

Dalhousie University

Project Report

CSCI 5409 – Cloud Computing

Team Members

#	Name	B00#
1	Ueli Haltner	B00526617
2	Smit Ashish Saraiya	B00811636
3	Chintan Patel	B00826089
4	Shrey Amin	B00822245
5	Ravi Zala	B00805073
6	Deep Nimesh Shah	B00796368

Team Nimbus
7-29-2019

Table of Contents

Table of Figures.....	2
Project Description	4
Front-End Overview	4
Web Service Overview	11
Workflow Overview	19
Software Deficiencies	32
Application Host / URLs	32
Heroku	32
Microsoft Azure	32
Resources.....	32
Hardware	32
Software.....	33
Libraries & Frameworks	33
Tools.....	34
Software.....	34
Code Sources	35
Log Service Configuration	37
Testing.....	37
Project Plan Execution Summary	53
Scalability	54
Security Mechanisms	54
References	57

Table of Figures

Figure 1: MBR portal homepage.....	4
Figure 2: MBR funds request form.....	5
Figure 3: MBR funds request form continued	5
Figure 4: MBR login page for application status	6
Figure 5: MBR application status page	6
Figure 6: Employer portal homepage	7
Figure 7: Mortgage form of employer portal.....	8
Figure 8: Employer mortgage form application error status feedback	8
Figure 9: Real estate portal homepage.....	9
Figure 10: Property assessment application approval form	9
Figure 11: Appraiser login page	10
Figure 12: Property validation page for Appraiser	10
Figure 13: HTTP Request Trigger in Workflow	19
Figure 14: Initialize variable for workflow	20
Figure 15: Employer Web service URL testing for /authenticate Endpoint.....	20
Figure 16: Parsing Response Body JSON.....	21
Figure 17: Evaluating workflow condition	21
Figure 18: Email sent to the user	22
Figure 19: MBR Web service URL testing for /validateApplication Endpoint.....	23
Figure 20: Checking the JSON response.....	23
Figure 21: If status is Accepted, send email.....	24
Figure 22: Status is Rejected.....	24
Figure 23: Validate credential for appraiser	25
Figure 24: Appraiser value submitted to insurance service.....	25
Figure 25: Database server on Microsoft azure.....	26
Figure 26: Local instance of cloud database.....	26
Figure 27: Testing the database instance	27
Figure 28: Database adapters-1.....	27
Figure 29: Database adapters 2	28
Figure 30: MBR table	29
Figure 31: Data in the MBR table.....	29
Figure 32: Employer table.....	30
Figure 33: Data in employer table	30
Figure 34: Real Estate table	30
Figure 35: Data in the real estate table	30
Figure 36: Appraiser table.....	31
Figure 37: Data in the appraiser table	31
Figure 38: Logger table	31
Figure 39: Data in the logger table	31
Figure 40: Heroku dashboard	35
Figure 41: Docker images'.....	35
Figure 42: Log Table in MySQL Database.....	37
Figure 43: Registration form front-end validation.....	38

Figure 44: MBR login form front end validation	38
Figure 45 Postman request.....	39
Figure 46 Data inserted in database	39
Figure 47 Postman request for login	39
Figure 48: Application is approved	40
Figure 49: Confirmation from insurance company and employer.....	40
Figure 50: MBR table storing status of each application	41
Figure 51: Application status changed to rejected	41
Figure 52: Registration form front-end validation.....	42
Figure 53 Backend validation demo for missing parameters	42
Figure 54: MBR login form front end validation	43
Figure 55: Application is approved	43
Figure 56: Confirmation from insurance company and employer.....	44
Figure 57: MBR table storing status of each application	44
Figure 58: application status changed to rejected.....	45
Figure 59: Error message in mbr login form	45
Figure 60 Postman testing for wrong credentials.....	46
Figure 61: Approval form required field validation	46
Figure 62: Appraiser form required fields validation.....	47
Figure 63: Invalid credentials (Appraiser login form)	47
Figure 64: Invalid credentials (Appraiser login form) using postman.....	48
Figure 65: Successful registration at real state portal	49
Figure 66: Data added into real estate table	49
Figure 67: Property appraisal form submission error	50
Figure 68: Approval form testing using postman	50
Figure 69: Appraiser valuation page on successful login	51
Figure 70: entry in the real estate table before the valuation	51
Figure 71: Error message on submitting in appropriate value	51
Figure 72: Submitting the valuation of the property (after submission MlsID M309 will disappear)	52
Figure 73: Updated entry in the real estate table (for MlsID M309)	52
Figure 74: Database showing that application has not received any insurance information	52
Figure 75: Postman POST request to insurance controller with a valid set of inputs. This request is simulating a http request which is provided by the Real Estate portal. Returned message indicates successful execution.	53
Figure 76: Database showing that the application has successfully received the insurance quote.....	53
Figure 77: Token in URL	55
Figure 78: Authorization of Token	56
Figure 79: Token Generation	57

Project Description

Front-End Overview

MBR Portal

The home page of the MBR portal is shown in Figure 1. Here the user has two options, which are Sign-in and Register.

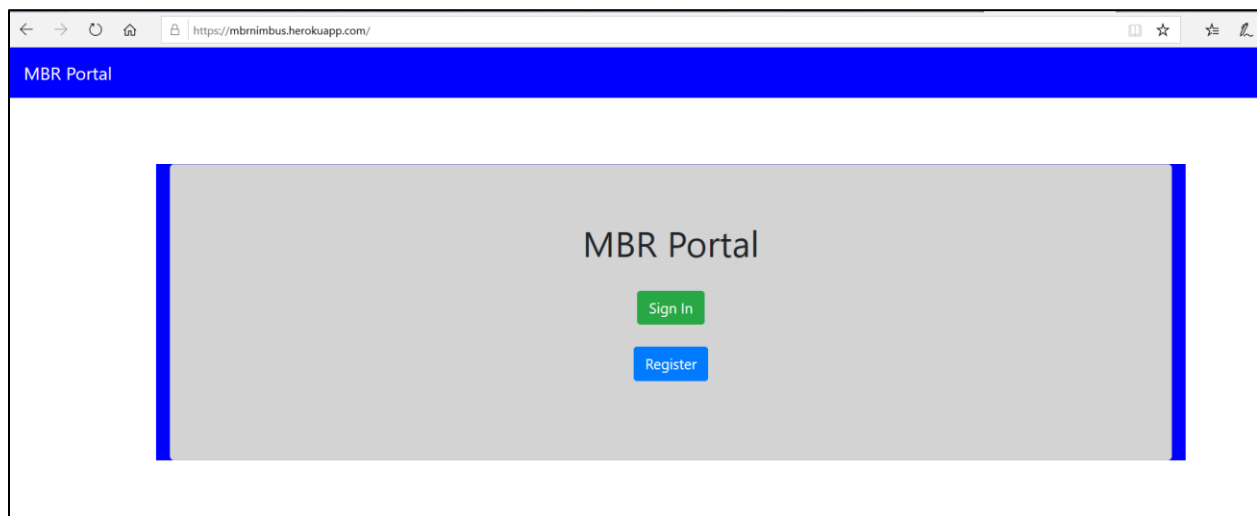
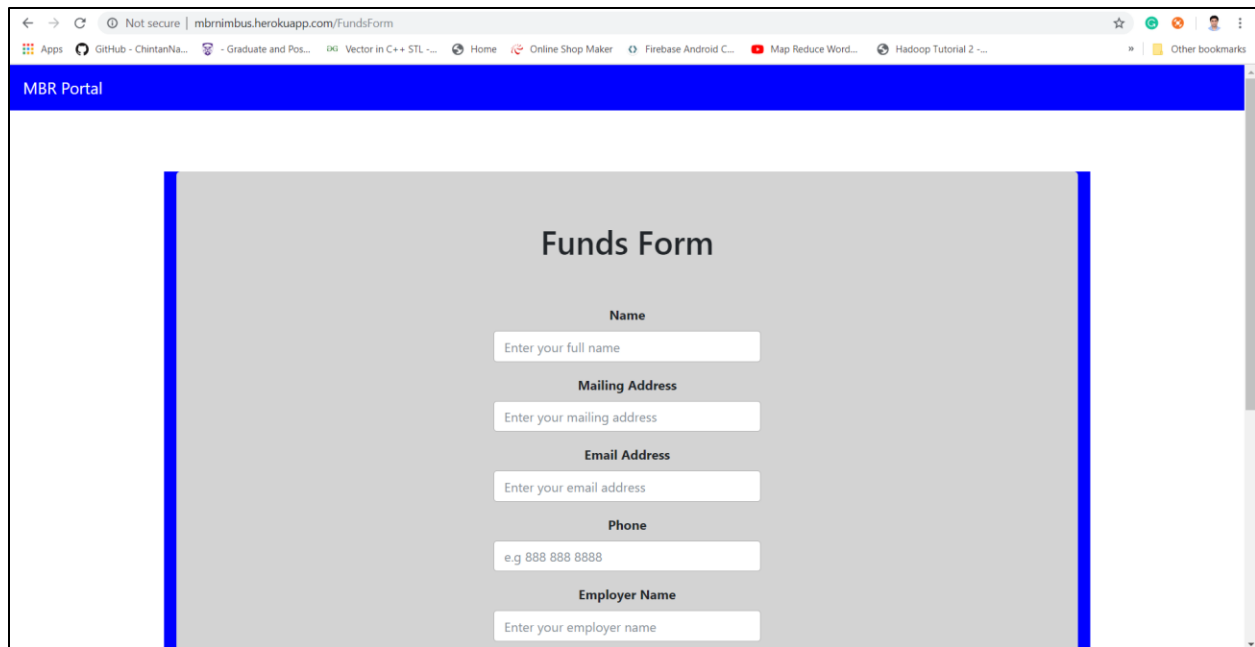


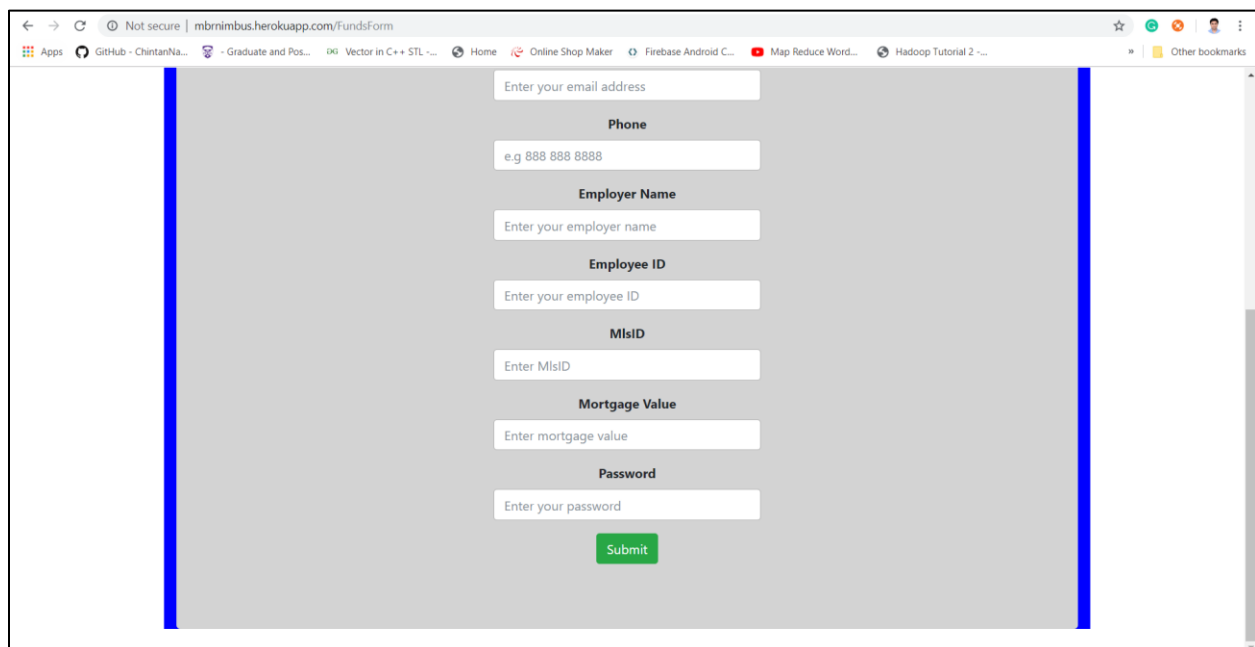
Figure 1: MBR portal homepage

The employee who is willing to purchase a house will provide details in the registration page. The registration form is as shown below. It requires to add a name, mailing address, email address, phone, employer name, employee ID, MIsID, mortgage value and password.



The screenshot shows a web browser window with the URL `mbrnimbus.herokuapp.com/FundsForm`. The page has a blue header bar labeled "MBR Portal". The main content area is a light gray rectangle with a blue border. Inside, the title "Funds Form" is centered. Below the title, there are six input fields, each with a label above it: "Name" (with placeholder "Enter your full name"), "Mailing Address" (with placeholder "Enter your mailing address"), "Email Address" (with placeholder "Enter your email address"), "Phone" (with placeholder "e.g 888 888 8888"), "Employer Name" (with placeholder "Enter your employer name"), and "Employee ID" (with placeholder "Enter your employee ID").

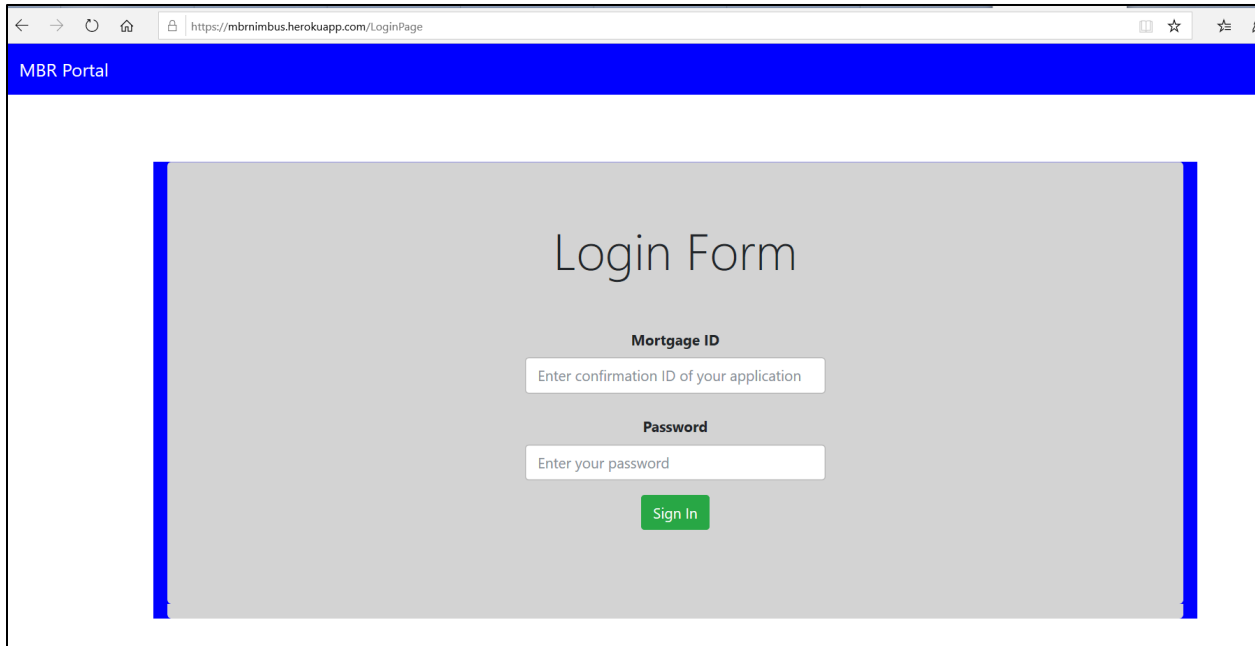
Figure 2: MBR funds request form



This screenshot continues the form from Figure 2. It shows the bottom half of the input fields: "Mortgage Value" (with placeholder "Enter mortgage value"), "Password" (with placeholder "Enter your password"), and a green "Submit" button. The "Employee ID" field is also visible at the top of this section. The layout is consistent with the previous figure, featuring a light gray background and a blue border.

Figure 3: MBR funds request form continued

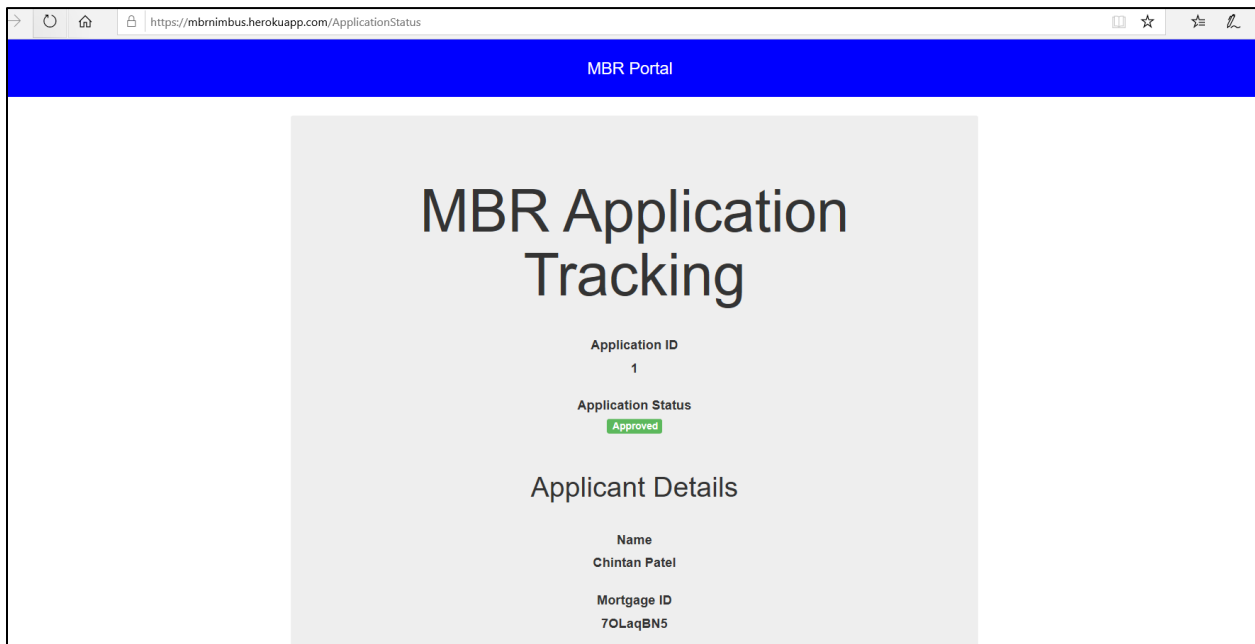
Once the employee has filled out the mortgage fund form and registered, they can log in on the MBR portal shown in below figure by entering the resulting application id of the mortgage and the chosen password. If the credentials are correct, they can check their application status. Otherwise, an error message will be shown.



The screenshot shows a web browser window with the URL <https://mbrnimbus.herokuapp.com/LoginPage>. The page has a blue header bar labeled "MBR Portal". The main content area is a light gray rectangle with a blue border. Inside, the text "Login Form" is centered at the top. Below it, there are two input fields: "Mortgage ID" with the placeholder text "Enter confirmation ID of your application", and "Password" with the placeholder text "Enter your password". A green "Sign In" button is positioned below the password field.

Figure 4: MBR login page for application status

After the login is successful, the applicants can check their status of the application on the application status page.



The screenshot shows a web browser window with the URL <https://mbrnimbus.herokuapp.com/ApplicationStatus>. The page has a blue header bar labeled "MBR Portal". The main content area is a light gray rectangle. Inside, the text "MBR Application Tracking" is centered at the top. Below it, there are two sections: "Application ID" with the value "1", and "Application Status" with the value "Approved" (highlighted in a green box). Below these, there is a section titled "Applicant Details" with the following information: "Name" (Chintan Patel), "Mortgage ID" (7OLaqBN5).

Figure 5: MBR application status page

Employer Portal

Employer Portal is shown in below figure. It contains two important fields, which are:

- Employee ID
- Password

When the user enters these fields, it will be checked into the database of the Employer, and if the record is present in the database, then the authentication is successful. If the user is not present in the database, the error message will be displayed saying that credentials are invalid. All the fields are required. Therefore, if there are any blank fields when a user submits the form, it will display the message "Please fill out the field."

Once successfully logged-in to the Employer portal, the user will be directed to the mortgage form and needs to submit the mortgage id and the link to the MBR portal. The mortgage form is shown in Figure 8. Once the user clicks the *Agree* button, the application status is displayed on the page in the form of the "Accepted," "Rejected," or "Error Message."

