IN PARTNERSHIP WITH PLYMOUTH UNIVERSITY

Name: Krishan Shanuka
Student Reference Number: 10707370

Module Code: PUSL3111	Module Name: API Software Development
Coursework Title: C1	
Deadline Date: 11/05/2022	Member of staff responsible for coursework: Dr. Rasika Ranaweera
Programme: BSc. Software Engineering	

Please note that University Academic Regulations are available under Rules and Regulations on the University website www.plymouth.ac.uk/studenthandbook.

Group work: please list all names of all participants formally associated with this work and state whether the work was undertaken alone or as part of a team. Please note you may be required to identify individual responsibility for component parts.

10707370 Krishan Shanuka
10707426 Sanjana Witharanage
10707273 Manuja Mallikarachchi
10707360 Kavindya Sandeepanie

All members contributed equally

We confirm that we have read and understood the Plymouth University regulations relating to Assessment Offences and that we are aware of the possible penalties for any breach of these regulations. We confirm that this is the independent work of the group.

Signed on behalf of the group: 

Krishan Shanuka

Individual assignment: ***I confirm that I have read and understood the Plymouth University regulations relating to Assessment Offences and that I am aware of the possible penalties for any breach of these regulations. I confirm that this is my own independent work.***
Signed:

Use of translation software: failure to declare that translation software or a similar writing aid has been used will be treated as an assessment offence.
I have not used translation software.
If used, please state name of software……

Overall mark _____% Assessors Initials _____ Date_____

*Please delete as appropriateSci/ps/d:/students/cwkfrontcover/2013/14

CONTENTS

FIGURES

1 PROJECT IDENTIFICATION

1.1 INTRODUCTION

The Sri Lanka Bureau of Foreign Employment (SLBFE) mission is to make "Sri Lanka the finest choice for qualified human resources for the international market." The agency assists citizens in finding work outside of the country. Still, it also helps businesses find qualified personnel, monitor the well-being of foreign workers, and resolve family disputes. The existing website is outdated, and the bureau plans to open up its facilities so that businesses and other parties can access the vast amount of data accumulated over the years.

There are three main parties in this system. They are admins, officers, and jobseekers.

Jobseekers

i. Any citizen can become a member through free online registration.
ii. The citizens who seek jobs must be able to update their qualifications and upload their birth certificates, CVs, and copies of the passports through the system.
iii. The citizens who have gone for foreign employment must update their current location as soon as they visit the foreign company.
iv. Any citizen can make a complaint

Officers

i. The bureau officers must be able to see and validate information provided by the job seekers.
ii. Foreign companies should be able to find workers based on their qualifications.
iii. The bureau officers should be able to see the complaint and reply accordingly

Admin

i. Create new users.
ii. Delete users.
iii. Get users matched for the qualifications.

This system is considered with two types of implementations. They are,

a) Web Application

b) Mobile Application

1.2 PROJECT GOALS

This project's primary goal is to develop a RESTful API web application and a mobile application for the given assignment scenario.

Our second goal was to learn something new and enhance our skills and technical knowledge as a team and collaborate.

1.3 PROJECT OBJECTIVES

- Analyze the given problem clearly and thoroughly before presenting a suitable, practical answer.

- A functional mobile application and a web application were designed and developed.

- Decision-making assistance

  This method provides information about the covid positive patients' direct contacts. Then, because they have the appropriate information, healthcare workers may contact them soon and make vital decisions.

- friendly to the user

  This system was created for three primary users: the general public(job seekers), officers and admins. The system is somewhat complicated according to the functionalities included. The system's functionality will be evident to any user, and working with it will be simple.

- Responsive designs

  The User Interface (UI) of this system is of outstanding quality. As a result, this system relied on graphics, simple button types, and font sizes.

- Productivity

  The system aims to track the covid positive patients' immediate contacts. As a result, the spread of the covid virus is halted. The QR scanner in the system saves the consumers time.

- User Interaction

  This system should provide consumers with a positive user experience.

- Documentation of the project, which is properly done.

1.4 PROJECT SCOPE

i. Any citizen can become a member through free online registration.
ii. The citizens who seek jobs must be able to update their qualifications and upload their birth certificates, CVs, and copies of the passports through the system.
iii. The bureau officers must be able to see and validate information provided by the job seekers.
iv. Foreign companies should be able to find workers based on their qualifications.
v. The citizens who have gone for foreign employment must update their current location as soon as they visit the foreign company.
vi. Any citizen can make complaints, and the bureau officers should be able to see the content and reply accordingly

1.5 ASSUMPTIONS

For the development of this API, the designers of this system had to come up with assumptions. Those assumptions are as follows,

- The whole system consists of three user account types, and those roles were named ‘Citizen’, Officer’ and ‘Staff’. The ‘Citizen’ user type handles the role of the ‘Job Seeker’. The ‘Officer’ user type handles the role of ‘Foreign Hiring Company Staff’. The ‘Staff’ user type handles the role of ‘SLBFE Staff’.

- The ‘Job Seeker’, also known as the ‘Citizen’ in the system, must upload their CV during the signup process.

2 PLANNING

Project planning is a field that deals with how to complete a project within a set time frame, usually using pre-determined stages and resources. One approach to project planning splits the process into the following steps: establishing measurable goals, determining the deliverables, and scheduling and Organizing tasks. Team members have to clarify the project's scope, objectives, goals, and schedule to go through those steps. As the project's planning was done at first successfully, team members were able to finish the project the same as the plan.

3 ANALYZE

3.1 FUNCTIONAL REQUIREMENTS

From the jobseeker aspect

- Citizen registration

- Citizen login
- Update personal information
- Update job seeker information
- Add new complaints, reply to the solutions
- Change password

From the officer aspect

- Officer registration
- Officer login
- Find jobseekers
- Change password

From the admin aspect

- Admin login
- Admin registration
- Find jobseekers
- Add, edit, view and delete officers
- Add, edit, view and delete admins
- Receive complaints from the job seekers, reply to the complaints and mark them as resolved
- View logs

3.2 NON FUNCTIONAL REQUIREMENTS

- Reliability

Reliability means how the system continues to run continuously and performs specified functions without harming the system.

- Maintainability

Maintainability refers to how easily a product may be managed to correct flaws or causes.

- Performance

The software application or its components should reply to the user under a specific workload in a certain amount of time.

- Usability

Anyone should be able to learn and use a system. This software system was constructed by gathering user requirements and is designed for anyone to use.

- Security

The system should be safe and protected from third-party access to the user's data and information.

- Safety

The system should protect people and the environment from harm.

4 DESIGNING

4.1 SYSTEM ARCHITECTURE DIAGRAM

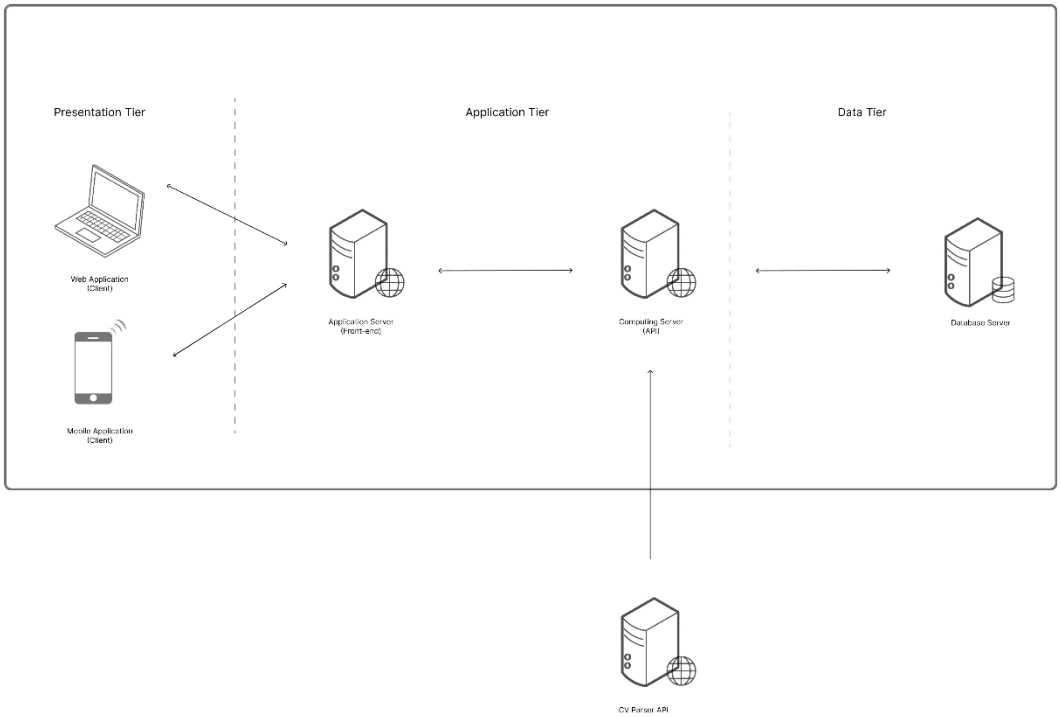


Figure 4.1: System Architecture Diagram

According to the above diagram, this system mainly contains a Mobile Application, Web Application, Rest API, Database and Cloud Storage. So, most of the components of the system should be hosted in the cloud.

Developers of this project decided to select Azure related services to host this system because azure has unlimited support to host Microsoft's own ASP. Net APIs on their App Services.

- Web Application deployed to Azure static website container, with the help of GitHub actions pipeline.

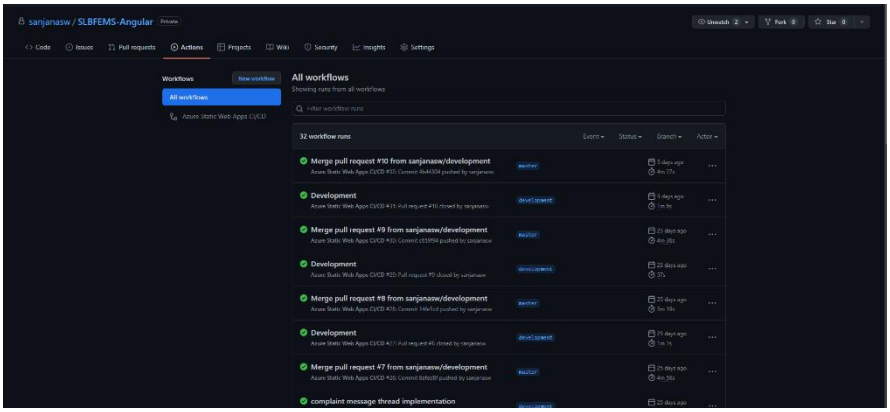


Figure 4.2: Github Action for Client Application

- Rest API deployed to Azure app service, with the help of GitHub actions pipeline.

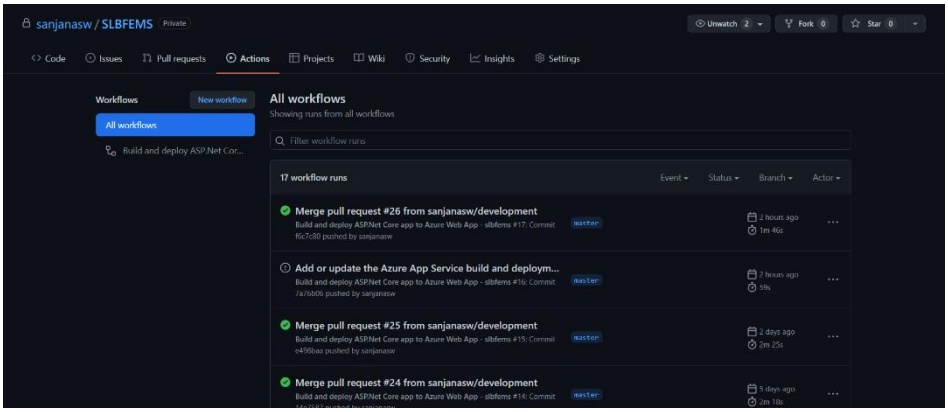


Figure 4.3: Github Action for API

- To store the data, this system uses Azure MSSQL Database Server, and for storing larger files like CVs, Passports, and Birth Certificates, this system connects with the Azure Storage bucket.

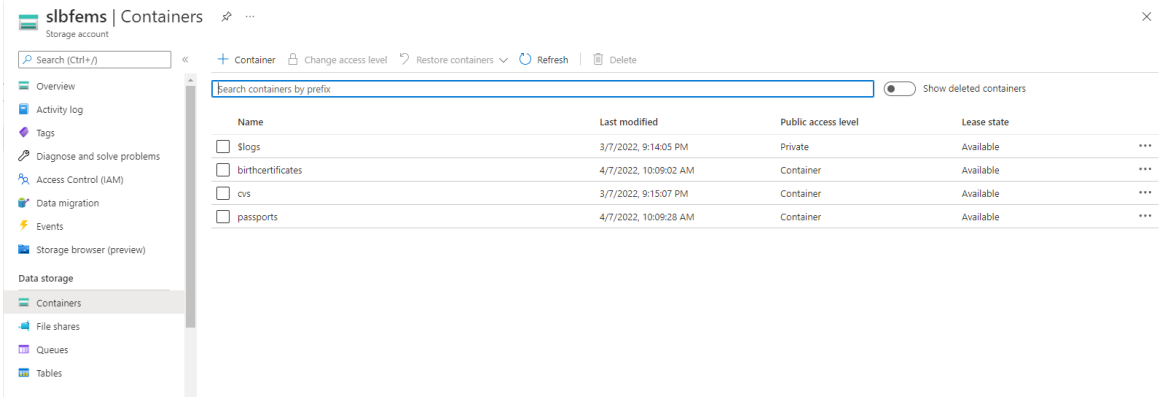


Figure 4.4: Azure Container Structure

4.2 USE CASE DIAGRAM

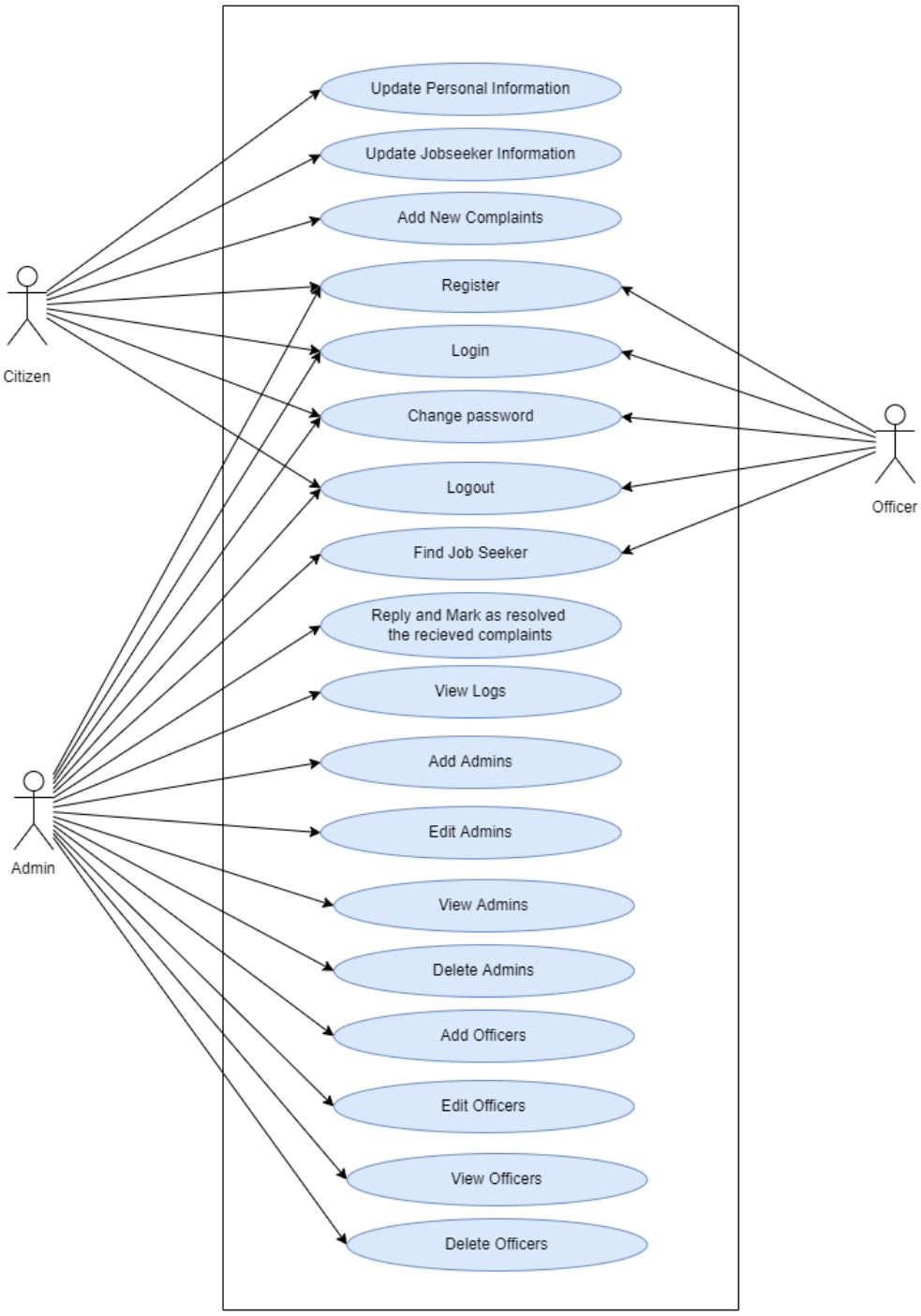


Figure 4.5: Use-case diagram

4.3 DATABASE DESIGN

4.3.1 ENTITY-RELATIONSHIP DIAGRAM

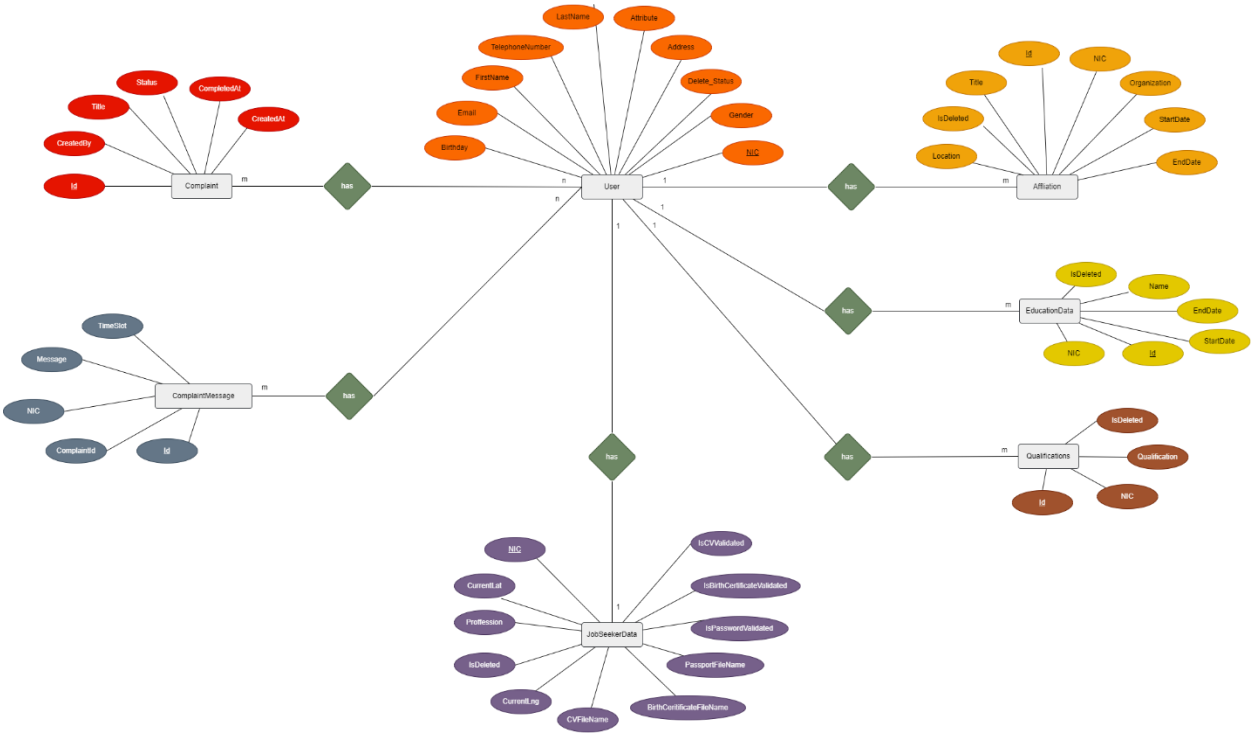


Figure 4.6: ER diagram

5 TECHNOLOGIES USED

Technologies used in this project,

- For backend: ASP.NET core web API
- For mobile application front-end: Flutter
- For web application front-end: angular

5.1 FRONT-END

5.1.1 FLUTTER

Flutter, React Native, Swiftic, Xamarin, Ionic, Corona, JQuery Mobile, and Java are some of the current front-end technologies for mobile applications. Although there are many different technologies, there aren't as many resources to refer to them as there are in React Native, Flutter, and Java. (Flutter Developers & Contributors, n.d.) The main reason is that while designing the front-end, the author is likely to encounter a slew of mistakes, defects, and other issues resulting from differences in operating systems, IDEs, and versions. If there are previously established solutions available at that time, it will benefit the author. Flutter, React Native, and Java has a higher presence on YouTube, StackOverflow, and other popular websites than the other technologies mentioned above. The author's project was chosen since the Flutter community is significant now and will continue to grow in the future. In addition, there is adequate documentation for this. Android Studio, Visual Studio Code, IntelliJ IDEA, and the Emacs IDE are good options for developing Flutter apps. (Flutter Developers & Contributors, n.d.)

Flutter is open-source and a user interface software development kit produced by Google. It's used to create cross-platform apps from a single codebase for Android, iOS, Linux, macOS, Windows, Google Fuchsia[4], and the web. (Wikipedia Contributors, 2019)

5.1.2 ANGULAR

The front-end of websites, like mobile apps, is built using a variety of technologies. Vue.js (JavaScript library), Npm, Vue.js (web framework), Ionic 4, Bootstrap, Chrome DevTools, HTML5 Boilerplate, Grunt, Angular, Blockchain Testnet, Grid Guide, Meteor, Git Extensions, Backbone, CodePen, and Foundation. It is straightforward to grasp because Angular is mostly used for web application development and is a well-developed language. (Angular, 2022)

5.2 BACKEND

5.2.1 ASP.NET CORE WEB API

Backend development for websites and mobile applications is being done in various languages. Some examples are JavaScript, Python, PHP, Java, Golang, C#, and Perl. Any of these languages can be used to create standard backends for developers. However, when linking it to the front-end, the author has two: one for the website and one for the mobile app. It is impossible to utilize the same code for both if the author traditionally writes the backend. The author chose web APIs as a solution, utilizing ASP.NET Core Web API as the backend technology. Asp.net Core web API 6.0 is the newest version. However, the author chose the 5.0 version because the resources accessible on online platforms are limited compared to other languages, and there are fewer courses in the most recent version. Because the author can utilize ASP.NET or ASP.NET core as the backend development technology, there is no need to use another platform like angular, Vue.js, React.js, or Flutter. (Microsoft, 2016)

5.3 DATABASE

5.3.1 MSSQL

Microsoft SQL Server is created by Microsoft, and its a relational database management system. It is a database server, which is a software product whose principal role is to retrieve and store data as required by other software applications, which may run on the same computer or on a networked computer. (Wikipedia, Wikimedia Foundation, 2022)

5.4 MIDDLEWARE JUSTIFICATION

We can define Middleware as software that sits in the middle of an operating system and the programs that execute on it. Middleware allows communication and data management for distributed applications by acting as a covert translation layer. (Microsoft Azure, n.d.)

Middleware aids in the faster development of apps. It is the glue that holds programs, data, and users together. Middleware can help enterprises with multi-cloud and containerized environments develop and run large-scale applications at a lower cost. (Microsoft Azure, n.d.)

So, in this vaccination management system API project uses,

1. UseDeveloperExceptionPage Middleware - used to report app runtime errors.
2. UseHttpsRedirection Middleware - used to redirects HTTP requests to HTTPS.
3. UseRoutnig - used to route requests.
4. UseCors - used to enable cross-origin resource sharing.

5. UseSwagger & UseSwaggerUI - used to generate Swagger API documentation. (Configured to handle
6. UseAuthentication - used to authenticate users. (Configured to use JSON Web Tokens)
7. UseAuthorization - used to authorize users to access secure resources. (Configured to use JSON Web Tokens)
8. UseEndpoints - used to map controllers to the request pipeline.

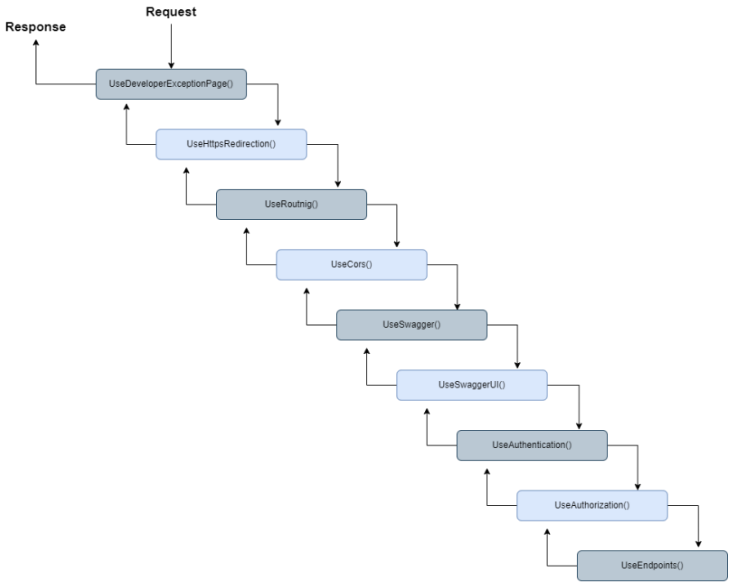


Figure 5.1: Middleware Flow

5.5 3RD PARTY LIBRARIES JUSTIFICATION

5.5.1 API

1. Mailkit

MailKit is an Open-Source, cross-platform .NET mail-client library based on MimeKit and optimized for mobile devices. Using this library system can send required emails to the user.

2. Serilog

Logging frameworks make sending logs to different places easy via simple configurations. Serilog uses sinks to send records to a text file, database, log management solutions, or potentially dozens of other sites, all without changing code.

3. Swashbuckle.AspNetCore

This library can automatically generate interactive testable API documentation for ASP .NET core end-points. This library will help developers to work with the front end.

5.5.2 WEB APPLICATION

1. Data tables

JQuery DataTables is the plugin that powers the Angular DataTables framework. Angular 2+ is supported and set up to support TypeScript.
When using Angular DataTables,

- Developers have a vast dataset coming in from one or more API end-points
- It would help if developers had customized data sorting/filtering

There are two primary categories of Angular DataTable features: basic and sophisticated. From there, Angular DataTables has a variety of extensions that may be added.

2. Moment

JavaScript developers can use Moment.js to parse, validate, manipulate, and display dates and times.

3. NGX-Avatar

If the system has information about the user, the system may use this universal avatar component to get or generate an avatar.

4. NGX-Spinner

For Angular 4+ versions, an animated loading spinner informs the user that an operation occurred.

5. NGX-Device-Detector

AOT-compatible device detector powered by Angular 6+ that helps identify the browser, operating system, and other important information about the device accessing the app. User-agent is used to automate the process of analyzing the data.

6. NGX Chart

Charting framework ngx-charts is open source and declarative for angular 2+. Swimlane is responsible for its maintenance. SVG elements are rendered and animated with the help of Angular's binding and speed, and d3's math functions, scales, axes, and form generators are used to make the animations even more impressive.

5.5.3 MOBILE APPLICATION

1. HTTP

HTTP package makes it easy to send and process HTTP requests made by the mobile application to the server.

2. Shared Preferences

Shared preferences help to store data temporarily in the app, which can later be used throughout the application.

3. QR flutter

QR flutter package makes it simple to paint a QR code with the provided information.

4. Cool Alert

Allows making custom mobile alerts to notify the user.

6 IMPLEMENTATION

6.1 WEB APPLICATION IMPLEMENTATION

The web application is implemented for officers, admins and jobseekers.

6.1.1 FEATURES OF THE WEB APPLICATION

Admins, jobseekers and officers have a common login interface. After the registration, users can log in to the system.

- Common login page
  Admins, jobseekers and officers have a common login interface. After registering to the system, they can log in to the system.
- The registration page is different for each role.
  The process of registration differs for each role according to the requirement. So, the interfaces also differ from each other.
- Admin dashboard
  Admin has permission to view, edit, delete and add admins and officers. Also, the admin can edit, delete and view jobseekers.
- Officer dashboard
  Officers can view and filter the job seekers by qualifications.
- Jobseeker dashboard
  Jobseeker can update their CV and details and send complaints to the admin.

- Mapbox

  A Mapbox is added to get the user's location in the first phase of the registration process for job seekers. The user can edit the location by dragging the indicator to where the user needs it. Zoom out, Zoom in, reset bearing to North and find my location options also included on the map.

  Mapbox is a developer-friendly mapping and location cloud platform. We're the foundation, the SDKs and APIs that enable developers and designers to incorporate real-time location awareness into their projects. (Mapbox Developers, n.d.)

  Foursquare, Lonely Planet, the Financial Times, The Weather Channel, Instacart Inc., and Snapchat are among the companies that use Mapbox to create personalized online maps. (Mapbox, n.d.)

- Complaint sending

  Usually, the complaints are sent as a notification to the admin. But here, that feature is implemented as a chat option. The Jobseeker can send the complaint to the admin, and the admin can reply to that until it is resolved. In the end, the admin can toggle the resolved button.

- CV Parsing

  The job seeker can upload the CV at the first phase of the registration process. Then the process of CV parsing takes some time and then moves to the next step of the registration.

6.1.2 WEB APPLICATION SAMPLE SCREENSHOTS

6.1.2.1 LOGIN

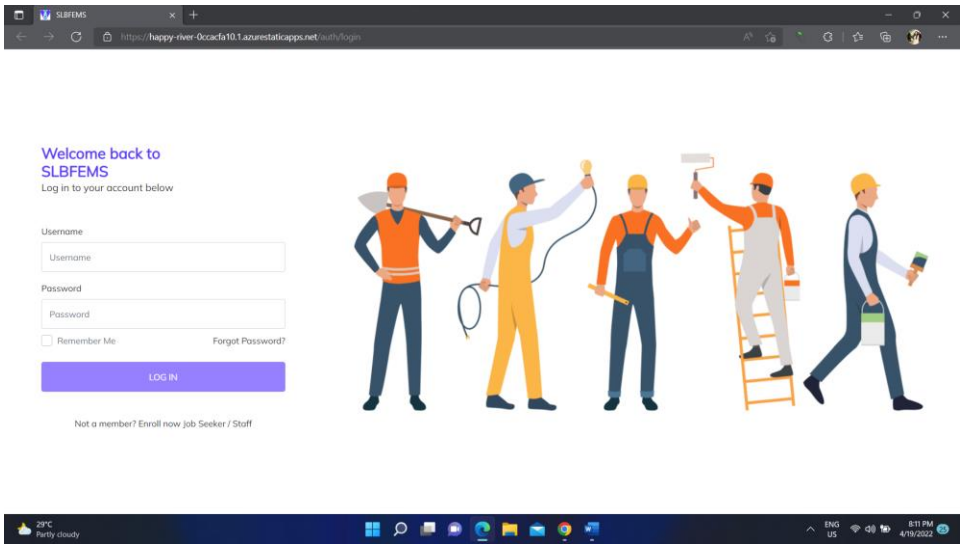


Figure 6.1: Login

This is the first interface users interact with when the user enters the system. The login form is provided here.

If the user cannot remind the user password, the user can click forgot password.

Users can log in if a user has an account already.

Otherwise, the user can click job seeker or staff according to the user's role.

6.1.2.2 FORGOT PASSWORD

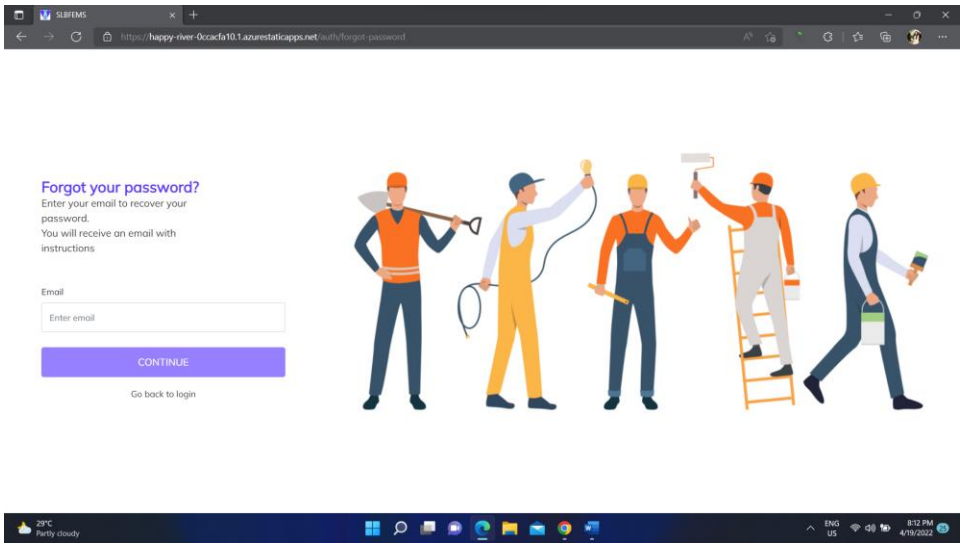


Figure 6.2: Forgot password

Users can enter their user email and go through the button in the received email sent to reset the password.

