create(appraisalRequest);
46         return exits.success({ message: 'Appraisal Request Successful' });
47     }
48 }
```

Figure 62: `request.js`