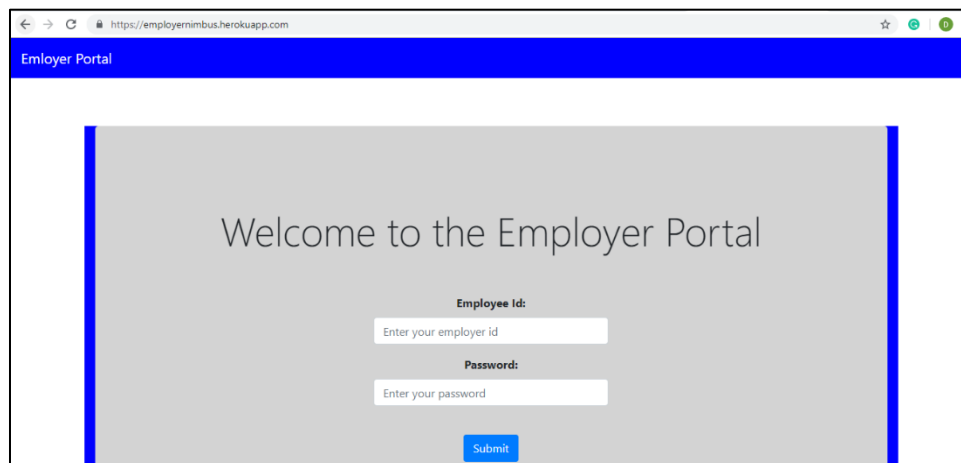
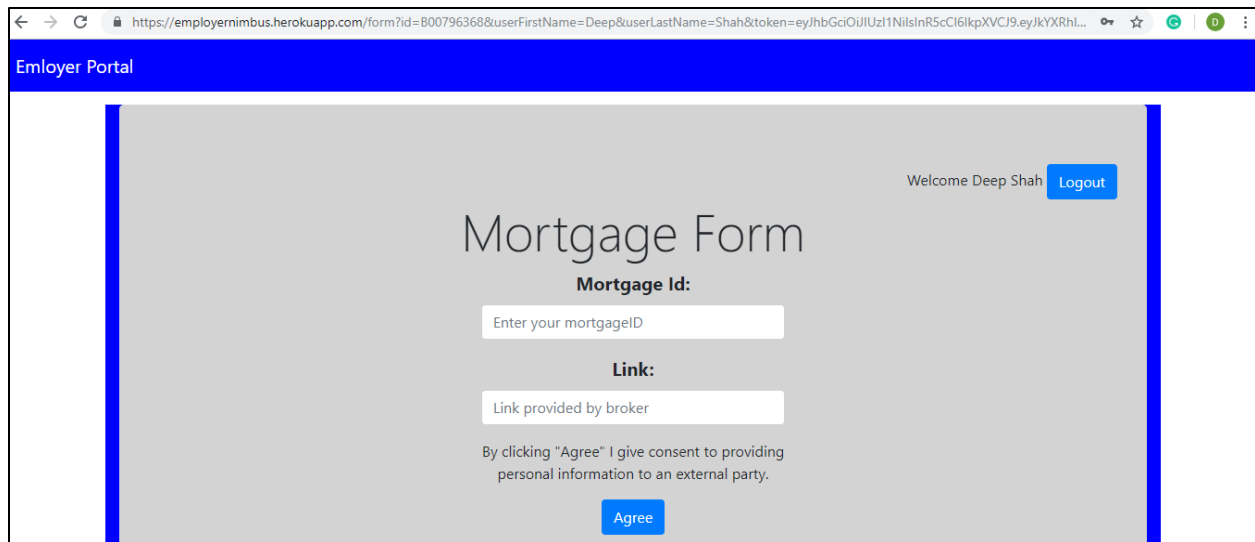
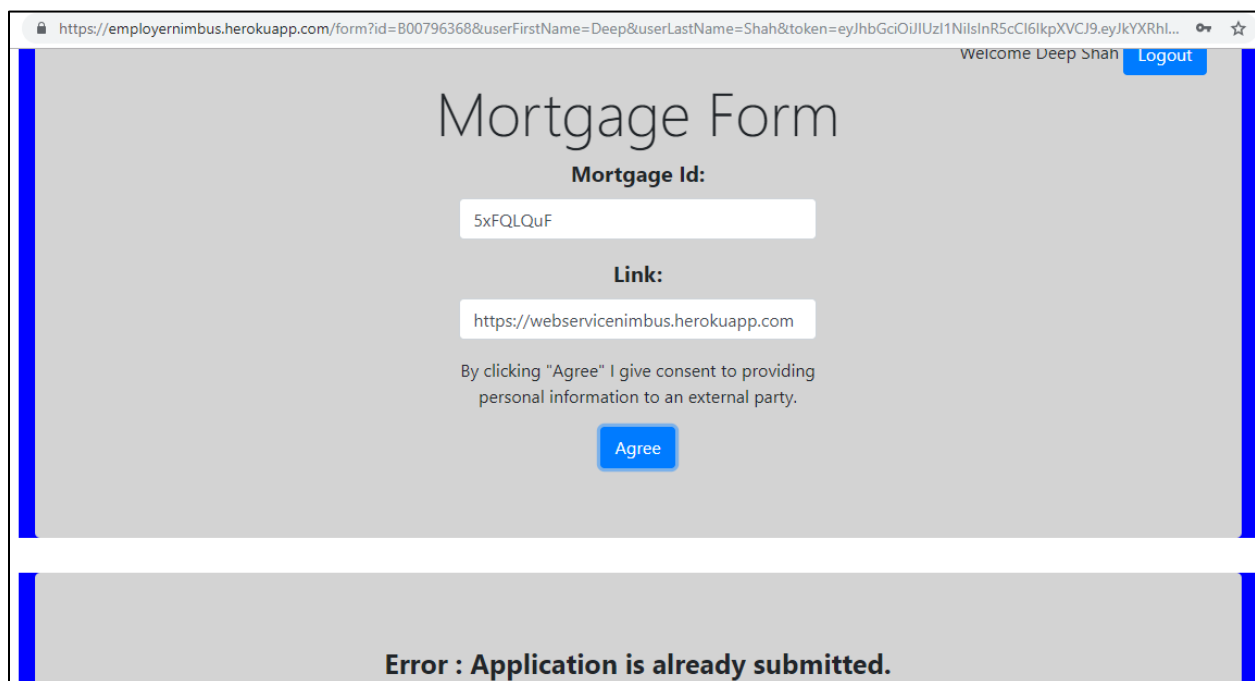
The image is a screenshot of a web browser displaying the 'Employer Portal' homepage. The browser's address bar shows the URL 'https://employernimbus.herokuapp.com'. The page has a blue header with the text 'Employer Portal'. The main content area has a light gray background and is framed by a blue border. It features a large heading 'Welcome to the Employer Portal'. Below this, there are two input fields: 'Employee Id:' with a placeholder 'Enter your employer id' and 'Password:' with a placeholder 'Enter your password'. A blue 'Submit' button is located at the bottom of the form.

Figure 6: Employer portal homepage



The screenshot shows a web browser window with the URL `https://employernimbus.herokuapp.com/form?id=B00796368&userFirstName=Deep&userLastName=Shah&token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRh...`. The page has a blue header bar labeled "Employer Portal". In the top right corner, it says "Welcome Deep Shah" next to a blue "Logout" button. The main content area has a light gray background and is framed by blue vertical bars on the left and right. The title "Mortgage Form" is centered at the top. Below it, the label "Mortgage Id:" is followed by a white input field containing the placeholder text "Enter your mortgageID". Below that, the label "Link:" is followed by a white input field containing the placeholder text "Link provided by broker". At the bottom of the form area, there is a line of text: "By clicking 'Agree' I give consent to providing personal information to an external party." followed by a blue "Agree" button.

Figure 7: Mortgage form of employer portal



This screenshot shows the same "Mortgage Form" page as Figure 7, but with data entered and an error message displayed. The "Mortgage Id:" input field now contains the text "5xFQLQuF". The "Link:" input field contains the URL "https://webservicesnimbus.herokuapp.com". The "Agree" button is still present. Below the form area, a gray box with a white border contains the error message: "Error : Application is already submitted." in bold black text. The browser window and header elements are identical to the previous figure.

Figure 8: Employer mortgage form application error status feedback

Real Estate Broker Portal

The portal of the real estate broker has 2 parts, which are property appraisal form and sign in for appraiser.

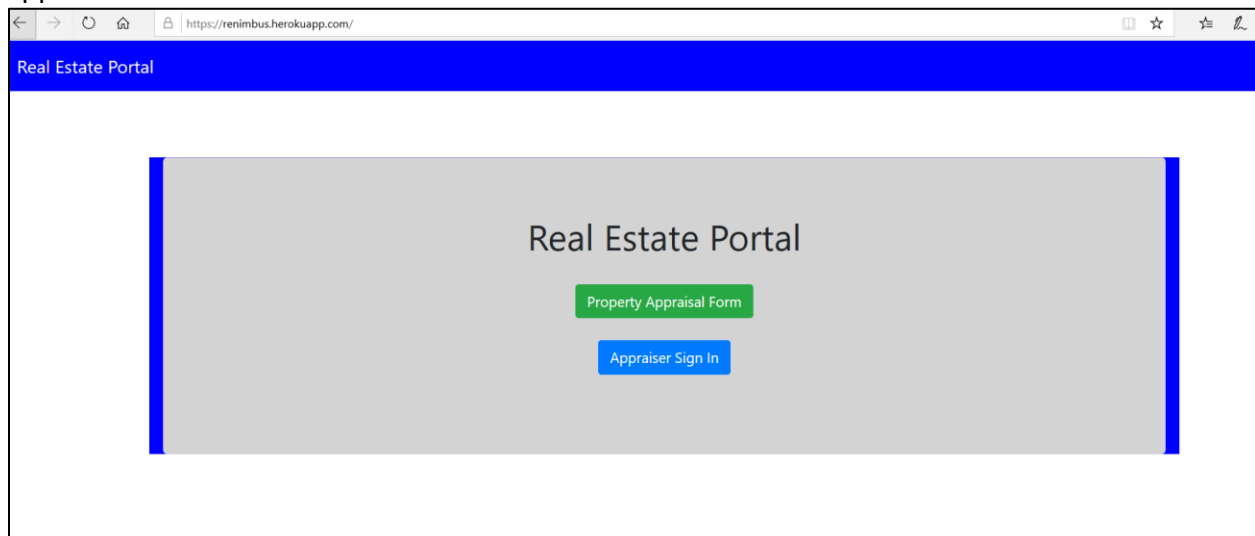


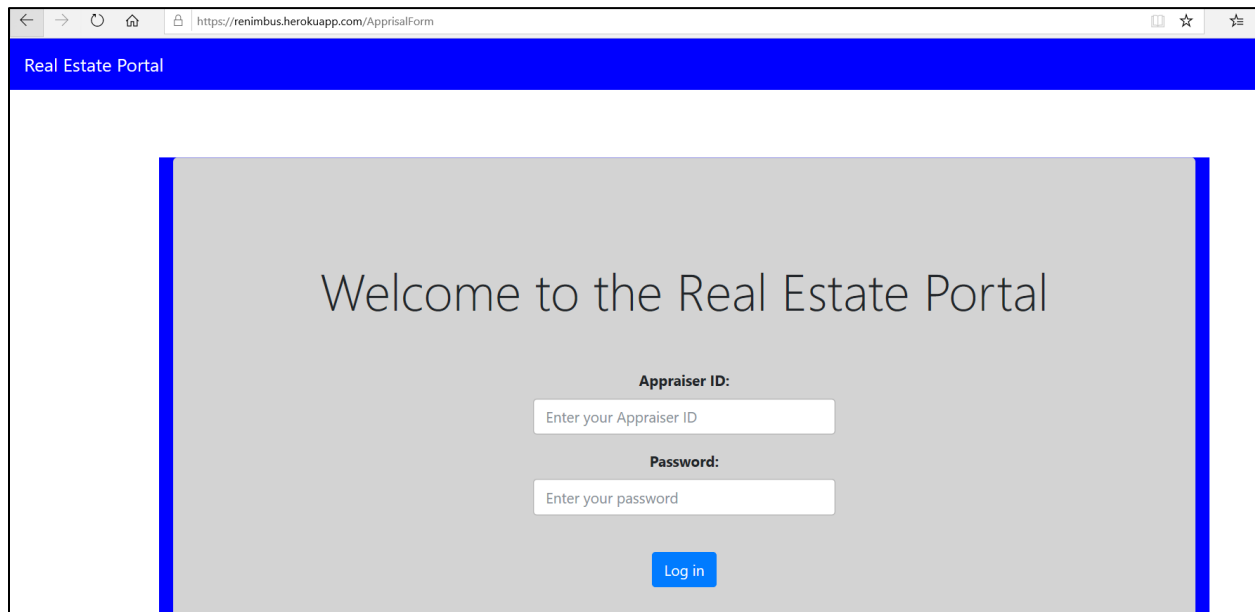
Figure 9: Real estate portal homepage

The employee who has registered for a mortgage into MBR portal needs to provide application details to the real estate (RE) broker. The employee needs to provide MlsID, name and mortgageID of the application. These details will be added to the RE broker database to evaluate.

A screenshot of a web browser showing the 'Approval Form' for a property assessment application. The browser's address bar displays 'https://renimbus.herokuapp.com/ApprovalForm'. The page has a blue header with the text 'Real Estate Portal'. The main content area is a light gray rectangle with a blue border. Inside this rectangle, the text 'Approval Form' is centered. Below the text, there are three input fields: 'MlsID:' with a placeholder 'Enter your MlsID', 'Name:' with a placeholder 'Full name provided at the MBR portal', and 'MortgageID:' with a placeholder 'Enter your MortgageID'. Below these fields is a blue button labeled 'Agree'.

Figure 10: Property assessment application approval form

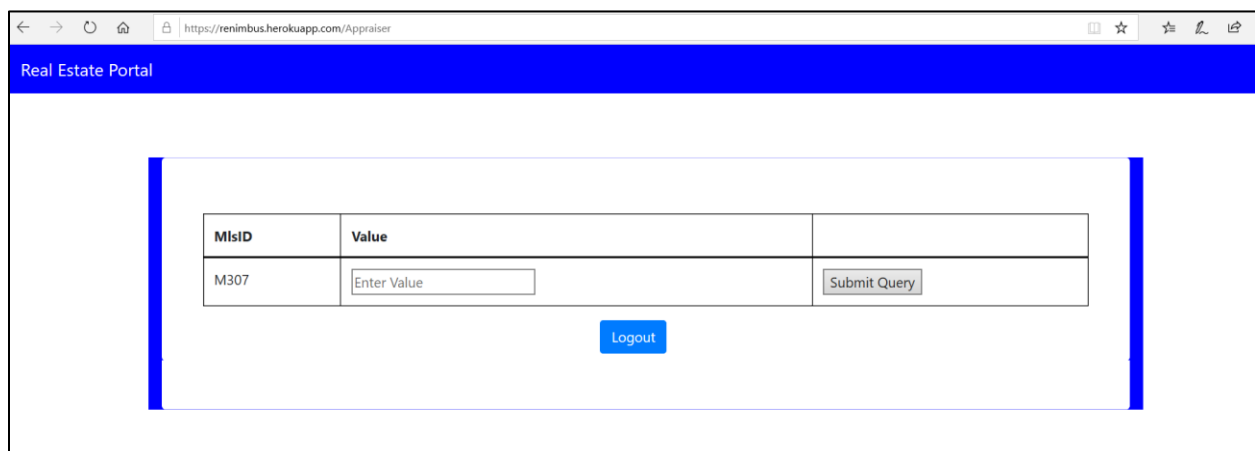
RE broker has appraisers who evaluate the house by MlsID. The appraiser has to provide their ID and password in order to estimate valuation for a property on the real estate portal.



The screenshot shows a web browser window with the URL <https://renimbus.herokuapp.com/AppraisalForm>. The page has a blue header bar with the text "Real Estate Portal". The main content area is a light gray rectangle with a blue border. Inside, the text "Welcome to the Real Estate Portal" is centered. Below it, there are two input fields: "Appraiser ID:" with a placeholder "Enter your Appraiser ID" and "Password:" with a placeholder "Enter your password". A blue "Log in" button is positioned below the password field.

Figure 11: Appraiser login page

Once the appraiser has successfully logged in, the properties are yet to be evaluated are shown in Figure 12. Further, property values estimated by the appraiser will be sent to the insurance company for completing the further process.



The screenshot shows a web browser window with the URL <https://renimbus.herokuapp.com/Appraiser>. The page has a blue header bar with the text "Real Estate Portal". The main content area is a light gray rectangle with a blue border. Inside, there is a table with two columns: "MlsID" and "Value". The first row has "M307" in the "MlsID" column and an input field with the placeholder "Enter Value" in the "Value" column. To the right of the table is a "Submit Query" button. Below the table is a blue "Logout" button.

MlsID	Value
M307	<input type="text" value="Enter Value"/>

Figure 12: Property validation page for Appraiser

Web Service Overview

MBR Controller

1. newApplication

This function is written to insert new mortgage application data into MBR database. To call this function, the request needs to be made via /FundsForm URL. It accepts JSON data. The sample JSON request is as mentioned below.

```
{
  "email": <Email address>,
  "address": <Mailing address>,
  "name": <name of the user>,
  "phone": <phone number>,
  "employer": <name of the employer>,
  "password": <password which will be used to check application status>,
  "employee_ID": <employee ID>,
  "mortgage_value": <mortgage value>,
  "MIsID": <MIsID of house>
}
```

The function will store the passed data into MBR database. In response, the function will return JSON data containing either an error message or data about the user. For example, an error occurs, the data would be as indicated below.

```
{
  "error_message": <detailed error message>,
}
```

If the data is successfully inserted into the database, the function will return data which is inserted into the database along with mortgageID and status of the application. The sample JSON response is as indicated below.

```
{
  data: {
    "id": <primary ID>,
    "email": <Email address>,
    "phone": <phone number>,
    "mailing_address": <Mailing address>,
    "name": <name of the user>,
    "employee_ID": <employee ID of employee>,
```

```

    "MIsID": <MIsID>,
    "mortgage_value": <mortgage value>,
    "employer_name": <name of the employer>,
    "employment_duration": <duration of employment>,
    "employee_salary": <salary of the employee>,
    "insured_value": <insured value which will be passed by insurance company>,
    "deductible_value": <deductible value which will be passed by insurance company>,
    "EMP_confirmation ": <Boolean value to indicate if employer has passed the details>,
    "INSinc_confirmation ": <Boolean value to indicate if INSinc has passed the details>,
    "status": <status of the application>
  }
}

```

2. check_credentials

The user can check the application status of the application submitted in MBR portal by logging in using the mortgageID and password. The check_credentials function is developed for checking credentials the user has entered in the login page of MBR portal. To call the function, the request needs to send via check_credentials request. It accepts JSON data. Sample JSON request is as mentioned below.

```

{
  "user_id": <mortgageID of the application>,
  "password": <password>
}

```

The function returns a JSON response. If the credentials are passed, the response will return JSON data with two keys data and error_message. The sample response is as mentioned below. The error_message will be blank if credentials are matched.

```

{
  data: {
    "id": <primary ID>,
    "mortgage_ID": <mortgage ID>,
    "name": <name of the user>,
    "email": <Email address>,
    "mailing_address": <Mailing address>,
    "phone": <phone number>,
    "MIsID": <MIsID>,
    "mortgage_value": <mortgage value of application>,
    "employer_name": <name of the employer>,
    "employment_duration": <employment duration>,
    "employee_salary": <employee salary>,
    "employee_ID": <employee ID>,
    "insured_value": <insured value>,
    "deductible_value": <deductible value>,

```

```

        "EMP_confirmation ": <Boolean value to indicate if employer has passed the details>,
        "INSinc_confirmation ": <Boolean value to indicate if INSinc has passed the details>,
        "status": <status of the application>
    },
    error_message:{
        ""
    }
}

```

If the credentials are not matched, the web service will return error_message. The sample JSON response in case of wrong credentials are passed is as mentioned below.

```

{
    error_message:{
        <error message>
    }
}

```

3. validateApplication

This function is designed for validating applications which are passed from the employer. This function will be called from the employer portal. The function can be called via validateApplication. The function accepts JSON data. The sample data which needs to be passed in JSON request of validateApplication is as mentioned below.

```

{
    "employer_name": <name of employer>,
    "mortgageID": <mortgageID>,
    "employeeName": <name of the employee>,
    "salary": <salary of employee>,
    "employment_length": <length of employment>
    "employeeID": <ID of the employee>
}

```

In response, the validateApplication returns three status messages. The first status message is ACCEPTED. This status message will be returned if the data passed by an employer is stored in the MBR database. The error_message will be blank if the status is ACCEPTED. In this case, the EMP_confirmation flag will be set to true. If the INSinc_confirmation flag is also true, the status of the application will be updated to “accepted.” The sample JSON response is as indicated below.

```

{
    "status": "ACCEPTED",
    "error_message": ""
}

```

The second status is REJECTED. This status will be returned if the data passed from the employer is not matched with the MBR database. In this case, the application status will be changed to “rejected” in the MBR database. The sample JSON response is as indicated below.

```
{
  "status": "REJECTED",
  "error_message": <error message>
}
```

The third status is ERROR. This status will be returned if something wrong occurs in the web service. It may occur in many cases like the database is not accessible or if the application status is already updated in the MBR database. The sample JSON response is as indicated below.

```
{
  "status": "ERROR",
  "error_message": <error message>
}
```

4. insuranceUpdate

This function is designed for validating applications which are passed from the insurance company. This function will be called from the INSinc portal. The function can be called via insuranceUpdate. The function accepts JSON data. The sample data which needs to be passed in JSON request of insuranceUpdate is as mentioned below.

```
{
  "MortID": <Mortgage ID of the application>,
  "FullName": <name of the employee>,
  "MIsID": <MIsID>,
  "DeductableValue": <deductible value which is determined by insurance company>
  "InsuredValue": <insured value which is determined by insurance company>
}
```

The response of insuranceUpdate has two types. The first type is OK. This status message will be returned if the data passed by insurance is stored in the MBR database. In this case, the INSinc_confirmation flag will be set to true. If the EMP_confirmation flag is also true, the status of the application will be updated to “accepted.”