6.1.2.3 ALERT FOR USERS WHO DON'T HAVE PERMISSION

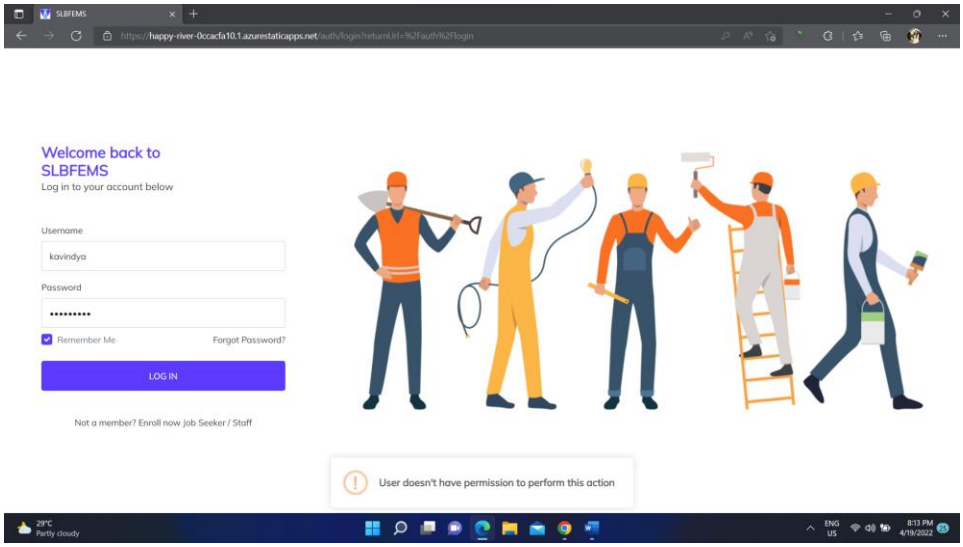


Figure 6.3: Alert

Users will get this alert if they try to log in to the system without confirming the user's email address.

6.1.2.4 STAFF REGISTRATION - STEP 1

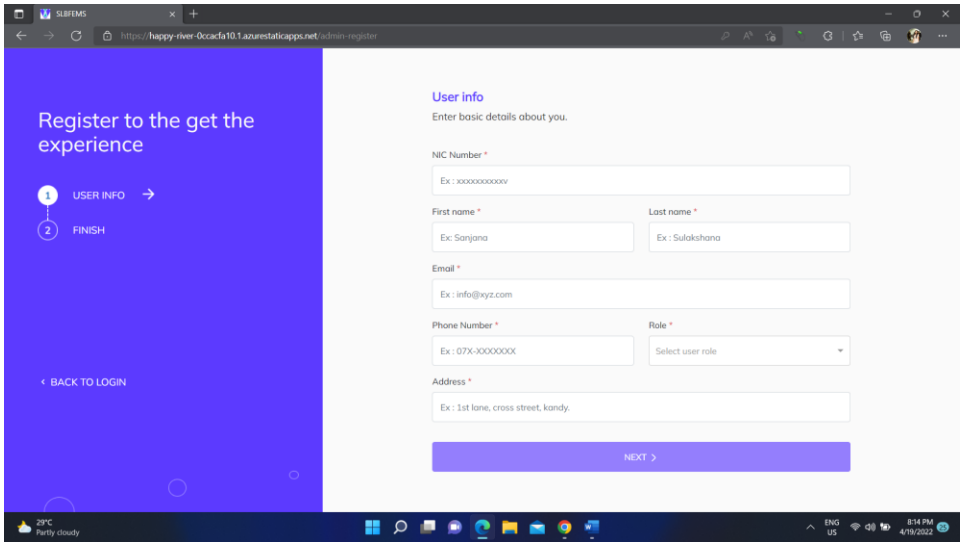


Figure 6.4: staff registration-step 1

In the first registration phase, fill all the fields and click next.

6.1.2.5 STAFF REGISTRATION- STEP 2

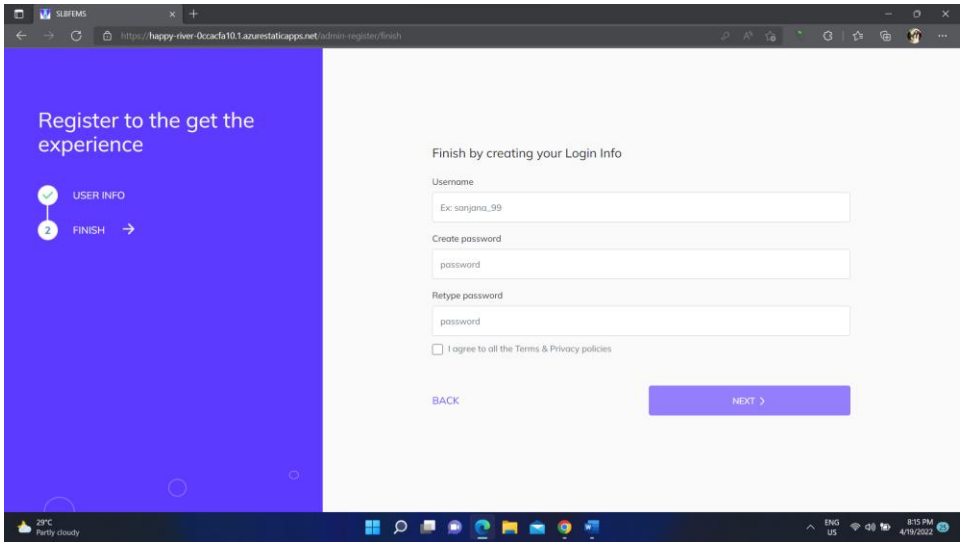


Figure 6.5: staff registration-step 2

In the second phase, users must define user credentials and tick to agree with the terms and privacy policies.

6.1.2.6 JOBSEEKER REGISTRATION – STEP 1

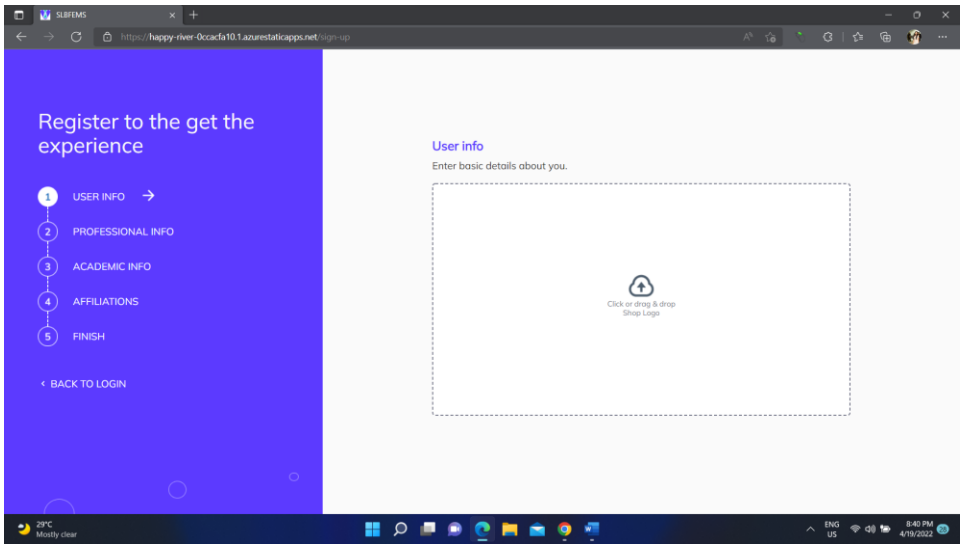


Figure 6.6: job seeker registration-step 1

This is the first phase of job seeker registration. Users have to submit user CVs here.

6.1.2.7 JOBSEEKER REGISTRATION – STEP 1

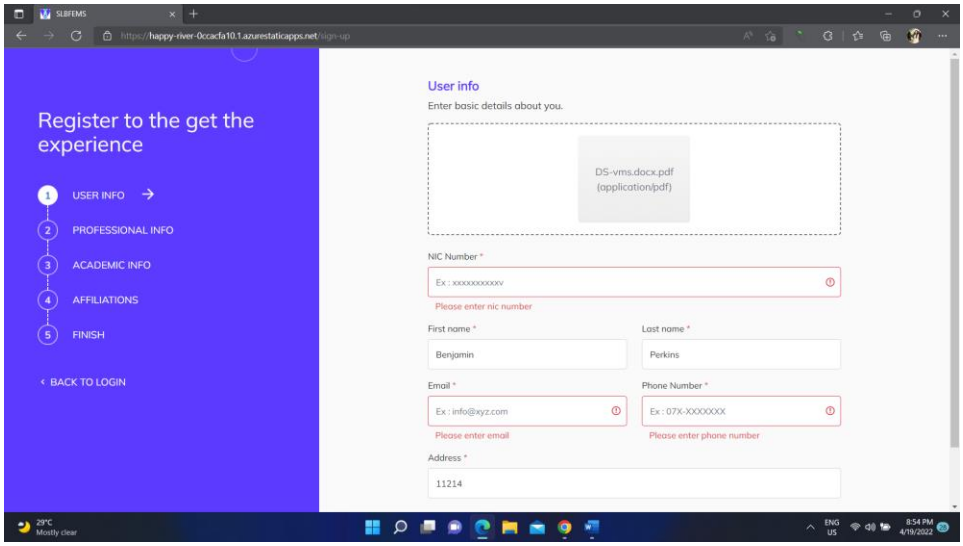


Figure 6.7: job seeker registration-step 1

Some of the fields are automatically filled according to the uploaded CV, and the user has to fill the rest of the areas.

6.1.2.8 JOBSEEKER REGISTRATION – STEP 2

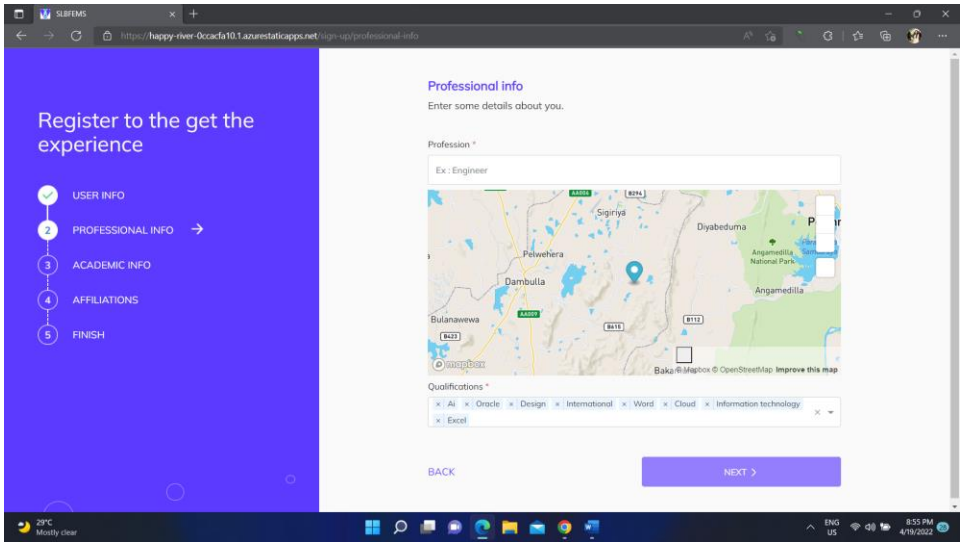


Figure 6.8: job seeker registration- step 2

In the second phase, the user has to input the user's professional information and click next.

6.1.2.9 JOBSEEKER REGISTRATION – STEP 3

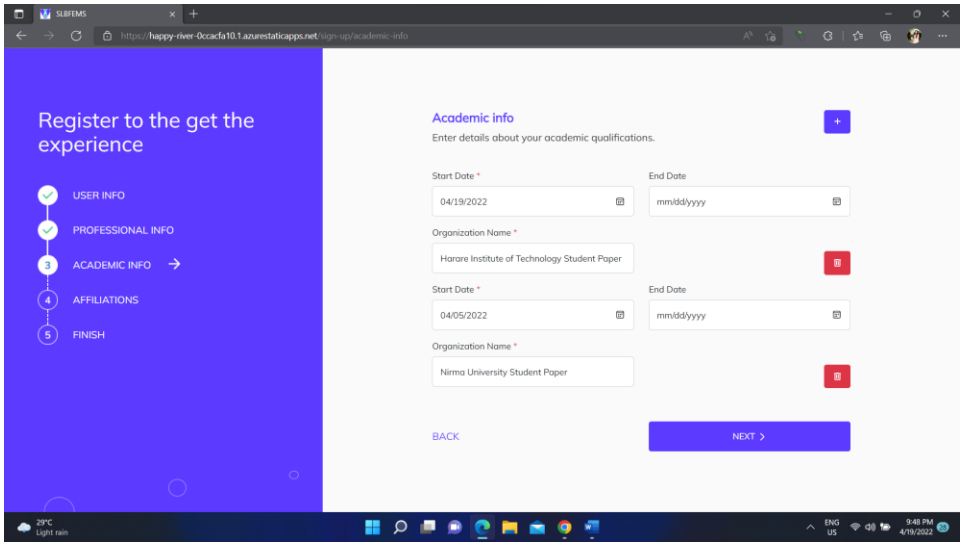


Figure 6.9: job seeker registration - step 3

Users have to enter user academic information in the third interface. Some are filled according to the uploaded CV, and the rest of the things have to add by the user itself.

6.1.2.10 JOBSEEKER REGISTRATION – STEP 4

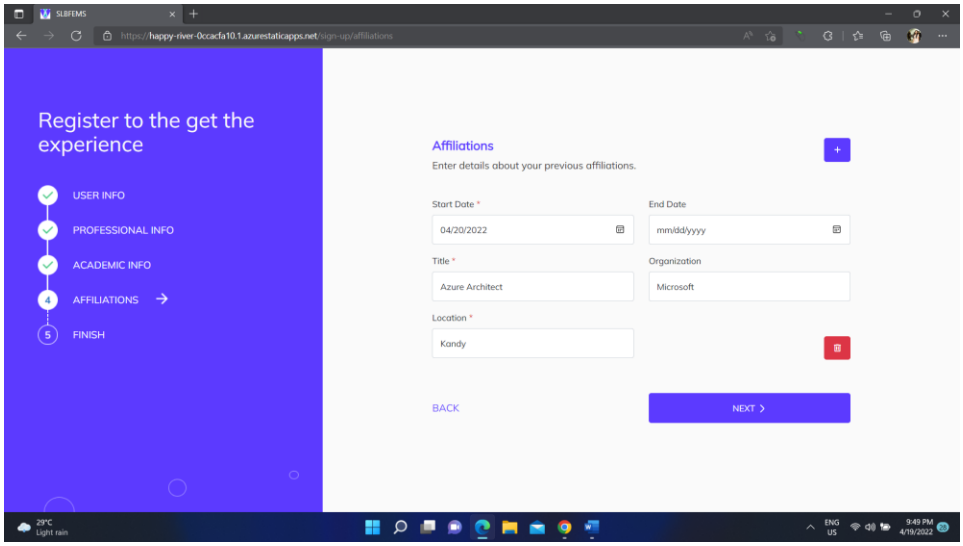


Figure 6.10: job seeker registration -step 4

In the fourth phase, the user has to add affiliations.

6.1.2.11 JOBSEEKER REGISTRATION – STEP 5

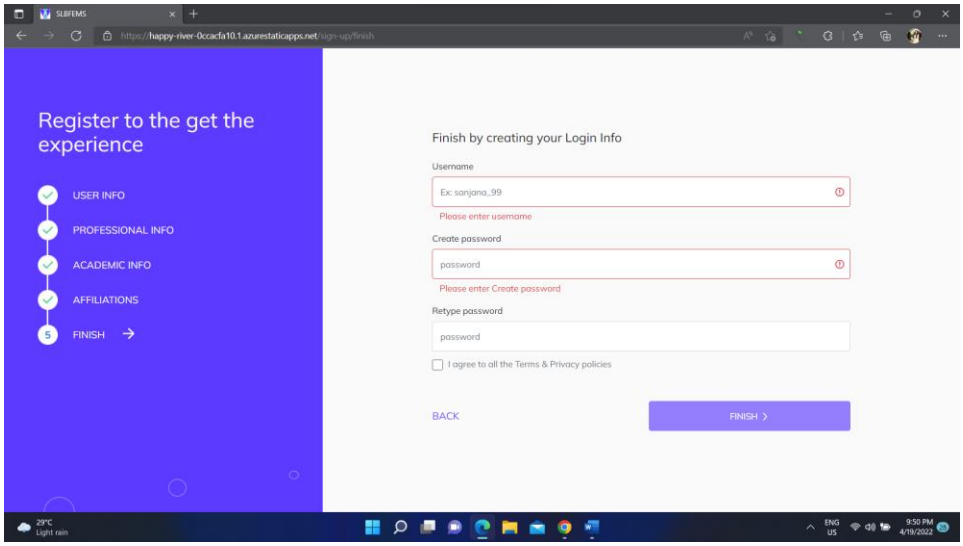


Figure 6.11: job seeker registration- step 5

In the final phase of the registration process, users have to define user credentials and tick to agree with the terms and privacy policies.

ADMIN

6.1.2.12 VIEW JOB SEEKERS

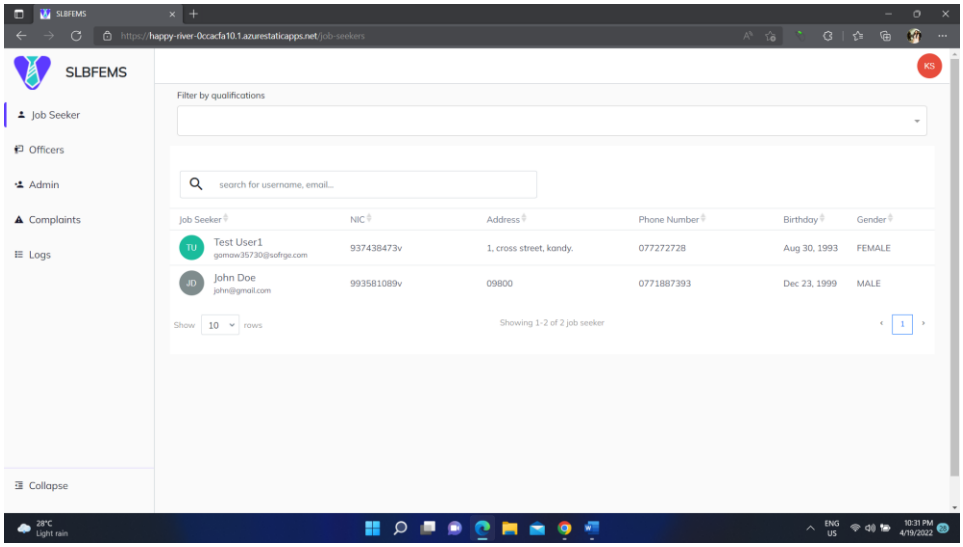


Figure 6.12: view job seekers

This is the first interface user get when the user logs in as a job seeker.

Details of the job seeker are displayed here, and users can filter the job seekers by qualifications.

The side menu bar includes the navigations to job seekers, officers, admin, complaints and logs.

Also, the user can collapse the menu bar by the collapse option.

6.1.2.13 VIEW OFFICERS

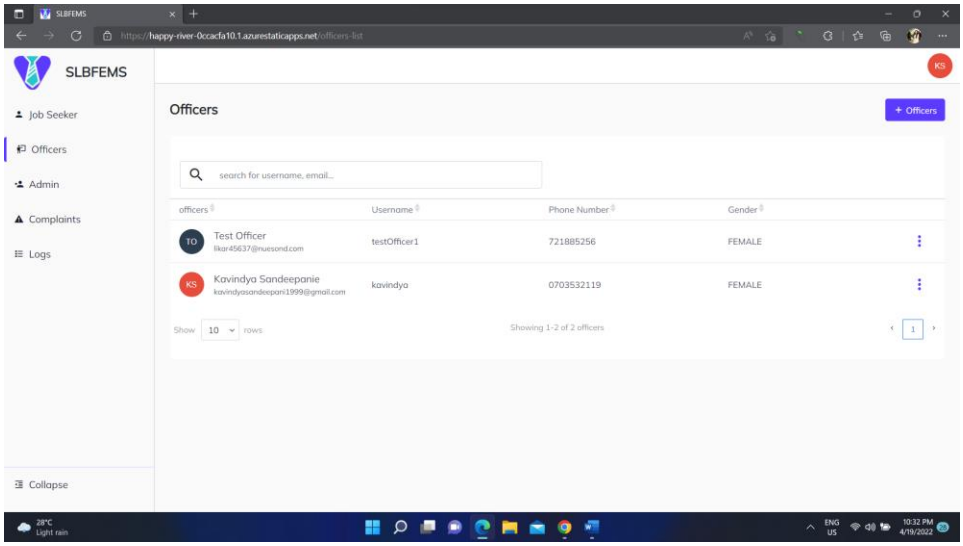


Figure 6.13: view officers

Officers are displayed here. Users can add new officers to the system.

6.1.2.14 ADD OFFICERS

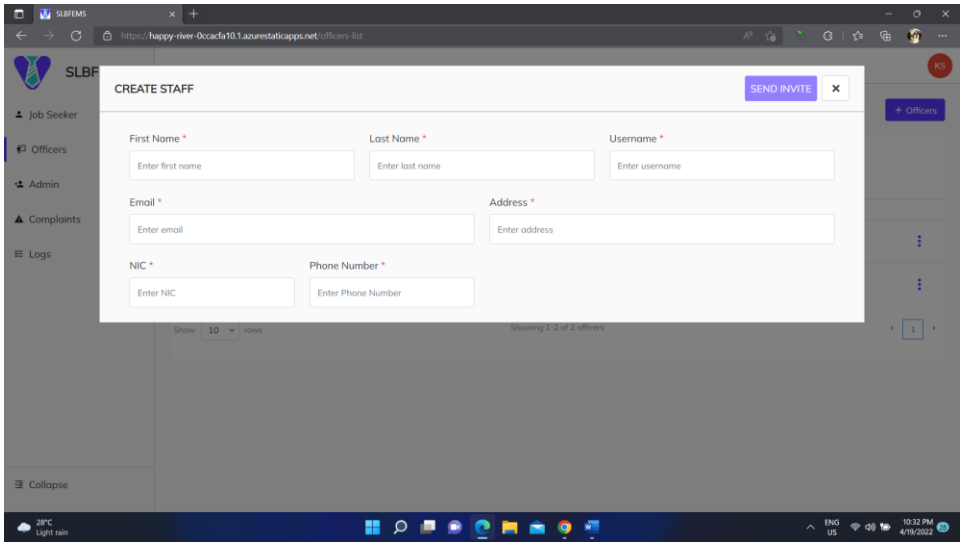


Figure 6.14: add officers

Users have to fill out this form and click send invite button to invite a new officer.

6.1.2.15 VIEW ADMINS

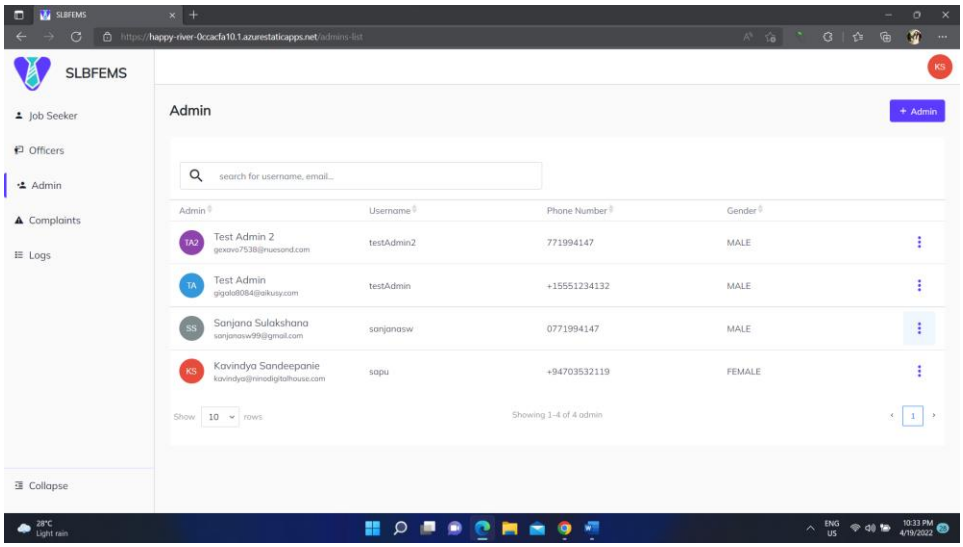


Figure 6.15: view admins

Admins are displayed in this interface. Users can add new admins by clicking on the admin button.

6.1.2.16 ADD NEW ADMIN

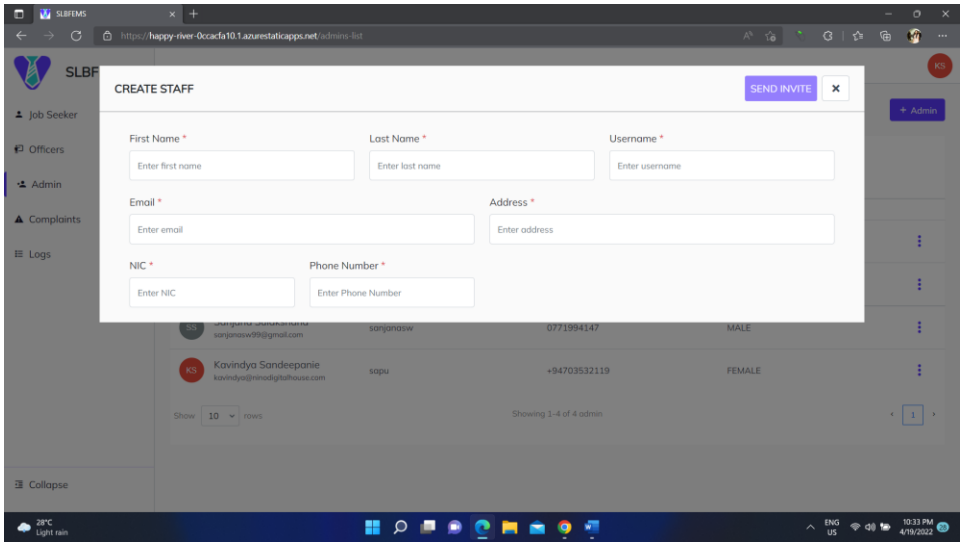


Figure 6.16: add new admin

Users can add new staff members by filling in this form.

6.1.2.17 KEBAB MENU FOR THE ADMIN

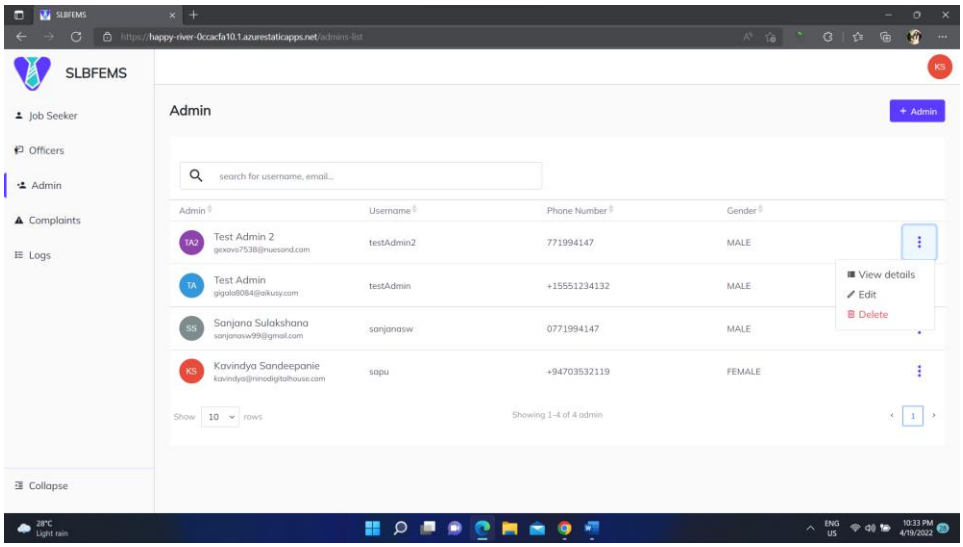


Figure 6.17: kebab menu

Users can view details, edit, and delete admins.

6.1.2.18 VIEW ADMIN DETAILS

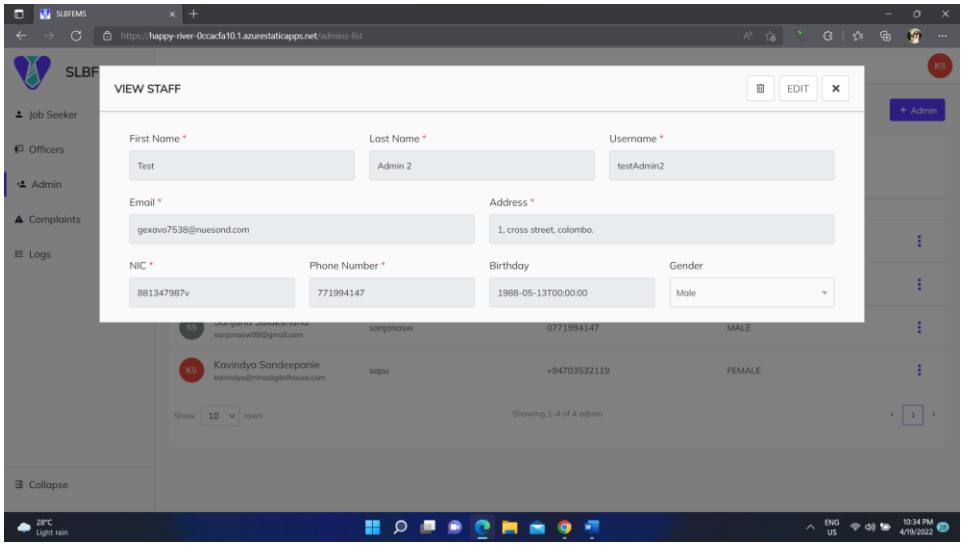


Figure 6.18: view admin detail

Users can view the details of the officer.

6.1.2.19 EDIT ADMIN DETAILS

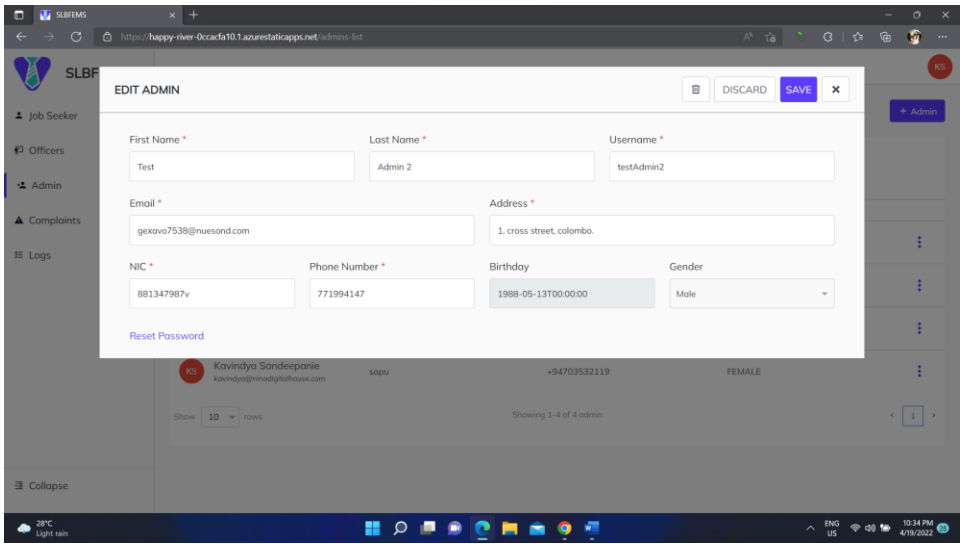


Figure 6.19: edit admin details

Users can edit the details of the admin.

6.1.2.20 RESET PASSWORD LINK SENT

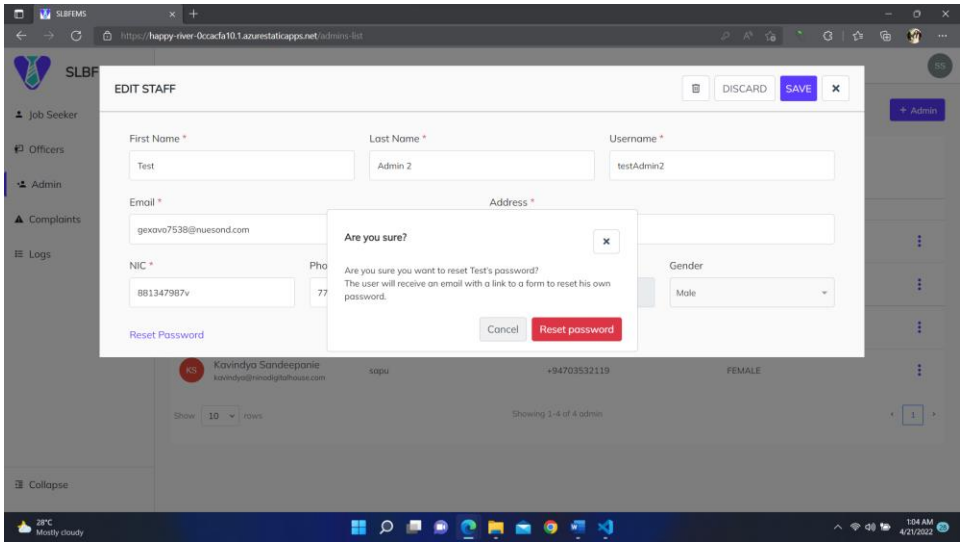


Figure 6.20: edit staff

Users can send the reset password link to the admin by clicking on the reset password button.

6.1.2.21 DELETE ADMIN

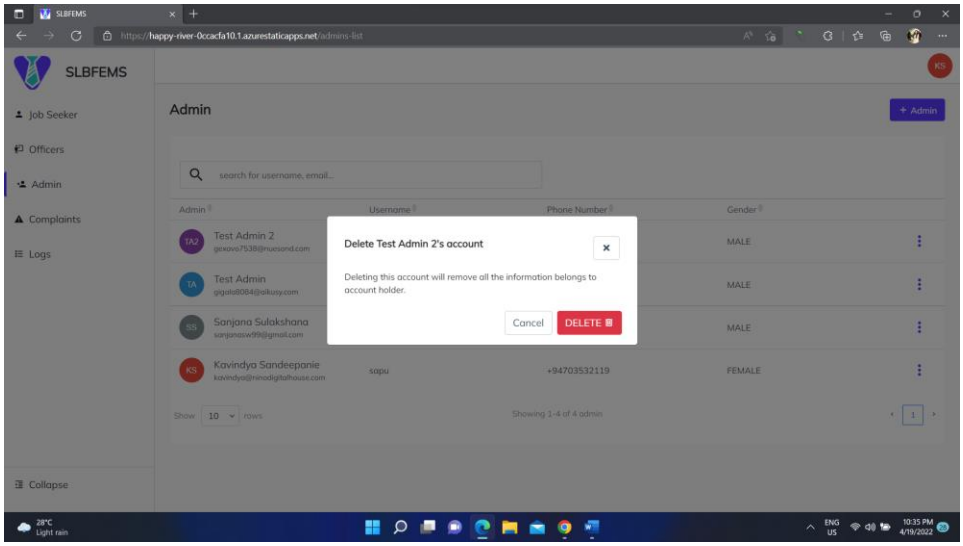


Figure 6.21: delete admin

Users have to confirm before deleting the admin.

6.1.2.22 VIEW COMPLAINTS

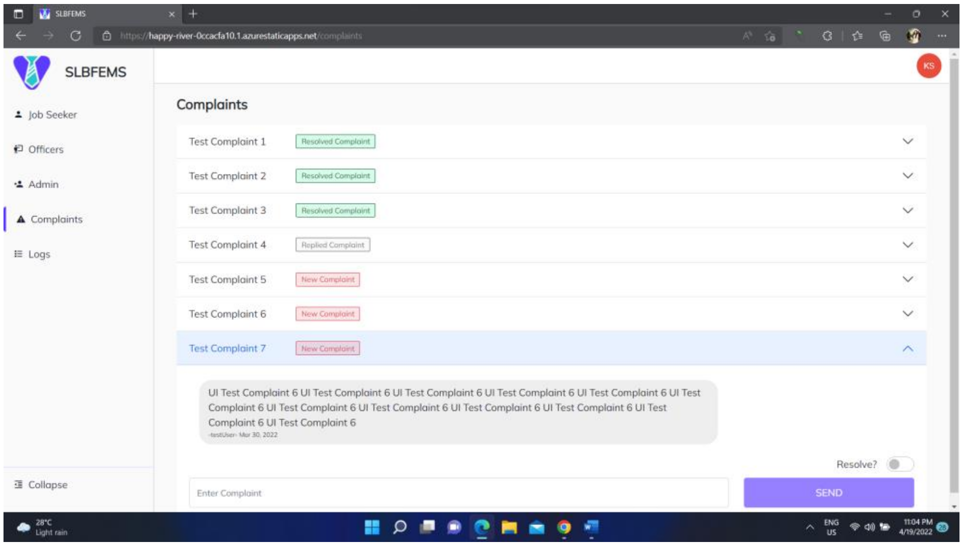


Figure 6.22: view complaints

In this interface, the user can see the complaints. Click on the complaint to view the complaint.

6.1.2.23 RESOLVED COMPLAINT

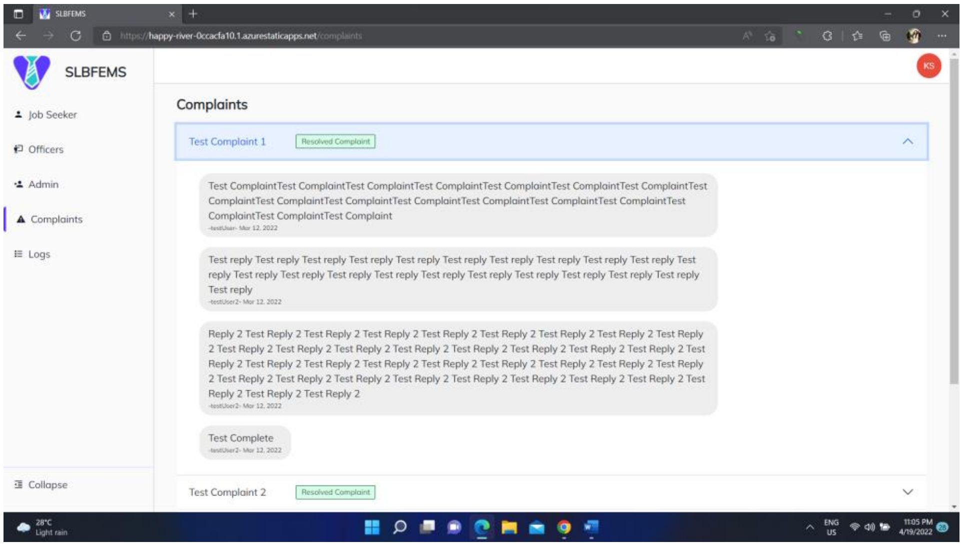


Figure 6.23: resolved complaints

Here, the user can't reply because the user marked the complaint as resolved. But the user can see the conversation.

6.1.2.24 NEW COMPLAINT

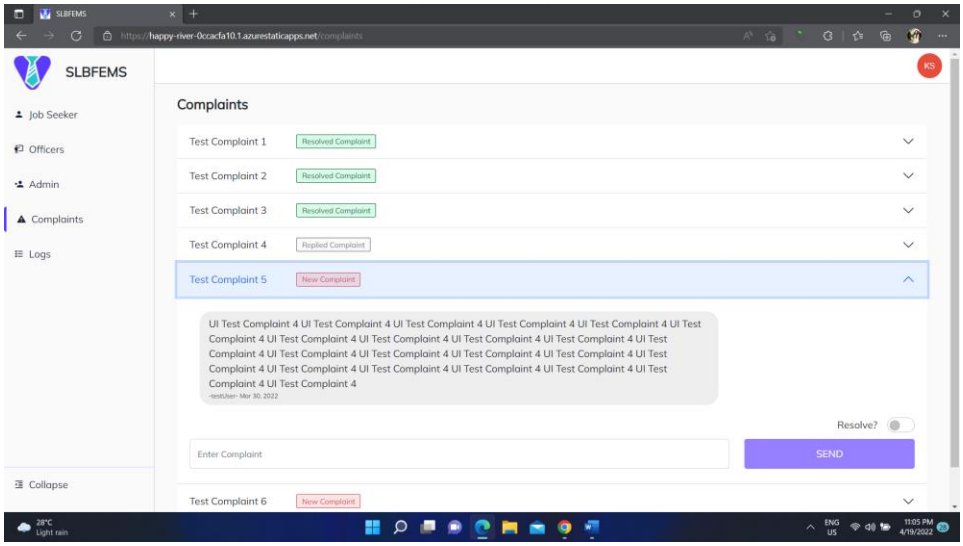


Figure 6.24: new complaint

Here, the admin didn't reply to the complaint. So still, it is labelled as a new complaint. Users can respond to the complaint or mark it as resolved.

6.1.2.25 INFORMATION LOGS

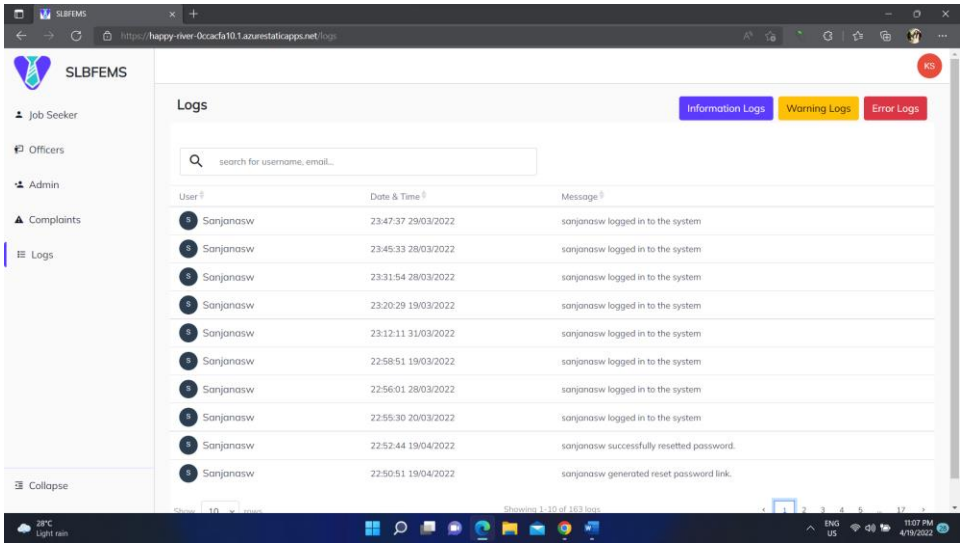


Figure 6.25: information logs

Incoming logs are displayed here.

6.1.2.26 WARNING LOGS

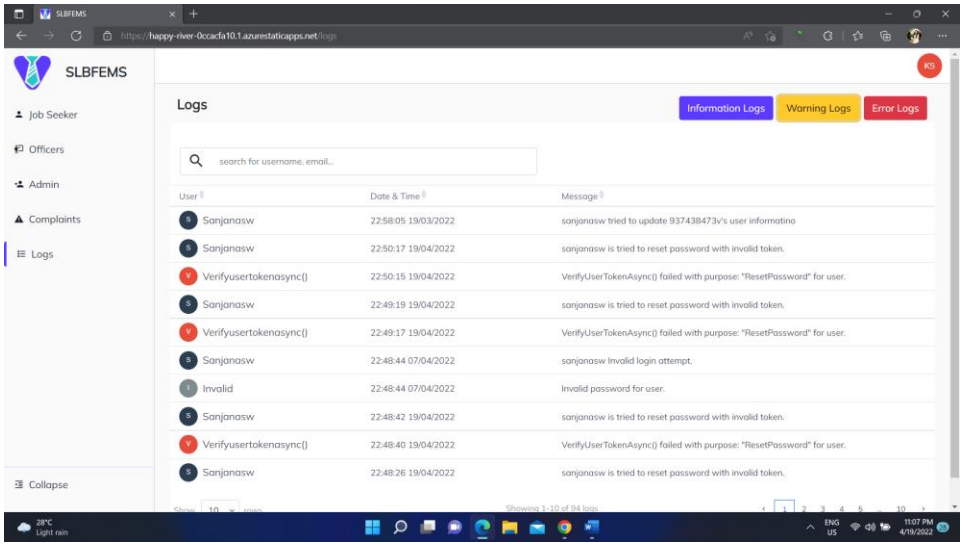


Figure 6.26: warning logs

Warning logs are displayed here.

6.1.2.27 ERROR LOGS

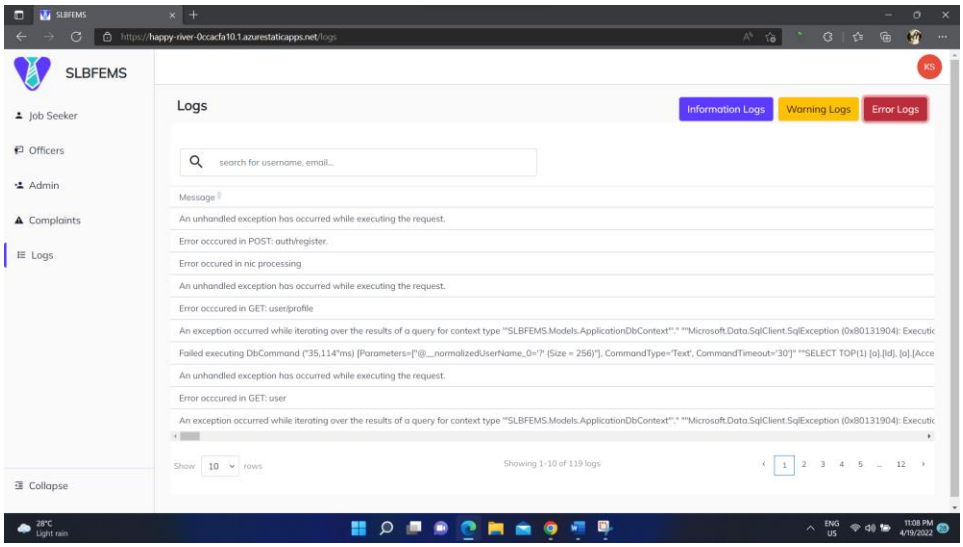


Figure 6.27: error logs

Error logs are displayed here.

JOBSEEKER

6.1.2.28 PERSONAL AND JOBSEEKER INFORMATION

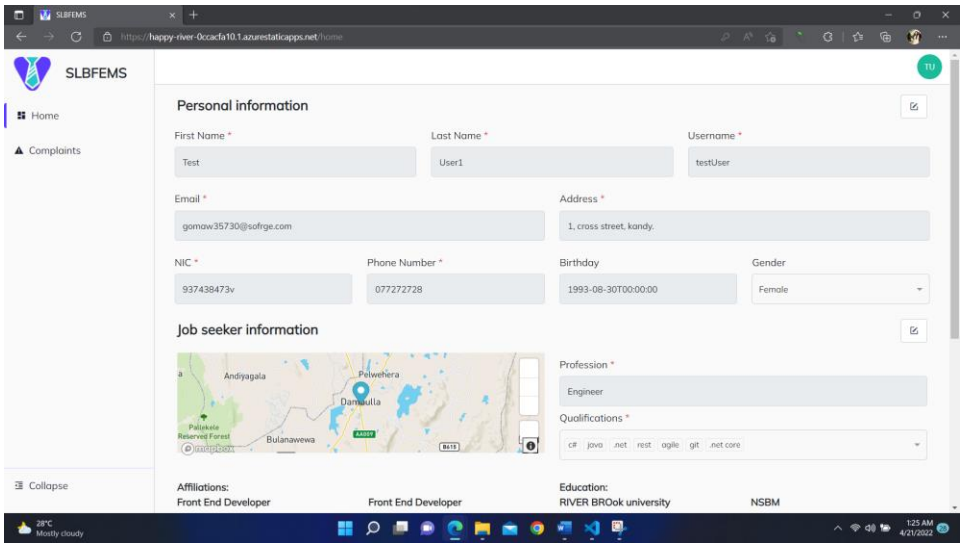


Figure 6.28: personal and job seeker info

This is the first-page user interacts with when the user is logged as a jobseeker. Personal and job seeker information is displayed here.

6.1.2.29 EDIT PERSONAL INFORMATION

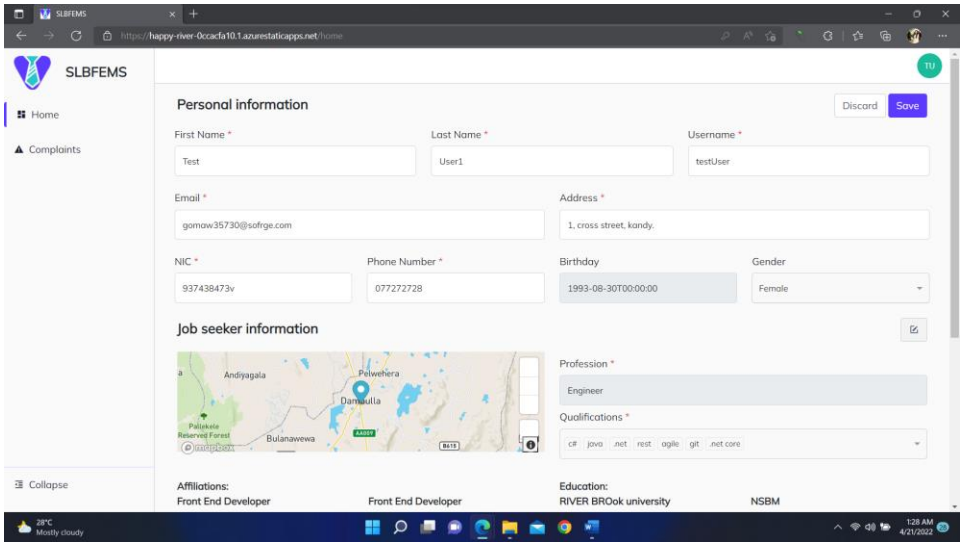


Figure 6.29: edit personal info

Users can edit users personal information here.

6.1.2.30 EDIT JOB SEEKER INFORMATION

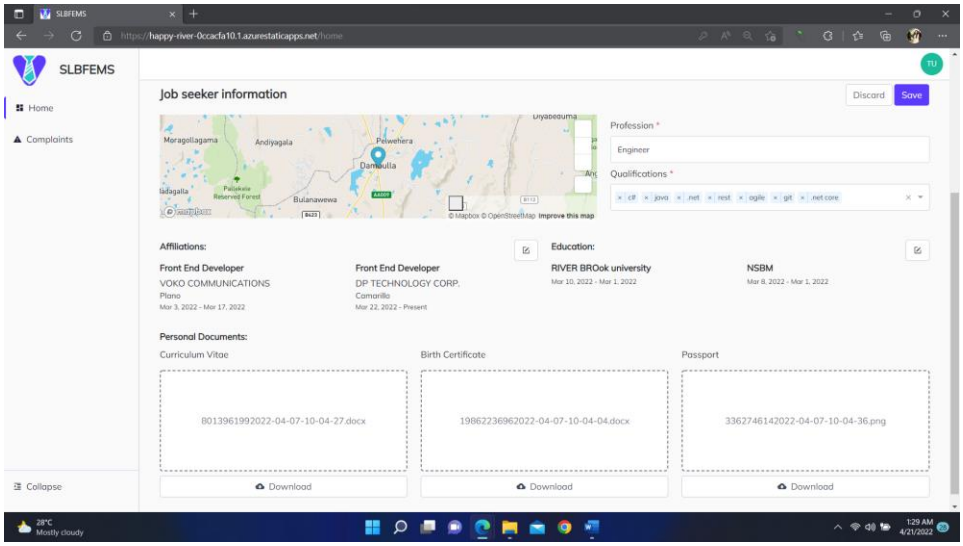


Figure 6.30: edit job seeker info

Users can edit job seeker information here.

6.1.2.31 COMPLAINTS

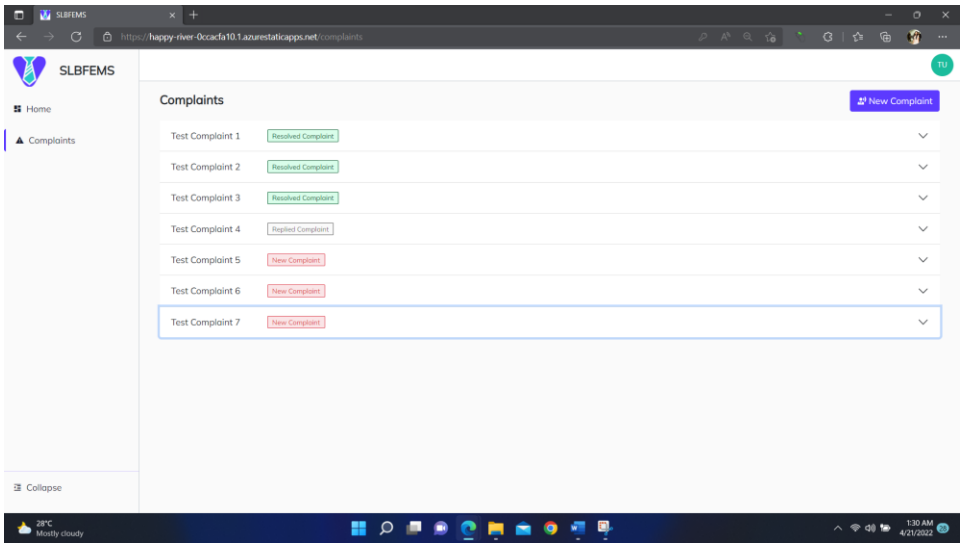


Figure 6.31: complaints

All the complaints users send are listed here. Users can view the complaints by clicking on the complaint.

6.1.2.32 ADD COMPLAINT

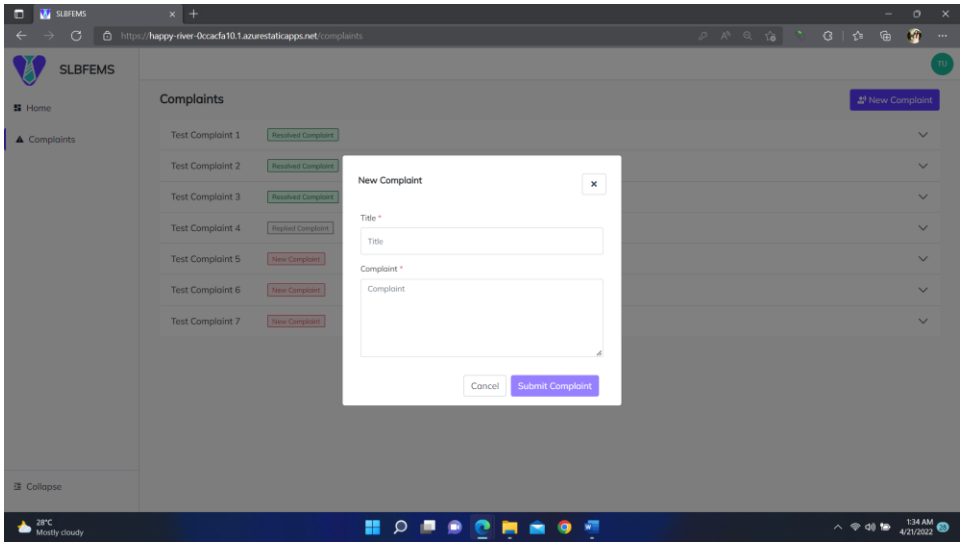


Figure 6.32: add a complaint

Users can add new complaints.

OFFICER

6.1.2.33 VIEW JOBSEEKERS

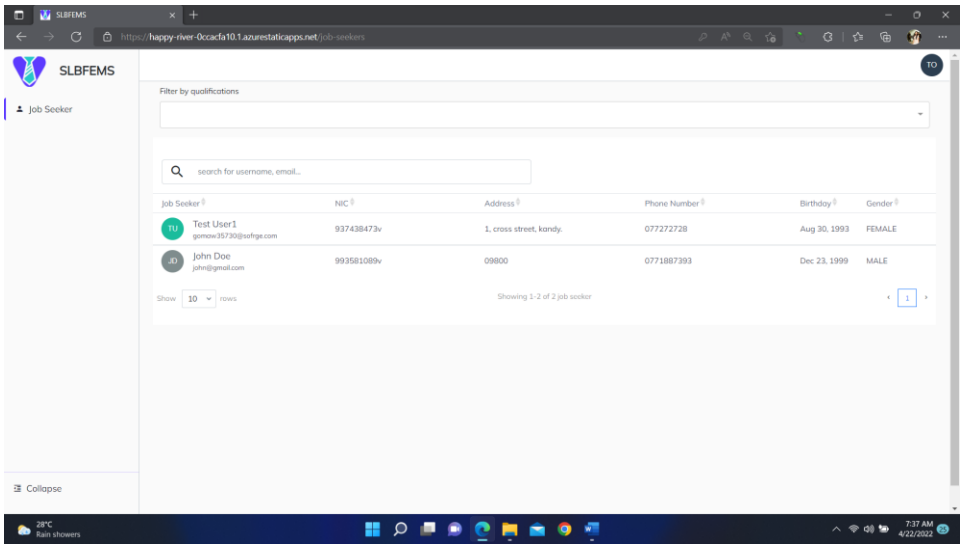


Figure 6.33: view jobseekers

A list of job seekers is listed here with their details.

Users can filter by the qualification also.

6.1.2.34 CHANGE PASSWORD

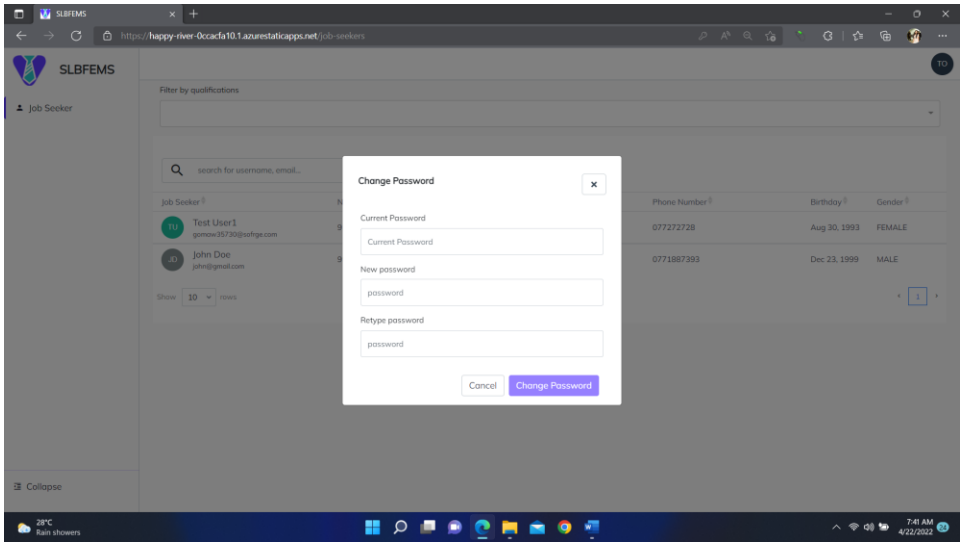


Figure 6.34: change password

Users can change their password here.

6.2 MOBILE APPLICATION IMPLEMENTATION

6.2.1 FEATURES OF THE MOBILE APPLICATION

- Common login page

  Admins, jobseekers and officers have a common login interface. After registering to the system, they can log in to the system.

- The registration page is different for each role.

  The process of registration differs for each role according to the requirement. So, the interfaces also differ from each other.

- Admin dashboard

  Admin has permission to view, edit, delete and add admins and officers. Also, the admin can edit, delete and view jobseekers.

- Officer dashboard

  Officers can view and filter the job seekers by qualifications.

- Jobseeker dashboard

  Jobseeker can update their CV and details and send complaints to the admin.

- Mapbox

  A Mapbox is added to get the user's location in the first phase of the registration process for job seekers. The user can edit the location by

dragging the indicator to where the user needs it. Zoom in, reset bearing to North, zoom out, and find my location options also included on the map.

Mapbox is a developer-friendly mapping and location cloud platform. We're the foundation, the SDKs and APIs that enable developers and designers to incorporate real-time location awareness into their projects. (Mapbox, n.d.)

Foursquare, Lonely Planet, the Financial Times, The Weather Channel, Instacart Inc., and Snapchat are among the companies that use Mapbox to create personalized online maps. (Mapbox Developers, n.d.)

- Complaint sending

  Usually, the complaints are sent as a notification to the admin. But here, that feature is implemented as a chat option. The Jobseeker can send the complaint to the admin, and the admin can reply to that until it is resolved. In the end, the admin can toggle the resolved button.

6.2.2 MOBILE APPLICATION SAMPLE SCREENSHOTS

6.2.2.1 LOGIN

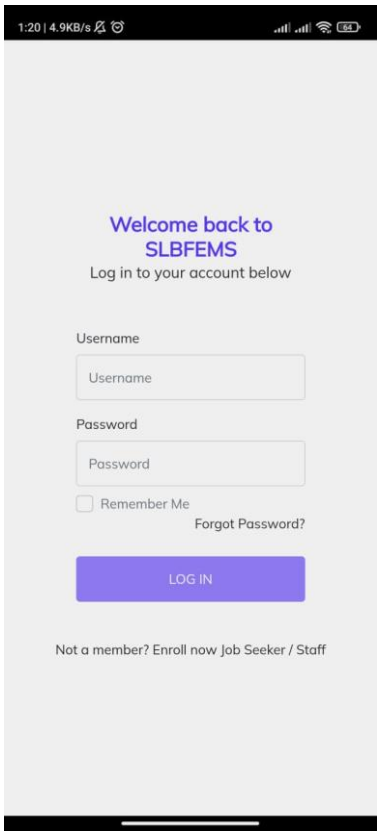


Figure 6.35: login

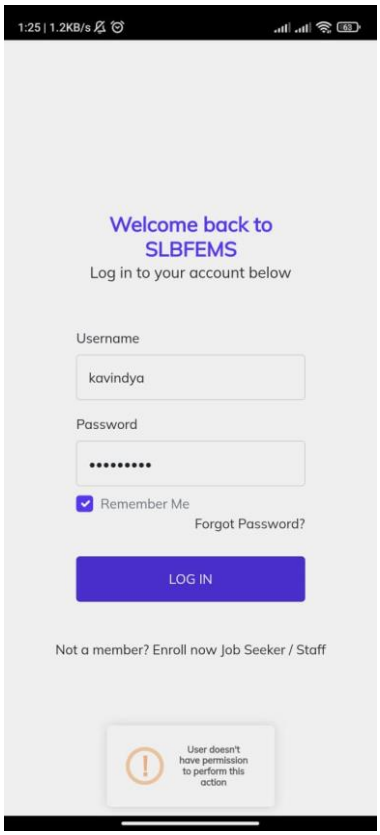


Figure 6.36: alert for errors

This is the login interface. Users have to enter the user's password and username to log in to the system.
If the user entered the wrong credentials, the user would get an error.
If the user didn't confirm the email, the user would get this error.

6.2.2.2 STAFF REGISTRATION STEP 1

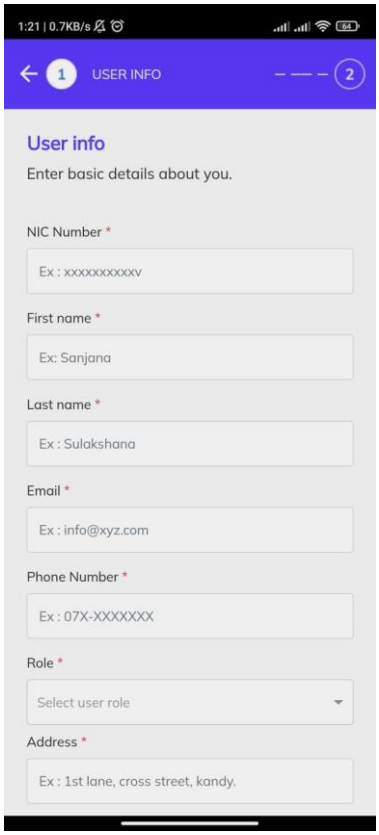


Figure 6.37: staff registration - step 1

As all the fields are required, the user must fill all the fields in the first step.

6.2.2.3 STAFF REGISTRATION STEP 2

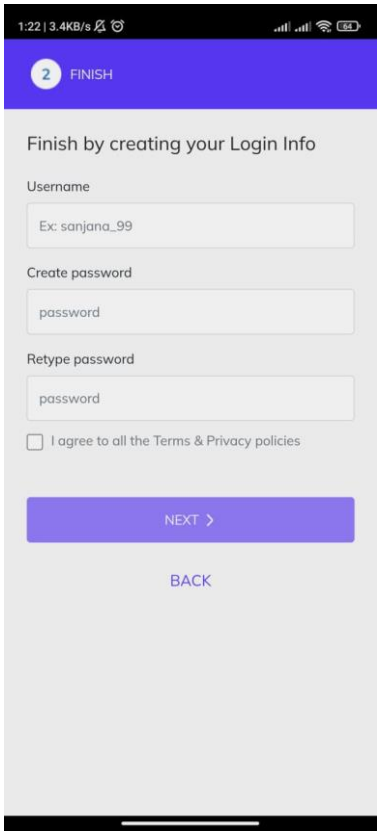


Figure 6.38: staff registration- step 2

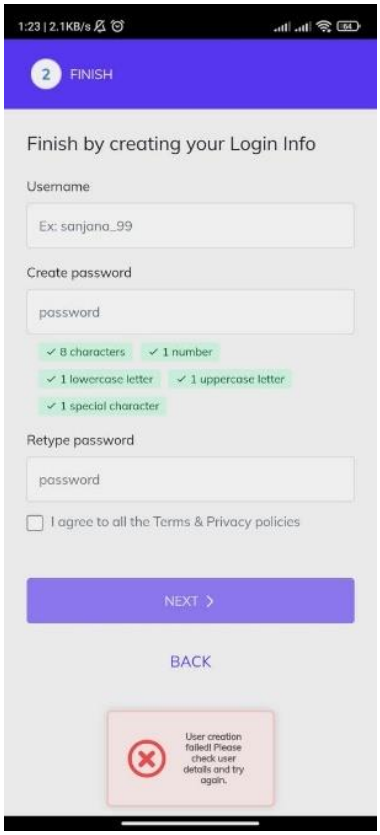


Figure 6.39: error displaying the alert

In the second step, users must define the user credentials and tick the agreement for terms and privacy policies.