The second type of JSON is an array. This type will be returned if the data passed from the insurance is not matched with the MBR database. In this case, the application status will be changed to “rejected” in the MBR database. The sample JSON response is as indicated below. The below response can also be returned if something is wrong with the database or any other issues occur.

```
{
  [<error message>]
}
```

Logger Controller

The Logger Controller consists of a single function to meet the assignments logging of web service requests/response requirement. More details about the logging can be found in the upcoming *Log Service Configuration* section.

Employer Controller

The employer controller consists of two functions.

1. authenticate

The authenticate function is called once the user has submitted all the sign-in credentials on the employer portal. The function accepts two parameters, which are the user ID and password. Both parameters are stored in a database prior to signing in as the assumption is made that a user is not required to sign up. The employer should have this information entered and available in the database prior to the mortgage request.

The function searches the Employer database for a single match with the given user ID and password. The password is encrypted prior to sending to the database to search the combination of employee ID and password. The sample JSON request which needs to be made is as below.

```
{
  "id": <employee ID>,
  "pwd": <password>
}
```

The response of the authenticate function is JSON data. It returns two types of JSON data. If some parameters are missing, the function will return a JSON message with a detailed error message. The sample response is as described below.

```
{
  "ErrorMessage": <error message in detail>
}
```

If all parameters are passed to the function, the function returns JSON response as below.

```
{
  "Authorized": <Boolean to indicate if employee data is available in database>,
  "ErrorMessage": <error message if any>,
  "Token": <token generated if any>,
  "Employee": <employee data which is stored in database>,
}
```

2. retrieveData

The retrieveData function is called by the Employer portal once the user accepts the terms of providing personal information to a third party. Once the user accepts, the user ID is passed to this function, and all employee details are returned. The function searches the database for an exact match and returns an Employee object which contains all the employee information. This object is then returned to the employer portal. The sample JSON request is as below.

```
{
  "id": <Employee ID>,
}
```


Like the previous function, if an error occurs with the database, then an error is logged and returned. A success message is returned if the user data is successfully returned. There is also a condition to handle an invalid user, but this is unlikely to happen as the user has already been authenticated at this stage. The following is a sample string of the JSON response by the function.

```
{
    "Employee": <Employee Object>,
    "ErrorMessage": <error message if any>
}
```

The error message has three possible strings. The first is “Success” which will be returned if the employee data is available in the database. The second possible error string is “Employee not found” which will be returned if the passed employee ID is not available in the database. The error message is database connectivity error which contains an error message as “We are having troubles connecting to the database, please try again later.”

Please refer to the testing section of the report for further details on all the parameters that are included in the Employee object.

INSinc Controller

The INSinc controller is developed to handle employer request to generate an insurance quote and pass the data to the MBR controller. It contains three functions. However, only one function is written to handle web service calls. Other two functions are for internal use. The first one is to calculate the deductible amount and the insured amount by using appraisal value, which is passed from the employer controller.

1. insuranceQuote

This function is written with an aim to handle employer controller request to generate insurance quotes. This function requires mortgage ID, appraisal value, MIsID and name of the person to generate insurance quotes. The function accepts a JSON request. The sample JSON request is as described below.

```
{
    "MortID": <mortgage ID >,
    "AppraisalValue": <appraisal value which will be used to calculate deductible and insured value>,
    "MIsID": <MIsID>,
    "FullName": <full name of the person>
}
```

All of the above parameters are required. The insuranceQuote function will calculate the deductible and insured value with the help of calculateDeductibleAmount and calculateInsuredAmount functions. These two functions are internal functions.

The insuranceQuote function has two main tasks to do. One is sending data to MBR web service, and the other is to return a response to the employer web service, which called this function. The

function sends mortgage ID, MlsID, name of the person, insured value and deductible value to the MBR web service's insuranceUpdate function.

The insuranceQuote function returns three types of responses to the employer web service. The first is server error which will be returned if an error occurs while sending data to web service of MBR. The second type of response is OK response, which will be returned if the MBR web service returns OK response. The third type of return is a string containing an error message. The string contains message as "[Error] MBR return indicates an error with the data [MortId]: <mortgage id which is passed from employer controller>."

Real Estate Controller

The real estate controller consists of four functions. This controller is designed for evaluating house from MlsID and submitting the data to the insurance company portal.

1. REApprovalForm

This function is written to insert property evaluation requests to the real estate table. When the employee submits a new request to MBR portal, it is needed to evaluate property by MlsID. The person needs to submit details in the real estate portal. The submitted details are inserted into the table in order to evaluate. The function accepts JSON data, which is as described below.

```
{
  "MortgageID": <mortgage ID which is given by MBR>,
  "MlsID": <MlsID of the property>,
  "Name": <full name of the person>
}
```

The function will insert the data into the table if it passes validations. The function returns response in JSON format. Two types of responses are sent by the function. One is the error message which consists of a detailed error message if any issue occurs. The sample JSON response if an error occurs is as below.

```
{
  "error_message": <detailed error message>,
}
```

The second type of response is a JSON with data as a key. The value of the data key will be "Your mortgage property details are sent for approval."

2. checkAppraisalCredentials

This function is written to check if appraisal credentials are valid or not. The appraiser needs to login to real estate in order to evaluate the property. The real estate table has pre-defined login credentials of the appraiser. checkAppraisalCredentials function accepts JSON request, which consists of user ID and password. The sample JSON request is as mentioned below.

```
{
```

```

    "UserID": <UserID of the appraiser>,
    "Password ": <password>
  }

```

The function will encrypt the password using the Caesar cipher technique prior to matching with the table values. The response of checkAppraisalCredentials function is JSON response. The response consists of two types. The first type contains error message if any and data. The sample response is as below.

```

{
  "data": <data fetched from table>,
  "error_message ": <error message is any>
}

```

The second type consists of the only error message in the above response. It will be returned if the credentials are not matched.

3. getREData

Once the appraiser logs in successfully, the properties which need to be evaluated should be displayed. This function is written with the purpose of fetching data from the table. This function does not require any data to be passed in order to fetch property data from the table. The function will return those property data which are yet to be evaluated. The response is of JSON type, which consists of data and flag if the error is present or not. The error may occur if the database connection is unsuccessful. The sample JSON response is as mentioned below.

```

{
  "data": <data fetched from table>,
  "errorPresent ": <Boolean to indicate if an error occurred or not>
}

```

4. updateREData

Once the appraiser enters the value of the property by MlsID, the data needs to be submitted to the insurance company in order to generate insurance quotes. The function requires MlsID and appraiser value to be passed. The sample JSON request is as detailed below.

```

{
  "id": <MlsID of the property>,
  "value ": <appraiser value>
}

```

It is possible that multiple users have submitted same MlsID to evaluate. Thus, it is required to pass all requests to insurance web service. Thus, the response of updateREData function consists of an array. The function first updates real estate table with appraiser value and fetches user data from RE table matching MlsID. The response is either data with detailed error message or data with user data. The sample JSON response if the error is present is as below.

```
{
  "data": <error message>,
}
```

If an error is not present, data will be returned by the updateREData function. This data can be passed to the insurance company portal, which can be used to generate insurance quotes.

Workflow Overview

Workflow is used to Schedule, integrate, orchestrate tasks using Azure Logic Apps. It is used to create a trigger like when HTTP Request is received and define the JSON schema of request parameters. Also, we can Initialize the variables to make the workflow dynamic and create the action afterwards like HTTP callback methods or HTTP API Testing, which will be called by Logic App Instance created after trigger execution.

For example, HTTP action testing API endpoint /authenticate of Employer web service URL <https://webservicenimbus.herokuapp.com/employer/authenticate>, as shown in **Figure 15**. We can also Parse the response (Authorized true or false) from the Response Body and add the conditional flow action to the workflow for Authorization if successful send the login success email and if fails send the login fail email to that employee.

1. Creating trigger titled “When an HTTP request is received” with request JSON body schema.

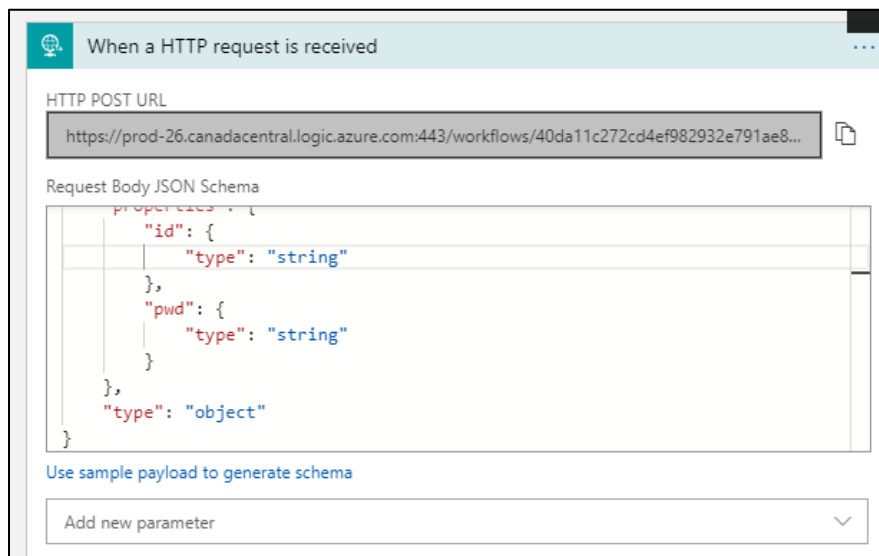


Figure 13: HTTP Request Trigger in Workflow

2. Initializing variables like employeename = “Ueli Haltner” to change the values dynamically.

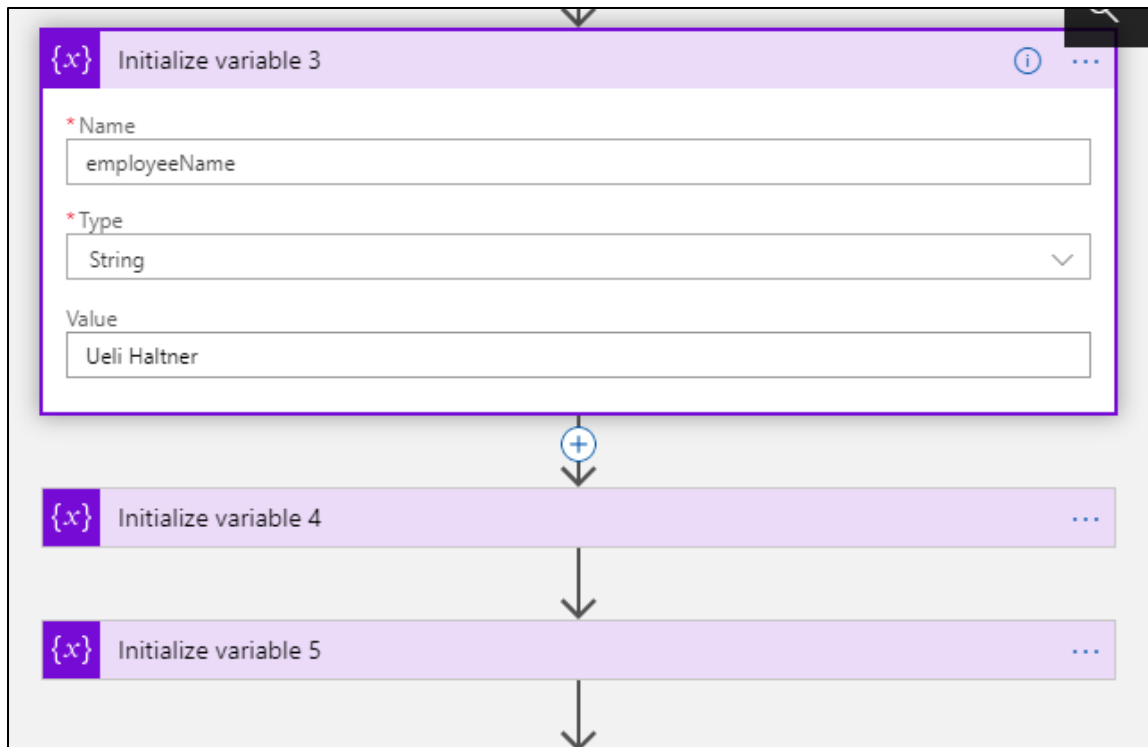


Figure 14: Initialize variable for workflow

3. Creating the HTTP action for the /authenticate endpoint of Employer web service

The screenshot shows the configuration for an HTTP action. The Method is set to POST. The URI is `https://webservicenimbus.herokuapp.com/employer/authenticate`. The Headers section is empty. The Queries section contains two entries: 'id' with value '{x} id x' and 'pwd' with value '{x} pwd x'. The Body section is empty. The Authentication is set to None.

Queries	
id	{x} id x
pwd	{x} pwd x
Enter key	Enter value

Figure 15: Employer Web service URL testing for /authenticate Endpoint

4. Adding the Parse JSON action to parse the JSON



Figure 16: Parsing Response Body JSON

5. Adding the conditional flow to check if the Authorized is True or False. If true, send the login success email to Employee and if false, send the login failed email to Employee

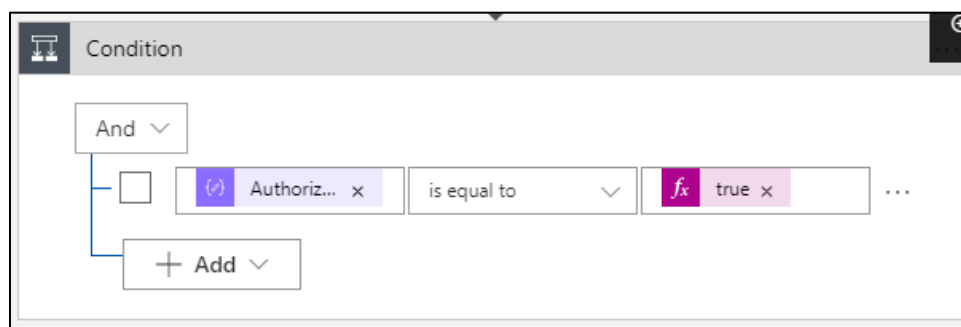


Figure 17: Evaluating workflow condition

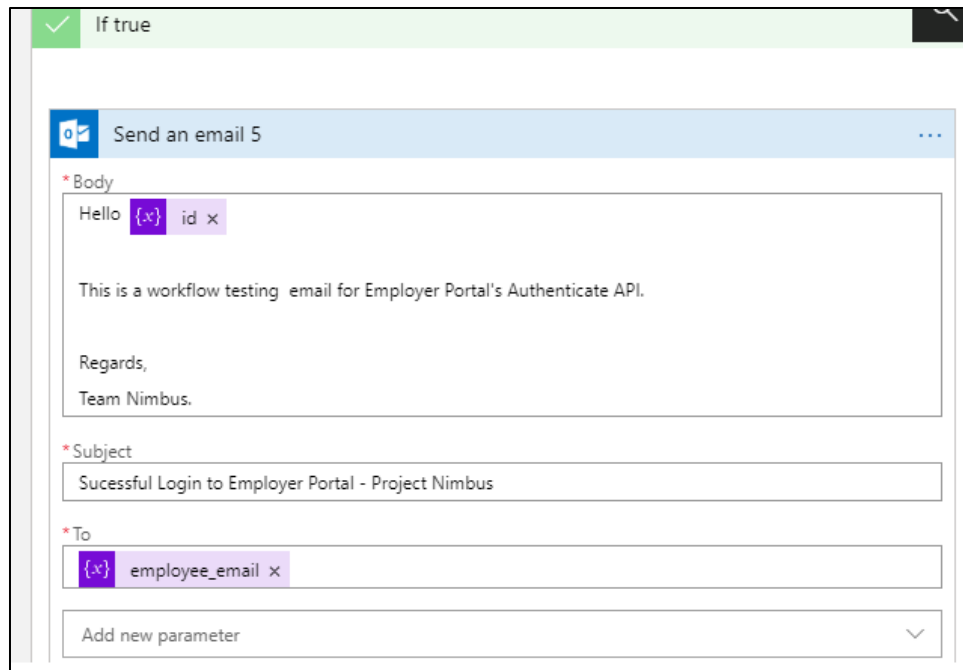
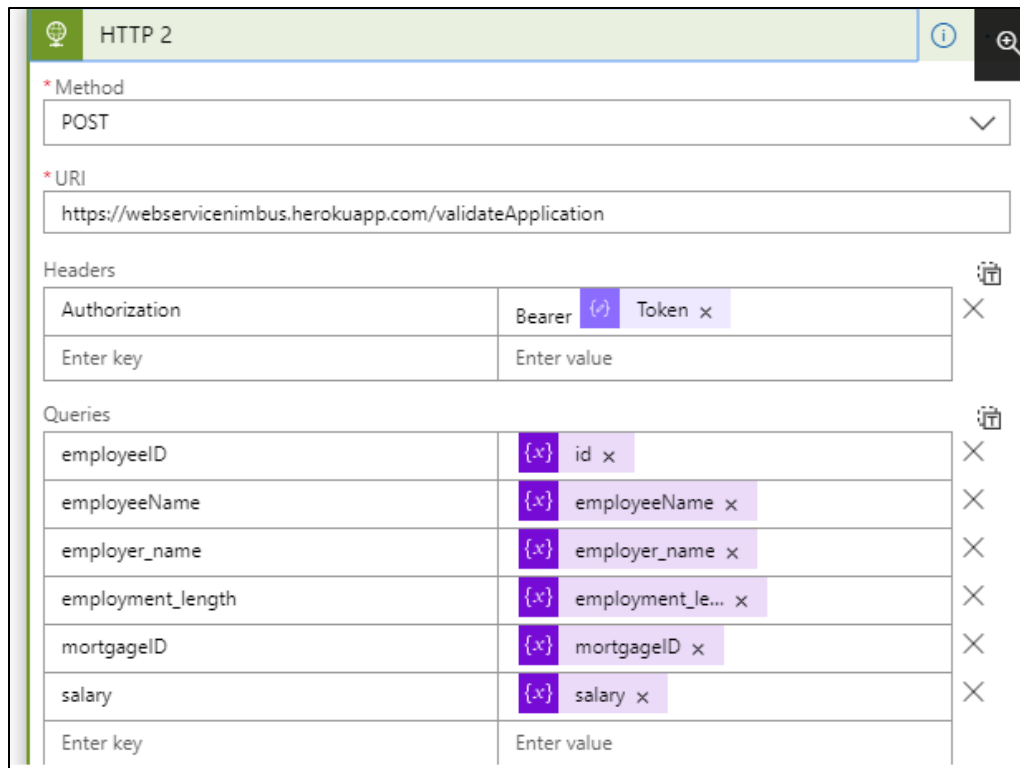


Figure 18: Email sent to the user

6. Adding one more HTTP action to test the API endpoint `/validateApplication` of MBRweb service with query parameter shown in the below screenshots.



The screenshot shows the configuration for an HTTP 2 test in a testing tool. The Method is set to POST and the URI is `https://webservicenimbus.herokuapp.com/validateApplication`. The Headers section contains an Authorization header with the value `Bearer {x} Token x`. The Queries section contains several parameters, each with a value starting with `{x}`:

Key	Value
employeeID	{x} id x
employeeName	{x} employeeName x
employer_name	{x} employer_name x
employment_length	{x} employment_le... x
mortgageID	{x} mortgageID x
salary	{x} salary x

Figure 19: MBR Web service URL testing for /validateApplication Endpoint

7. Parsing JSON response to get the Status and checking the JSON response of the Status

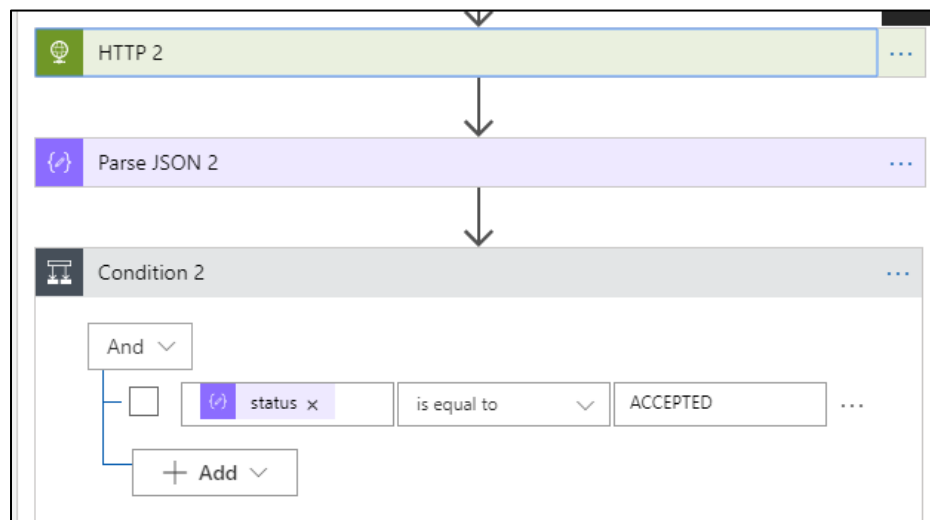


Figure 20: Checking the JSON response

8. If the status is Accepted, send the Mortgage Application Acceptance Success email.

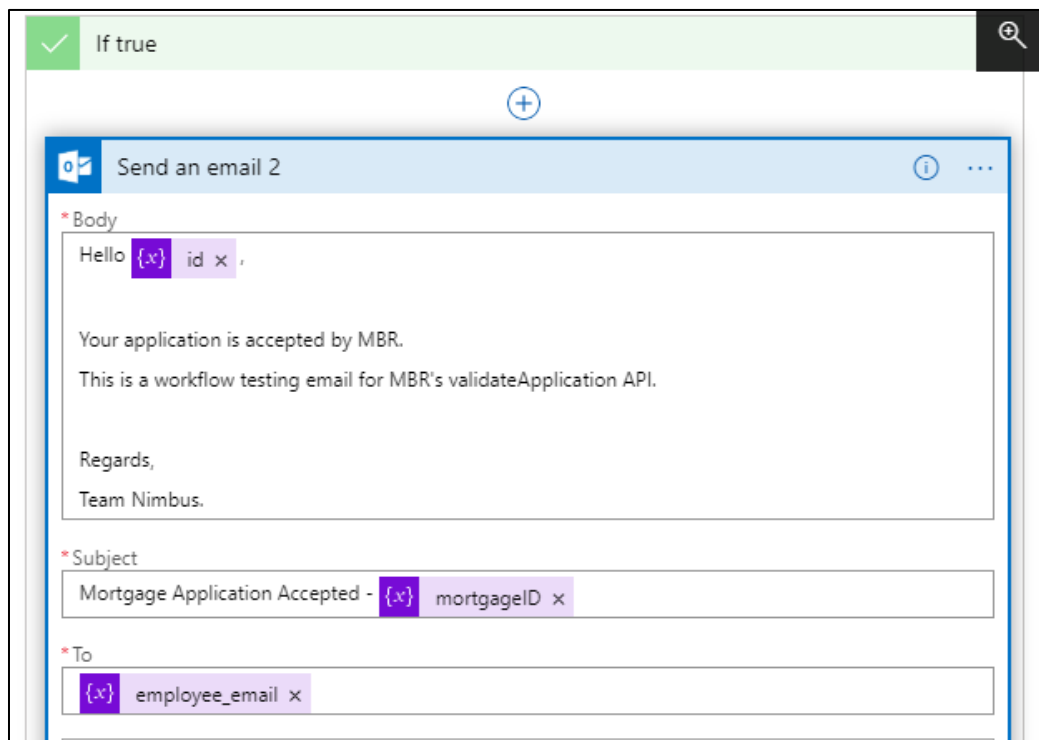


Figure 21: If status is Accepted, send email

9. If the status is Rejected, send the Mortgage Application Rejection Error email to employee.

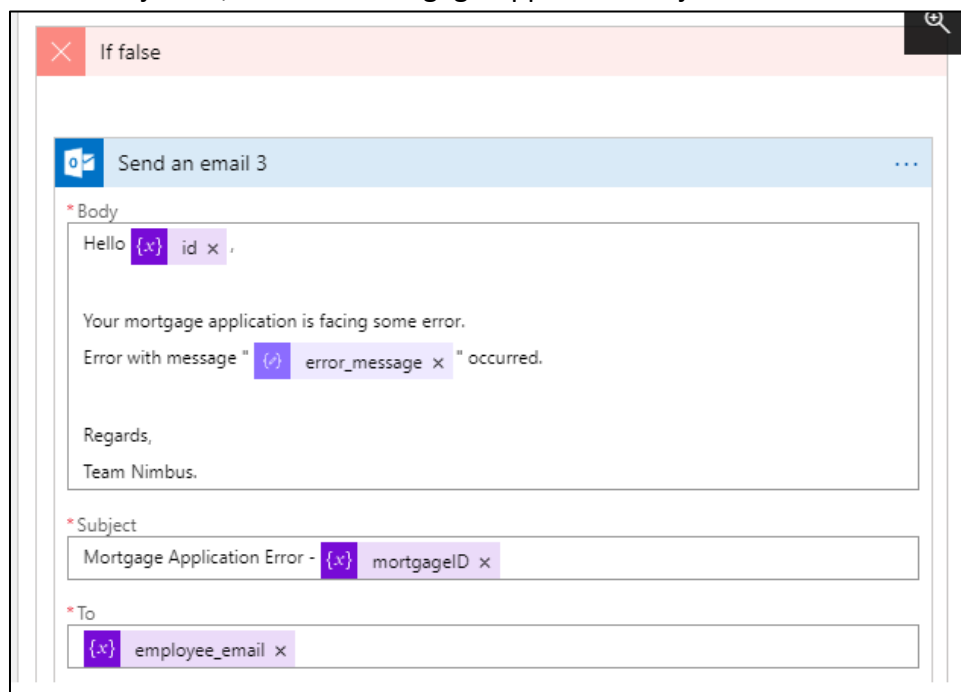


Figure 22: Status is Rejected

Similar way, we have implemented the workflow logic which checks appraiser Userid and password in Real Estate portal from the database and on success condition it asks appraiser to submit the appraise value which is directed to Insurance portal.

The screenshot shows an HTTP client interface for a POST request. The title bar is labeled 'HTTP'. The 'Method' is set to 'POST'. The 'URI' is 'https://webservicenimbus.herokuapp.com/appraisalForm/checkAppraisalCredentials'. The 'Headers' section is empty. The 'Queries' section contains two entries: 'Password' with value '{x}' and 'UserID' with value '{x}'. The 'Body' section is empty. The 'Authentication' section is empty.

Enter key	Enter value
Method	POST
URI	https://webservicenimbus.herokuapp.com/appraisalForm/checkAppraisalCredentials
Header	
Query	
Password	{x} Password x
UserID	{x} UserID x
Header	
Body	
Authentication	

Figure 23: Validate credential for appraiser

The screenshot shows an HTTP client interface for a POST request. The title bar is labeled 'HTTP 2'. The 'Method' is set to 'POST'. The 'URI' is 'https://webservicenimbus.herokuapp.com/insuranceQuote'. The 'Headers' section is empty. The 'Queries' section contains four entries: 'AppraisalValue' with value '{x}', 'FullName' with value '{x}', 'MIsID' with value '{x}', and 'MortID' with value '{x}'. The 'Body' section is empty. The 'Authentication' section is empty.

Enter key	Enter value
Method	POST
URI	https://webservicenimbus.herokuapp.com/insuranceQuote
Header	
Query	
AppraisalValue	{x} AppraisalValue x
FullName	{x} FullName x
MIsID	{x} MIsID x
MortID	{x} MortID x
Header	
Body	
Authentication	

Figure 24: Appraiser value submitted to insurance service

Database Overview & Database Schemas

For the project, we have created a MySQL database server on Microsoft Azure. Furthermore, we have created a local instance using MySQL Workbench to access the data stored in the database server.

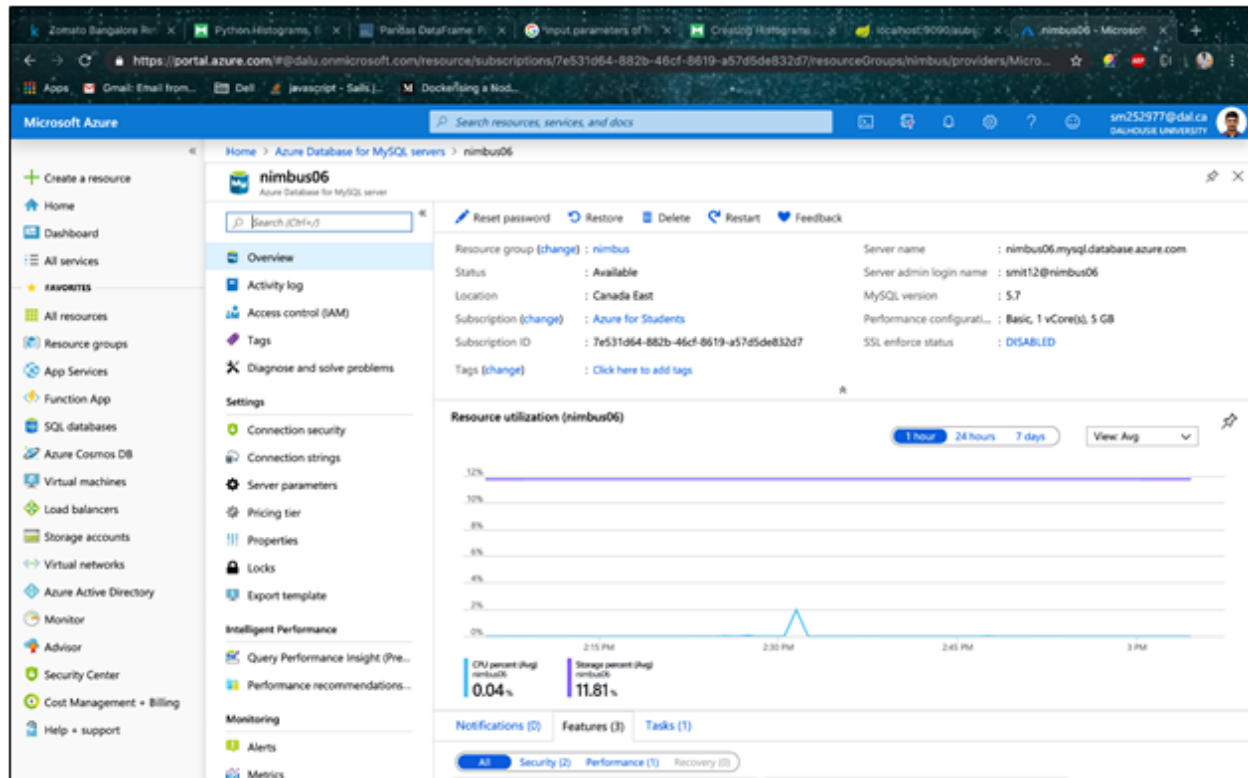


Figure 25: Database server on Microsoft azure

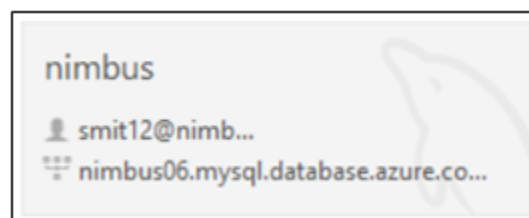


Figure 26: Local instance of cloud database

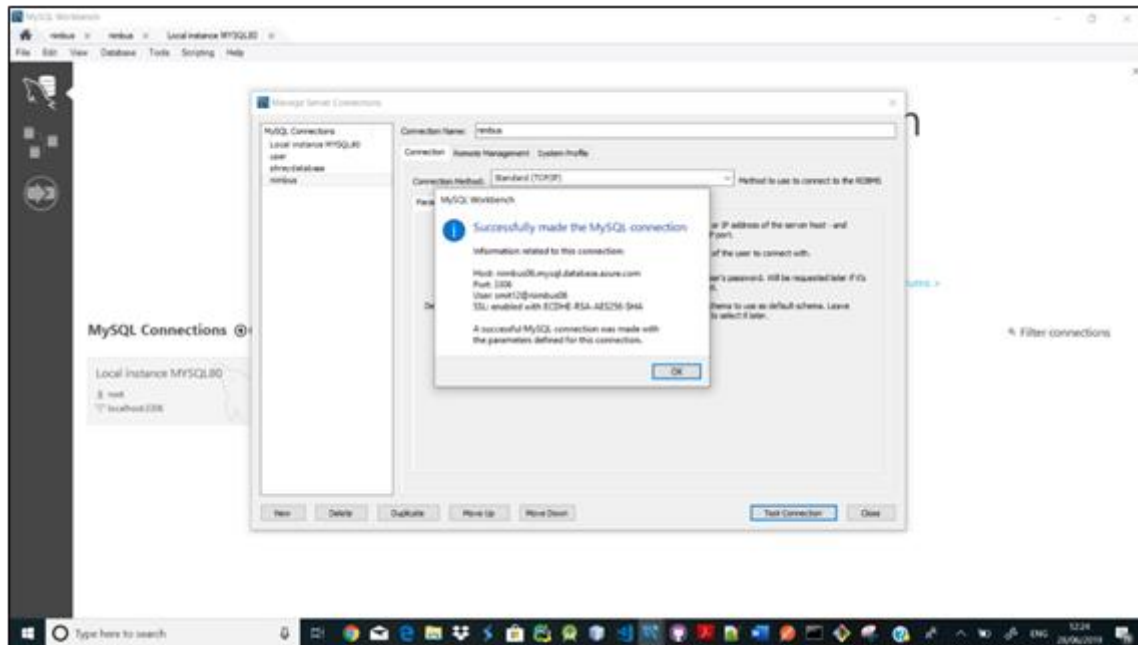


Figure 27: Testing the database instance

As shown in the Figure 28, we have created 3 separate database adapters which are MbrDb, employerDB, loggerDB, REDb and AppraiserDb. This allows us to isolate the connections to the database server and provides the flexibility to transfer any of the connections to a separate database server.

```

MbrDb: {
  adapter: 'sails-mysql',
  host: 'nimbus06.mysql.database.azure.com',
  port: 3306,
  user: 'smit12@nimbus06',
  password: 'Smit@1208',
  database: 'mbr',
},

employerDB:{
  adapter: 'sails-mysql',
  host: 'nimbus06.mysql.database.azure.com',
  port: 3306,
  user: 'smit12@nimbus06',
  password: 'Smit@1208',
  database: 'employer',
},

loggerDB:{
  adapter: 'sails-mysql',
  host: 'nimbus06.mysql.database.azure.com',
  port: 3306,
  user: 'smit12@nimbus06',
  password: 'Smit@1208',
  database: 'logs',
},

```

Figure 28: Database adapters-1

```
REDb:{
  adapter: 'sails-mysql',
  host: 'nimbus06.mysql.database.azure.com',
  port: 3306,
  user: 'smit12@nimbus06',
  password: 'Smit@1208',
  database: 'realestate',
},
AppraiserDb:{
  adapter: 'sails-mysql',
  host: 'nimbus06.mysql.database.azure.com',
  port: 3306,
  user: 'smit12@nimbus06',
  password: 'Smit@1208',
  database: 'appraiser',
},
```

Figure 29: Database adapters 2

As shown in the Figure 30 and Figure 31, MBR table consists of personal details and application details of the employee.

mbr	
Columns	
id	
mortgage_ID	
name	
email	
password	
phone	
mailing_address	
employee_ID	
employer_name	
MIsID	
mortgage_value	
employment_duration	
employee_salary	
insured_value	
deductible_value	
EMP_confirmation	
INSinc_confirmation	
status	

Figure 30: MBR table

id	mortgage_ID	name	email	password	phone	mailing_address	employee_ID	employer_name	MIsID	mortgage
1	70LaqBN5	Chintan Patel	chintan.patel@dal.ca	123456789	9601722640	306, 1333, South Park Street	B00826089	DAL	M306	7
2	zWlpfzeg	Ravi Tulsi Zala	ravi.zala@dal.ca	123456789	9024125695	6221, Duncan Street	B00805073	DAL	M307	8

Figure 31: Data in the MBR table

As shown in the Figure 32 and Figure 33, employer table will store information of every employer. This table will be used to validate the details filled by the employee on the MBR portal while creating the mortgage application.

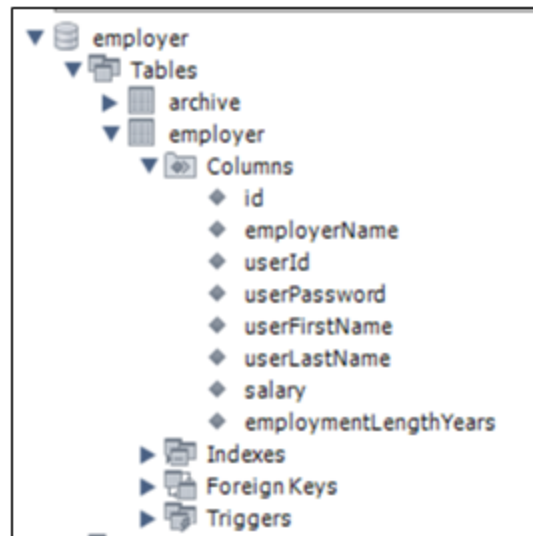


Figure 32: Employer table

id	employerName	employeeID	userPassword	userFirstName	userLastName	salary	employmentLengthYears
1	DAL	B00826089	1234567	Chintan	Patel	10000	2
2	DAL	B00805074	1234567	Ravi Tulsi	Zala	5000	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 33: Data in employer table

As shown in the Figure 34 and Figure 35 the real estate table will store details of the applicants that have registered their property for the value on the real estate portal.

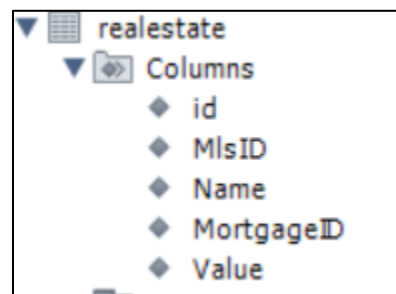
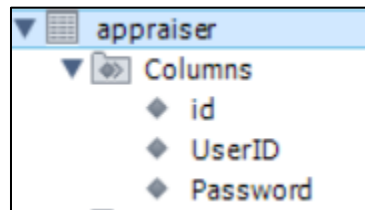


Figure 34: Real Estate table

id	MlsID	Name	MortgageID	Value
1	M306	Chintan Patel	70LaqBN5	1000
2	M307	Ravi Tulsi Zala	zWlpfzeg	0
NULL	NULL	NULL	NULL	NULL

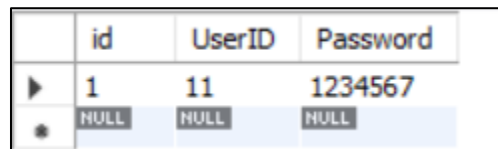
Figure 35: Data in the real estate table

As shown in the Figure 36 and Figure 37 appraiser table will store information of the appraiser.



appraiser		
Columns		
id		
UserID		
Password		

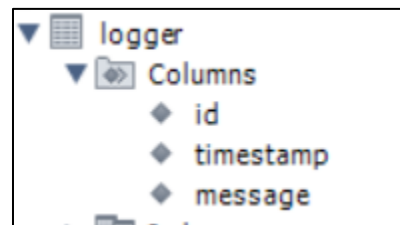
Figure 36: Appraiser table



	id	UserID	Password
▶	1	11	1234567
•	NULL	NULL	NULL

Figure 37: Data in the appraiser table

The logger table is used to maintain the logs it stores the details of all the activities carried out on all the portals along with message and the timestamp.



logger		
Columns		
id		
timestamp		
message		

Figure 38: Logger table

id	timestamp	message
1	2019-08-01 05:00:00	[Employer] [Authentication] Employee successfully found in Employer database
2	2019-08-01 02:00:12	[Employer] [retrieveData] Employee data successfully retrieved found in Employer database
3	2019-07-31 23:02:05	[Employer] [Authentication] Employee successfully found in Employer database
4	2019-07-31 23:02:09	[Employer] [retrieveData] Employee data successfully retrieved found in Employer database
5	2019-07-31 23:02:10	[MBR] [ValidationError] All parameters not passed from Employer
6	2019-07-31 23:02:38	[Employer] [Authentication] Employee successfully found in Employer database
7	2019-07-31 23:02:44	[Employer] [retrieveData] Employee data successfully retrieved found in Employer database
8	2019-07-31 23:02:44	[MBR] [ValidationError] All parameters not passed from Employer
9	2019-07-31 23:06:37	[Employer] [Authentication] Employee successfully found in Employer database
10	2019-07-31 23:06:43	[Employer] [retrieveData] Employee data successfully retrieved found in Employer database
11	2019-07-31 23:06:43	[MBR] [ValidationError] All parameters not passed from Employer
12	2019-07-31 23:09:13	[Employer] [Authentication] Employee successfully found in Employer database

Figure 39: Data in the logger table

Software Deficiencies

In the application, we have included all the required test cases. We have implemented token for employer portal. The token implementation can be extended for MBR and real estate portal too. Other than that, there are no software deficiencies in our application.