If the user enters an invalid value, the user won't create the account. The system will alert users, as in the above image.

6.2.2.4 REGISTRATION SUCCESS ALERT

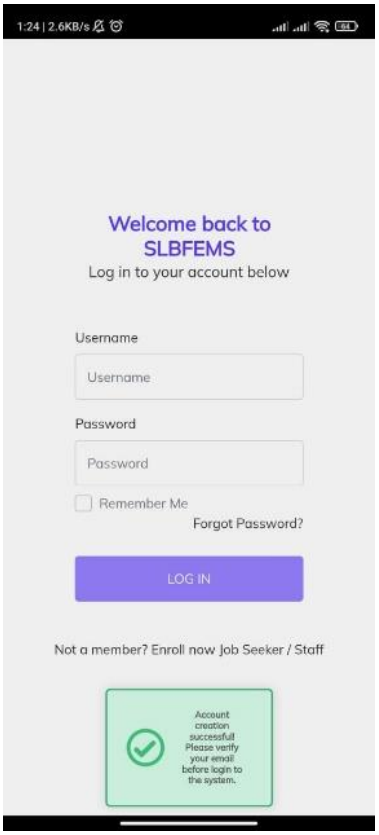


Figure 6.40: account creation successful message and redirect to login

After creating the account, an alert will display and automatically redirect to the login interface.

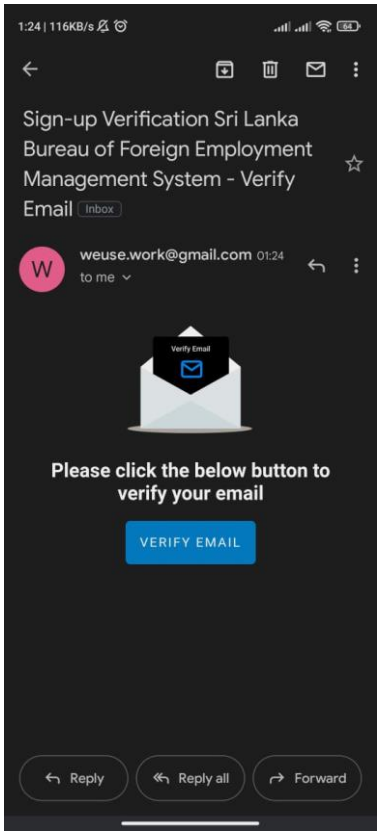


Figure 6.41: email to verify the email address in the signup process

After creating the account, users have to verify users email address by clicking this verify email button.

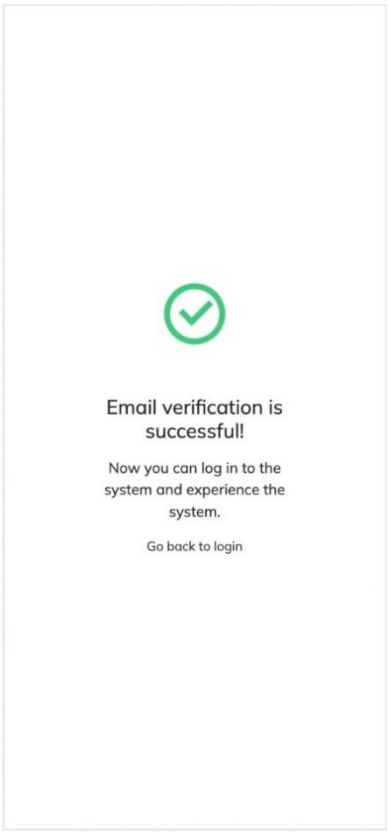


Figure 6.42:email verification

After verifying the email, the user will get this alert. The user can log in to the system by going back to log in.

6.2.2.5 JOBSEEKER REGISTRATION STEP 1

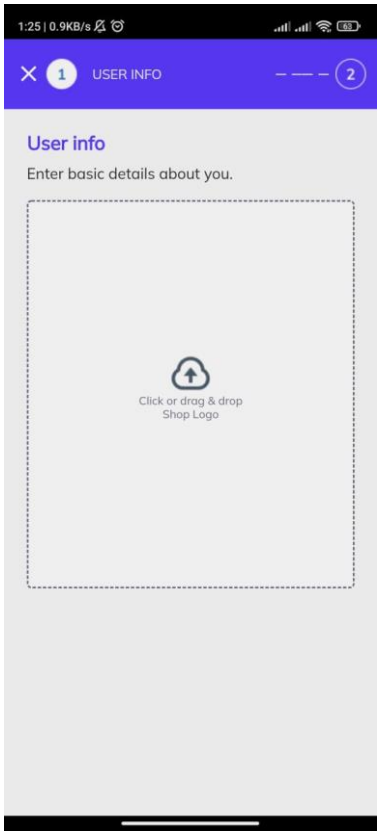


Figure 6.43: Job Seeker Registration - Step 1

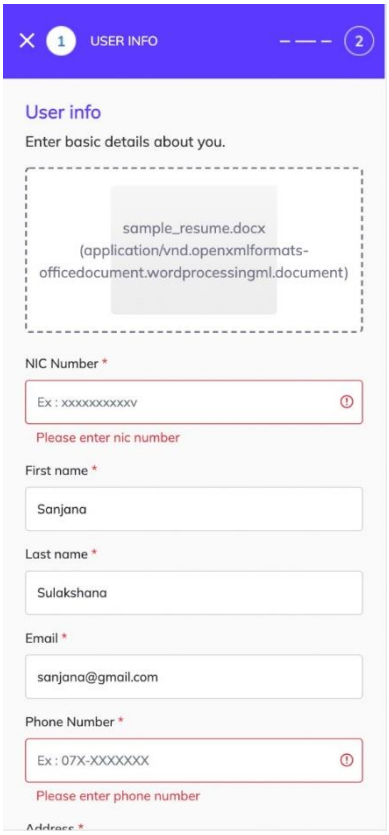


Figure 6.44: Job Seeker Registration- Step 1

As the first step of the job seeker registration process, users have to submit a user CV.

Then some of the fields are filled automatically according to the CV, and the rest of the areas have to reload by the user itself.

6.2.2.6 JOBSEEKER REGISTRATION STEP 2

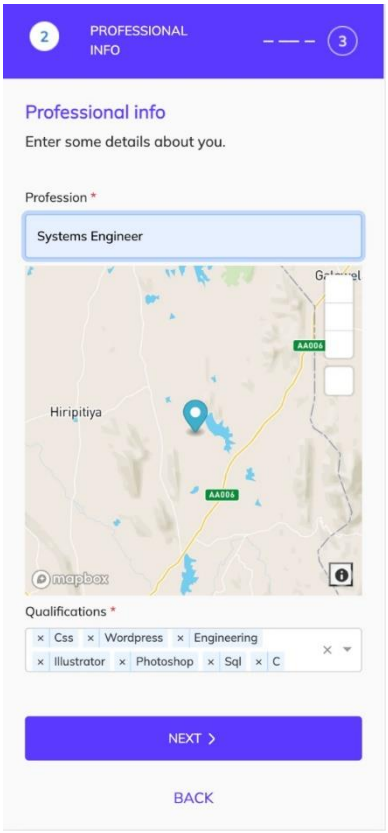


Figure 6.45: Job Seeker Registration-Step2

Enter user personal information here and click next.

6.2.2.7 JOBSEEKER REGISTRATION STEP 3

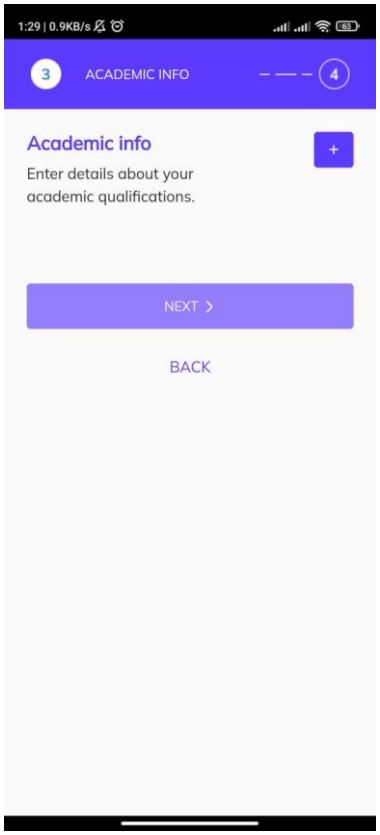


Figure 6.46: Job Seeker Registration-Step 3

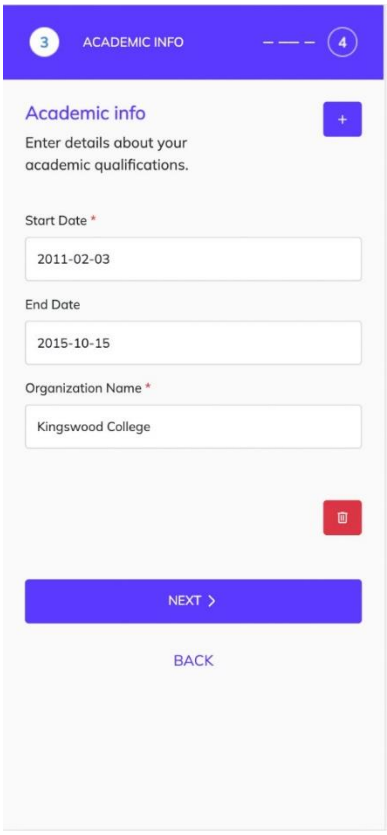


Figure 6.47: Job Seeker Registration-Step3

Click plus button to add academic info.

Users can add users academic information in the third step.

Users can go back to the second step by the back button.

6.2.2.8 JOBSEEKER REGISTRATION STEP 4 AND STEP 5

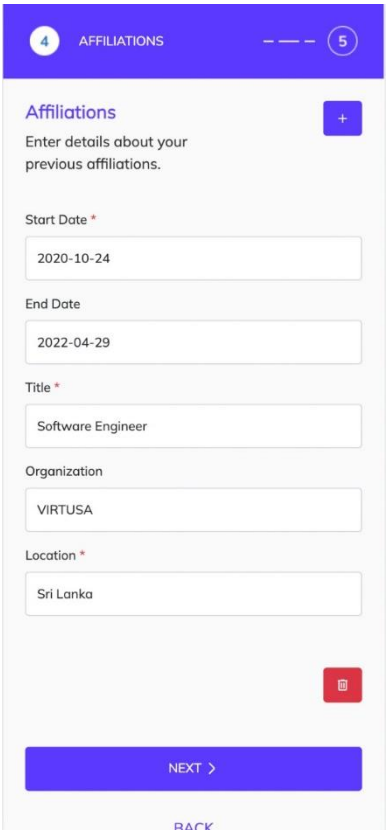


Figure 6.48: Job Seeker Registration-Step 4

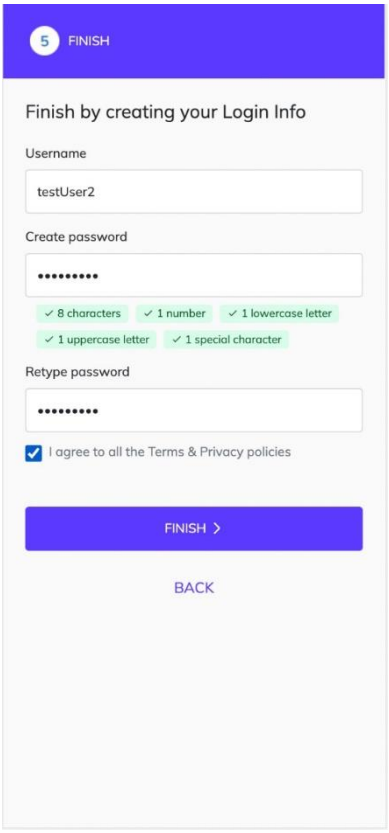


Figure 6.49: Job Seeker Registration -Step 5

Users can add affiliation if they have affiliations other than those included in the CV.

Also can delete them.

The user defines user login credentials at the last step and finishes the registration process.

6.2.2.9 JOBSEEKER DETAILS

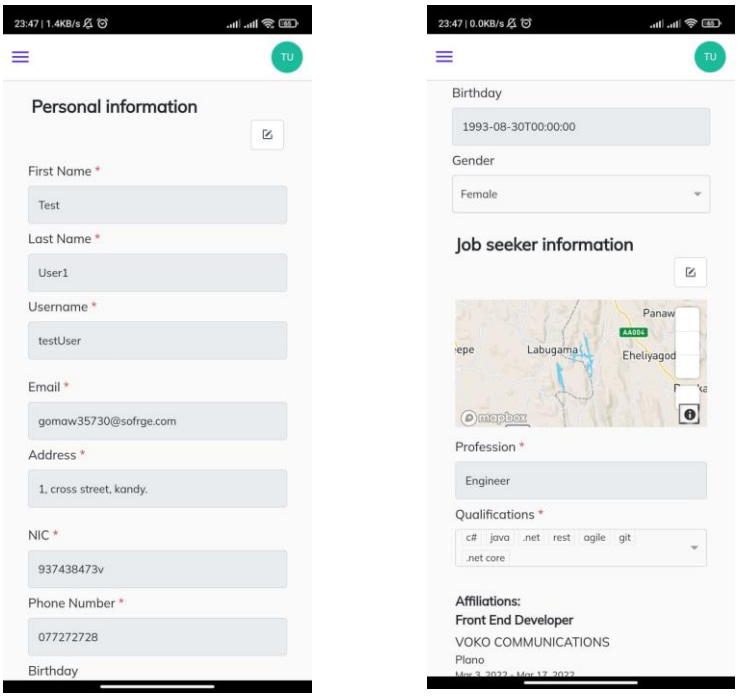


Figure 6.50: Personal Info

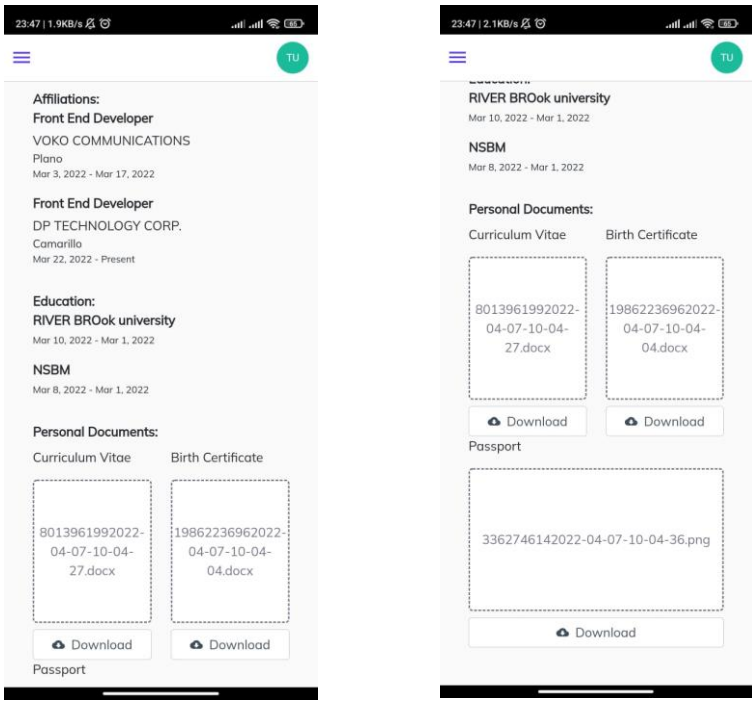


Figure 6.52: Affiliation Info

Jobseeker details are displayed here.

6.2.2.10 JOBSEEKER SIDE MENU AND OPTIONS

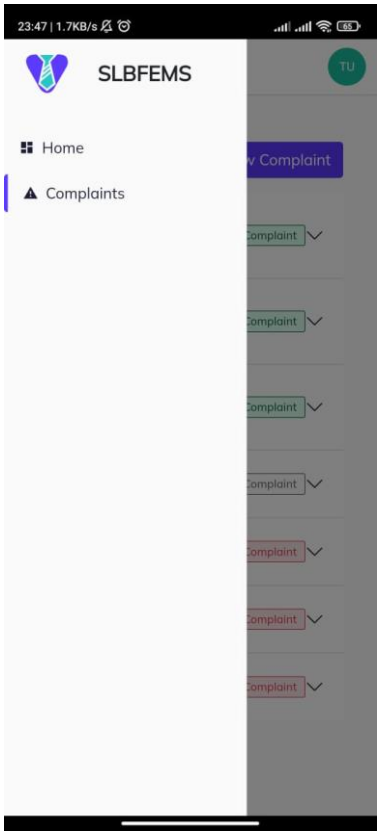


Figure 6.54: Job Seeker Side Menu

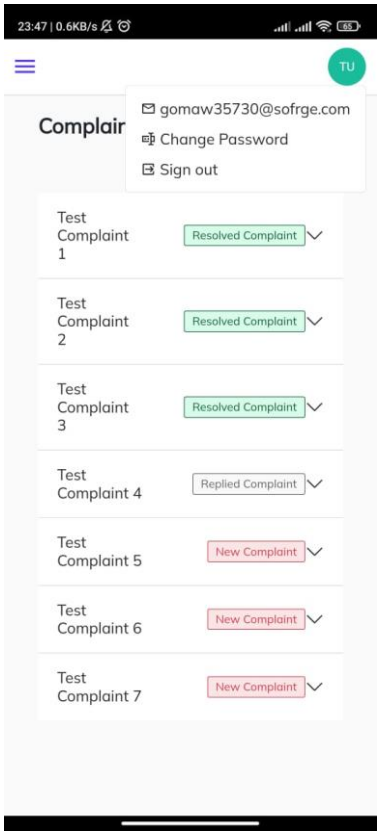


Figure 6.55: Job Seeker Options

This is the side menu in the mobile application.
Navigations to the home and complaints are displayed here.

Change password and sign out options are provided here.

6.2.2.11 COMPLAINTS

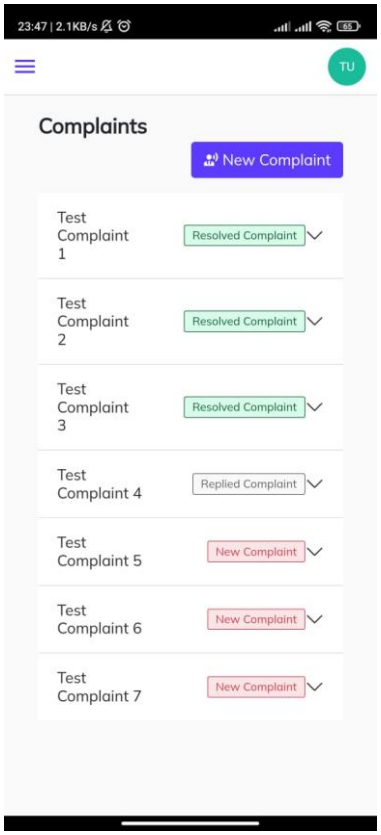


Figure 6.56: complaints

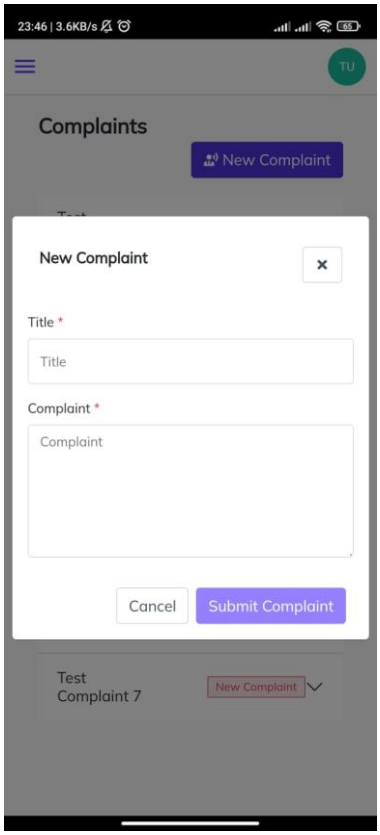


Figure 6.57: new complaint

Users can see the complaints here. To view the complaint, click on the complaint.

When the user clicks the new component button, the user can add a recent complaint by entering a title and complaint.

ADMIN

6.2.2.12 VIEW JOBSEEKERS

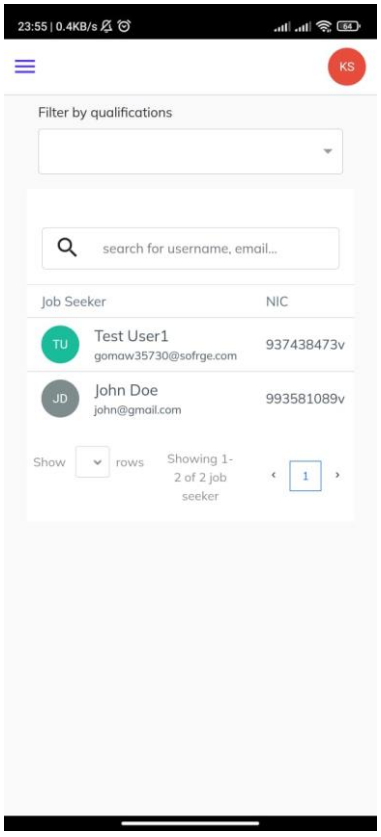


Figure 6.58: view job seekers

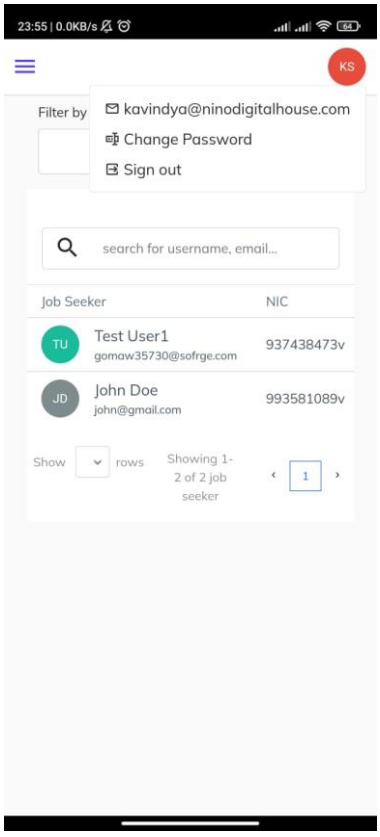


Figure 6.59: options

Users can view the job seekers with their details.

Change password and signout can do by this dropdown menu.

6.2.2.13 COLLAPSIBLE SIDE MENU

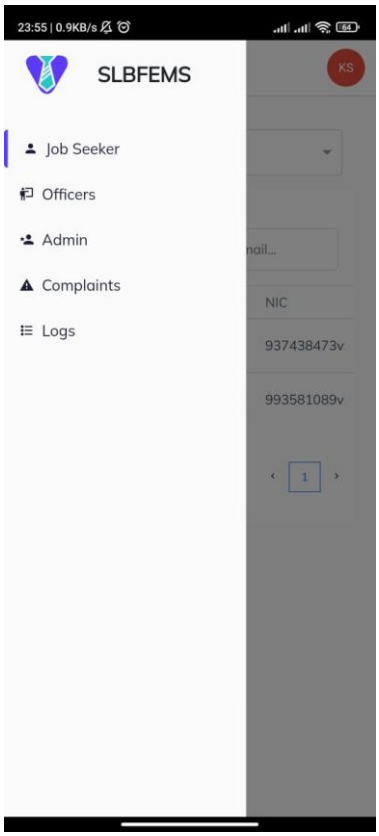


Figure 6.60: admin side menu

When it comes to mobile view, a collapsible menu is handy. Users can navigate to any page through the navigation items.

6.2.2.14 VIEW OFFICERS

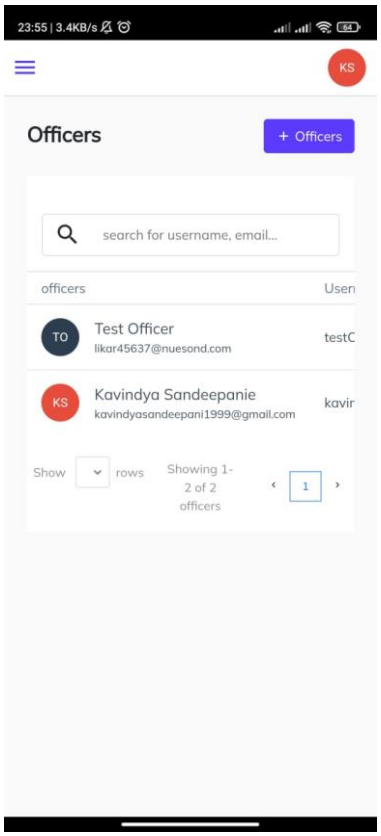


Figure 6.61: view officers

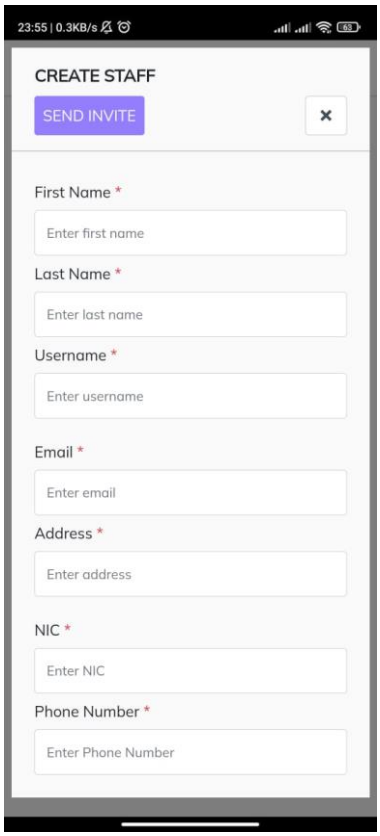


Figure 6.62: add a new officer

Users can view the officers' list with their details.

When the user clicks the officer's button, the user can add a new officer to the system by inviting the member.

6.2.2.15 KEBAB MENU FOR OFFICERS

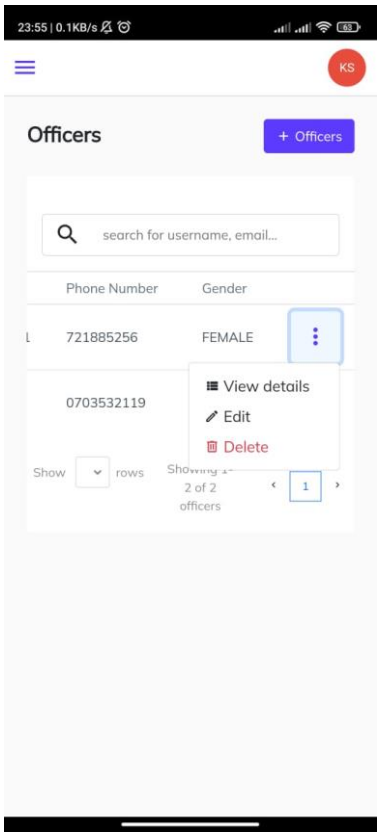


Figure 6.63: kebab menu officers

Users can edit, view details and delete officers from here.

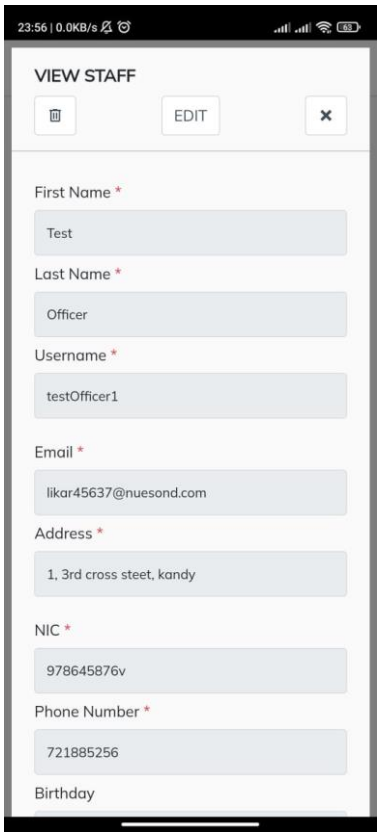


Figure 6.64: view officer details

Details of the officer are displayed here. Those options are also provided if the user wants to edit or delete them.

6.2.2.16 DELETE STAFF MEMBER

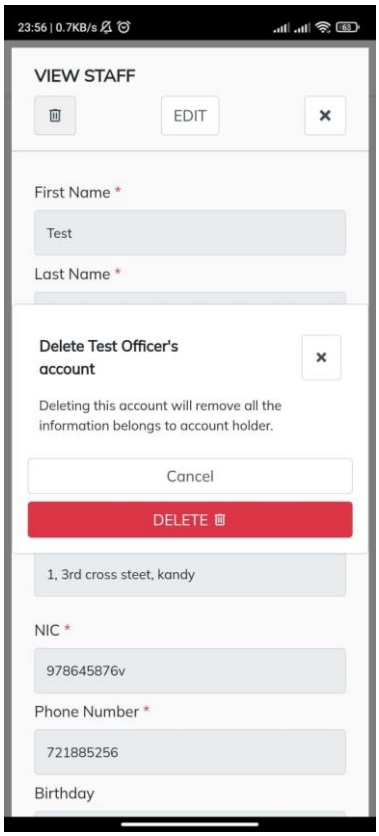


Figure 6.65: delete staff member

Users can delete the officer by clicking the delete button here.

6.2.2.17 VIEW ADMINS

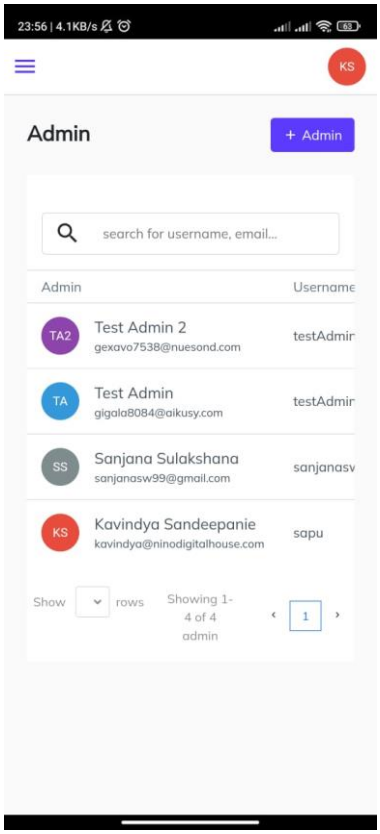


Figure 6.66: view admins

Users can view the admin details by scrolling right. To add a new admin, the user can click the admin button and fill out the form.

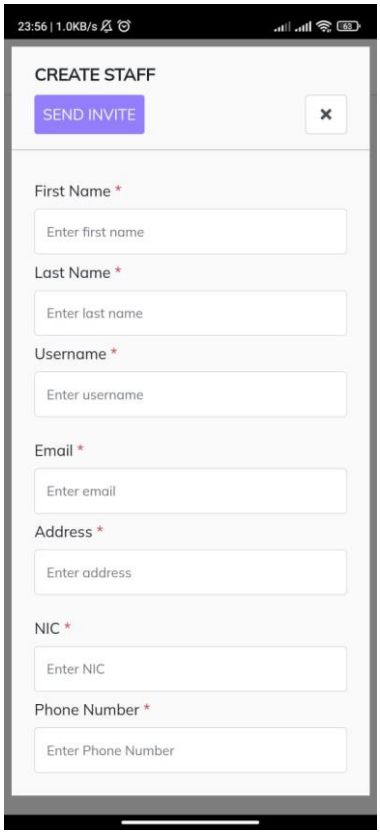


Figure 6.67: admin details

Here is the form. The user has to fill out the form and click the send invite button to the user.

6.2.2.18 KEBAB MENU IN ADMIN VIEW

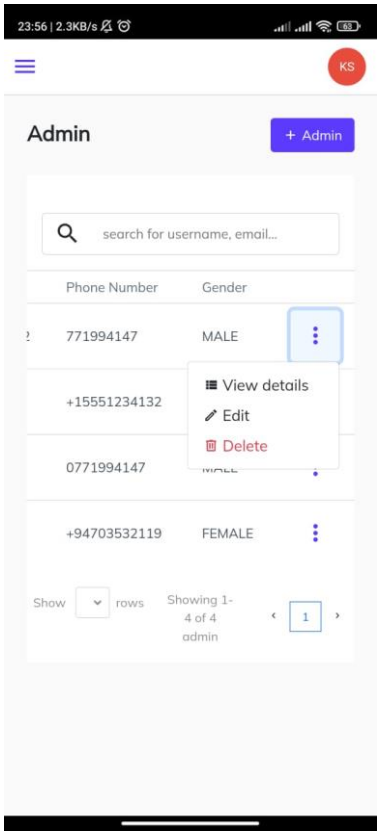


Figure 6.68: kebab menu

Users can view details and edit or delete them using the kebab menu.