Application Host / URLs

Heroku

Application Name	URL
Employernimbus	https://employernimbus.herokuapp.com/
mbrnimbus	https://mbrnimbus.herokuapp.com/
RealEstateBrokerNimbus	https://renimbus.herokuapp.com/
webservicenimbus	https://webservicenimbus.herokuapp.com/

Microsoft Azure

Application Name	URL/Host
MySQL Database	nimbus06.mysql.database.azure.com

Resources

Hardware

Hardware	
System Manufacturer	Apple Inc.
System Model	MacBookPro8,1
System Type	X64-based PC (Laptop)
Processor	Intel Core i7 (I7-2620M) CPU @ 2.70 GHz
RAM	4 GB
Software	
OS Name	macOS High Sierra
OS Manufacturer	Apple Inc.

Version	10.13.5
---------	---------

Software

Libraries & Frameworks

The framework which is used to build the web application is Sails.js. Sails.js is a Node.js based development framework which follows the MVC (Model View Controller) pattern [1]. It is inspired by the Ruby on Rails web framework and allows us to build REST APIs, single-page application and real-time (WebSocket-based) apps. It follows a “convention over configuration” principle and extensively uses a powerful code generator (blueprint.js) which facilitates us to write less amount of code to build our web application [1].

Sails use ExpressJS in the backend, which is a web-based application framework that provides us with an API to build websites and web apps and handles authentication more efficiently [2]. It also incorporates open source server environment named as Node.js which allows our JavaScript to run on the server. We can say that Sails sits on the top of Node.js and Express.js.

Sails provide a higher layer of abstraction as compared to the express JavaScript in terms of middleware and routing. Like node and angular, it is also a single page application. The flow of the applications is determined through routing, which is explicitly defined. One key feature of Sails.js is Waterline, which is a powerful object-relational mapper (ORM) for SQL-based databases objects document mapper for NoSQL databases [2]. Sails also provide the facility of routing to mapping URL from the actions and view. It guides sails on what to do next when encountering a specified request [1]. User can provide the route explicitly and Sails also provides automatically route using the blueprint. We can also connect MySQL, MongoDB and many more database with Sails without a robust connection to perform the business logic in the web application.

- **JavaScript:** - JavaScript is the weak-type, high-level, multi-paradigm, an interpreted language used to make web pages interactive and dynamic at its core. It does not require any compiler or the interpreter and is also supported in almost all the web browsers [3].
- **jQuery:** - jQuery is a JavaScript library which is used to simplify HTML Document Object Model (DOM) traversal and manipulation, event handling, CSS, and Ajax call. It is also light weight as compared to another framework of the JavaScript [3].
- **Bootstrap:** - Bootstrap provides responsiveness on the website. If we want to develop our web application without writing much CSS code, we can use bootstrap which provides direct blocks of codes. Using Bootstrap, we can create attractive web pages with minimal effort [4].
- **Popper JS:** - It is useful to provide dynamic positioning and viewport detection. It is mainly used for positioning. Many bootstrap components require this library.

Tools

- **Gitlab**
Version control software used for collaborative software development. Link to group repository: <https://git.cs.dal.ca/zala/project--csci-5409---nimbus->
- **Slack**
Team collaboration software tool used for daily communication.
<https://app.slack.com/client/TJW22A6JU/CJW22BGAC>
- **Trello**
Web-based list-making application used for tracking ongoing work progress.
- **Postman**
Postman is an API development tool to test REST APIs. In this assignment, Postman was used to testing the back-end rest API on different environments (Local, Local Container, Cloud Container).

Software

Microsoft Azure – Azure is Microsoft's cloud computing service for building, testing, and deploying applications on the cloud environment. For this application, we use this service to host the MySQL database and perform workflow operations using the Azure Logic App Designer.

Heroku [5] – Heroku is a container-based Platform as a Service (PaaS). Developers uses Heroku to deploy, manage, and scale modern apps. Our platform is flexible, elegant and easy to use, offering developers the easiest path to publish their application on the market. We have deployed all the following applications on the Heroku:

- MBR
- Employer
- Real Estate Broker
- Web service (includes MBR, Employer, Real Estate Broker and Insurance Company)

Steps to deploy web service into Heroku.

- Create account into Heroku
- Login using Heroku CLI.
- Create an application into the Heroku.
- Create and push the image into the Heroku.
- Release the application and container into the web.

We have deployed the following four web-application on the Heroku.

1. Employer Portal
2. Real Estate Portal
3. MBR Portal

4. Webservice

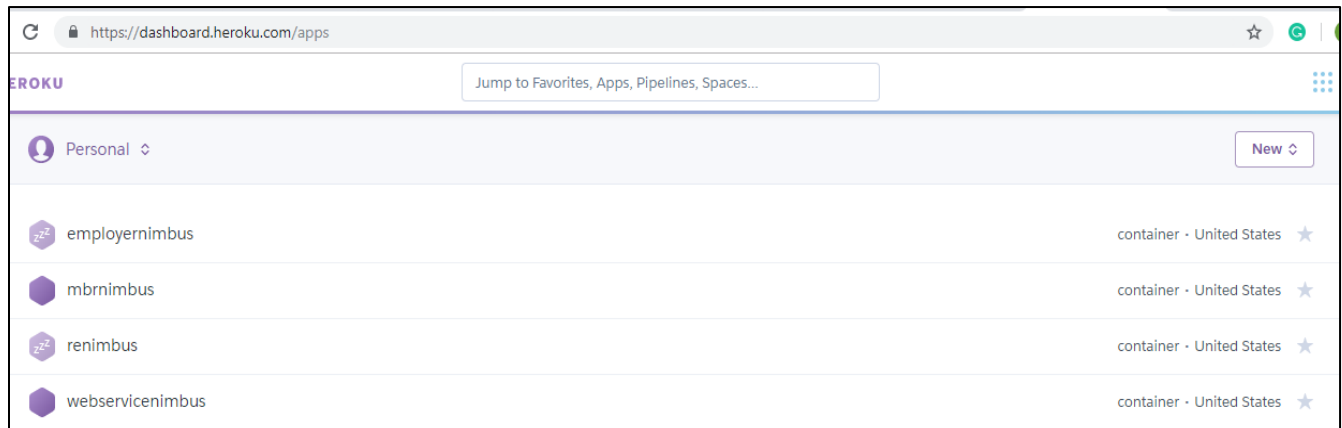


Figure 40: Heroku dashboard

Docker

We have used Docker toolbox for deploying the web service in the local container. Docker provides an environment to deploy and run an application with the help of containers [1]. Further, it allows developers to package the application along with the dependencies and libraries. Thus, docker helps us in deploying the application as a single package and provides an environment to run the application on different platforms. As a result, developers can just focus on writing the code of their application instead of focussing on which machine the application will run. We have created the docker containers for web service, MBR application and employer application.

```

Deep@DESKTOP-P26AKG9 MINGW64 /c/Program Files/Docker Toolbox
$ docker images
REPOSITORY                                TAG                IMAGE ID           CREATED            SIZE
renimbus                                  latest            00d403a15619      7 hours ago       1.18GB
registry.heroku.com/renimbus/web          latest            00d403a15619      7 hours ago       1.18GB
mbrportalnimbus                           latest            637febce9d5f      7 hours ago       1.18GB
registry.heroku.com/mbrnimbus/web          latest            637febce9d5f      7 hours ago       1.18GB
employerportalnimbus                      latest            1a167b2c9dc0      9 hours ago       1.18GB
registry.heroku.com/employernimbus/web     latest            1a167b2c9dc0      9 hours ago       1.18GB
employerproject                           latest            62928bd55406      12 hours ago      1.18GB
webservice                                latest            8c56739d6bc7      12 hours ago      1.21GB
mbr                                         latest            5a25f1c7eb97      4 weeks ago       1.18GB
registry.heroku.com/webservicenimbus/web   latest            f7369346b2d3      4 weeks ago       1.2GB
frontend368                               latest            67aa99545110      8 weeks ago       1.18GB
bmiapp                                     latest            3c266dda6589      8 weeks ago       1.18GB
registry.heroku.com/bmicalculator368/web  latest            3c266dda6589      8 weeks ago       1.18GB
<none>                                     <none>            1773602f7c        8 weeks ago       1.18GB
appdocker                                  latest            e32dd84d6b0b      2 months ago      939MB
registry.heroku.com/young-sands-83416/web  latest            e32dd84d6b0b      2 months ago      939MB
node                                       latest            6be2fabd4196      2 months ago      908MB
node                                       9                08a8c8089ab1      12 months ago     673MB

```

Figure 41: Docker images'

Code Sources

The following code sources were used to solve issues encountered in the project. More details of the issue and how it resolved the issue can be found for each source.

HTTP Requests in Sails JS Controller

One of the troubles we encountered was that the INSinc Controller required to perform a sequential HTTP request to the MBR controller once the Real Estate request arrived. As the INSinc controller was not requested by a view (front end), there was not proper means of redirecting to a view and calling a stored AJAX function. The following source helped install and develop a HTTP request in the INSinc controller itself.

Source: <https://stackoverflow.com/questions/30523872/make-a-http-request-in-your-controller-sails-js>

Password encryption

Plain text password may get exposed. To improve the security, it is essential to encrypt the password into cipher text. In our application, we are storing passwords in encrypted format. We have used Caesar cipher technique to encrypt the password. The password encryption is applied to Employer login, MBR registration and the appraiser login. The password is encrypted prior to adding into the database and making find queries. The following source was used to get an insight about encryption of password in JavaScript.

Source: <http://codeniro.com/caesars-cipher-algorithm-javascript/>

Email validation

During the registration for new mortgage, the user needs to provide email address which can be used by the broker for further communication. Thus, the validation of email is a vital part during registration. We have added a regular expression in order to validate the email address provided by the user. The following source was used to write the regular expression in JavaScript.

Source: <https://stackoverflow.com/a/46181/8243992>

Application status

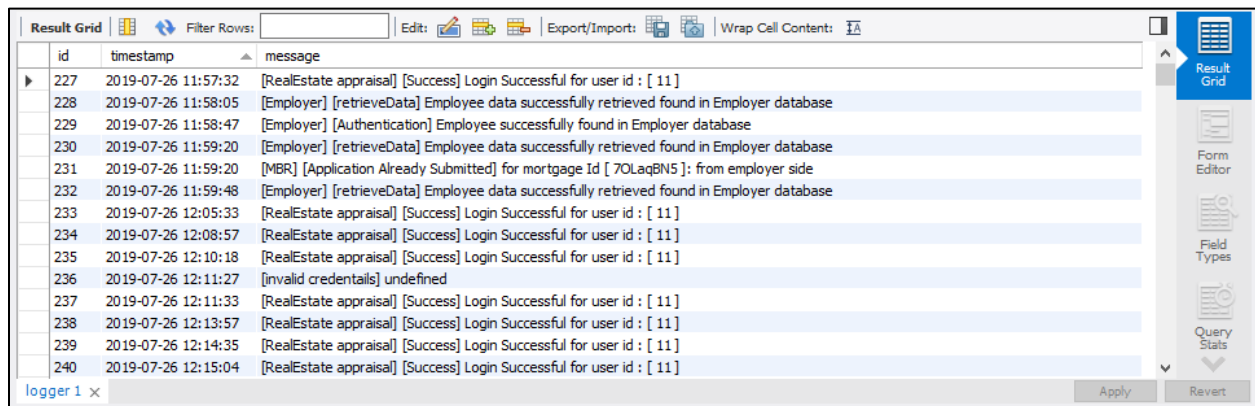
The broker provides a facility to view mortgage application status. The user needs to login to view the status of mortgage application. The application status can be 'pending', 'accepted' or 'rejected'. The following source is used to provide attractive look and feel to the status messages. For example, 'pending' status will be shown in Yellow box, 'accepted' in Green box and 'rejected' in Red box.

Source: https://www.w3schools.com/bootstrap/bootstrap_badges_labels.asp

Log Service Configuration

The log service was configured using a separate logging controller on the web service to generate a history of various logs from the collective application. The function acts as a utility function which can be used by any other controllers in the web service application. The function accepts two inputs which are the location calling the log function and a message describing the log. A sample of the log can be seen in **Figure 42**.

The logging controller houses the log functions which concatenates the two inputs together into a special format for easy readability. The function creates a new entry in the logger database which uses an independent database adapter. This allows us to quickly manage and update the database in case the use of the temporary cloud environment expires.



id	timestamp	message
227	2019-07-26 11:57:32	[RealEstate appraisal] [Success] Login Successful for user id : [11]
228	2019-07-26 11:58:05	[Employer] [retrieveData] Employee data successfully retrieved found in Employer database
229	2019-07-26 11:58:47	[Employer] [Authentication] Employee successfully found in Employer database
230	2019-07-26 11:59:20	[Employer] [retrieveData] Employee data successfully retrieved found in Employer database
231	2019-07-26 11:59:20	[MBR] [Application Already Submitted] for mortgage Id [70LaqBN5]: from employer side
232	2019-07-26 11:59:48	[Employer] [retrieveData] Employee data successfully retrieved found in Employer database
233	2019-07-26 12:05:33	[RealEstate appraisal] [Success] Login Successful for user id : [11]
234	2019-07-26 12:08:57	[RealEstate appraisal] [Success] Login Successful for user id : [11]
235	2019-07-26 12:10:18	[RealEstate appraisal] [Success] Login Successful for user id : [11]
236	2019-07-26 12:11:27	[invalid credentials] undefined
237	2019-07-26 12:11:33	[RealEstate appraisal] [Success] Login Successful for user id : [11]
238	2019-07-26 12:13:57	[RealEstate appraisal] [Success] Login Successful for user id : [11]
239	2019-07-26 12:14:35	[RealEstate appraisal] [Success] Login Successful for user id : [11]
240	2019-07-26 12:15:04	[RealEstate appraisal] [Success] Login Successful for user id : [11]

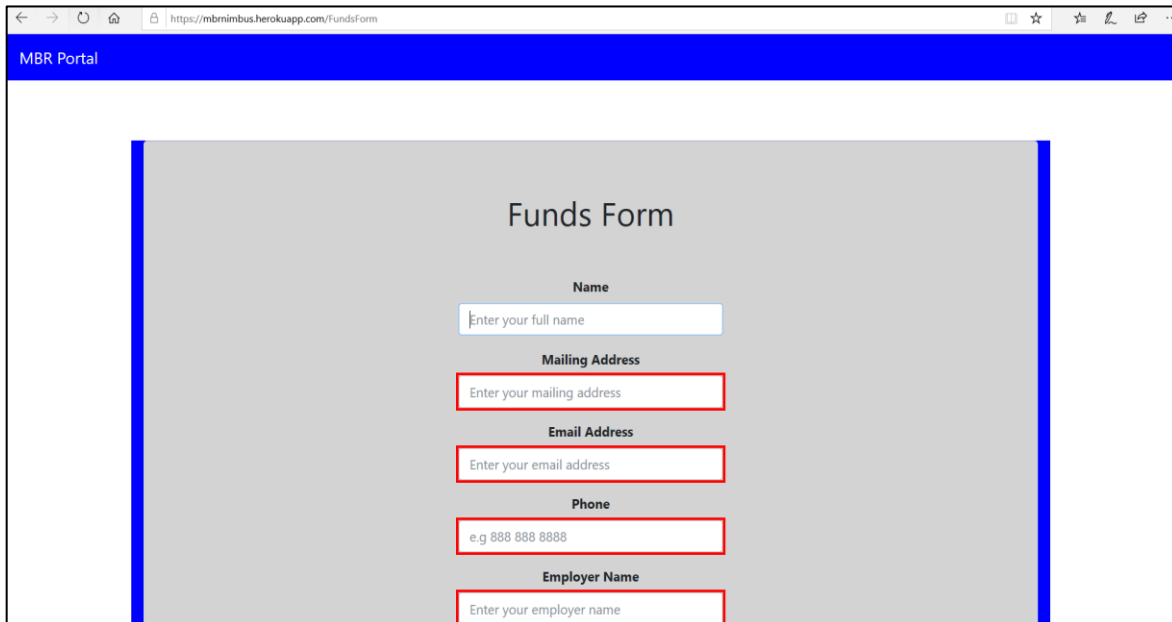
Figure 42: Log Table in MySQL Database

Testing

The following is a list of different test cases to verify proper functionality of each controller function.

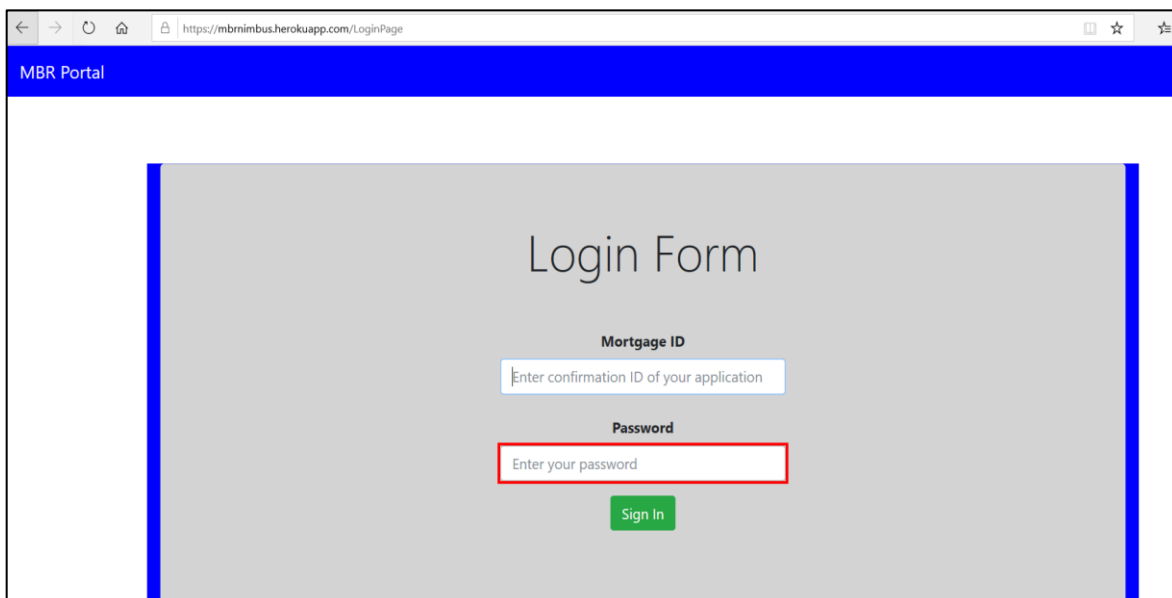
Test Name: Front-end and back-end Testing for MBR portal (new application and login)

The Figure 43 shows that all the fields in the Registration form and login form are mandatory to fill. If any of the field is empty, then error message will be shown.



The screenshot shows a web browser window with the URL <https://mbrnimbus.herokuapp.com/FundsForm>. The page has a blue header bar labeled "MBR Portal". The main content area is a light gray box titled "Funds Form". Inside this box, there are five input fields, each with a red border indicating a validation error. The fields are labeled "Name", "Mailing Address", "Email Address", "Phone", and "Employer Name". The "Name" field contains the placeholder text "Enter your full name". The "Mailing Address" field contains the placeholder text "Enter your mailing address". The "Email Address" field contains the placeholder text "Enter your email address". The "Phone" field contains the placeholder text "e.g 888 888 8888". The "Employer Name" field contains the placeholder text "Enter your employer name".

Figure 43: Registration form front-end validation



The screenshot shows a web browser window with the URL <https://mbrnimbus.herokuapp.com/LoginPage>. The page has a blue header bar labeled "MBR Portal". The main content area is a light gray box titled "Login Form". Inside this box, there are two input fields, each with a red border indicating a validation error. The first field is labeled "Mortgage ID" and contains the placeholder text "Enter confirmation ID of your application". The second field is labeled "Password" and contains the placeholder text "Enter your password". Below the password field is a green button labeled "Sign In".

Figure 44: MBR login form front end validation

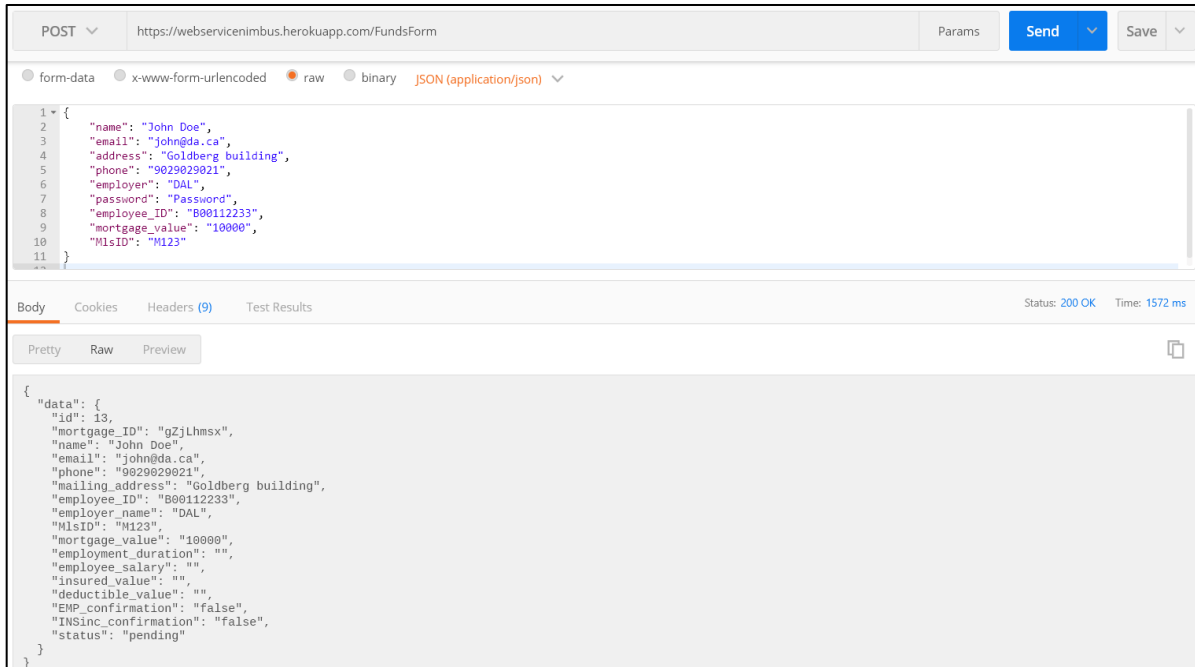


Figure 45 Postman request

id	mortgage_ID	name	email	password	phone	mailing_address	employee_ID	employer_name	MIsID	mortgage_value	employ
13	gZjLhmsx	John Doe	john@da.ca	Cnffjbeq	9029029021	Goldberg building	B00112233	DAL	M123	10000	

Figure 46 Data inserted in database

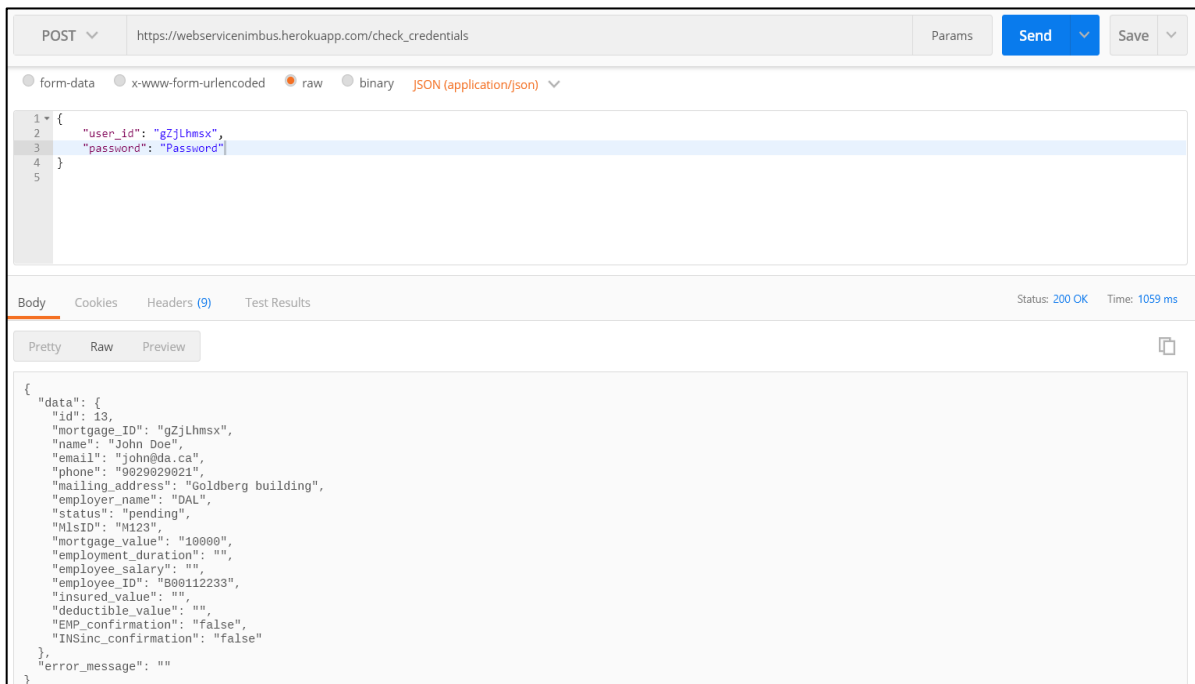


Figure 47 Postman request for login

Test Name: Showing application status to the applicant

An applicant must get confirmation from the employer and the insurance company for the application to be approved. Once the confirmation is received application status will be accepted and database entry will also be updated. In the back-end side, when the user logs in to the portal, the web service returns data as shown in login request of Postman in previous steps (Figure 47). This data can be used to render application status page.

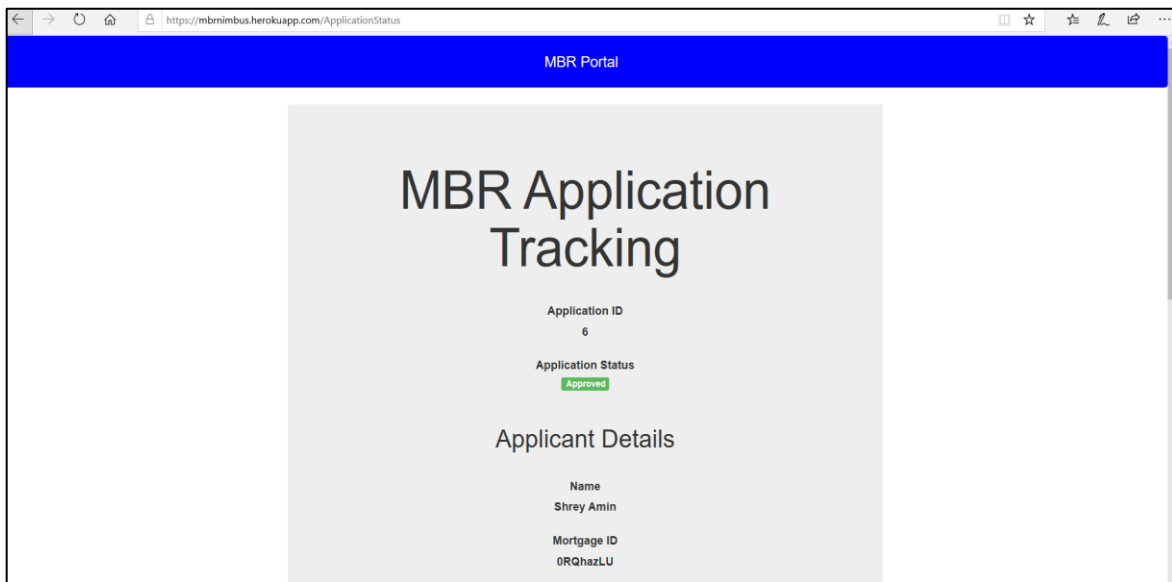


Figure 48: Application is approved

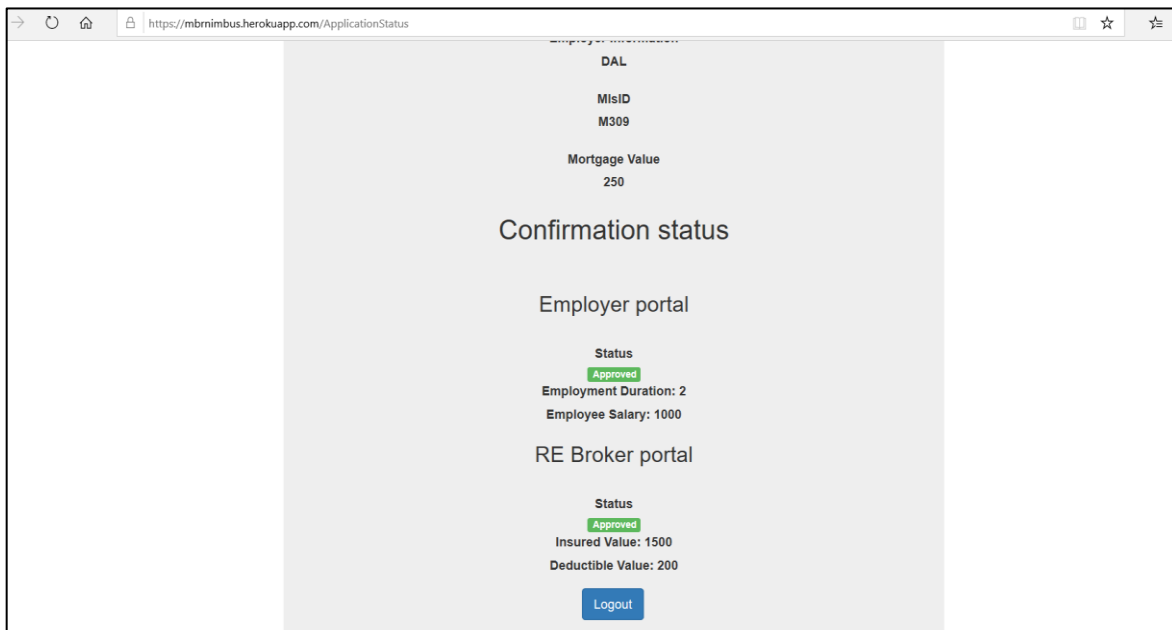


Figure 49: Confirmation from insurance company and employer

insured_value	deductible_value	EMP_confirmation	INSinc_confirmation	status
75000	10000	true	true	accepted
75	10	true	true	accepted
		false	false	rejected
		true	false	pending
3006	400.8	true	true	accepted
1500	200	true	true	accepted

Figure 50: MBR table storing status of each application

If details entered by the user on the employer portal and real estate portal does not match, then application of the applicant will be rejected.

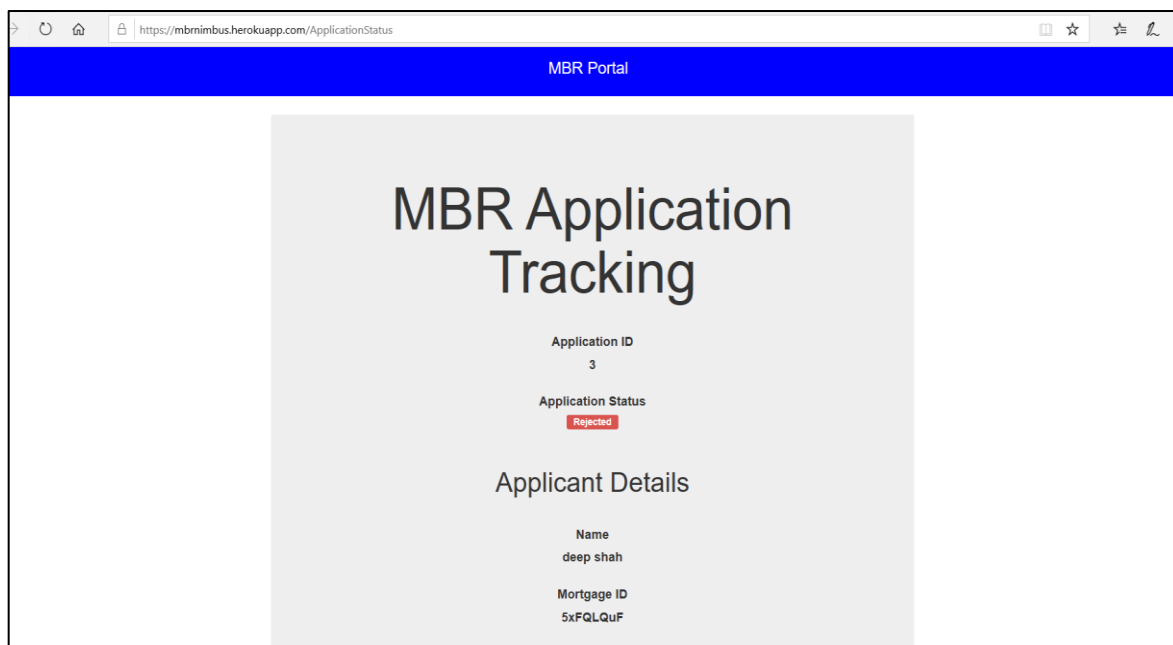


Figure 51: Application status changed to rejected

Test Name: Front-end Testing for MBR portal (validations)

The Figure 52 shows that all the fields in the Registration form and login form are mandatory to fill. If any of the field is empty, then error message will be shown.

The screenshot shows a web browser window with the URL <https://mbnimbus.herokuapp.com/FundsForm>. The page has a blue header bar labeled "MBR Portal". The main content area is a light gray box titled "Funds Form". Inside this box, there are five input fields, each with a label above it: "Name" (with placeholder "Enter your full name"), "Mailing Address" (with placeholder "Enter your mailing address"), "Email Address" (with placeholder "Enter your email address"), "Phone" (with placeholder "e.g 888 888 8888"), and "Employer Name" (with placeholder "Enter your employer name"). Each input field is outlined with a red border, indicating a validation error.

Figure 52: Registration form front-end validation

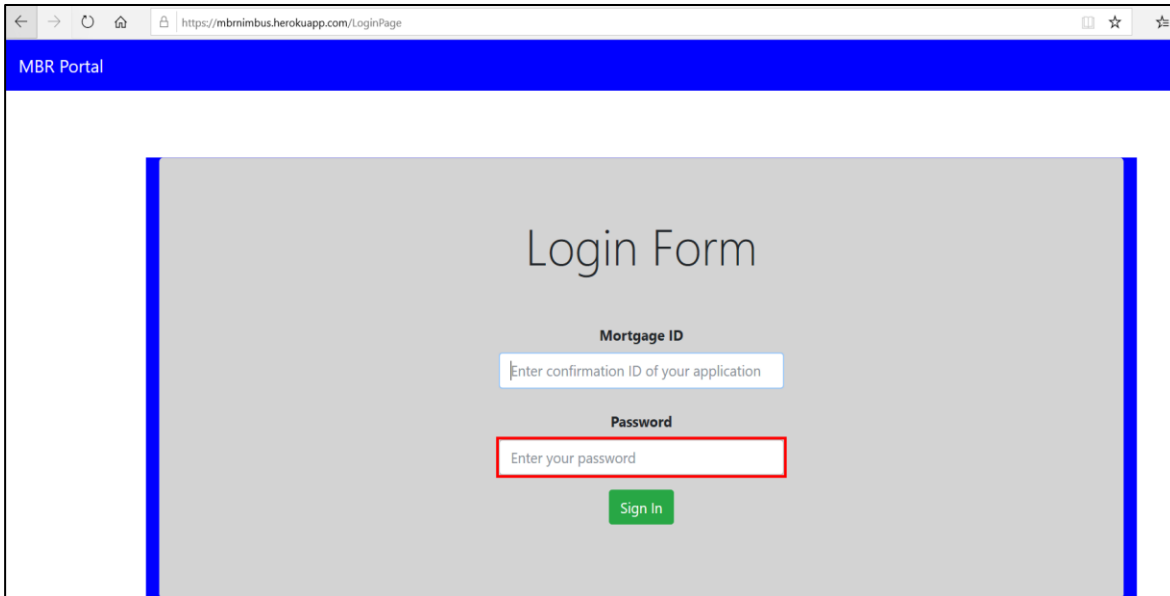
The screenshot shows a REST client interface. The top bar indicates a POST request to <https://webservicenimbus.herokuapp.com/FundsForm>. Below the bar, there are radio buttons for different content types: "form-data", "x-www-form-urlencoded", "raw" (selected), and "binary". To the right of these is a dropdown menu showing "JSON (application/json)". The main area displays a JSON object with the following structure:

```
1 {
2   "user_id": "Sr8XqV1j",
3   "email": "john@da.ca",
4   "address": "Goldberg building"
5 }
6
```

Below the JSON editor, there are tabs for "Body", "Cookies", "Headers (9)", and "Test Results". The "Body" tab is selected. Under the "Body" tab, there are three sub-tabs: "Pretty", "Raw", and "Preview". The "Raw" sub-tab is selected, showing the following JSON response:

```
{
  "error_message": "Please enter all the details!"
}
```

Figure 53 Backend validation demo for missing parameters

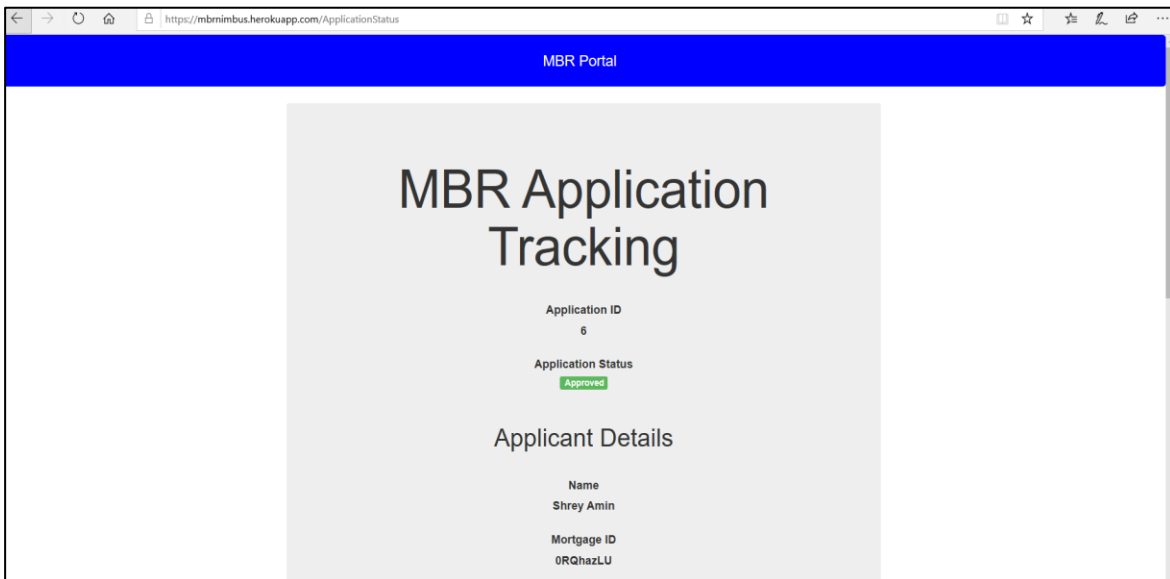


The screenshot shows a web browser window with the URL <https://mbrnimbus.herokuapp.com/LoginPage>. The page has a blue header bar labeled "MBR Portal". The main content area is a light gray rectangle with a blue border. Inside, the text "Login Form" is centered. Below it, there are two input fields: "Mortgage ID" with the placeholder text "Enter confirmation ID of your application" and "Password" with the placeholder text "Enter your password". The "Password" field is highlighted with a red border. Below the input fields is a green "Sign In" button.

Figure 54: MBR login form front end validation

Test Name: Showing application status to the applicant

An applicant has to get confirmation from the employer and the insurance company for the application to be approved. Once the confirmation is received application status will be accepted and database entry will also be updated.



The screenshot shows a web browser window with the URL <https://mbrnimbus.herokuapp.com/ApplicationStatus>. The page has a blue header bar labeled "MBR Portal". The main content area is a light gray rectangle with a blue border. Inside, the text "MBR Application Tracking" is centered. Below it, there are two sections: "Application ID" with the value "6" and "Application Status" with the value "Approved" (highlighted in green). Below these is a section titled "Applicant Details" with the following information: Name: Shrey Amin, Mortgage ID: 0RQhazLU.

Figure 55: Application is approved

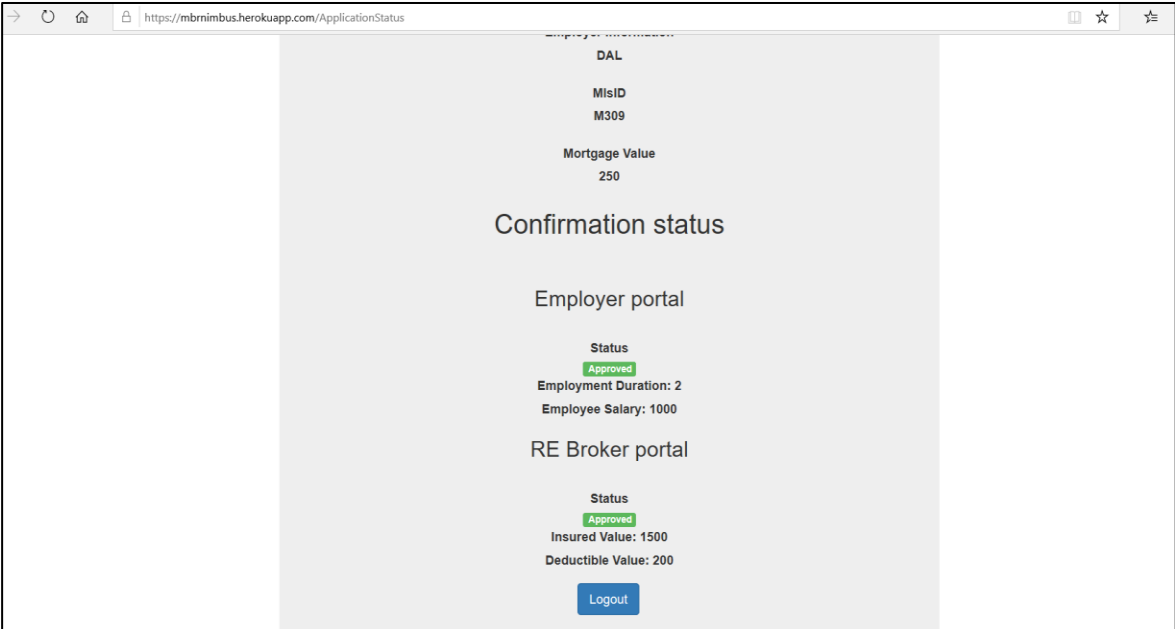


Figure 56: Confirmation from insurance company and employer

insured_value	deductible_value	EMP_confirmation	INSinc_confirmation	status
75000	10000	true	true	accepted
75	10	true	true	accepted
		false	false	rejected
		true	false	pending
3006	400.8	true	true	accepted
1500	200	true	true	accepted

Figure 57: MBR table storing status of each application

If details entered by the user on the employer portal and real estate portal does not match, then application of the applicant will be rejected.