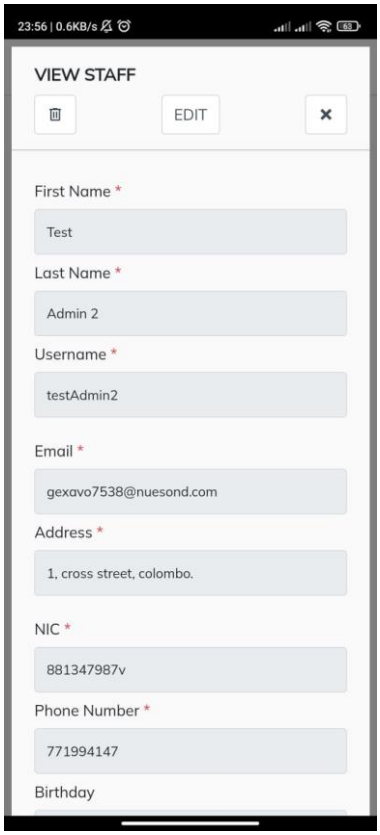
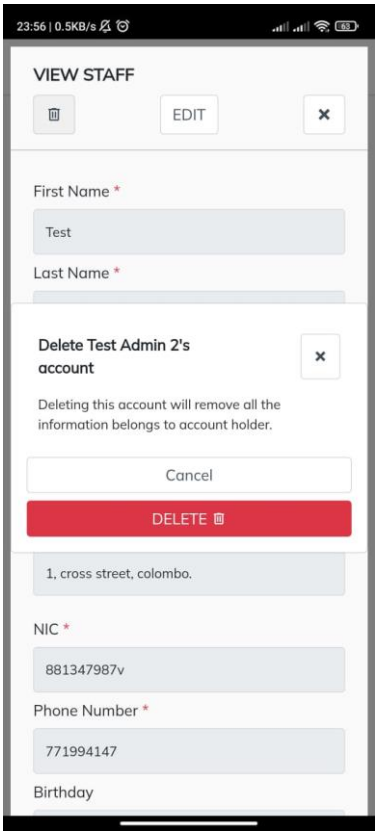


Figure 6.69: view staff details

Details of the admin are displayed clearly.

6.2.2.19 DELETE AN ADMIN



Users can delete the admin.

Figure 6.70: delete an admin

6.2.2.20 VIEW COMPLAINTS

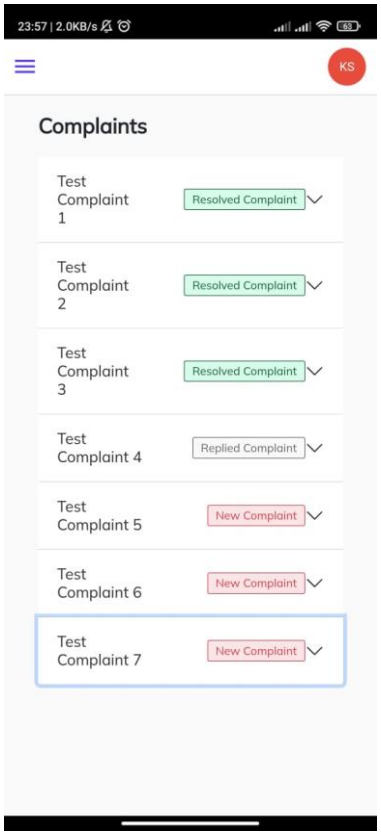


Figure 6.71: view complaints

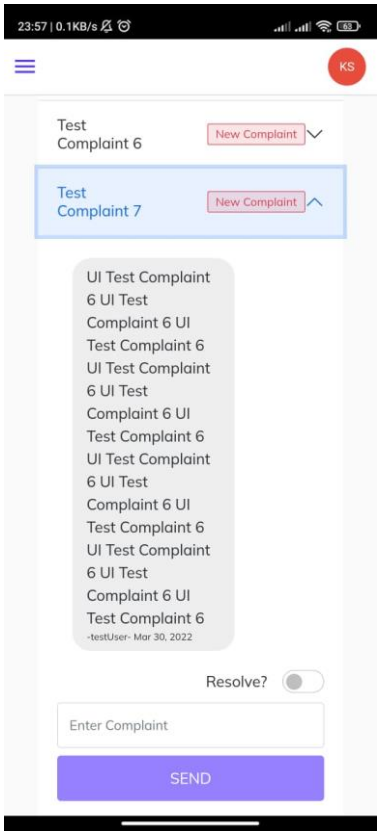


Figure 6.72: new complaint

Users can see the list of complaints with a label. So it is easy to understand. Users can click on the complaint and view the complaint.

The user can reply to the complaint. Also can mark it as resolved.
Then the label new complaint changes to the resolved complaint.

6.2.2.21 LOGS

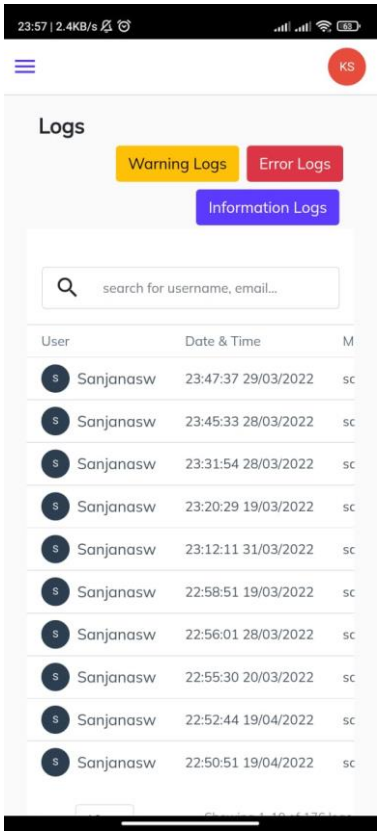


Figure 6.73: logs

Users can see the logs separated as warning, error, and information logs.

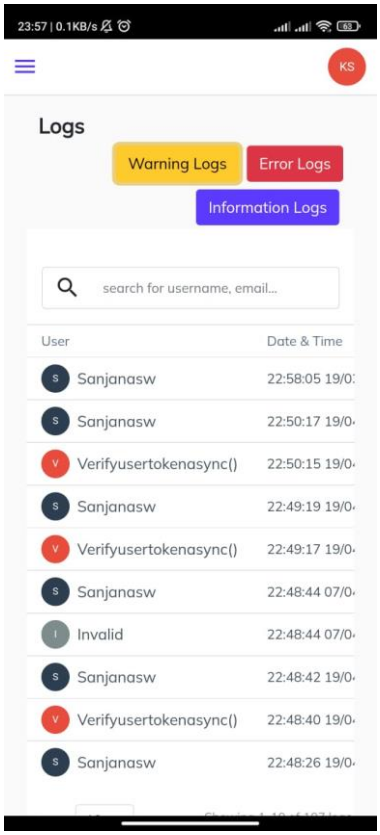


Figure 6.74: warning logs

Warning logs are displayed here.

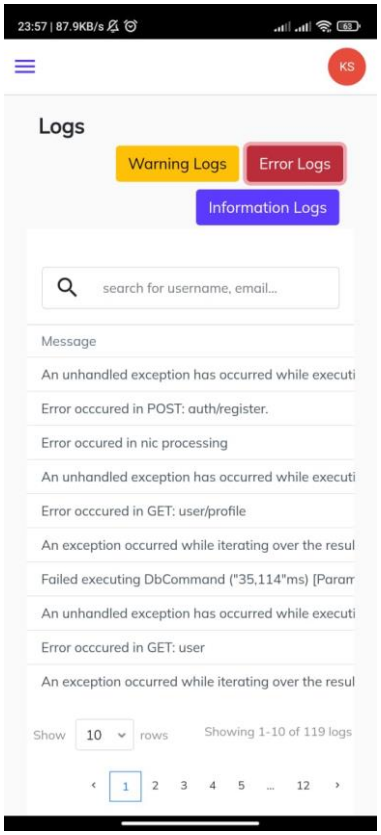


Figure 6.75: error logs

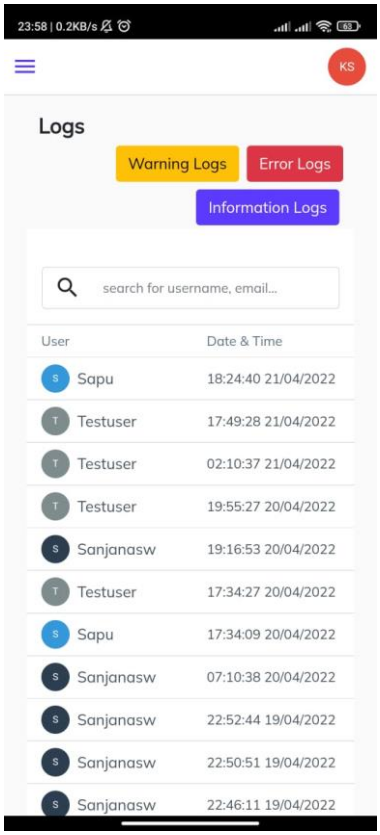
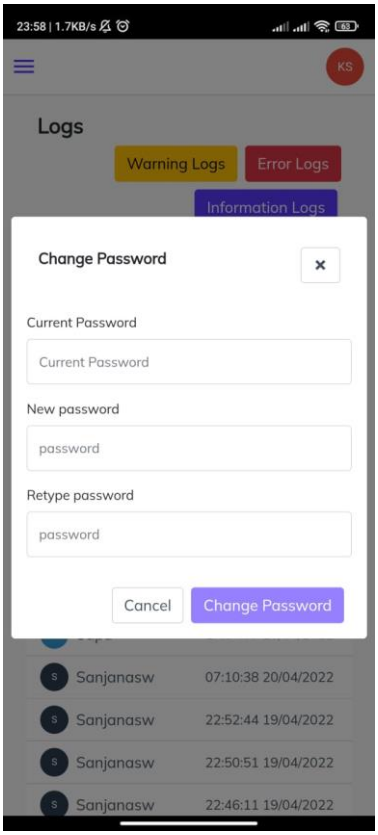


Figure 6.76: information logs

Error logs are displayed here.

All the logs are in this interface.

6.2.2.22 ADMIN PASSWORD CHANGE



Users can change users password from here.

Figure 6.77:change password

OFFICER

6.2.2.23 JOBSEEKER VIEW

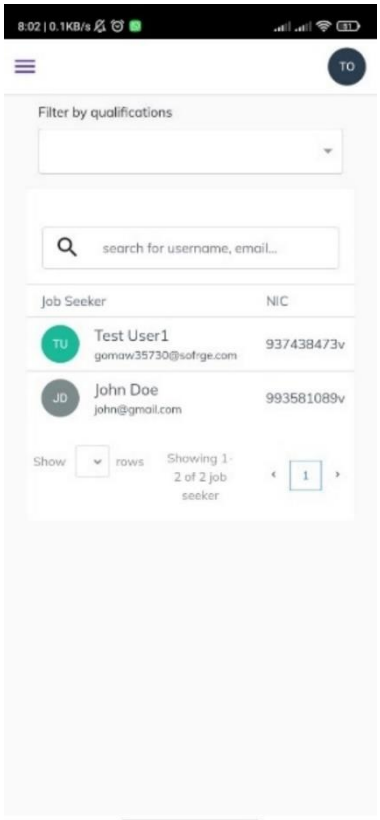


Figure 6.78:job seeker view

The list of the jobseekers is displayed here with their details.

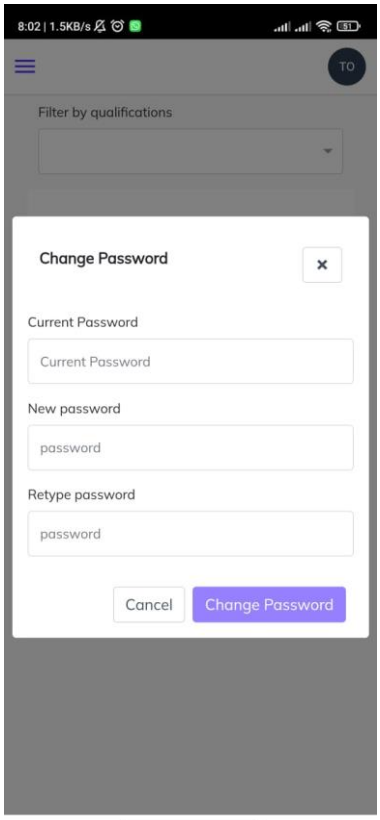
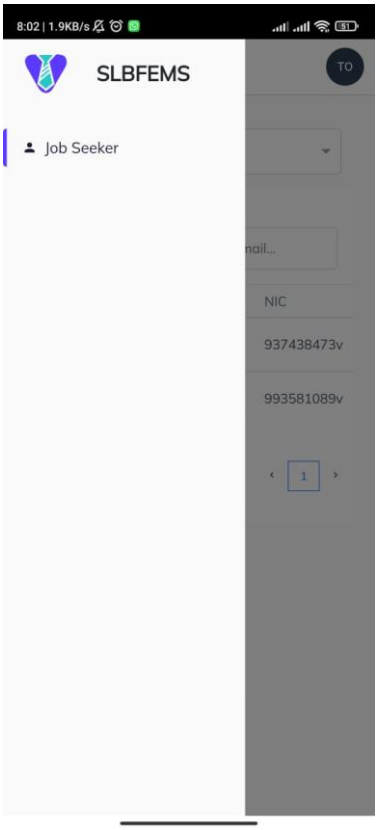


Figure 6.79:change password

Officers can change the password also from this interface.

6.2.2.24 OFFICER SIDE MENU



The side menu of the officer is displayed with the navigation items.

Figure 6.80:officer side menu

7 ISSUES AND APPROACHES TO RESOLVING

When the system was being implemented, each team member encountered a unique set of issues. At first, some of the technologies were unfamiliar.

- Git conflicts
- System errors
- Technical issues
- Designing issues
- Database syncing issues

Methods used to resolve the issues

- We moved to feature branches to overcome that instead of using flocked repositories. We maintained a proper flow of git to minimize code conflicts.
- Team members shared their knowledge with other team members to correct system flaws and minimize issues.
- To keep the project safe as a backup and collaborate, OneDrive and GITHUB were utilized.
- When we started the development with our laptops without hosting the system on the cloud, there was an issue with syncing data in different local Databases into a single database. To overcome that, we hosted our application in Azure app service and SQL server from the initial step and configured CI/CD pipelines with GitHub action to do quick deployments. And configured Discord notification channel to notify front-end developers about releases

8 API DOCUMENTATION

Swagger documentation: slbfems.azurewebsites.net/swagger

All end-points are described with all responses possible with status codes and response types in the Swagger documentation.

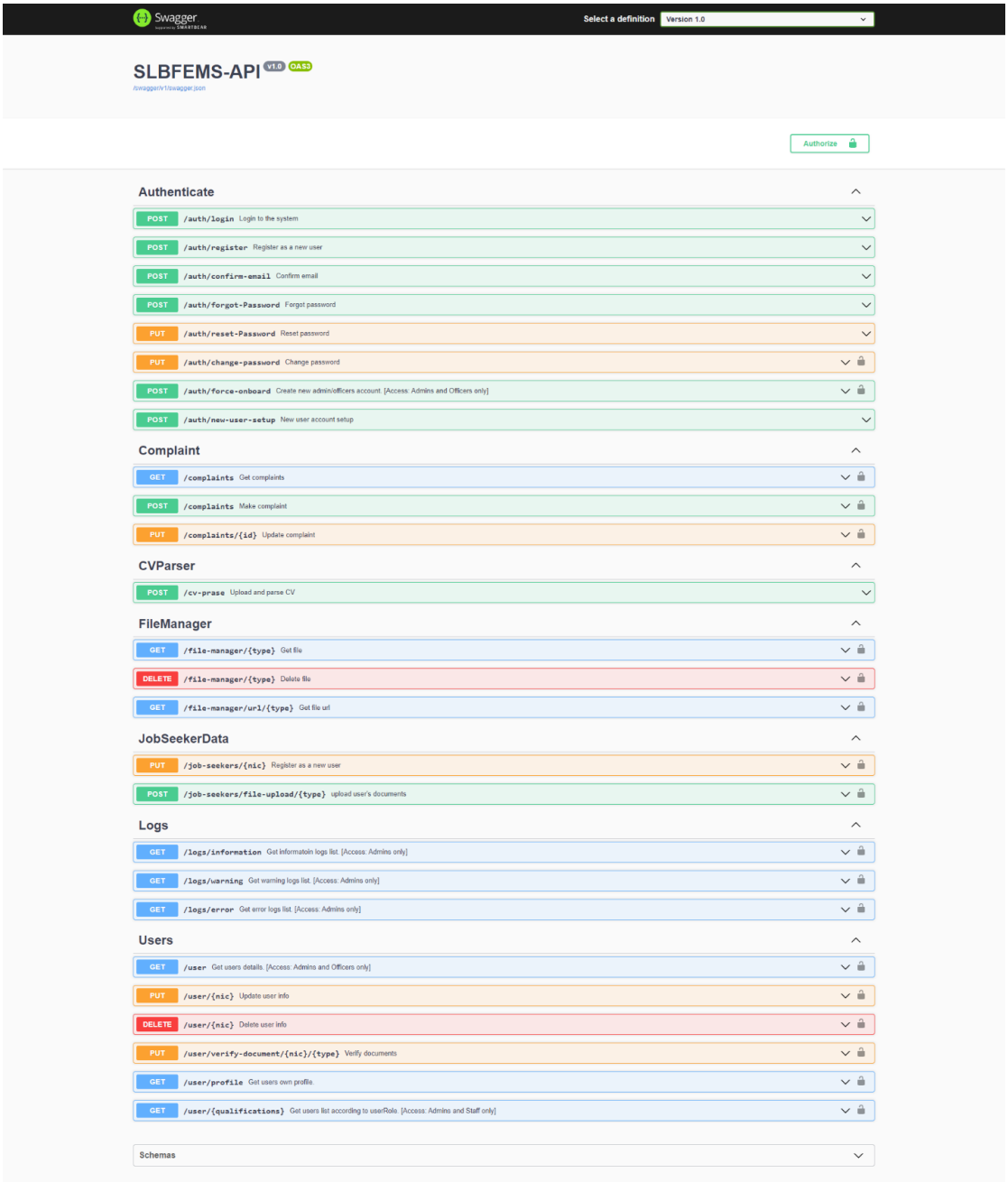


Figure 8.1: API documentation(swagger)

The API supports both JSON and XML requests and responses. It works by handling the request type through the Request Header.

JSON

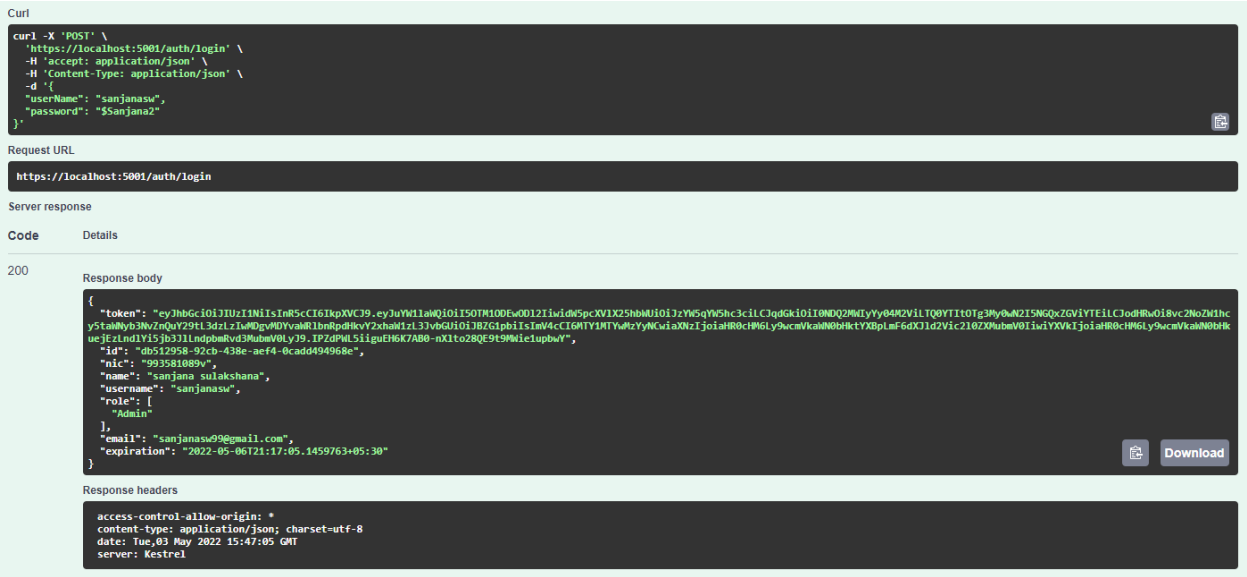


Figure 8.2: JSON Response

XML

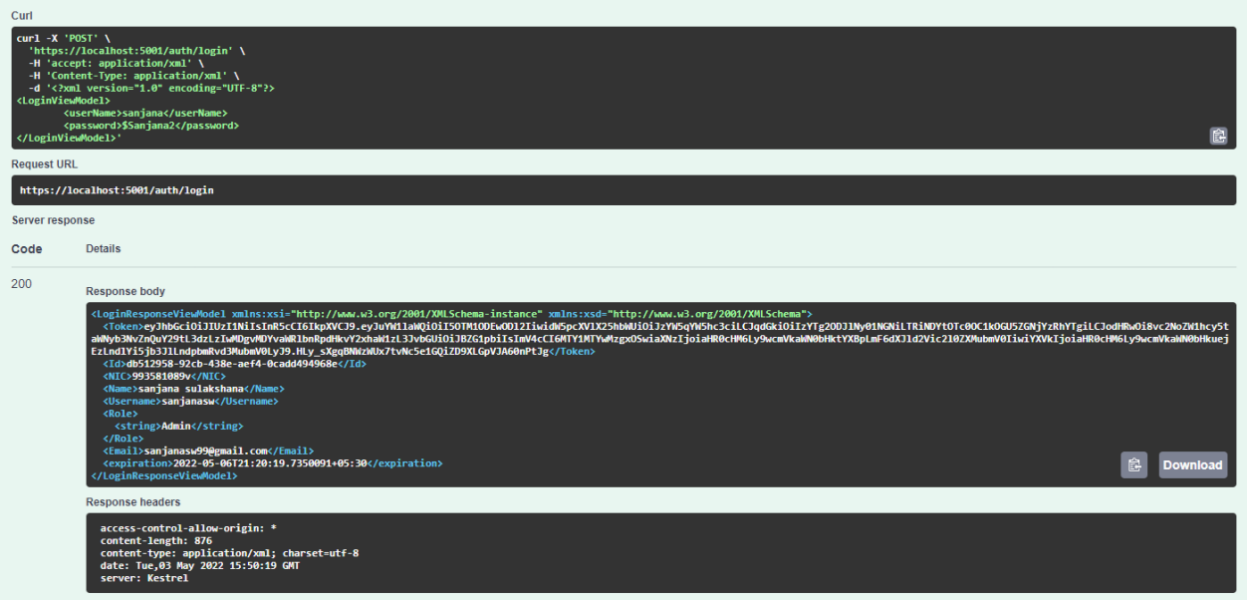


Figure 8.3: XML Response

8.1 AUTHENTICATION

8.1.1 LOGIN

Endpoint - POST: /auth/login

Users can get the JWT token by sending a username and password to the system through this end-point.

Request body –

```
{
 "userName": "string",
 "password": "string"
}
```

Success response -

```
{
  "token": "string",
  "id": "string",
  "nic": "string",
  "name": "string",
  "username": "string",
  "role": [
    "string"
  ],
  "email": "string",
  "expiration": "2022-04-23T02:23:16.890Z"
}
```

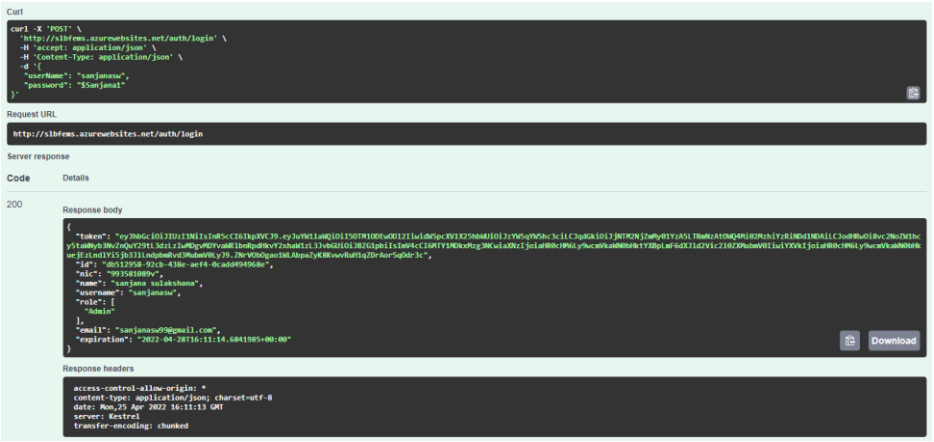


Figure 8.4: Success Response For Login

Other responses :

1. Code 401: Unauthorized users
2. Code 403: When the user hasn't verified email after registering

8.1.2 REGISTRATION

Endpoint - POST: /auth/register

Users can register to the system by entering a user NIC number, a username, user first name, user last name, a proper email address, user phone number, gender, user birthday, user address and a valid password which does not fail against the validations included for the password(at least eight characters, At least one letter in Uppercase, At least one letter in Lowercase and At least one unique character)

Request body –

```
{
  "userInfo": {
    "nic": "string",
    "firstName": "string",
    "lastName": "string",
    "email": "user@example.com",
    "phoneNumber": "string",
    "address": "string",
    "username": "string",
    "password": "string",
    "role": 0
  },
  "jobSeekerData": {
    "currentLat": "string",
    "currentLong": "string",
    "profession": "string",
    "affiliations": [
      {
        "start": "string",
        "end": "string",
        "location": "string",
        "organization": "string",
        "title": "string"
      }
    ],
    "qualifications": [
      "string"
    ],
    "education": [
      {
        "start": "string",
        "end": "string",
        "name": "string"
      }
    ],
    "cvFileName": "string"
  }
}
```

Success response -

```
{
  "status": "string",
  "message": "string"
}
```

Figure 8.5: Success Response For Registration

Other responses:

1. Code 400: Jobseeker register request without providing job seeker data
2. Code 409: Username or email already exists
3. Code 500: Internal server error

8.1.3 CONFIRM EMAIL

Endpoint - POST: /auth/confirm-email

After registering to the system, a confirmation email is sent to the email user entered in the registration form. Users have to Confirm the email by clicking the confirm button before the link expires (the link will be expired after 24 hours).

Request body –

```
{
  "userId": "string",
  "token": "string"
}
```

Success response -

```
{
  "status": "string",
  "message": "string"
}
```

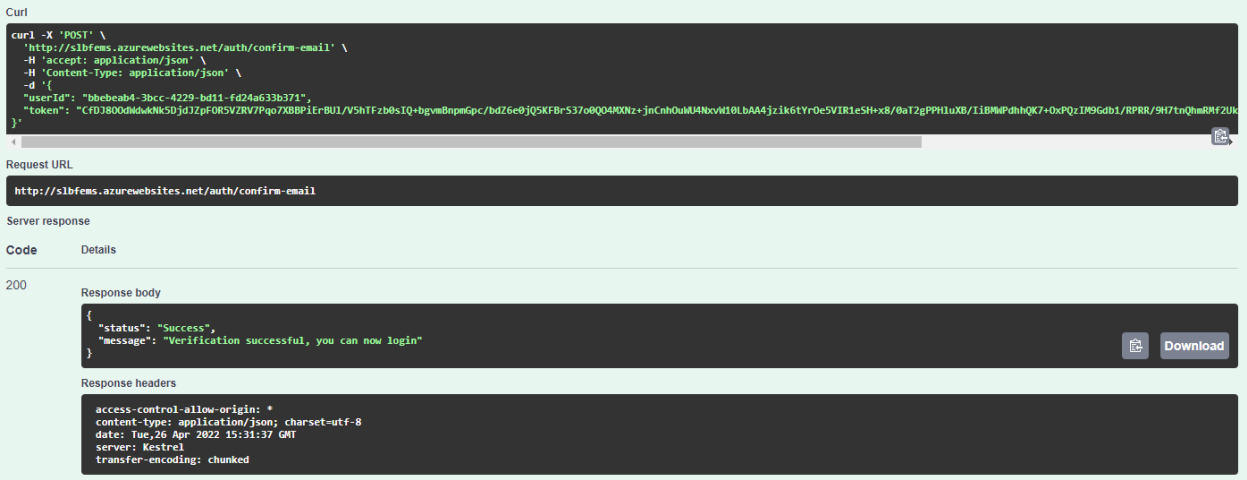


Figure 8.6: Success Response For Confirm Email

Other responses:

1. Code 400: Invalid token
2. Code 404: Username or email already exists

8.1.4 FORGOT PASSWORD

Endpoint - POST: /auth/forgot-password

If the user forgot the user password, the user could go for the resetting by going through the link sent to the user's email. The user has to enter the user's email, which the user entered in the registration process.

Request body –

```
{
  "email": "user@example.com"
}
```

Success response -

```
{
  "status": "string",
  "message": "string"
}
```

Figure 8.7: Success Response For Forgot Password

Other responses:

1. Code 404: User not found

8.1.5 RESET PASSWORD

Endpoint - PUT: /auth/reset-password

Users can reset their passwords by entering a new password for the user account as a user-entered user email to get the reset password link.

Request body –

```
{
  "userid": "string",
  "token": "string",
  "password": "string"
}
```

Success response -

```
{
  "status": "string",
  "message": "string"
}
```

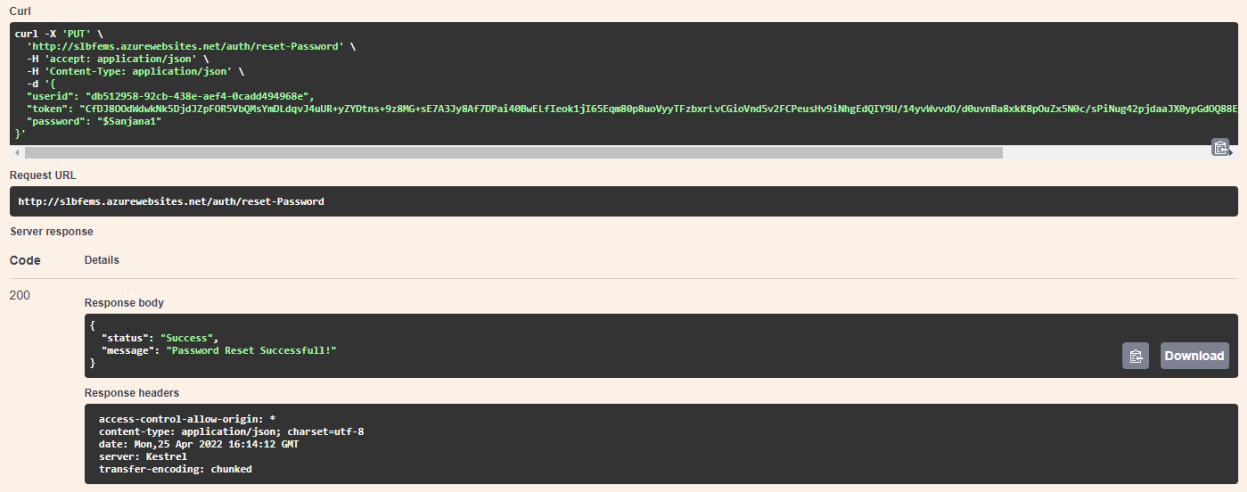


Figure 8.8: Success Response For Reset Password

Other responses:

1. Code 400: Invalid token
2. Code 404: User not found

8.1.6 CHANGE PASSWORD

Endpoint - PUT: /auth/change-password

Users can change users current password to a new password by entering the current password and new password in the given two fields

Request body –

```
{
  "currentPassword": "string",
  "newPassword": "string"
}
```

Success response -

```
{
  "status": "string",
  "message": "string"
}
```

Figure 8.9: Success Response For Change Password

Other responses:

1. Code 400: Password doesn't meet minimum requirements
2. Code 401: Unauthorized
3. Code 403: Current password incorrect
4. Code 404: User not found

8.1.7 ADD NEW USER

Endpoint - POST: /auth/force-onboard

Admin and Officers can add new users to the system by entering the user's NIC number, email, last name, first name, phone number, address, username and user's role into the relevant fields.

Request body –

```
{
  "nic": "string",
  "firstName": "string",
  "lastName": "string",
  "email": "user@example.com",
  "phoneNumber": 0,
  "gender": 0,
  "address": "string",
  "birthday": "2022-04-23T04:54:09.580Z",
  "username": "string",
  "role": 0
}
```

Success response –

```
{
  "status": "string",
  "message": "string"
}
```

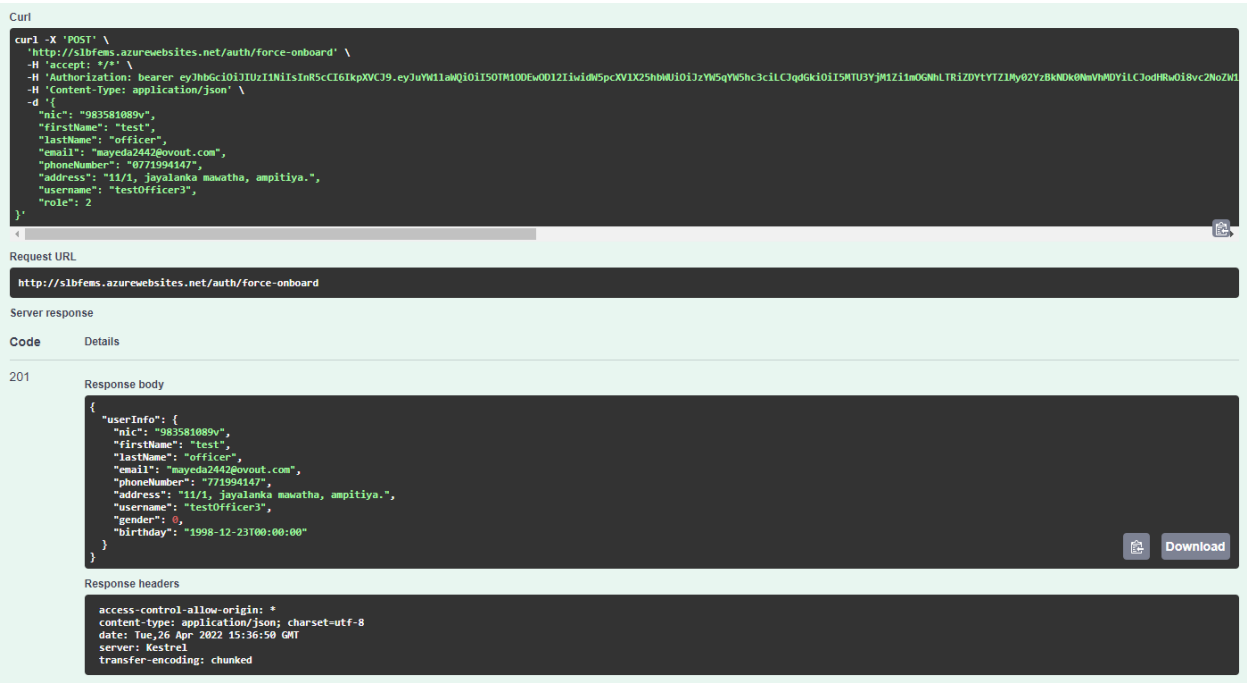


Figure 8.10: Success Response For Add New User

Other responses:

1. Code 400: Invalid user role
2. Code 401: Unauthorized
3. Code 403: Forbidden
4. Code 409: Username or email already exists
5. Code 500: Internal server error

8.1.8 NEW USER SETUP

Endpoint - POST: /auth/new-user-setup

If the user is added to the system by an Admin or a Staff member, the user can add a password to the account by going through the email sent to the user's email.

Request body –

```
{
  "email": "user@example.com"
}
```

Success response -

```
{
  "userid": "string",
  "token": "string",
  "password": "string"
}
```

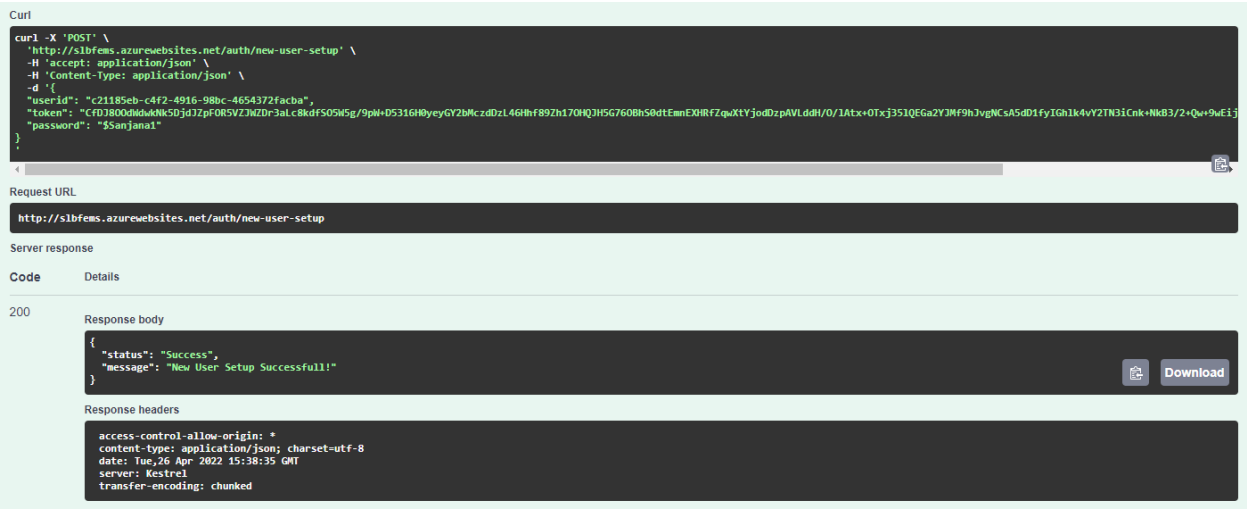


Figure 8.11: Success Response For New User Setup

Other responses:

1. Code 400: Invalid user role
2. Code 404: User not found

8.2 COMPLAINT

8.2.1 RETRIEVE COMPLAINTS

Endpoint - GET: /complaints
GET: /complaint?id={id}

Users can Get the message included in the complaint and the NIC of the complaint sender, bypassing the Id of the Complaint. If no ID is passed, all the complaints are returning

Success response –


```
[
  {
    "complaintStatus": {
      "id": 0,
      "title": "string",
      "status": 0,
      "createdBy": "string",
      "createdAt": "2022-04-23T05:34:29.930Z",
      "completedAt": "2022-04-23T05:34:29.930Z"
    },
    "messageThread": [
      {
        "complaint": "string",
        "nic": "string",
        "timeStamp": "2022-04-23T05:34:29.930Z",
        "name": "string"
      }
    ]
  }
]
```

Figure 8.12: Success Response For Retrieve Complaints

Other responses:

1. Code 401 : Unauthorized

8.2.2 ADD COMPLAINT

Endpoint - POST: / complaints

Users can Make Complaints to the system by entering the message. NIC of the logged-in user is getting by the system.

Request body –

```
{
  "title": "string",
  "complaint": "string"
}
```

Success response -

```
{
  "complaintStatus": {
    "id": 0,
    "title": "string",
    "status": 0,
```

```
    "createdBy": "string",
    "createdAt": "2022-04-23T05:35:46.013Z",
    "completedAt": "2022-04-23T05:35:46.013Z"
  },
  "messageThread": [
    {
      "complaint": "string",
      "nic": "string",
      "timeStamp": "2022-04-23T05:35:46.013Z",
      "name": "string"
    }
  ]
}
```

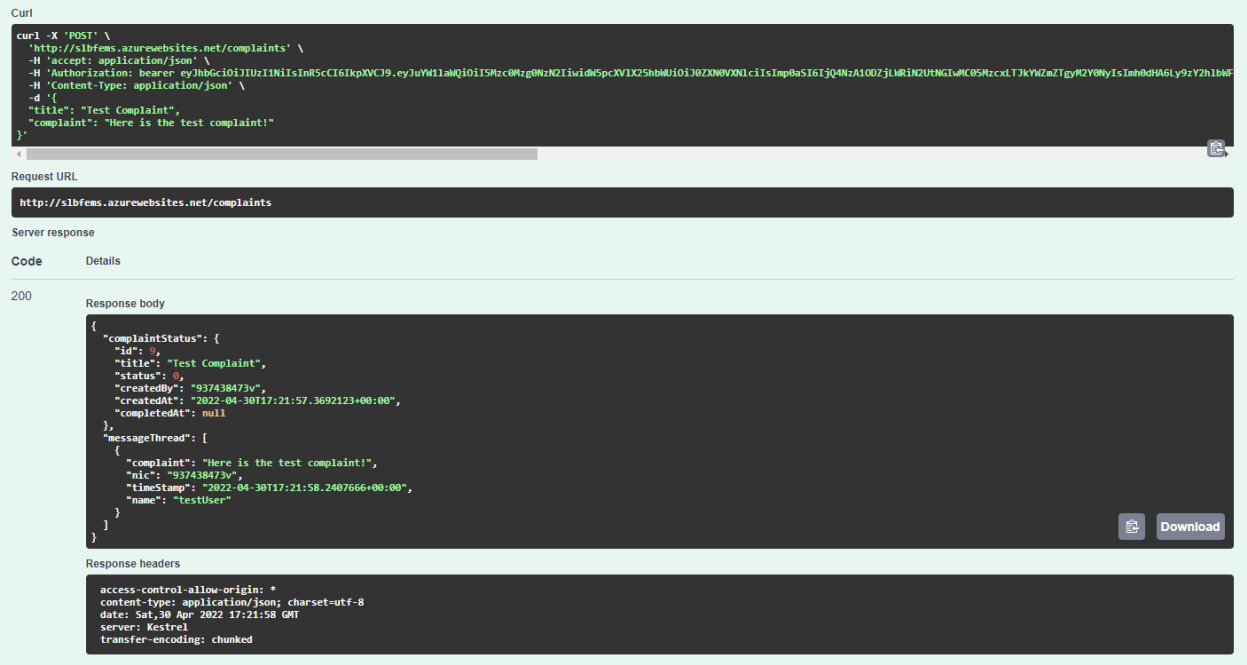



Figure 8.13: Success Response For Add complaint

Other responses:

1. Code 401: Unauthorized
2. Code 404: User not found

8.2.3 EDIT COMPLAINT

Endpoint - PUT: / complaints /{id}

If the user wants to Edit the Complaint that the user sent, the user can edit the message the user sent.

Request body –

```
{
  "title": "string",
  "complaint": "string",
  "isComplete": true
}
```

Success response –

```
{
  "status": "string",
  "message": "string"
}
```

Figure 8.14: Success Response For Edit Complaint

Other responses:

1. Code 401: Unauthorized
2. Code 404: Complaint or user not found

8.3 CVPARSER

8.3.1 UPLOAD CV

Endpoint - POST: /cv-prase

User can Upload userr CV to the system which is in a valid format(.docx or .pdf)

Request body –

```
{
   "File”: IFormFile
}
```

Success response -

```
{
  "cvData": {
    "name": "string",
    "email": "string",
    "address": "string",
    "education": [
      {
        "dates": [
          "string"
        ],
        "name": "string"
      }
    ],
    "affiliation": [
      {
        "dates": [
          "string"
        ],
        "location": "string",
        "organization": "string",
        "title": "string"
      }
    ],
    "skills": [
      "string"
    ]
  },
  "fileName": "string"
}
```

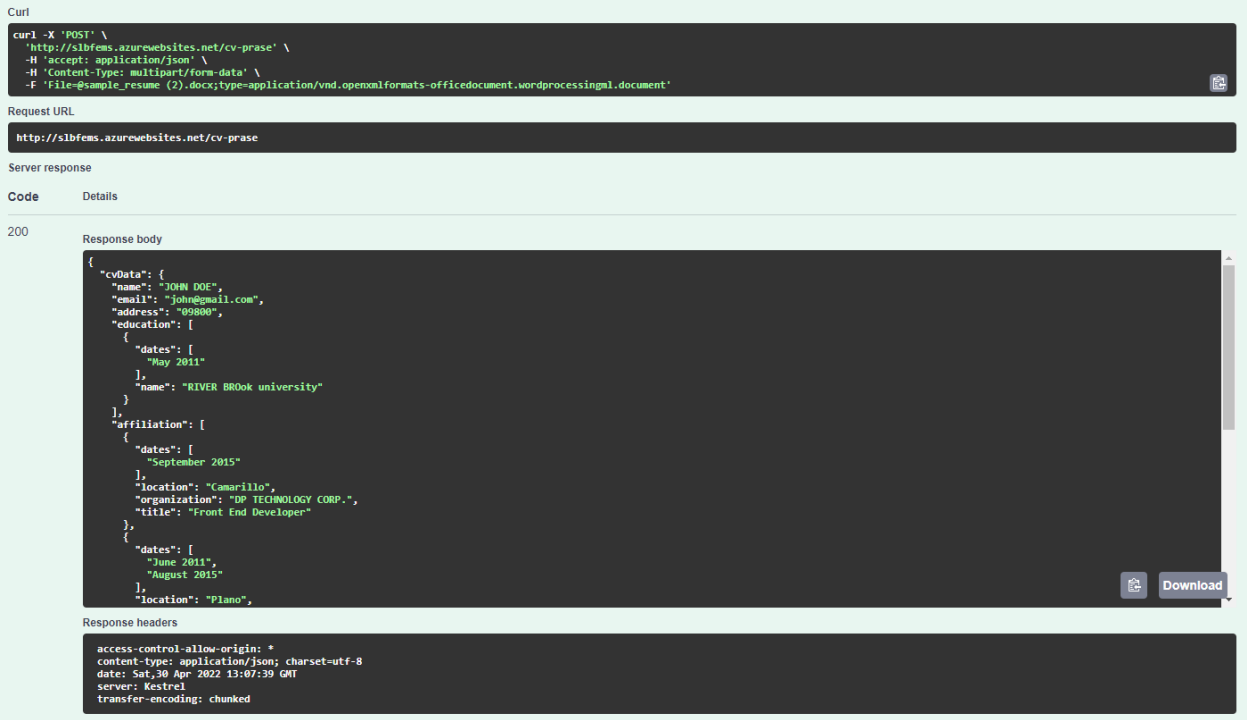


Figure 8.15: Success Response For Upload CV

Other responses:

1. Code 400: Unsupported file format
2. Code 500: Internal server error

8.4 FILEMANAGER

8.4.1 GET FILES

Endpoint - GET: /file-manager/{type}?fileName={name}

User can retrive the file by entering the category and the file name.

Success response -

`Returns IFormFiles`

Figure 8.16: Success Response For Get files

Other responses:

1. Code 401: Unauthorized
2. Code 404: File not found

8.4.2 DELETE FILES

Endpoint - DELETE: /file-manager/{type}?fileName={name}

User can Delete the file by entering the type of the file and name of the file.

Success response -

```
{
  "status": "string",
  "message": "string"
}
```

Figure 8.17: Success Response For Delete Files

Other responses:

1. Code 401: Unauthorized
2. Code 404: File not found

8.4.3 GET FILE URL

Endpoint - GET: /file-manager/{type}?fileName={name}

According to the file category can get the file url.

Success response -

```
{
  "url": "string"
}
```

Figure 8.18: Success Response For Get File URL

Other responses:

1. Code 401: Unauthorized
2. Code 404: File not found

8.5 JOBSEEKERDATA

8.5.1 UPLOAD USER'S DOCUMENTS

Endpoint - POST: /job-seekers/file-upload/{type}

Users can upload file documents after Selecting the type of the uploading file

Request body –

```
{
  "File" : IFormFile
}
```

Success response -

```
{
  "status": "string",
  "message": "string"
}
```

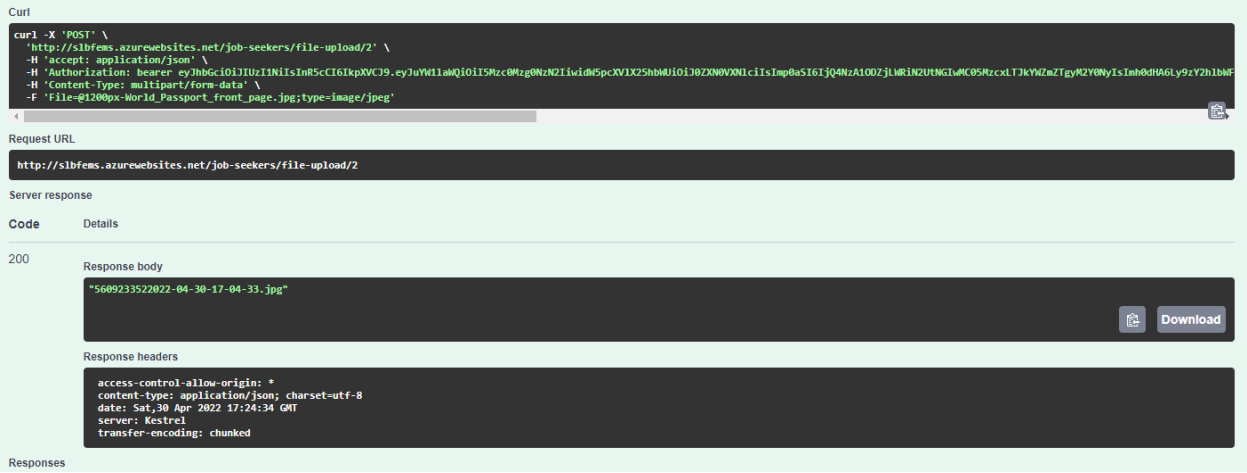


Figure 8.19: Success Response For Upload user's documents

Other responses :

1. Code 400: Invalid user role
2. Code 401: Unauthorized
3. Code 403: Can't update others' data

8.5.2 UPDATE JOBSEEKERS DATA

Endpoint - PUT: / jobseeker/ {NIC}

Users can Update the NIC, address, email, first name, last name, username, phone number, birthday and gender

Request body –

```
{
  "nic": "string",
  "currentLat": "string",
  "currentLong": "string",
  "profession": "string",
  "affiliations": [
    {
      "start": "string",
      "end": "string",
      "location": "string",
      "organization": "string",
      "title": "string"
    }
  ],
  "qualifications": [
    "string"
  ],
  "education": [
    {
      "start": "string",
      "end": "string",
      "name": "string"
    }
  ]
}
```

Success response –

```
{
  "status": "string",
  "message": "string"
}
```

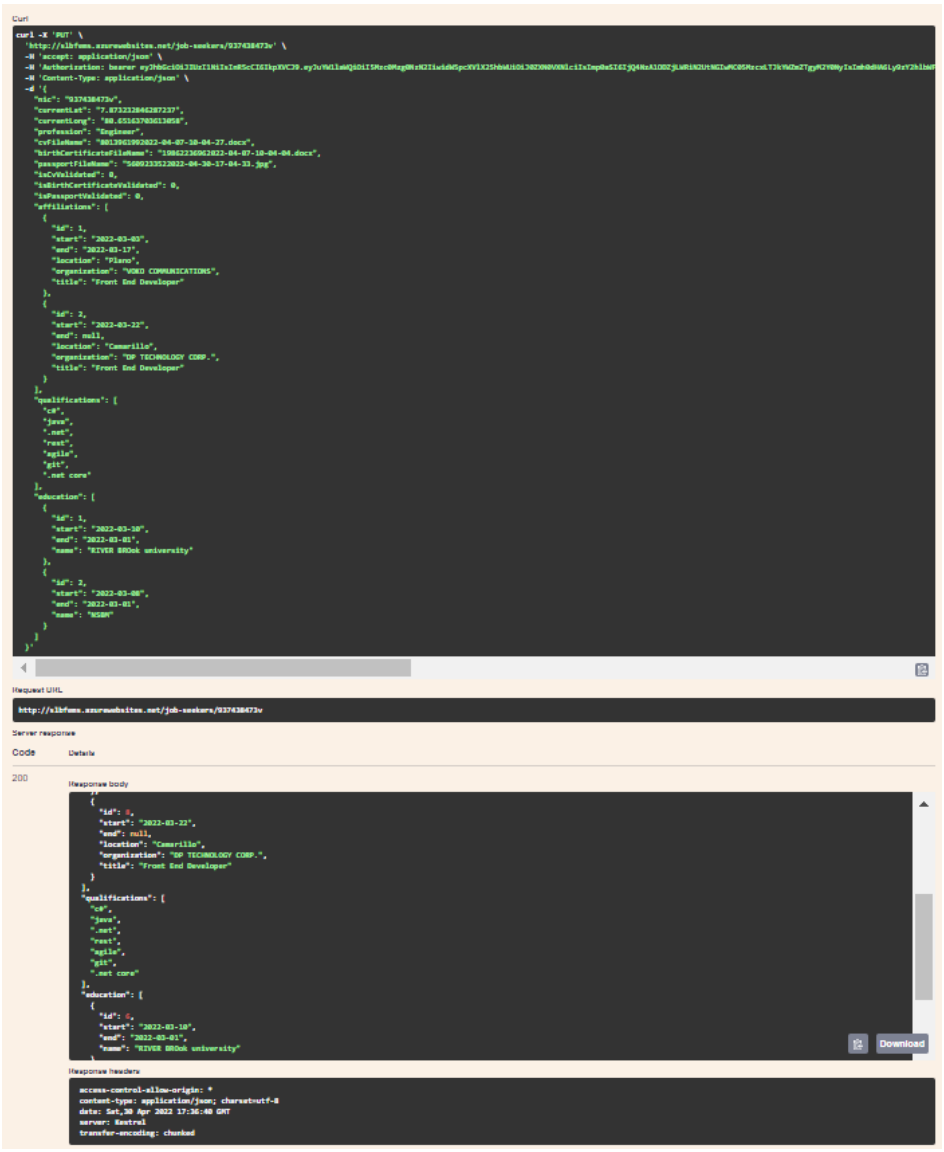


Figure 8.20: Success Response For Update Jobseeker's data

Other responses:

1. Code 401: Unauthorized
2. Code 404: User not found

8.6 USERS

8.6.1 GET USER DETAILS

Endpoint - GET: /user?role={role}&nic={nic}

Admins and Staff members can retrieve the NIC, address, email, first name, last name, username, phone number, birthday and gender of the list of users regarding the entering role. If NIC is also entered, the address, email, first name, last name, username, phone number, birthday and gender of the owner of the NIC number are retrieved. If no role is selected, the role is taken as the user.

Success response –


```
[
  {
    "userInfo": {
      "nic": "string",
      "firstName": "string",
      "lastName": "string",
      "email": "string",
      "phoneNumber": "string",
      "address": "string",
      "username": "string",
      "gender": 0,
      "birthday": "2022-04-23T07:18:58.390Z"
    },
    "jobSeekerData": {
      "currentLat": "string",
      "currentLong": "string",
      "profession": "string",
      "cvFileName": "string",
      "birthCertificateFileName": "string",
      "passportFileName": "string",
      "isCvValidated": 0,
      "isBirthCertificateValidated": 0,
      "isPassportValidated": 0,
      "affiliations": [
        {
          "id": 0,
          "start": "string",
          "end": "string",
          "location": "string",
          "organization": "string",
          "title": "string"
        }
      ],
      "qualifications": [
        "string"
      ],
      "education": [
```

```
        {
          "id": 0,
          "start": "string",
          "end": "string",
          "name": "string"
        }
      ]
    }
  }
]
```

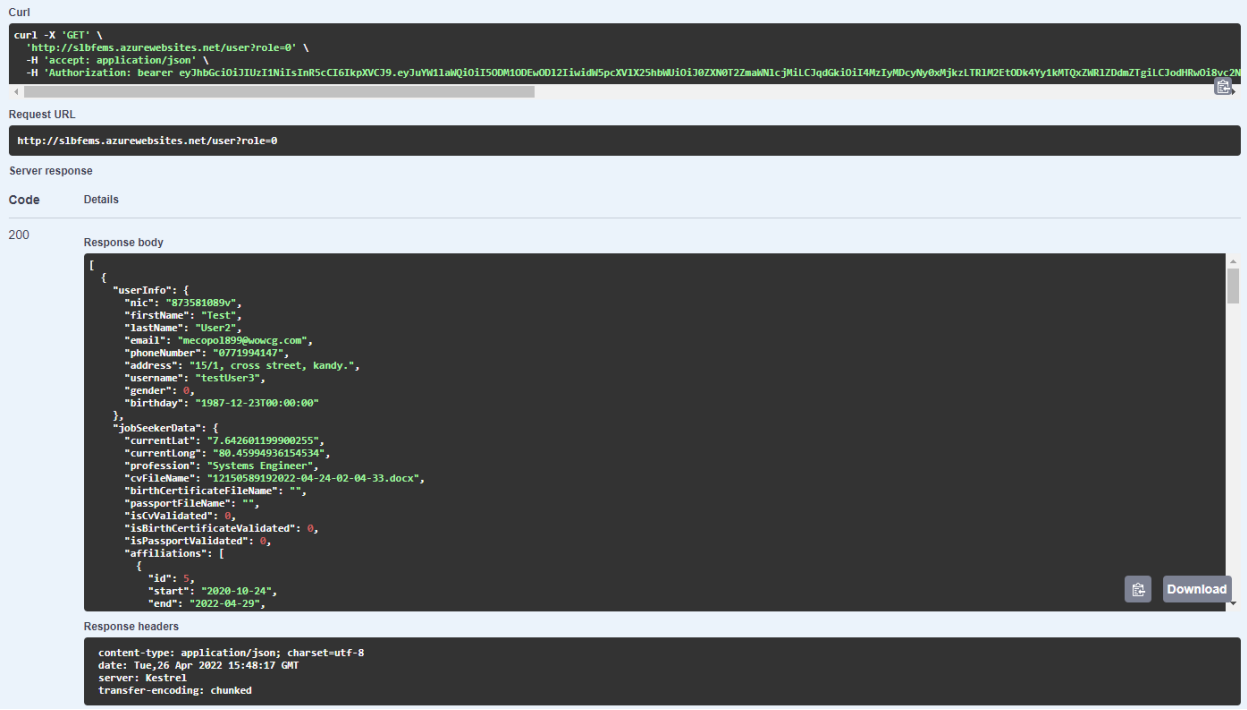



Figure 8.21: Success Response For Get User Details

Other responses:

1. Code 401: Unauthorized
2. Code 403: Forbidden
3. Code 404: Requested user not found

8.6.2 UPDATE USER INFORMATION

Endpoint - PUT: / user/ {NIC}

Users can Update the NIC, address, email, first name, last name, username, phone number, birthday and gender

Request body –

```
{
  "nic": "string",
  "firstName": "string",
  "lastName": "string",
  "email": "user@example.com",
  "phoneNumber": "string",
  "address": "string",
  "username": "string"
}
```

Success response –

```
{
  "status": "string",
  "message": "string"
}
```

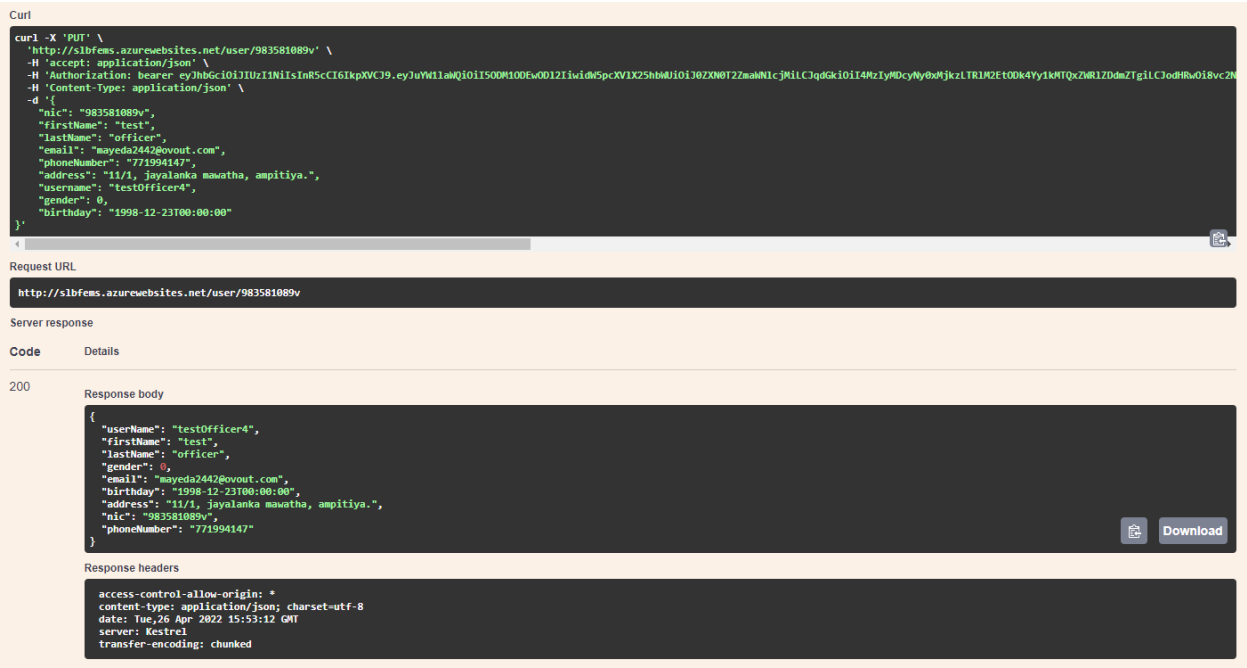


Figure 8.22: Success Response For Update User Information

Other responses:

1. Code 400: Provided nic's are not matching
2. Code 401: Unauthorized
3. Code 403: Users can't update others' data
4. Code 500: Internal server error

8.6.3 DELETE USER INFORMATION

Endpoint - DELETE: / user/ {NIC}

Admin can Delete the user by entering the NIC of the user

Success response –

```
{
  "status": "string",
  "message": "string"
}
```

Figure 8.23: Success Response For Delete User Information

Other responses:

1. Code 401: Unauthorized
2. Code 403: Forbidden
3. Code 404: Use not found
4. Code 500: Internal server error

8.6.4 VERIFY DOCUMENTS

Endpoint - PUT: /user/verify-document/{nic}/{type}?isapproved={isApproved}

Staff members can Verify the uploaded documents if all the required documents are uploaded to the system

Success response –

```
{
  "status": "string",
  "message": "string"
}
```

Figure 8.24: Success Response For Verify Documents

Other responses:

1. Code 401: Unauthorized
2. Code 403: Forbidden
3. Code 404: User not found

8.6.5 GET THE USER'S PROFILE

Endpoint - GET: /user/profile

Get users' profiles.

Success response –


```
{
  "userInfo": {
    "nic": "string",
    "firstName": "string",
    "lastName": "string",
    "email": "string",
    "phoneNumber": "string",
    "address": "string",
    "username": "string",
    "gender": 0,
    "birthday": "2022-04-23T07:59:33.528Z"
  },
  "jobSeekerData": {
    "currentLat": "string",
    "currentLong": "string",
    "profession": "string",
    "cvFileName": "string",
    "birthCertificateFileName": "string",
    "passportFileName": "string",
    "isCvValidated": 0,
    "isBirthCertificateValidated": 0,
    "isPassportValidated": 0,
    "affiliations": [
      {
        "id": 0,
        "start": "string",
        "end": "string",
        "location": "string",
        "organization": "string",
        "title": "string"
      }
    ],
    "qualifications": [
      "string"
    ],
    "education": [
      {
        "id": 0,
        "start": "string",
        "end": "string",
        "name": "string"
      }
    ]
  }
}
```

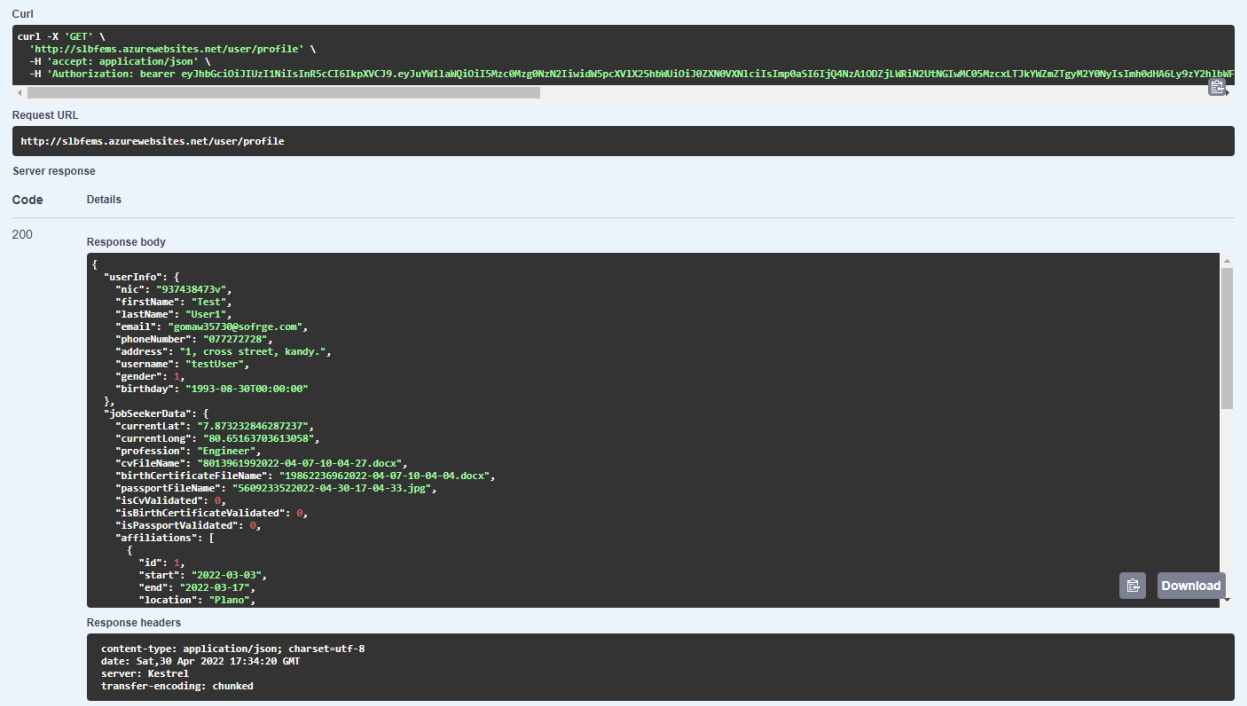


Figure 8.25: Success Response For Get the user's profile

Other responses:

1. Code 401: Unauthorized
2. Code 404: Users not found

8.6.6 GET USER LIST FILTERED BY QUALIFICATIONS

Endpoint - GET: /user/{qualifications}

Admins and Staff members can get the list of users according to the list of qualifications entered, and the items on the list should be separated by a comma(',').