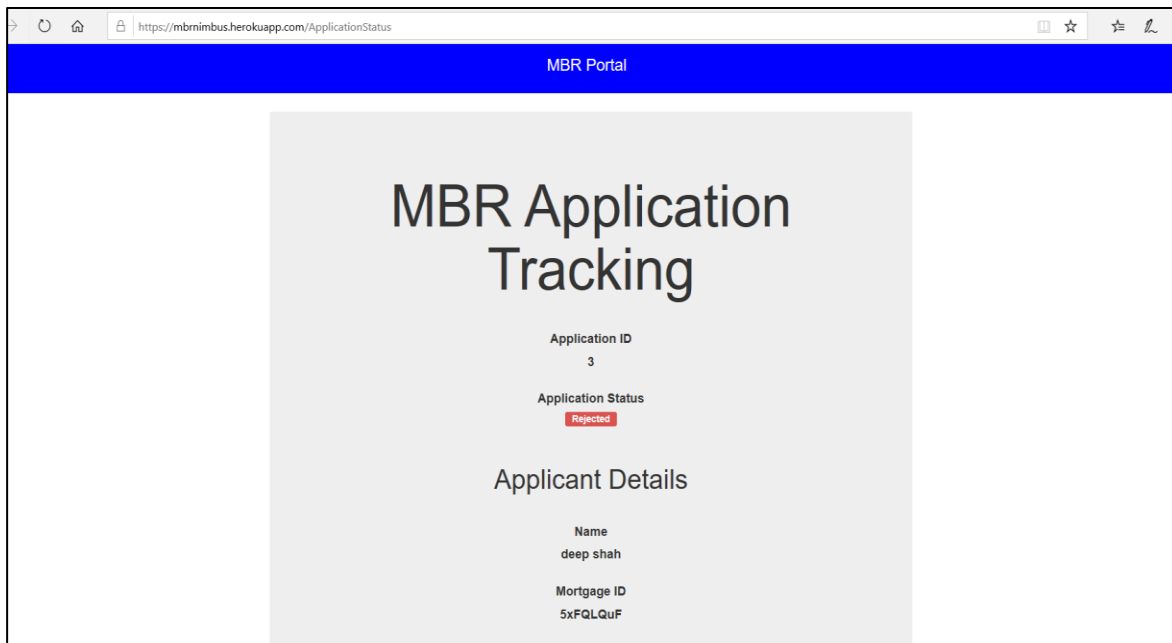


Figure 58: application status changed to rejected

Test Name: Credential testing for mbr login form

As shown in Figure 59, if credentials entered by user are wrong then error message will be shown.

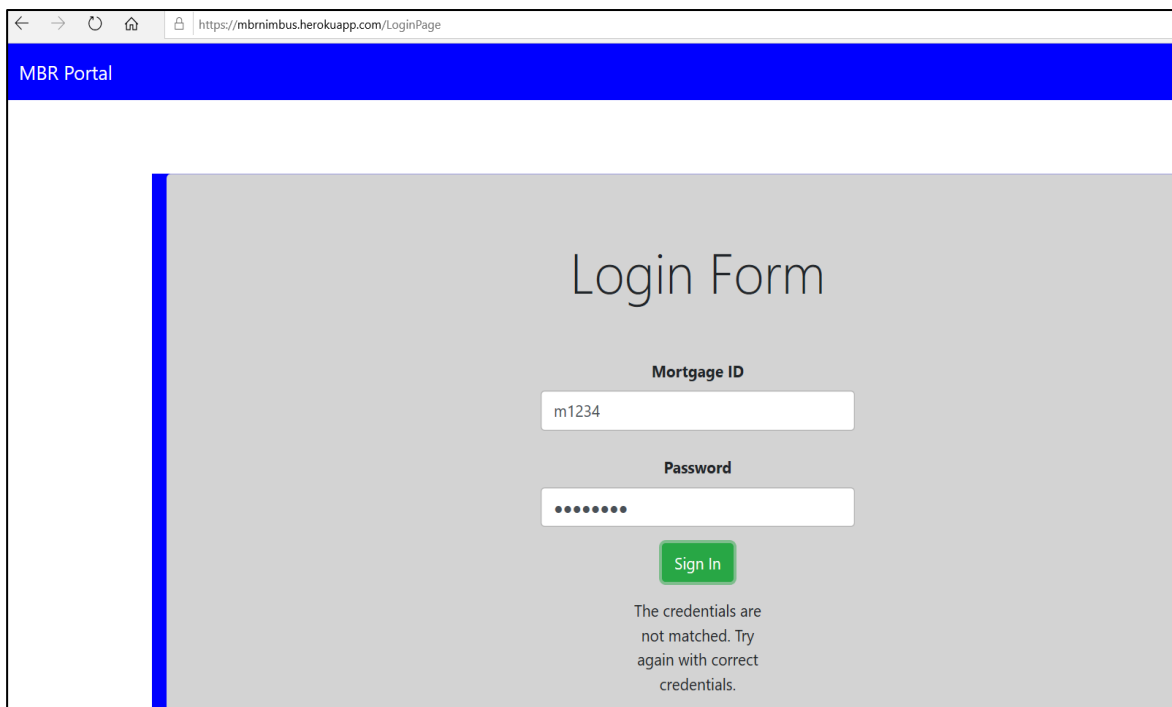


Figure 59: Error message in mbr login form

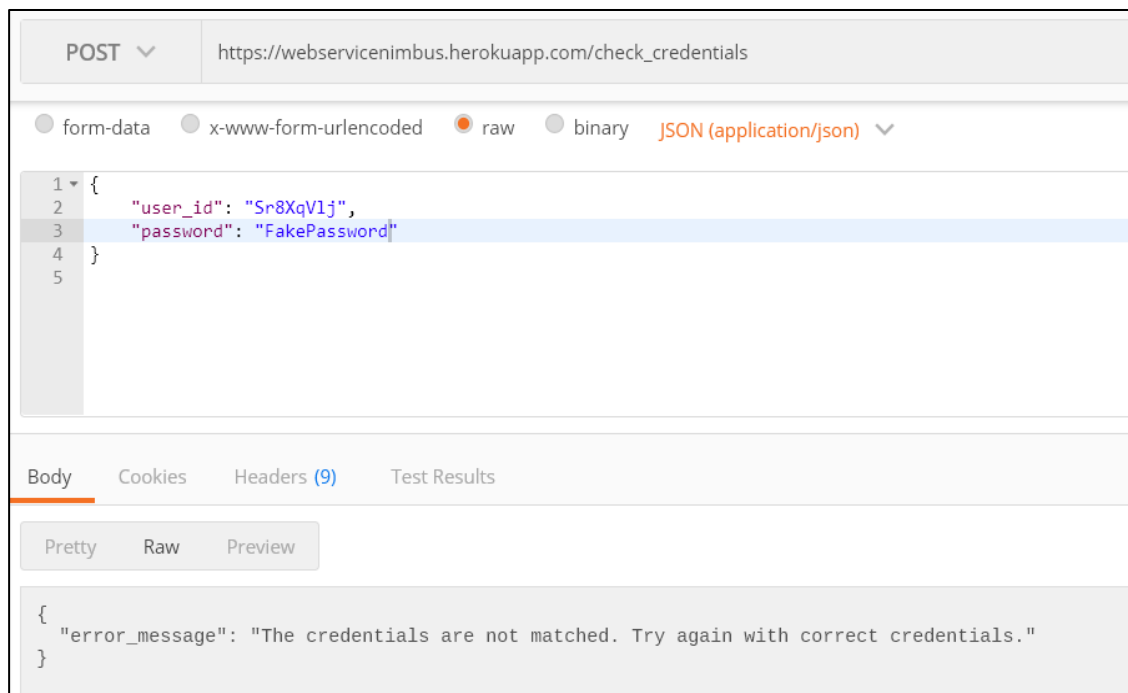


Figure 60 Postman testing for wrong credentials

Test Name: Front-end Testing for Real Estate Portal

As shown in Figure 61, all the fields in the approval form and appraiser login are mandatory to fill. If any of the field is empty, then error message will be shown.

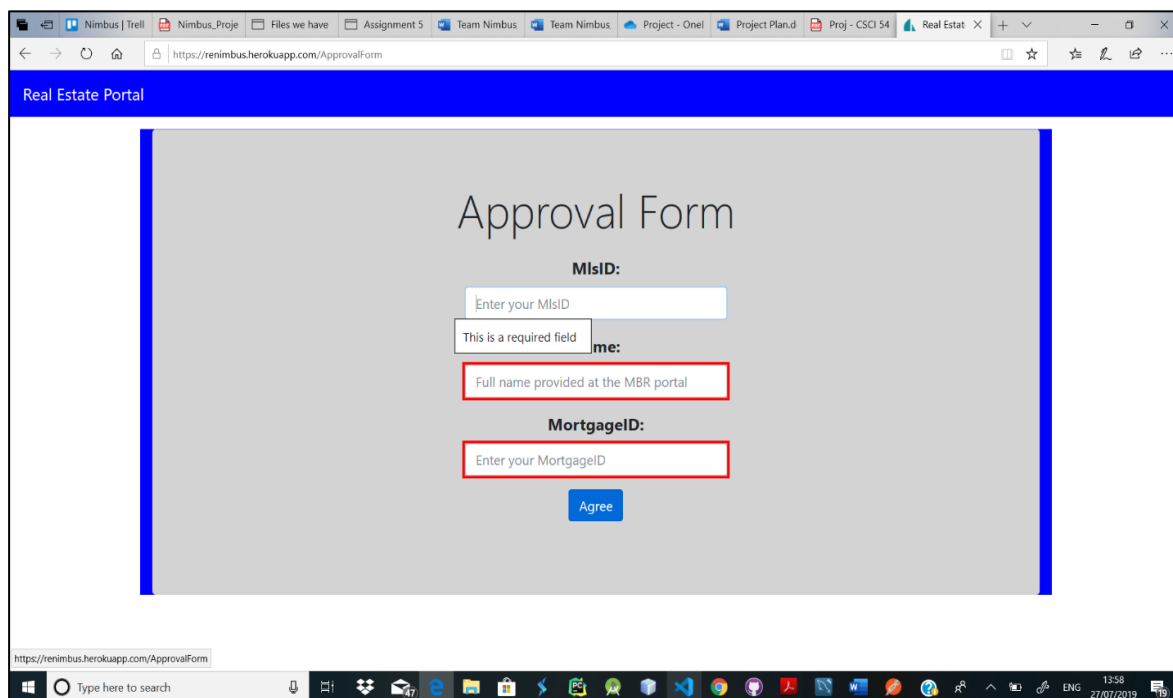


Figure 61: Approval form required field validation

The screenshot shows a web browser window with the URL <https://renimbus.herokuapp.com/AppraisalForm>. The page has a blue header bar with the text "Real Estate Portal". The main content area is a light gray rectangle with a blue border. Inside, the text "Welcome to the Real Estate Portal" is centered. Below it, the label "Appraiser ID:" is followed by a text input field containing "Enter your Appraiser ID". A red border highlights the input field, and a small red box with the text "This is a required field" is positioned to its left. Below the ID field is the label "word:" followed by a text input field containing "Enter your password". A red border highlights the password input field. Below the password field is a blue "Log in" button.

Figure 62: Appraiser form required fields validation

Test Name: Credential Testing for Appraiser form

If appraiser Id or password entered by the appraiser is incorrect error message will be shown in Figure 63.

The screenshot shows the same web browser window as Figure 62. The "Appraiser ID:" field now contains the value "11". The "word:" field is now labeled "Password:" and contains six dots. The "Log in" button is still present. Below the "Log in" button, a red rectangular box contains the text "Incorrect credentials. Please try again".

Figure 63: Invalid credentials (Appraiser login form)

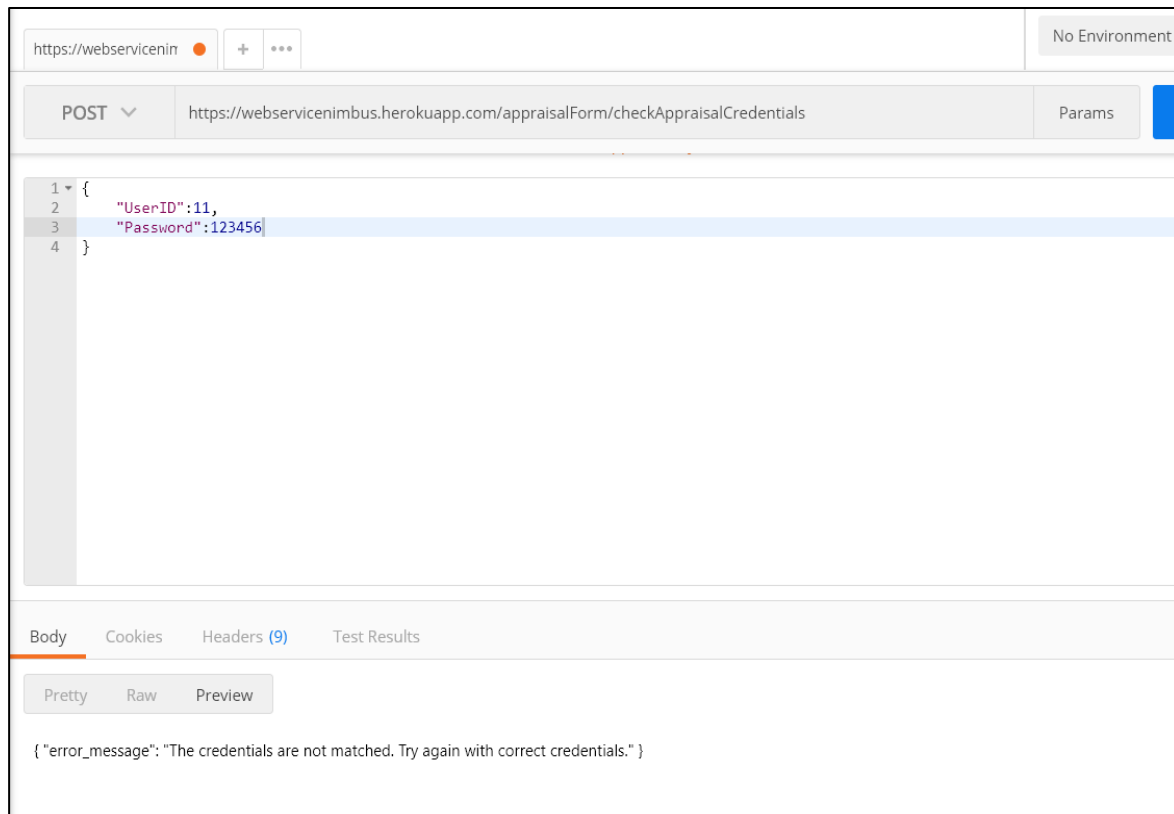
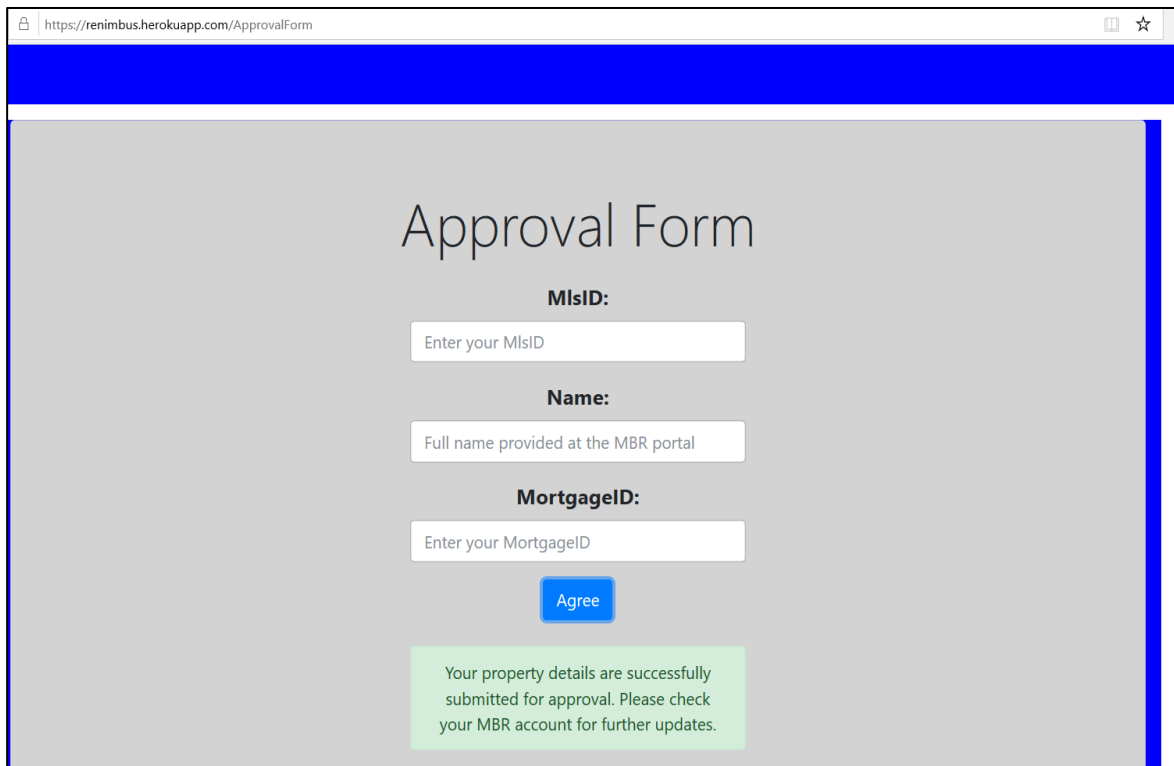


Figure 64: Invalid credentials (Appraiser login form) using postman

Test Name: Testing of Real Estate Approval form

Once the applicants have registered on the MBR portal then they can register at the Real Estate portal. An applicant can register once using their Mortgage ID or else error message will be shown (Figure 65). On successful registration data of the applicant will be stored in the real estate table.



https://renimbus.herokuapp.com/ApprovalForm

Approval Form

MlsID:
Enter your MlsID

Name:
Full name provided at the MBR portal

MortgageID:
Enter your MortgageID

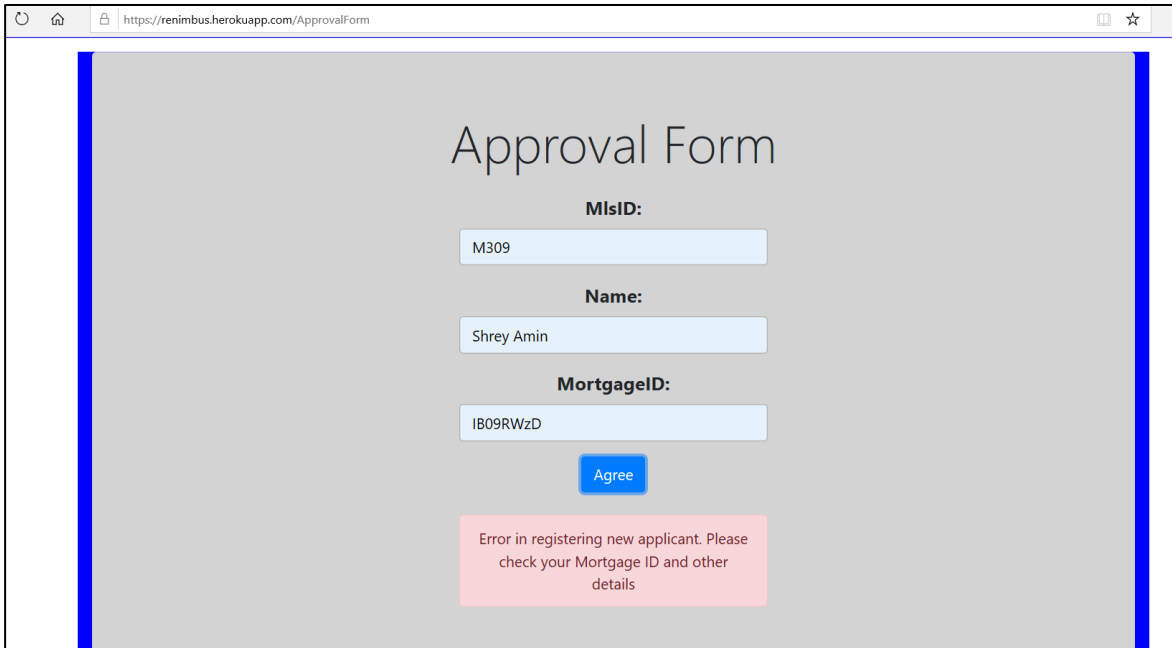
Agree

Your property details are successfully submitted for approval. Please check your MBR account for further updates.

Figure 65: Successful registration at real state portal

	id	MlsID	Name	MortgageID	Value
▶	1	M306	Chintan Patel	7OLaqBN5	1000
	2	M307	Ravi Tulsi Zala	zWlpfzeg	0
	3	M308	deep shah	5xFQLQuF	200
	4	M309	Shrey Amin	IB09RWzD	0
		NULL	NULL	NULL	NULL

Figure 66: Data added into real estate table



A screenshot of a web browser displaying the "Approval Form" at the URL <https://renimbus.herokuapp.com/ApprovalForm>. The form contains three input fields: "MIsID:" with the value "M309", "Name:" with the value "Shrey Amin", and "MortgageID:" with the value "IB09RWzD". Below these fields is a blue "Agree" button. A red error message box at the bottom states: "Error in registering new applicant. Please check your Mortgage ID and other details".

Figure 67: Property appraisal form submission error

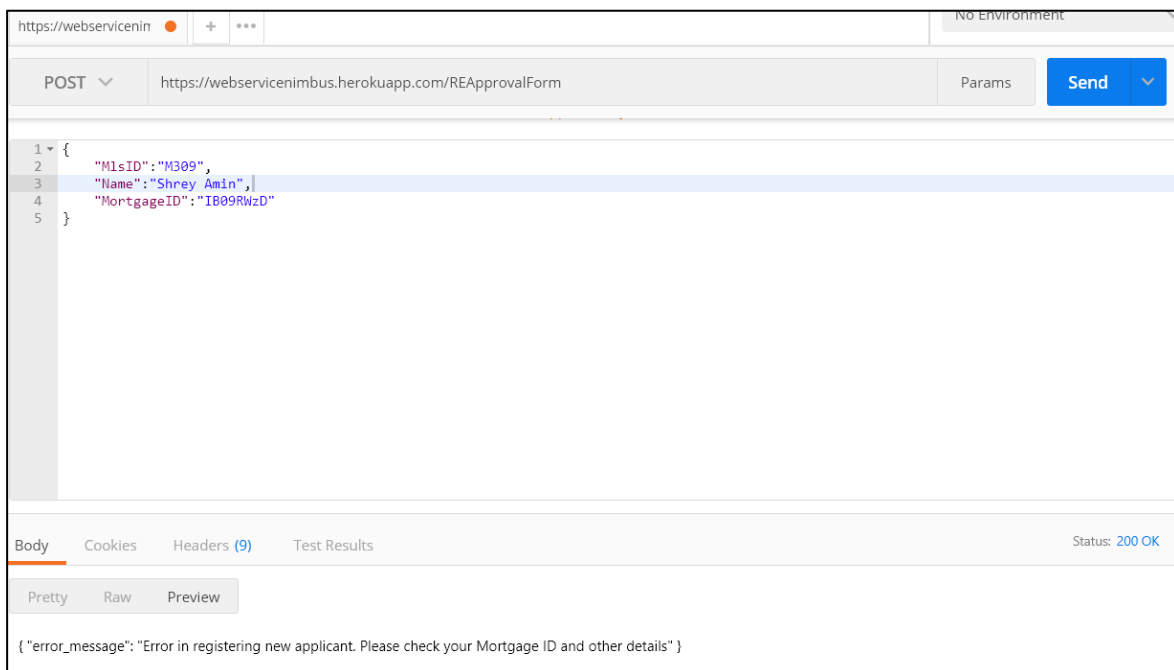


Figure 68: Approval form testing using postman

Test name: Testing Appraiser valuation page

Once the appraiser has successfully logged in, he can provide valuation of all the properties.

MlsID	Value
M307	<input type="text" value="Enter Value"/> <input type="button" value="Submit Query"/>
M309	<input type="text" value="Enter Value"/> <input type="button" value="Submit Query"/>

Figure 69: Appraiser valuation page on successful login

3	M308	deep shah	5xFQLQuF	200
4	M309	Shrey Amin	IB09RWzD	0

Figure 70: entry in the real estate table before the valuation

When appraiser enters the value of the property and clicks submit query button then the records with that MlsId will be updated in the real estate table, and these records will be forwarded to the insurance company for further processing. Here if the value entered by the appraiser is inappropriate, then an error message will be shown(Figure 71).

MlsID	Value
M307	<input type="text" value="Enter Value"/> <input type="button" value="Submit Query"/>
M309	<input type="text" value="Enter Value"/> <input type="button" value="Submit Query"/>

This site says...

Please pass value

Figure 71: Error message on submitting in appropriate value

MIsID	Value
M307	<input type="text" value="Enter Value"/>

Figure 72: Submitting the valuation of the property (after submission MIsID M309 will disappear)

3	M308	deep shah	5xFQLQuF	200
4	M309	Shrey Amin	IB09RWzD	2000

Figure 73: Updated entry in the real estate table (for MIsID M309)

Test: Insurance Quote Test

This test case showcases that the INSinc Controller on the web service successfully handles incoming requests from the Real Estate broker and forwards the requests to the Mortgage Broker with the newly calculated deductible and insured values.

The following **figure 74** shows that the current application for mortgage id 'xiM1wYuX' has not received any information from the Insurance company.

Result Grid						
Filter Rows: <input type="text"/>						
Edit: <input type="button" value="Edit"/> <input type="button" value="Add"/> <input type="button" value="Delete"/>						
Export/Import: <input type="button" value="Export"/> <input type="button" value="Import"/>						
Wrap Cell Content: <input type="button" value="Wrap"/> <input type="button" value="Unwrap"/>						
	id	mortgage_id	MIsID	insured_value	deductible_value	INSinc_confirmation
7	xiM1wYuX		M524			false
	NULL	NULL	NULL	NULL	NULL	NULL
						pending
						NULL

Figure 74: Database showing that application has not received any insurance information

A postman request is prepared as shown in **figure 75**. with a valid set of parameters for the Insurance company to parse and process. The function *insuranceQuote()* is invoked and 'OK' response message is returned to indicate that all information has successfully been processed in the Insurance Controller.

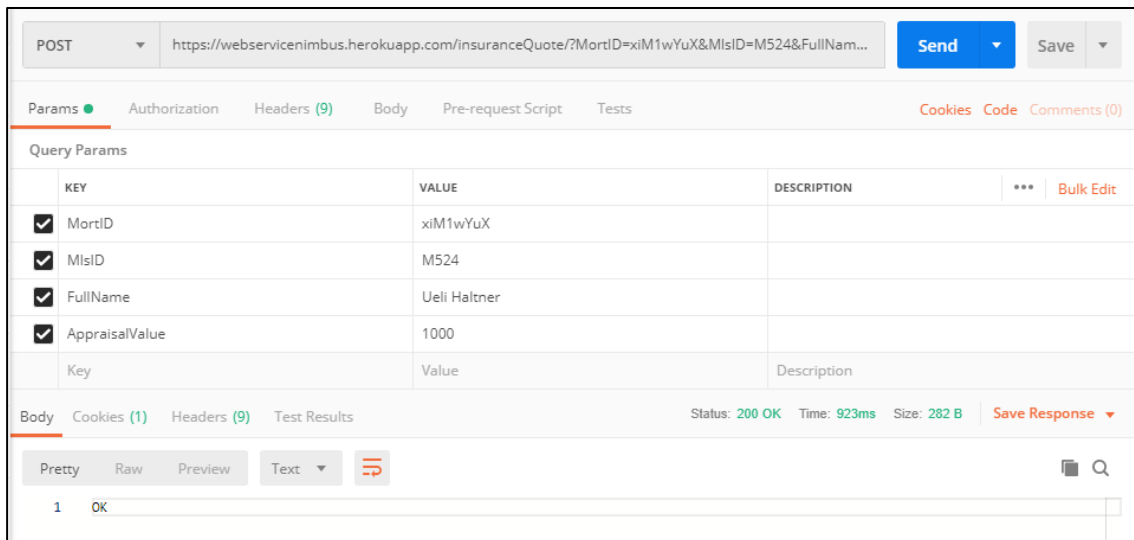


Figure 75: Postman POST request to insurance controller with a valid set of inputs. This request is simulating a http request which is provided by the Real Estate portal. Returned message indicates successful execution.

The newly processed information is then forward to the MBR portal and updated in the database as shown in figure 76.

Result Grid							
Filter Rows:							
	id	mortgage_id	MlsID	insured_value	deductible_value	INSinc_confirmation	status
▶	7	xiM1wYuX	M524	1500	200	true	pending
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 76: Database showing that the application has successfully received the insurance quote.

Project Plan Execution Summary

The initial plan for this assignment included breaking up the work into different categories as demonstrated in the Project Plan document. The Project Plan document can be found as a separate attachment.

Each category was divided into smaller segments which could be distributed among team members. The expertise of each team member was taken into consideration based on work demonstrated in previous assignments. The project plant shows which team member was assigned to each task as well as any additional supporting roles. Furthermore, tasks such as documentation and testing requirements were assigned to all team members.

Difficulties Encountered:

- We faced major difficulties while generating the token in the login for this project as Initially we were not able to fetch the authorization header.

- Initially we were using the auto increment index of the rows as the mortgage id in the MBR database. This caused an issue where one could simply create a typo and enter another person's mortgage id. As they were ordered, the chances of the previous and next mortgage ID being active (pending state) was extremely high. Therefore, a mistake in the submitted mortgage ID could result in rejection status of another request. Our solution was to generate a random mortgage ID which consisted of 8 characters and included lower case, upper case, and numeric. Also, this value is checked in the database to ensure no duplication. If a duplicate value is found it generates a new ID and repeats the check until it is unique.

Deviations from the Original Plan:

- As per our initial plan, we were storing the raw password in the database. During the implementation, we realized that it is not good practice to store the original value of the password for security reasons. Therefore, we now encrypt the password in the Caesar cipher format and stored it in the database.

Scalability

There are different parts of our project like MBR, Employer, Real Estate Broker and Insurance company. So, in future we need to consider scaling of the web service deployed on the cloud. Currently, there is only one common web service with different controllers of each company. We will be using the automated horizontal scaling for increasing the performance of the web service. The main idea in automated scaling is to increase the number of instances of web service when threshold capacity of the single instance has reached.

We will be using Microsoft Azure for scaling as it provides an internal load balancer which will evenly distribute number of requests across all the instances of the web service. Moreover, will be also create separate web service for each company so that requests can be handle by different web service based on the task to be performed. This will reduce will workload on the single web service.

Security Mechanisms

Password Encryption (Caesar Cipher)

The passwords encryption is a vital part of security measures. If the password is stored in plain text, it may get exposed if any security breach occurs. Thus, we decided to encrypt the password prior to storing in the database. We are using Caesar cipher algorithm to encrypt the plain text password into cipher text. The password encryption is implemented in MBRController, EmployerController and RealEstateController.

Security Tokens

We have generated token for authentication in Employer Portal. Token generation is done using json web token package which is provided by node. Once the employee logs in the portal then the token will be generated and then stored in the session storage which is valid for 1 minute. Once the token gets expired then employee will be automatically redirected to the login page and needs to sign in again and new token is generated now. Json web token works with authentication headers in the request which means that the token will be sent in the headers with “Bearer” in start and followed by token, which will be then decoded at the time of verification. In policies.js all the routes for that particular webservice are made private and only authorized users will be allowed to access those.

Below figure shows that the token is appended in the URL

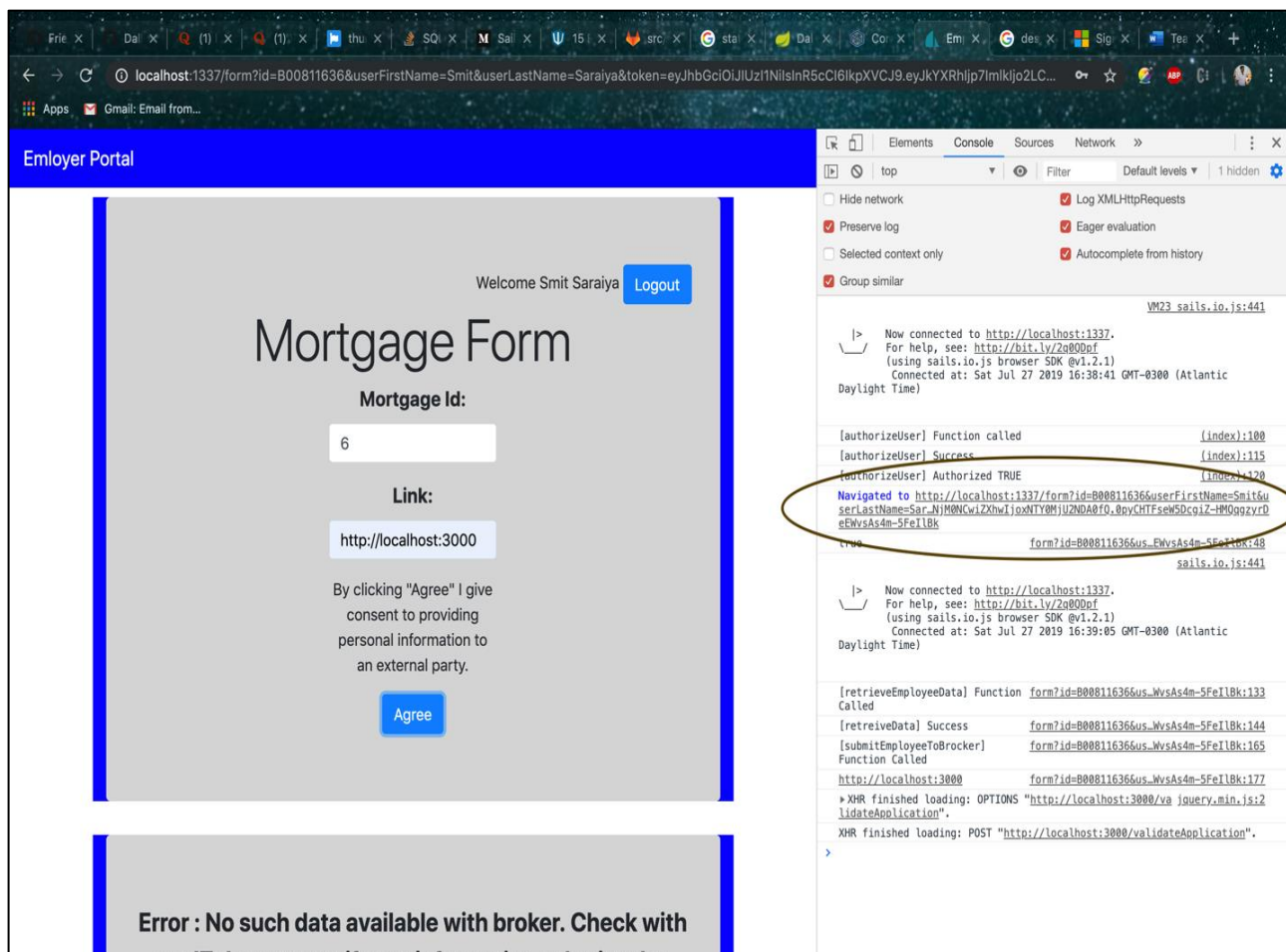


Figure 77: Token in URL

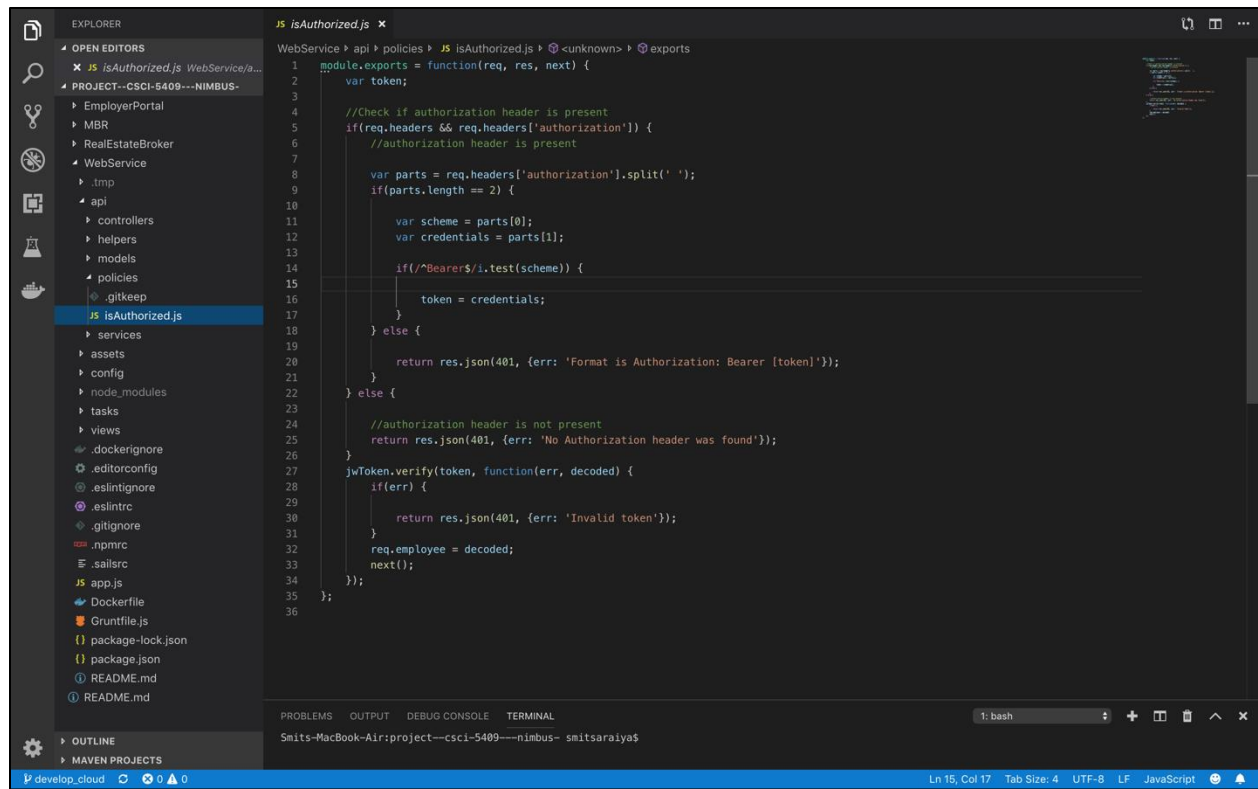


Figure 78: Authorization of Token

```
1
2   var jwt = require('jsonwebtoken');
3
4   module.exports = {
5     'sign': function(payload) {
6       return jwt.sign({
7         data: payload
8       }, 'mysecret', {expiresIn: '1m'});
9     },
10    'verify': function(token, callback) {
11      jwt.verify(token, 'mysecret', callback);
12    }
13  };
```

Figure 79: Token Generation

References

- [1]"What is Sails.js and what's new with the upcoming 1.0 version", CloudBoost, 2019. [Online]. Available: <https://blog.cloudboost.io/what-is-sails-js-and-whats-newwith-the-upcoming-1-0-version-38672f41ea2d>
- [2]T. Node.js IDEs, "An Introduction to Sails.js — SitePoint", SitePoint, 2019. [Online]. Available: <https://www.sitepoint.com/an-introduction-to-sails-js/>
- [3]"WHAT ARE THE BENEFITS OF LEARNING JAVASCRIPT," Boostlog, 2019. [Online]. Available: <https://boostlog.io/@sonuton/what-are-the-benefits-of-learningjavascript-5a87b3669837780090b3e833>
- [4]a. Mark Otto, "Introduction," Getbootstrap.com, 2019. [Online]. Available: <https://getbootstrap.com/docs/4.3/getting-started/introduction/>
- [5]"What is Heroku | Heroku", *Heroku.com*, 2019. [Online]. Available: <https://www.heroku.com/what>. [Accessed: 29- Jun- 2019].
- [6]"Sails Backend for Angular2+ Auth", *Medium*, 2019. [Online]. Available: <https://medium.com/@theophilus.omoregbee827/sails-backend-for-angular2-auth-2c6126b07e9d>. [Accessed: 27- Jul- 2019].
- [7] U. Haltner, C. Patel, S. Amin, D. Shah, S. Saraiya, R. Zala, "Assignment-5", CSCI5409 – Advanced topics in Cloud Computing, 2019. [Accessed: 27- Jul- 2019].