Success response –

```
[
  {
    "userInfo": {
      "nic": "string",
      "firstName": "string",
      "lastName": "string",
      "email": "string",
      "phoneNumber": "string",
      "address": "string",
      "username": "string",
      "gender": 0,
```

```
      "birthday": "2022-04-23T08:01:03.693Z"
    },
    "jobSeekerData": {
      "currentLat": "string",
      "currentLong": "string",
      "profession": "string",
      "cvFileName": "string",
      "birthCertificateFileName": "string",
      "passportFileName": "string",
      "isCvValidated": 0,
      "isBirthCertificateValidated": 0,
      "isPassportValidated": 0,
      "affiliations": [
        {
          "id": 0,
          "start": "string",
          "end": "string",
          "location": "string",
          "organization": "string",
          "title": "string"
        }
      ],
      "qualifications": [
        "string"
      ],
      "education": [
        {
          "id": 0,
          "start": "string",
          "end": "string",
          "name": "string"
        }
      ]
    }
  }
]
```

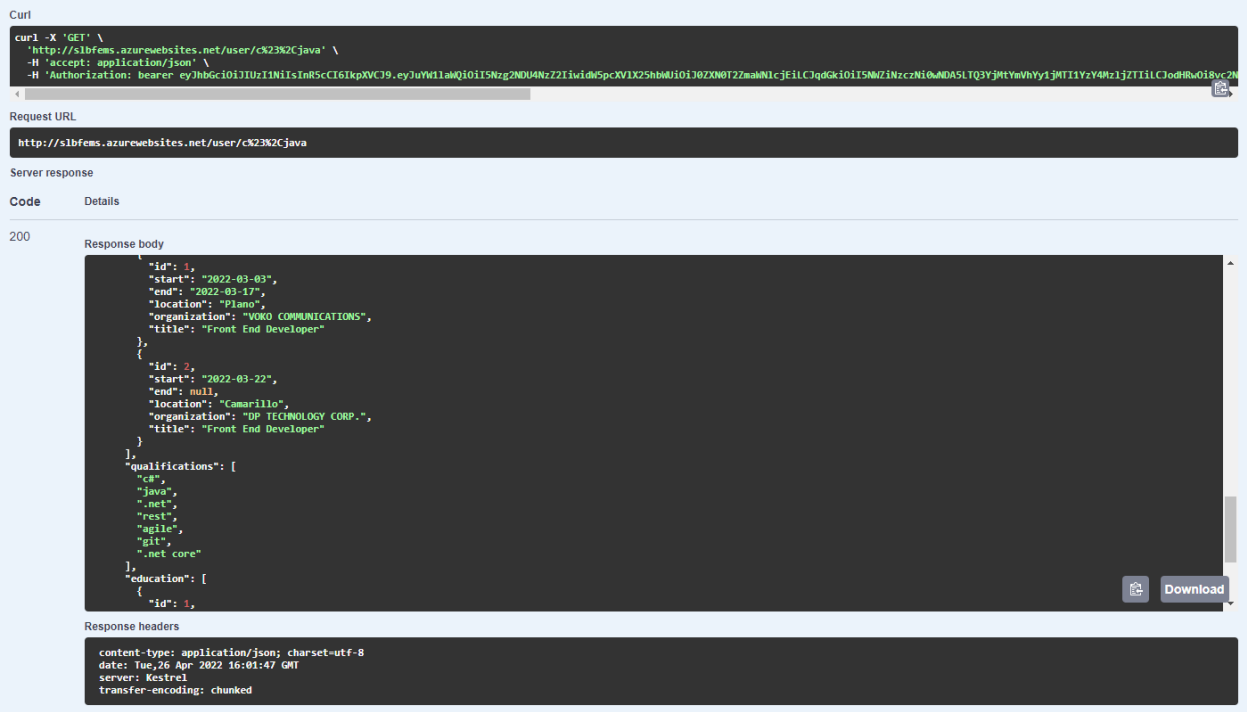


Figure 8.26: Success Response For Get User List Filtered by Qualifications

Other responses:

1. Code 401: Unauthorized
2. Code 403: Forbidden
3. Code 404: N0 users found

8.7 LOGS

8.7.1 GET INFORMATION LOGS LIST

Endpoint - GET: /logs/information

Get information logs list

Success response –

```
{
  "message": "string",
  "timeStamp": "string",
  "exception": "string"
}
```

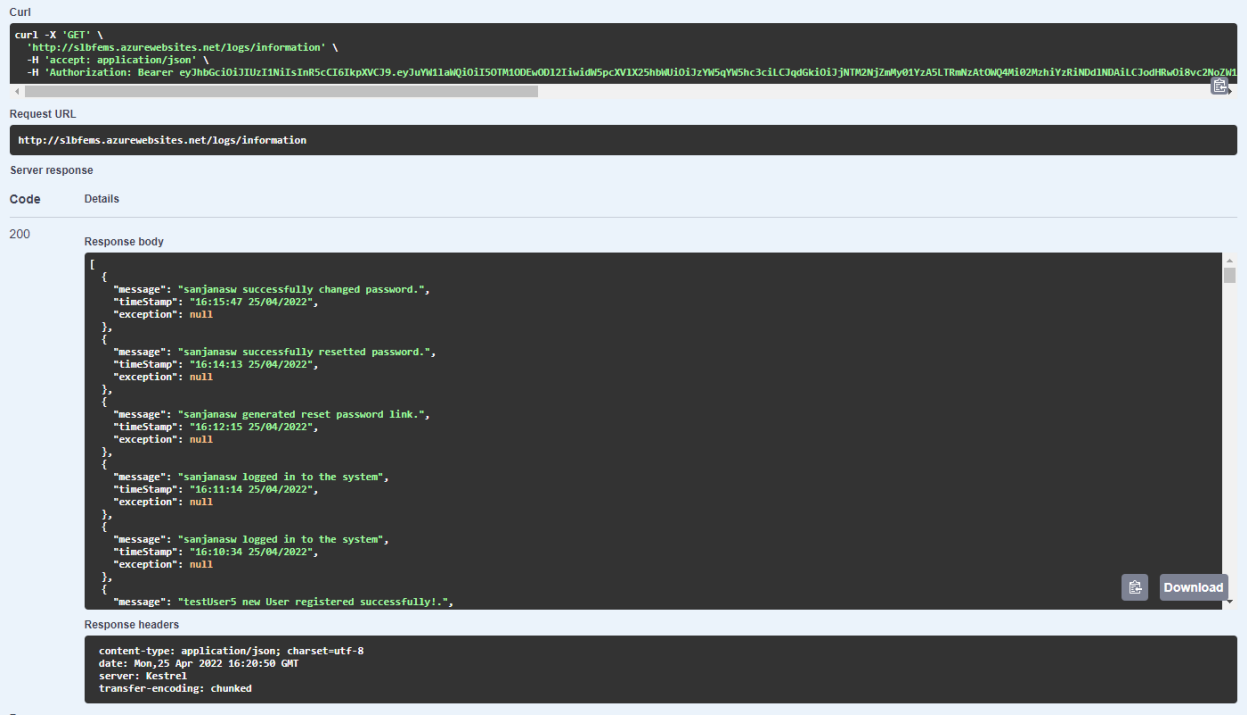


Figure 8.27: Success Response For Get Information Logs List

Other responses:

1. Code 401: Unauthorized
2. Code 403: Forbidden

8.7.2 GET WARNING LOGS LIST

Endpoint - GET: /logs/warning

Get warning logs list

Success response –

```
{
  "message": "string",
  "timeStamp": "string",
  "exception": "string"
}
```

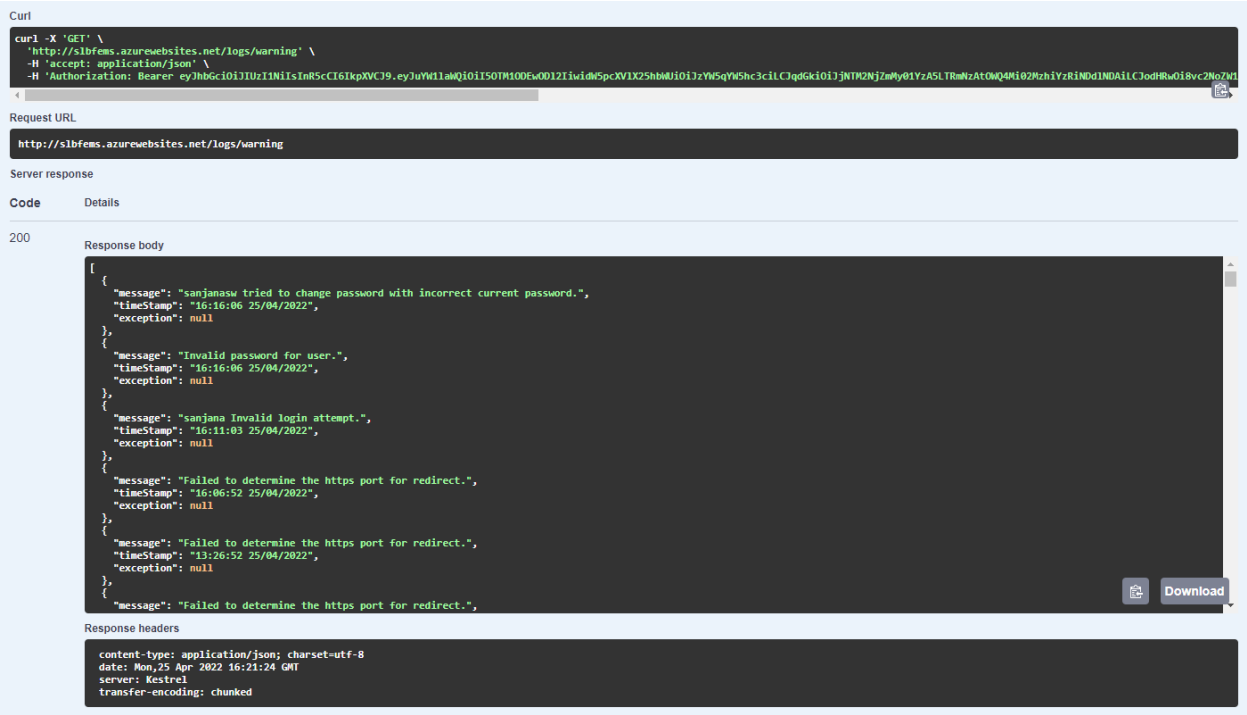


Figure 8.28: Success Response For Get Warning Logs List

Other responses:

1. Code 401: Unauthorized
2. Code 403: Forbidden

8.7.3 GET ERROR LOGS LIST

Endpoint - GET: /logs/error

Get error logs list

Success response –

```
{
  "message": "string",
  "timeStamp": "string",
  "exception": "string"
}
```

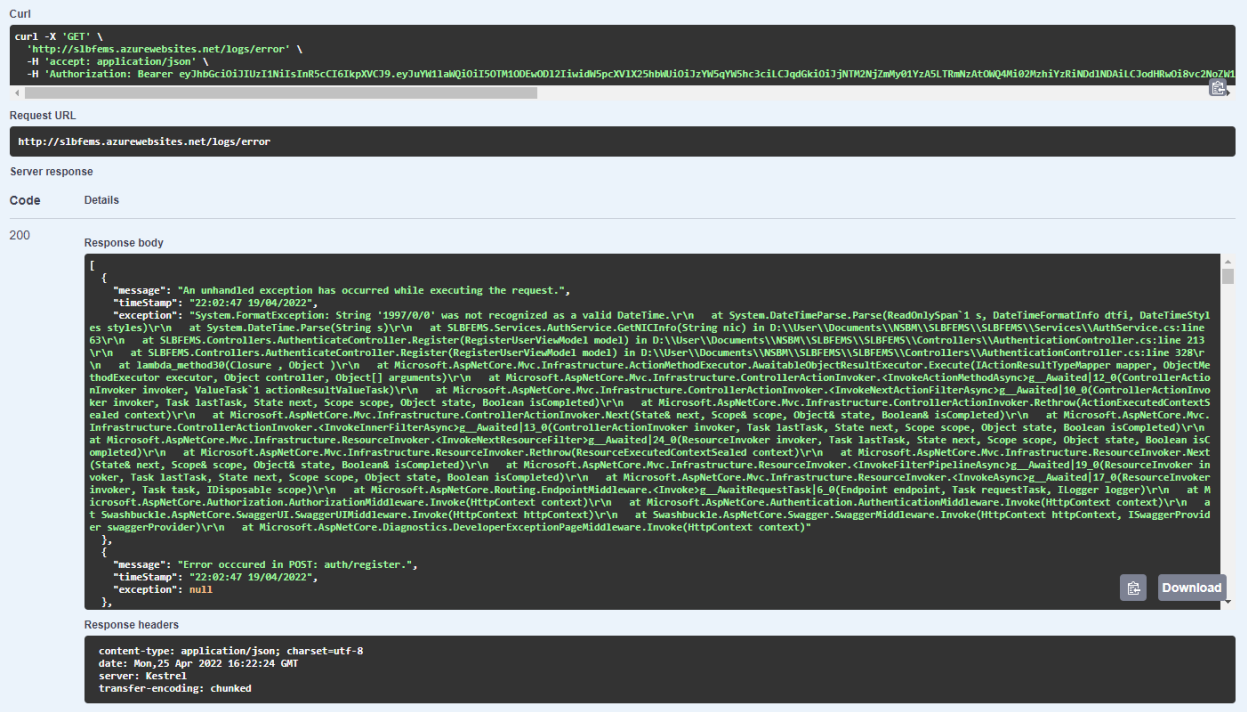


Figure 8.29: Success Response For Get Error Logs List

Other responses:

1. Code 401: Unauthorized
2. Code 403: Forbidden

9 CONTRIBUTION

9.1 SANJANA

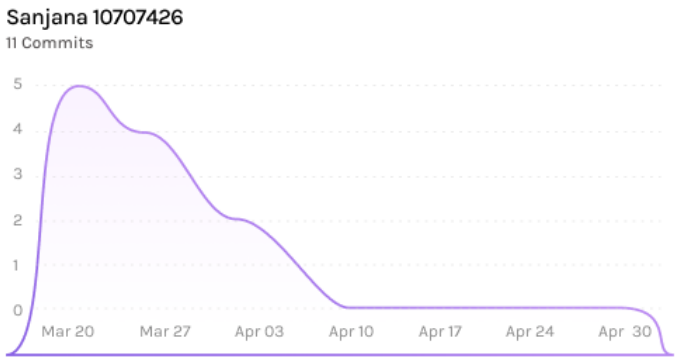


Figure 9.1: Sanjana - Contribution Graph

Endpoints

POST: /auth/login

- Enter your username and password into the fields given below to Login to the system, which you entered in the registration phase according to the validations

POST: /auth/register

- You can Register to the system by entering your NIC number, a username, your first name, your last name, a proper email address, your phone number, gender, your birthday, your address and a proper password which does not fail against the validations included for the password (at least eight characters, At least one letter in Uppercase, At least one letter in Lowercase and At Least one special character)

POST: /auth/confirm-email

- After registering to the system, a confirmation email is sent to the email you entered in the registration form. You have to Confirm the email by clicking the confirm button before the link expires (the link will expire after 24 hours).

POST: /auth/forgot-password

- If you forgot your password, you can go for the resetting by going through the link sent to your email. For that, you have to enter your email, which you entered in the registration process.

PUT: /auth/reset-password

- You can reset your password by entering a new password for your account as you entered your email to get the reset password link.

PUT: /auth/change-password

- You can change your current password to a new password by entering the current password and new password in the given two fields

POST: /auth/force-onboard

- Admin and Officers can add new users to the system by entering the user's NIC number, first name, last name, email, phone number, address, username and user's role into the relevant fields.

POST: /auth/new-user-setup

- If the user is added to the system by an Admin or a Staff member, the user can add a password to the account by going through the email sent to the user's email.

POST: /cv-parser

- Users can upload pdf/word cv and get categorized cv data in text format using Natural Language Processing. Then no need to add those data manually to the system.

Services & interfaces

AuthService
CvParserService

ViewModels

RegisterViewModel
NewUserViewModel
LoginViewModel
LoginResponseViewModel
ForgotPasswordViewModel
ConfirmEmailViewModel
ChangePasswordViewModel
NICInfoViewModel
ResetPasswordViewModel
CvParserExternalResponseViewModel
CvParserResponseViewModel
CvResponseViewModel

Overview

Worked on authentication-related end-points and cv parser functionality. In the CV parser, the user can upload pdf/word cv and get categorized cv data in text format using Natural Language Processing. Refer to xx figure. In user authentication and authorization, I have used JSON web token to do that. All end-points integrated with angular web application and flutter mobile application. And deployed API And Web Application on Azure App service and created CI/CD pipelines to boost the development process, and also created a discord notification channel to notify developers

Figure 9.2: Sanjana - LinkedIn Certificate

9.2 MANUJA

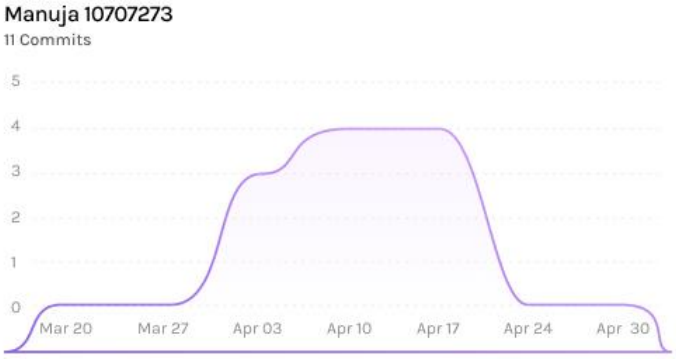


Figure 9.3: Manuja - Contribution Graph

Endpoints
GET: /user/{qualifications}

- Admins and Staff members can get the list of users according to the list of qualifications entered, and the items on the list should be separated by a comma(',').

GET: /user

- Admins and Staff members can retrieve the NIC, address, email, first name, last name, username, phone number, birthday and gender of the list of users which regard to the entering role. If NIC is also entered, the address, email, first name, last name, username, phone number, birthday and gender of the owner of the NIC number are retrieved. If no role is selected, the role is taken as the user.

PUT: /user/{nic}

- Option to Update the NIC, address, email, first name, last name, username, phone number, birthday and gender

DELETE: /user/{nic}

- Admin can Delete the user by entering the NIC of the user

PUT: /user/verify-document/{nic}/{type}

- Staff members can verify uploaded documents and determine if they comply with the requirements.

GET: /user/profile

- Returns data according to the user's authorization token(JSON Web Token).

Services & Interfaces
User Service

ViewModels
UserUpdateViewModel
UserViewModel

Overview
Worked with an end-point where Admins and staff members can retrieve a list of users based on the qualifications filter, with each item separated by a comma. The qualification filtering end-point works with multiple qualifications at a single time. Also, Worked with user data, such as retrieving, updating and deleting users. Also added a Database connection to the system with Entity Framework.

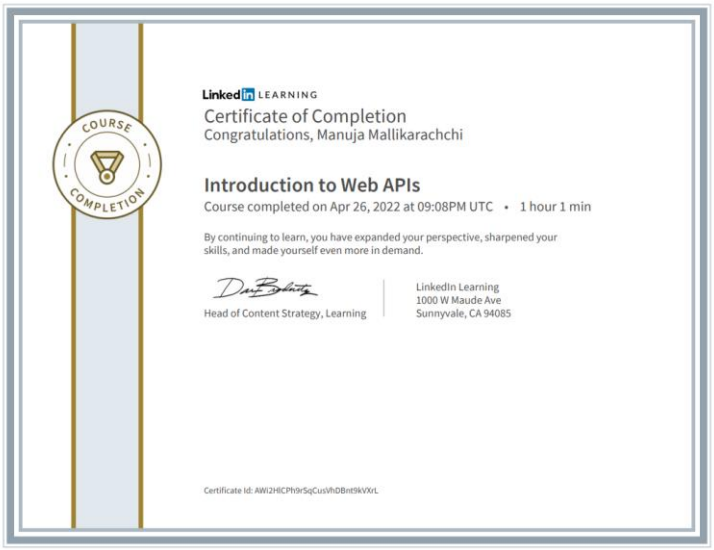


Figure 9.4: Manuja - LinkedIn Certificate

9.3 KRISHAN

Figure 9.5: Krishan - Contribution Graph

Endpoints
GET: /file-manager/{type}

- Provide a method for retrieving the uploaded file by selecting the category first, followed by the file name.

DELETE: /file-manager/{type}

- By passing the file type and name to the delete end-point, we can delete any file that is stored in the system

GET: /file-manager/url/{type}

- End-point that returns the file's URL when the file category is specified in the request URL.

PUT: /job-seekers/{nic}

- Users can update NIC, Address, Email, FirstName, LastName, Phone Number, and Username by entering the current NIC in the system.

POST: /job-seekers/file-upload/{type}

- An End-point that enables users to upload files. First, the user is required to choose the type of file to upload, and then the user may begin uploading documents.

Services & interfaces
FileManagerService

ViewModels
FileDownloadViewModel
FileUploadResponseViewModel
JobSeekerDataViewModel

Overview
Worked on file management end-points, which enable users to upload files to cloud storage in accordance with specified specifications and later retrieve the uploaded files or delete files by providing the file name. I have used Azure blob storage as cloud storage because we are using

Azure services for better performances and integration. Also worked with job seeker end-points where a job seeker can update their personal information

All end-points integrated with angular web application and flutter mobile application.
The token interceptor was added to the web application to inject JWT into each HTTP request. It will assist the API in authorizing users to access end-points.
Also added Azure blob storage connection to the system with the help of Azure NuGet package

Figure 9.6: Krishan- LinkedIn Certificate

9.4 KAVINDYA

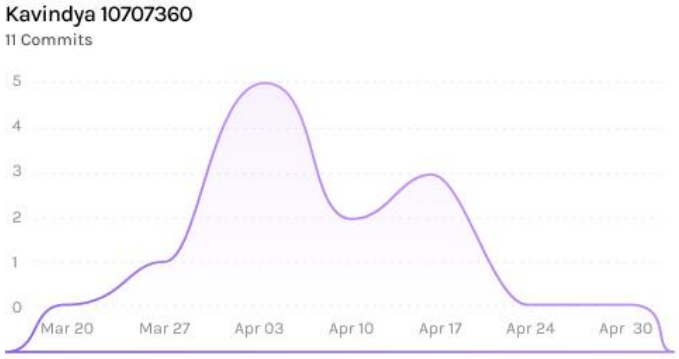


Figure 9.7: Kavindya - Contribution Graph

End-points
GET: /complaints

- By passing the Id of the complaint, job searchers and administrators can obtain a list of complaints along with the sender's NIC. If no identification is presented, all complaints are returned.

POST: /complaints

- Users can submit a complaint by inputting the message and title into the system. The system obtains the NIC of the currently logged-in user.

PUT: /complaints/{id}

- This end-point can be used by a user to add an additional comment to the respective complaint thread.

GET: /logs/informations

- Get information logs with user details and timestamps

GET: /logs/warning

- Get warning logs with user details and timestamps

GET: /logs/error

- Get error logs from the error table with exception details

Service & Interfaces
EmailService

ViewModels
ComplaintCreateViewModel
ComplaintUpdateViewModel
ComplaintViewModel
LogViewModel

Overview

Worked on a complaints management system that job seekers can raise complaints and give admins an option in resolving the pending complaints. And the system is built in a way to handle complaint message threads as well. Refer to figure xxx. And configured system logs using the Serilog package to write all system logs on the database. And also Configured Swagger documentation with the swagger NuGet package. All end-points integrated with angular web application and flutter mobile application.

Figure 9.8: Kavindya - LinkedIn Certificate

REFERENCES

Angular (2022). *Angular*. [online] angular.io. Available at: https://angular.io/guide/what-is-angular.

Codaffection (2018). *Angular 5 Login and Logout with Web API Using Token Based Authentication - CodAffection*. [online] www.codaffection.com. Available at: https://www.codaffection.com/angular-article/angular-5-login-and-logout-with-web-api-using-token-based-authentication/.

Flutter Developers & Contributors (n.d.). *Flutter documentation*. [online] docs.flutter.dev. Available at: https://docs.flutter.dev.

Mapbox (n.d.). *Mapbox | Company*. [online] www.mapbox.com. Available at: https://www.mapbox.com/about/company/.

Mapbox Developers (n.d.). *Documentation*. [online] Mapbox. Available at: https://docs.mapbox.com [Accessed 3 May 2022].

Microsoft (2016). *Visual Studio Code*. [online] Visualstudio.com. Available at: https://code.visualstudio.com/.

Microsoft Azure (n.d.). *What is Middleware - Definition and Examples | Microsoft Azure*. [online] azure.microsoft.com. Available at: https://azure.microsoft.com/en-in/overview/what-is-middleware/.

Shah, K. (2021). *ASP.NET Core 5.0 Web API*. [online] www.c-sharpcorner.com. Available at: https://www.c-sharpcorner.com/article/asp-net-core-5-0-web-api/#:~:text=In%20simple%20words%2C%20we%20can [Accessed 3 May 2022].

Wikipedia Contributors (2019). *Flutter (software)*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Flutter_(software).

Wikipedia, Wikimedia Foundation (2022). *Microsoft SQL Server*. [online] www.wikiwand.com. Available at: https://en.wikipedia.org/wiki/Microsoft_SQL_Server.

APPENDIX

AUTHENTICATION CONTROLLER

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.IdentityModel.Tokens.Jwt;
4.  using System.Security.Claims;
5.  using System.Text;
6.  using System.Threading.Tasks;
7.  using System.Linq;
8.  using System.Net.Mime;
9.  using Microsoft.AspNetCore.Http;
10. using Microsoft.AspNetCore.Identity;
11. using Microsoft.AspNetCore.Mvc;
12. using Microsoft.Extensions.Configuration;
13. using Microsoft.IdentityModel.Tokens;
14. using Microsoft.AspNetCore.Authentication;
15. using Microsoft.AspNetCore.Authorization;
16. using Microsoft.Extensions.Logging;
17. using SLBFEMS.Enums;
18. using SLBFEMS.Models;
19. using SLBFEMS.ViewModels.Authentication;
20. using SLBFEMS.Interfaces;
21. using SLBFEMS.ViewModels.User;
22.
23. namespace SLBFEMS.Controllers
24. {
25.     [Route("auth")]
26.     [ApiController]
27.     [Produces(MediaTypeNames.Application.Json, MediaTypeNames.Application.Xml)]
28.     [Consumes(MediaTypeNames.Application.Json, MediaTypeNames.Application.Xml)]
29.     public class AuthenticateController : ControllerBase
30.     {
31.         private readonly UserManager<ApplicationUserModel> _userManager;
32.         private readonly RoleManager<IdentityRole> _roleManager;
33.         private readonly ApplicationDbContext _context;
34.         private readonly IConfiguration _configuration;
35.         private readonly ILogger<AuthenticateController> _logger;
36.         private readonly IAuthService _authService;
37.
38.         public AuthenticateController(UserManager<ApplicationUserModel> userManager,
    RoleManager<IdentityRole> roleManager, ApplicationDbContext context,
39.             IConfiguration configuration, ILogger<AuthenticateController> logger,
    IAuthService authService)
40.         {
41.             _userManager = userManager;
42.             _roleManager = roleManager;
43.             _context = context;
44.             _configuration = configuration;
45.             _logger = logger;
46.             _authService = authService;
47.         }
48.
49.         /// <summary>
50.         /// Login to the system
51.         /// </summary>
52.         /// <remarks>
53.         /// Sample request:
54.         ///
55.         ///     POST /auth/login
56.         ///     {
57.         ///        "userName": "sanjana",
```

```
58.         ///         "password": "$Sanjana1"
59.         ///     }
60.         ///
61.         /// Enter your username and password into the fields given below to Login to the
    system, which you entered in the registration phase according to the validations
62.         /// </remarks>
63.         /// <response code="200">Returns user data with JWT</response>
64.         /// <response code="401">Unothorized user</response>
65.         /// <response code="403">User doesn't have access to this endpoint</response>
66.         [HttpPost]
67.         [Route("login")]
68.         public async Task<ActionResult<LoginResponseViewModel>> Login([FromBody]
    LoginViewModel model)
69.         {
70.             try
71.             {
72.                 var user = await _userManager.FindByNameAsync(model.UserName);
73.                 if (user != null && !user.DeleteStatus && await
    _userManager.CheckPasswordAsync(user, model.Password))
74.                 {
75.                     if (!user.EmailConfirmed)
76.                     {
77.                         _logger.LogWarning(string.Format("{0} is tried loggin to the system
    with out confirming email.", user.UserName));
78.                         return StatusCode(StatusCodes.Status403Forbidden, new ResponseModel
    { Status = "Error", Message = "Please verify your email!" });
79.                     }
80.
81.                     if (user.DeleteStatus)
82.                     {
83.                         _logger.LogWarning(string.Format("{0} deleted user is tried loggin
    to the system.", user.UserName));
84.                         return StatusCode(StatusCodes.Status403Forbidden, new ResponseModel
    { Status = "Error", Message = "Your account is deleted by admins. Please contact us ASAP!"
    });
85.                     }
86.
87.                     var userRoles = await _userManager.GetRolesAsync(user);
88.
89.                     var authClaims = new List<Claim>
90.                 {
91.                     new Claim(JwtRegisteredClaimNames.NameId, user.NIC),
92.                     new Claim(JwtRegisteredClaimNames.UniqueName, user.UserName),
93.                     new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
94.                 };
95.
96.                     foreach (var userRole in userRoles)
97.                     {
98.                         authClaims.Add(new Claim(ClaimTypes.Role, userRole));
99.                     }
100.
101.                      var authSigningKey = new
    SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["JWT:Secret"]));
102.
103.                     var token = new JwtSecurityToken(
104.                         issuer: _configuration["JWT:ValidIssuer"],
105.                         audience: _configuration["JWT:ValidAudience"],
106.                         expires: DateTime.Now.AddHours(3),
107.                         claims: authClaims,
108.                         signingCredentials: new SigningCredentials(authSigningKey,
    SecurityAlgorithms.HmacSha256)
109.                      );
110.
111.
```

```
                    _logger.LogInformation(string.Format("{0} logged in to the system", user.UserName));
                    return Ok(new LoginResponseViewModel
                    {
                        Token = new JwtSecurityTokenHandler().WriteToken(token),
                        Id = user.Id,
                        NIC = user.NIC,
                        Name = user.FirstName + ' ' + user.LastName,
                        Username = user.UserName,
                        Email = user.Email,
                        Role = (List<string>)userRoles,
                        expiration = DateTime.Now.AddDays(3)
                    });
                }
                _logger.LogWarning(string.Format("{0} Invalid login attempt.", model.UserName));
                return Unauthorized(new ResponseModel { Status = "Unauthorized", Message = "Username or password incorrect!" });
            }
            catch (Exception ex)
            {
                _logger.LogError("Error occcured in POST: auth/login.", ex.Message);
                throw;
            }
        }

        /// <summary>
        /// Register as a new user
        /// </summary>
        /// <remarks>
        /// Sample request:
        ///
        ///     POST /auth/register
        ///
        ///     "userInfo": {
        ///         "nic": "993581089v,
        ///         "firstName": "sanjana",
        ///         "lastName": "sulakshana",
        ///         "email": "sanajnasw@gmail.com",
        ///         "phoneNumber": "0771994147",
        ///         "address": "12/1, jayalanka mawatha, ampitiya.",
        ///         "username": "sanjanasw",
        ///         "password": "$Sanjan1",
        ///         "role": 0
        ///         },
        ///     "jobSeekerData"?: {
        ///         "currentLat": "1.232323",
        ///         "currentLong": "2.32323",
        ///         "profession": "Software Engineer",
        ///         "affiliations": [
        ///           {
        ///             "start": "jan 2021",
        ///             "end": "feb 2021",
        ///             "location": "kandy",
        ///             "organization": "built apps",
        ///             "title": "Associate Software Engineer"
        ///           }
        ///          ],
        ///          "education": [
        ///           {
        ///             "start": "jan 2021",
        ///             "end": "feb 2021",
        ///             "name": "Dharmaraja college"
        ///           }
        ///          ],
```

```
174.          ///         "qualifications": [
175.          ///           "java",
176.          ///           "Angular",
177.          ///           "SQL"
178.          ///          ],
179.          ///         "cvFileName": "7917212142022-03-07-23-03-37.docx"
180.          ///        }
181.          ///       }
182.          ///
183.          /// You can Register to the system by entering your NIC number, a username, your first name, your last name, a proper email address, your phone number, gender, your birthday, your address and a proper password which does not fail against the validations included for password(at least 8 characters, Atleast 1 letter in Uppercase, Atleast 1 letter in Lowercase and Atleast 1 special character)
184.          /// </remarks>
185.          /// <response code="200">Returns success message</response>
186.          /// <response code="400">Job seeker register request without providing job seeker data</response>
187.          /// <response code="409">Username or email already exists</response>
188.          /// <response code="500">Internal server error</response>
189.          [HttpPost]
190.          [Route("register")]
191.          public async Task<ActionResult<ResponseModel>> Register([FromBody] RegisterUserViewModel model)
192.          {
193.              using (var transaction = await _context.Database.BeginTransactionAsync())
194.              {
195.                  try
196.                  {
197.                      if (model.UserInfo.Role == UserRoles.User && model.JobSeekerData == null)
198.                      {
199.                          return BadRequest(new ResponseModel { Status = "Error", Message = "Job seekers must provide jobseekers data" });
200.                      }
201.                      var userExists = await _userManager.FindByNameAsync(model.UserInfo.Username);
202.                      var userEmailExists = await _userManager.FindByEmailAsync(model.UserInfo.Email);
203.                      if (userExists != null)
204.                      {
205.                          return StatusCode(StatusCodes.Status409Conflict, new ResponseModel { Status = "Error", Message = "Username is already exists!" });
206.                      }
207.                      else if (userEmailExists != null)
208.                      {
209.                          _logger.LogWarning(string.Format("{0} is tried to create another account.", model.UserInfo.Email));
210.                          return StatusCode(StatusCodes.Status409Conflict, new ResponseModel { Status = "Error", Message = "Email is already exists!" });
211.                      }
212.
213.                      var NICInfo = _authService.GetNICInfo(model.UserInfo.NIC);
214.
215.                      ApplicationUserModel user = new()
216.                      {
217.                          NIC = model.UserInfo.NIC,
218.                          UserName = model.UserInfo.Username,
219.                          FirstName = model.UserInfo.FirstName,
220.                          LastName = model.UserInfo.LastName,
221.                          Email = model.UserInfo.Email,
222.                          PhoneNumber = model.UserInfo.PhoneNumber,
223.                          Gender = NICInfo.Gender,
224.                          Birthday = NICInfo.Birthday,
225.                          Address = model.UserInfo.Address,
```

```
226.                         SecurityStamp = Guid.NewGuid().ToString(),
227.                     };
228.                     var result = await _userManager.CreateAsync(user,
    model.UserInfo.Password);
229.
230.                     if (!result.Succeeded)
231.                         return StatusCode(StatusCodes.Status500InternalServerError, new
    ResponseModel { Status = "Error", Message = "User creation failed! Please check user details
    and try again." });
232.
233.                     await RefreshRolesTableAsync();
234.
235.                     if (!string.IsNullOrEmpty(model.UserInfo.Role.ToString()) &&
    model.UserInfo.Role == UserRoles.User)
236.                     {
237.
238.                         if (await _roleManager.RoleExistsAsync(UserRoles.User.ToString()))
239.                         {
240.                             await _userManager.AddToRoleAsync(user,
    UserRoles.User.ToString());
241.                         }
242.                     }
243.                     else if (!string.IsNullOrEmpty(model.UserInfo.Role.ToString()) &&
    model.UserInfo.Role == UserRoles.Admin)
244.                     {
245.
246.                         if (await
    _roleManager.RoleExistsAsync(UserRoles.Admin.ToString()))
247.                         {
248.                             await _userManager.AddToRoleAsync(user,
    UserRoles.Admin.ToString());
249.                         }
250.                     }
251.                     else if (!string.IsNullOrEmpty(model.UserInfo.Role.ToString()) &&
    model.UserInfo.Role == UserRoles.Officer)
252.                     {
253.
254.                         if (await
    _roleManager.RoleExistsAsync(UserRoles.Officer.ToString()))
255.                         {
256.                             await _userManager.AddToRoleAsync(user,
    UserRoles.Officer.ToString());
257.                         }
258.                     }
259.
260.                     if (model.UserInfo.Role == UserRoles.User)
261.                     {
262.                         await _context.JobSeekerData.AddAsync(new JobSeekerDataModel
263.                         {
264.                             NIC = model.UserInfo.NIC,
265.                             CurrentLat = model.JobSeekerData.CurrentLat,
266.                             CurrentLong = model.JobSeekerData.CurrentLong,
267.                             Profession = model.JobSeekerData.Profession,
268.                             CvFileName = model.JobSeekerData.CvFileName
269.
270.                         });
271.
272.                         if (model.JobSeekerData.Affiliations.Count > 0)
273.                         {
274.                             foreach (var affilicate in model.JobSeekerData.Affiliations)
275.                             {
276.                                 await _context.AffiliationData.AddAsync(new
    AffiliationDataModel
277.                                 {
278.                                     NIC = model.UserInfo.NIC,
```

```
                                    Start = affilicate.Start,
                                    End = affilicate.End,
                                    Title = affilicate.Title,
                                    Location = affilicate.Location,
                                    Organization = affilicate.Organization,
                                });
                            }
                        }

                        if (model.JobSeekerData.Qualifications.Count > 0)
                        {
                            foreach (var qualification in
    model.JobSeekerData.Qualifications)
                            {
                                await _context.Qualifications.AddAsync(new
    QualificationsModel
                                {
                                    NIC = model.UserInfo.NIC,
                                    Qualification = qualification,
                                });
                            }
                        }

                        if (model.JobSeekerData.Education.Count > 0)
                        {
                            foreach (var education in model.JobSeekerData.Education)
                            {
                                await _context.EducationData.AddAsync(new
    EducationDataModel
                                {
                                    NIC = model.UserInfo.NIC,
                                    Start = education.Start,
                                    End = education.End,
                                    Name = education.Name
                                });
                            }
                        }
                    }

                    _context.SaveChanges();
                    await transaction.CommitAsync();


                    string confirmationToken = await
    _userManager.GenerateEmailConfirmationTokenAsync(user);
                    _authService.SendEmail("verify", null, user, confirmationToken);
                    _logger.LogInformation(string.Format("{0} new {1} registered
    successfully!.", user.UserName, model.UserInfo.Role.ToString()));
                    return Ok(new ResponseModel { Status = "Success", Message = "User
    created successfully!" });
                }
                catch (Exception ex)
                {
                    await transaction.RollbackAsync();
                    _logger.LogError("Error occcured in POST: auth/register.",
    ex.Message);
                    throw;
                }
            }
        }

        /// <summary>
        /// Confirm email
        /// </summary>
        /// <remarks>
```

```
337.          /// Sample request:
338.          ///
339.          ///     POST /auth/confirm-email
340.          ///     {
341.          ///         "userid": "gfuie-8feiufb-reufberf-rei",
342.          ///         "token": "kjufbkjdfuirefu8h4r94ruiuwb38dbnie844bu44bi"
343.          ///     }
344.          ///
345.          /// After registering to the system, a confirmation email is sending to the email
    you entered in the registration form. You have to Confirm the email by clicking the confirm
    button before the link expired(link will expired after 24 hours).
346.          /// </remarks>
347.          /// <response code="200">Returns success message</response>
348.          /// <response code="400">Invalid token</response>
349.          /// <response code="404">User not found</response>
350.          [HttpPost("confirm-email")]
351.          public async Task<ActionResult<ResponseModel>> ConfirmEmail([FromBody]
    ConfirmEmailViewModel model)
352.          {
353.              try
354.              {
355.                  ApplicationUserModel user = await
    _userManager.FindByIdAsync(model.UserId);
356.
357.                  if (user == null)
358.                  {
359.                      return StatusCode(StatusCodes.Status404NotFound, new ResponseModel {
    Status = "Error", Message = "User Not Found!" });
360.                  }
361.
362.                  IdentityResult result = await _userManager.ConfirmEmailAsync(user,
    model.Token);
363.                  if (!result.Succeeded)
364.                  {
365.                      _logger.LogWarning(string.Format("{0} is tried to comfirm email with
    invalid token.", user.UserName));
366.                      return StatusCode(StatusCodes.Status400BadRequest, new ResponseModel {
    Status = "Error", Message = "Token Invalid!" });
367.                  }
368.
369.                  _authService.SendEmail("verified", user.Email, null, null);
370.                  _logger.LogInformation(string.Format("{0} successfully confirmed email!",
    user.UserName));
371.                  return Ok(new ResponseModel { Status = "Success", Message = "Verification
    successful, you can now login" });
372.              }
373.              catch (Exception ex)
374.              {
375.                  _logger.LogError("Error occcured in POST: auth/confirm-email",
    ex.Message);
376.                  throw;
377.              }
378.          }
379.
380.          /// <summary>
381.          /// Forgot password
382.          /// </summary>
383.          /// <remarks>
384.          /// Sample request:
385.          ///
386.          ///     POST /auth/forgot-password
387.          ///     {
388.          ///         "email": "sanjanasw99@gmail.com"
389.          ///     }
```

```
390.         /// If you forgot your password, you can go for the resetting by going through the
    link send to your email. For that, you have to enter your email which you entered in the
    registration process.
391.         /// </remarks>
392.         /// <response code="200">Returns success message</response>
393.         /// <response code="404">User not found</response>
394.         [HttpPost("forgot-Password")]
395.         public async Task<ActionResult<ResponseModel>> ForgotPassword([FromBody]
    ForgetPasswordViewModel model)
396.         {
397.             try
398.             {
399.                 ApplicationUserModel user = await
    _userManager.FindByEmailAsync(model.Email);
400.
401.                 if (user == null || !(await _userManager.IsEmailConfirmedAsync(user)))
402.                 {
403.                     return StatusCode(StatusCodes.Status404NotFound, new ResponseModel {
    Status = "Error", Message = "User Not Found!" });
404.                 }
405.
406.                 var token = await _userManager.GeneratePasswordResetTokenAsync(user);
407.                 _authService.SendEmail("resetPass", null, user, token);
408.                 _logger.LogInformation(string.Format("{0} generated reset password link.",
    user.UserName));
409.                 return Ok(new ResponseModel { Status = "Success", Message = "Reset
    Password Link Sent!" });
410.             }
411.             catch (Exception ex)
412.             {
413.                 _logger.LogError("Error occcured in POST: auth/forgot-password.",
    ex.Message);
414.                 throw;
415.             }
416.         }
417.
418.         /// <summary>
419.         /// Reset password
420.         /// </summary>
421.         /// <remarks>
422.         /// Sample request:
423.         ///
424.         ///     POST /auth/reset-password
425.         ///     {
426.         ///         "userid": "gfuie-8feiufb-reufberf-rei",
427.         ///         "token": "kjufbkjdfuirefu8h4r94ruiuwb38dbnie844bu44bi",
428.         ///         "password": "Not@1234"
429.         ///     }
430.         ///
431.         /// You can reset your password by entering a new password for your account as you
    entered your email to get the reset password link.
432.         /// </remarks>
433.         /// <response code="200">Returns success message</response>
434.         /// <response code="400">Invalid token</response>
435.         /// <response code="404">User not found</response>
436.         [HttpPut("reset-Password")]
437.         public async Task<ActionResult<ResponseModel>> ResetPassword([FromBody]
    ResetPasswordViewModel model)
438.         {
439.             try
440.             {
441.                 ApplicationUserModel user = await
    _userManager.FindByIdAsync(model.Userid);
442.
443.                 if (user == null)
```

```
444.                 {
445.                     return StatusCode(StatusCodes.Status404NotFound, new ResponseModel {
    Status = "Error", Message = "User Not Found!" });
446.                 }
447.
448.                 IdentityResult result = await _userManager.ResetPasswordAsync(user,
    model.Token, model.Password);
449.
450.                 if (!result.Succeeded)
451.                 {
452.                     _logger.LogWarning(string.Format("{0} is tried to reset password with
    invalid token.", user.UserName));
453.                     return StatusCode(StatusCodes.Status400BadRequest, new ResponseModel {
    Status = "Error", Message = "Token Invalid!" });
454.                 }
455.
456.                 _authService.SendEmail("resetted", user.Email, null, null);
457.                 _logger.LogInformation(string.Format("{0} successfully resetted
    password.", user.UserName));
458.                 return Ok(new ResponseModel { Status = "Success", Message = "Password
    Reset Successfull!" });
459.             }
460.             catch (Exception ex)
461.             {
462.                 _logger.LogError("Error occcured in PUT: auth/reset-password",
    ex.Message);
463.                 throw;
464.             }
465.         }
466.
467.         /// <summary>
468.         /// Change password
469.         /// </summary>
470.         /// <remarks>
471.         /// Sample request:
472.         ///
473.         ///     PUT /auth/change-password
474.         ///     {
475.         ///         "currentPassword": "Not@1234",
476.         ///         "newPassword": "1234@Not"
477.         ///     }
478.         ///
479.         /// You can change your current password to a new password by entering the current
    password and new password in the given two fields
480.         /// </remarks>
481.         /// <response code="200">Returns success message</response>
482.         /// <response code="400">Password doesn't meet minimum requirements</response>
483.         /// <response code="403">Current password incorrect</response>
484.         /// <response code="404">User not found</response>
485.         [Authorize]
486.         [HttpPut("change-password")]
487.         public async Task<ActionResult<ResponseModel>> ChangePassword([FromBody]
    ChangePasswordViewModel model)
488.         {
489.             try
490.             {
491.                 var accessToken = await HttpContext.GetTokenAsync("access_token");
492.                 var token = new JwtSecurityTokenHandler().ReadJwtToken(accessToken) as
    JwtSecurityToken;
493.                 var loggedInUsername = token.Claims.First(claim => claim.Type ==
    "unique_name").Value;
494.                 var user = await _userManager.FindByNameAsync(loggedInUsername);
495.
496.                 if (user == null)
497.                 {
```

```
498.                     return StatusCode(StatusCodes.Status404NotFound, new ResponseModel {
    Status = "Error", Message = "User Not Found!" });
499.                 }
500.
501.                 var resetToken = await _userManager.GeneratePasswordResetTokenAsync(user);
502.
503.                 if (!await _userManager.CheckPasswordAsync(user, model.CurrentPassword))
504.                 {
505.                     _logger.LogWarning(string.Format("{0} tried to change password with
    incorrect current password.", user.UserName));
506.                     return StatusCode(StatusCodes.Status403Forbidden, new ResponseModel {
    Status = "Error", Message = "Current password is incorrect!" });
507.                 }
508.
509.                 IdentityResult result = await _userManager.ResetPasswordAsync(user,
    resetToken, model.NewPassword);
510.
511.                 if (!result.Succeeded)
512.                 {
513.                     return StatusCode(StatusCodes.Status400BadRequest, new ResponseModel {
    Status = "Error", Message = "Somethig went wrong!" });
514.                 }
515.
516.                 _authService.SendEmail("passwordChanged", user.Email, null, null);
517.                 _logger.LogInformation(string.Format("{0} successfully changed password.",
    user.UserName));
518.                 return Ok(new ResponseModel { Status = "Success", Message = "Password
    change Successfull!" });
519.             }
520.             catch (Exception ex)
521.             {
522.                 _logger.LogError("Error occcured in PUT: auth/change-password.",
    ex.Message);
523.                 throw;
524.             }
525.         }
526.
527.         /// <summary>
528.         /// Create new admin/officers account. [Access: Admins and Officers only]
529.         /// </summary>
530.         /// <remarks>
531.         /// Sample request:
532.         ///
533.         ///     POST /auth/force-onboard
534.         ///
535.         ///     "userInfo": {
536.         ///         "nic": "string",
537.         ///         "firstName": "string",
538.         ///         "lastName": "string",
539.         ///         "email": "user@example.com",
540.         ///         "phoneNumber": "string",
541.         ///         "address": "string",
542.         ///         "username": "string",
543.         ///         "role": 0
544.         ///         }
545.         ///       }
546.         ///
547.         /// Admin and Officers can add new users to the system by entering the user's NIC
    number, first name, last name, email, phone number, address, username and user's role into
    the relavant fields.
548.         /// </remarks>
549.         /// <response code="200">Returns new user details</response>
550.         /// <response code="400">Invalid user role</response>
551.         /// <response code="403">Forbidden</response>
552.         /// <response code="409">User details conflict</response>
```

```
553.          /// <response code="500">Internal server error</response>
554.          [Authorize(Roles = "Admin, Officers")]
555.          [HttpPost]
556.          [Route("force-onboard")]
557.          public async Task<ActionResult> NewUser([FromBody] NewUserViewModel model)
558.          {
559.              using (var transaction = await _context.Database.BeginTransactionAsync())
560.              {
561.                  try
562.                  {
563.                      if (model.Role == UserRoles.User)
564.                      {
565.                          return BadRequest();
566.                      }
567.
568.                      var accessToken = await HttpContext.GetTokenAsync("access_token");
569.                      var token = new JwtSecurityTokenHandler().ReadJwtToken(accessToken) as
    JwtSecurityToken;
570.                      var loggedInUsername = token.Claims.First(claim => claim.Type ==
    "unique_name").Value;
571.
572.                      var userExists = await _userManager.FindByNameAsync(model.Username);
573.                      var userEmailExists = await
    _userManager.FindByEmailAsync(model.Email);
574.
575.                      if (userExists != null)
576.                      {
577.                          _logger.LogWarning(string.Format("{0} is tried to create force-
    onboard account for existing username: {1}.", loggedInUsername, model.Username));
578.                          return StatusCode(StatusCodes.Status409Conflict, new ResponseModel
    { Status = "Error", Message = "Username is already exists!" });
579.                      }
580.                      else if (userEmailExists != null)
581.                      {
582.                          _logger.LogWarning(string.Format("{0} is tried to create force-
    onboard account for existing email: {1}.", loggedInUsername, model.Email));
583.                          return StatusCode(StatusCodes.Status409Conflict, new ResponseModel
    { Status = "Error", Message = "Email is already exists!" });
584.                      }
585.
586.                      var NICInfo = _authService.GetNICInfo(model.NIC);
587.
588.                      ApplicationUserModel user = new()
589.                      {
590.                          UserName = model.Username,
591.                          FirstName = model.FirstName,
592.                          LastName = model.LastName,
593.                          Email = model.Email,
594.                          NIC = model.NIC,
595.                          Address = model.Address,
596.                          PhoneNumber = model.PhoneNumber.ToString(),
597.                          Birthday = NICInfo.Birthday,
598.                          Gender = NICInfo.Gender,
599.
600.                          SecurityStamp = Guid.NewGuid().ToString(),
601.                      };
602.
603.                      var result = await _userManager.CreateAsync(user,
    "$NewUserPassword1Temp");
604.                      if (!result.Succeeded)
605.                          return StatusCode(StatusCodes.Status500InternalServerError, new
    ResponseModel { Status = "Error", Message = "User creation failed! Please check user details
    and try again." });
606.
607.                      await RefreshRolesTableAsync();
```

```
608.
609.                     if (!string.IsNullOrEmpty(model.Role.ToString()) && model.Role ==
    UserRoles.Admin)
610.                     {
611.
612.                         if (await
    _roleManager.RoleExistsAsync(UserRoles.Admin.ToString()))
613.                         {
614.                             await _userManager.AddToRoleAsync(user,
    UserRoles.Admin.ToString());
615.                         }
616.                     }
617.                     else if (!string.IsNullOrEmpty(model.Role.ToString()) && model.Role ==
    UserRoles.Officer)
618.                     {
619.
620.                         if (await
    _roleManager.RoleExistsAsync(UserRoles.Officer.ToString()))
621.                         {
622.                             await _userManager.AddToRoleAsync(user,
    UserRoles.Officer.ToString());
623.                         }
624.                     }
625.
626.                     var resetToken = await
    _userManager.GeneratePasswordResetTokenAsync(user);
627.                     await transaction.CommitAsync();
628.
629.                     _logger.LogInformation(string.Format("{0}, new user onborded to
    username: {1}", loggedInUsername, user.UserName));
630.                     _authService.SendEmail("newUser", null, user, resetToken);
631.                     var userPersonalInfo = new UserPersonalInfoViewModel
632.                     {
633.                         NIC = user.NIC,
634.                         Address = user.Address,
635.                         Email = user.Email,
636.                         FirstName = user.FirstName,
637.                         LastName = user.LastName,
638.                         PhoneNumber = user.PhoneNumber,
639.                         Username = user.UserName,
640.                         Birthday = user.Birthday,
641.                         Gender = user.Gender
642.                     };
643.                     return Ok(new{ UserInfo = userPersonalInfo });
644.                 }
645.                 catch (Exception ex)
646.                 {
647.                     await transaction.RollbackAsync();
648.                     _logger.LogError("Error occcured in POST: auth/force-onboard.",
    ex.Message);
649.                     throw;
650.                 }
651.             }
652.         }
653.
654.         /// <summary>
655.         /// New user account setup
656.         /// </summary>
657.         /// <remarks>
658.         /// Sample request:
659.         ///
660.         ///     POST /auth/new-user-setup
661.         ///     {
662.         ///         "userid": "gfuie-8feiufb-reufberf-rei",
663.         ///         "token": "kjufbkjdfuirefu8h4r94ruiuwb38dbnie844bu44bi",
```

```
664.          ///         "password": "Not@1234"
665.          ///     }
666.          ///
667.          /// If the user is added to the system by an Admin or a Staff member, user can add
    a password to the account by going through the email sent to the user's email.
668.          /// </remarks>
669.          /// <response code="200">Returns success message</response>
670.          /// <response code="400">Invalid token or password doesn't meet minimum
    requirements</response>
671.          /// <response code="404">User not found</response>
672.          [HttpPost("new-user-setup")]
673.          public async Task<ActionResult<ResponseModel>> NewUserSetup([FromBody]
    ResetPasswordViewModel model)
674.          {
675.              try
676.              {
677.                  ApplicationUserModel user = await
    _userManager.FindByIdAsync(model.Userid);
678.
679.                  if (user == null)
680.                  {
681.                      return StatusCode(StatusCodes.Status404NotFound, new ResponseModel {
    Status = "Error", Message = "User Not Found!" });
682.                  }
683.
684.                  IdentityResult result = await _userManager.ResetPasswordAsync(user,
    model.Token, model.Password);
685.                  user.EmailConfirmed = true;
686.                  await _userManager.UpdateAsync(user);
687.
688.                  if (!result.Succeeded)
689.                  {
690.                      _logger.LogWarning(string.Format("{0} is tried to setup fresh account
    with invalid token.", user.UserName));
691.                      return StatusCode(StatusCodes.Status400BadRequest, new ResponseModel {
    Status = "Error", Message = "Invalid token or password doesn't meet minimum requirements!"
    });
692.                  }
693.
694.                  _authService.SendEmail("newUserSetup", user.Email, null, null);
695.                  _logger.LogInformation(string.Format("{0} is onboarded successfully.",
    user.UserName));
696.                  return Ok(new ResponseModel { Status = "Success", Message = "New User
    Setup Successfull!" });
697.              }
698.              catch (Exception ex)
699.              {
700.                  _logger.LogError("Error occcured in POST: auth/new-user-setup",
    ex.Message);
701.                  throw;
702.              }
703.          }
704.
705.
706.          private async Task RefreshRolesTableAsync()
707.          {
708.              if (!await _roleManager.RoleExistsAsync(UserRoles.Admin.ToString()))
709.                  await _roleManager.CreateAsync(new
    IdentityRole(UserRoles.Admin.ToString()));
710.              if (!await _roleManager.RoleExistsAsync(UserRoles.Officer.ToString()))
711.                  await _roleManager.CreateAsync(new
    IdentityRole(UserRoles.Officer.ToString()));
712.              if (!await _roleManager.RoleExistsAsync(UserRoles.User.ToString()))
713.                  await _roleManager.CreateAsync(new
    IdentityRole(UserRoles.User.ToString()));
```

```
714.          }
715.
716.      }
717.  }
718.
```

USERS CONTROLLER

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.IdentityModel.Tokens.Jwt;
4.  using System.Linq;
5.  using System.Net.Mime;
6.  using System.Security.Claims;
7.  using System.Threading.Tasks;
8.  using Microsoft.AspNetCore.Authentication;
9.  using Microsoft.AspNetCore.Authorization;
10. using Microsoft.AspNetCore.Http;
11. using Microsoft.AspNetCore.Identity;
12. using Microsoft.AspNetCore.Mvc;
13. using Microsoft.EntityFrameworkCore;
14. using Microsoft.Extensions.Logging;
15. using SLBFEMS.Enums;
16. using SLBFEMS.Interfaces;
17. using SLBFEMS.Models;
18. using SLBFEMS.ViewModels.User;
19.
20. namespace SLBFEMS.Controllers
21. {
22.     [Authorize]
23.     [Route("user")]
24.     [ApiController]
25.     [Produces(MediaTypeNames.Application.Json, MediaTypeNames.Application.Xml)]
26.     [Consumes(MediaTypeNames.Application.Json, MediaTypeNames.Application.Xml)]
27.     public class UsersController : ControllerBase
28.     {
29.         private readonly UserManager<ApplicationUserModel> _userManager;
30.         private readonly ApplicationDbContext _context;
31.         private readonly IUserService _userService;
32.         private readonly ILogger<UsersController> _logger;
33.         private readonly IAuthService _authService;
34.
35.         public UsersController(UserManager<ApplicationUserModel> userManager,
    ApplicationDbContext context, IUserService userService, ILogger<UsersController> logger,
    IAuthService authService)
36.         {
37.             _userManager = userManager;
38.             _context = context;
39.             _logger = logger;
40.             _userService = userService;
41.             _authService = authService;
42.         }
43.
44.         /// <summary>
45.         /// Get users details. [Access: Admins and Officers only]
46.         /// </summary>
47.         /// <remarks>
48.         /// Admins and Staff members can retrieve the NIC, address, email, first name, last
    name, username, phone number, birthday and gender of the list of users which regard to the
    entering role. If NIC also entered, the address, email, first name, last name, username,
```

```
    phone number, birthday and gender of the owner of NIC number are retrieved. If no role is
    selected, role is taken as the user.
        /// </remarks>
        /// <response code="200">Returns user's details</response>
        /// <response code="403">Forbidden</response>
        /// <response code="404">Requested user not found</response>
        [Authorize(Roles = "Admin, Officer")]
        [HttpGet]
        public async Task<ActionResult<IEnumerable<UserViewModel>>> GetUsers([FromQuery]
    UserRoles? role, [FromQuery] string nic = null)
        {
            try
            {
                if (nic == null)
                {
                    if (role == null)
                        role = UserRoles.User;
                    var usersInfoList = new List<UserViewModel>();
                    var usersList = await _userManager.GetUsersInRoleAsync(role.ToString());
                    foreach (var item in usersList)
                    {
                        if (item.DeleteStatus == true)
                            continue;
                        var itemJobSeekerData = await
    _userService.GetJobSeekerData(item.NIC);
                        var itemPersonalInfo = new UserPersonalInfoViewModel
                        {
                            NIC = item.NIC,
                            Address = item.Address,
                            Email = item.Email,
                            FirstName = item.FirstName,
                            LastName = item.LastName,
                            PhoneNumber = item.PhoneNumber,
                            Username = item.UserName,
                            Birthday = item.Birthday,
                            Gender = item.Gender,
                        };

                        var itemData = new UserViewModel
                        {
                            UserInfo = itemPersonalInfo,
                            JobSeekerData = itemJobSeekerData
                        };

                        usersInfoList.Add(itemData);
                    }

                    return Ok(usersInfoList);
                }
                else
                {
                    var userInfo = new UserViewModel();
                    var userJobSeekerDataViewModel = new UserJobSeekerDataViewModel();
                    var userData = await _context.Users.Where(x => x.NIC ==
    nic).FirstOrDefaultAsync();
                    if (userData.DeleteStatus == true)
                          return BadRequest();
                      if (role != UserRoles.Admin && role != UserRoles.Officer)
                        userJobSeekerDataViewModel = await
    _userService.GetJobSeekerData(nic);

                      var userPersonalInfo = new UserPersonalInfoViewModel
                      {
                          NIC = nic,
                          Address = userData.Address,
```

```
                        Email = userData.Email,
                        FirstName = userData.FirstName,
                        LastName = userData.LastName,
                        PhoneNumber = userData.PhoneNumber,
                        Username = userData.UserName
                    };

                    userInfo = new UserViewModel
                    {
                        UserInfo = userPersonalInfo,
                        JobSeekerData = userJobSeekerDataViewModel
                    };
                    return Ok(userInfo);
                }

            }
            catch (Exception ex)
            {
                _logger.LogError("Error occcured in GET: user", ex.Message);
                return BadRequest(new ResponseModel { Status = "Error", Message = "NIC validation failed" });
            }
        }

        /// <summary>
        /// Update user info
        /// </summary>
        /// <remarks>
        /// Sample request:
        ///
        ///     PUT /user
        ///
        ///     {
        ///         "nic": "993581089v,
        ///         "firstName": "sanjana",
        ///         "lastName": "sulakshana",
        ///         "email": "sanajnasw@gmail.com",
        ///         "phoneNumber": "0771994147",
        ///         "address": "12/1, jayalanka mawatha, ampitiya.",
        ///         "username": "sanjanasw",
        ///       }
        ///
        /// You can Update the NIC, address, email, first name, last name, username, phone number, birthday and gender
        /// </remarks>
        /// <response code="200">Returns success message</response>
        /// <response code="400">Provided nic's are not matching</response>
        /// <response code="403">Users can't update others information</response>
        /// <response code="500">Internal server error</response>
        [HttpPut("{nic}")]
        public async Task<ActionResult<ResponseModel>> PutUser([FromRoute] string nic, [FromBody] UserUpdateViewModel model)
        {
            try
            {
                if (nic != model.NIC)
                {
                    return BadRequest(new ResponseModel { Status = "Error", Message = "Something went wrong!" });
                }

                var accessToken = await HttpContext.GetTokenAsync("access_token");
                var token = new JwtSecurityTokenHandler().ReadJwtToken(accessToken) as JwtSecurityToken;
```

```
167.                 var loggedInUserNic = token.Claims.First(claim => claim.Type ==
    "nameid").Value;
168.                 var loggedInUsername = token.Claims.First(claim => claim.Type ==
    "unique_name").Value;
169.                 var role = token.Claims.First(claim => claim.Type ==
    ClaimTypes.Role).Value;
170.
171.                 if (role != UserRoles.Admin.ToString())
172.                 {
173.                     if (loggedInUserNic != nic)
174.                     {
175.                         _logger.LogWarning(string.Format("{0} tried to update {1}'s user
    informatino ", loggedInUsername, nic));
176.                         return StatusCode(StatusCodes.Status403Forbidden, new
    ResponseModel { Status = "Error", Message = "Can't update others information" });
177.                     }
178.                 }
179.
180.                 var findUser = await _context.Users.Where(x => x.NIC == nic).FirstAsync();
181.                 var NICInfo = _authService.GetNICInfo(nic);
182.
183.                 if (findUser == null)
184.                 {
185.                     return NotFound(new ResponseModel { Status = "Error", Message = "User
    not found!" });
186.                 }
187.
188.                 findUser.UserName = model.Username;
189.                 findUser.FirstName = model.FirstName;
190.                 findUser.LastName = model.LastName;
191.                 findUser.Gender = NICInfo.Gender;
192.                 findUser.Email = model.Email;
193.                 findUser.Birthday = NICInfo.Birthday;
194.                 findUser.Address = model.Address;
195.                 findUser.NIC = model.NIC;
196.                 findUser.PhoneNumber = model.PhoneNumber;
197.
198.                 var result = await _userManager.UpdateAsync(findUser);
199.
200.                 if (result.Succeeded)
201.                 {
202.
203.                     _logger.LogInformation(string.Format("{0} is updated own user
    information.", model.Username));
204.
205.                     return Ok(new
206.                     {
207.                         UserName = model.Username,
208.                         FirstName = model.FirstName,
209.                         LastName = model.LastName,
210.                         Gender = NICInfo.Gender,
211.                         Email = model.Email,
212.                         Birthday = NICInfo.Birthday,
213.                         Address = model.Address,
214.                         NIC = model.NIC,
215.                         PhoneNumber = model.PhoneNumber,
216.                     });
217.                 }
218.                 else
219.                 {
220.                     return StatusCode(StatusCodes.Status500InternalServerError, new
    ResponseModel { Status = "Error", Message = "User data updating failed!" });
221.                 }
222.             }
223.             catch (Exception ex)
```

```
224.             {
225.                 _logger.LogError("Error occcured in PUT: user/id", ex.Message);
226.                 throw;
227.             }
228.         }
229.
230.         /// <summary>
231.         /// Delete user info
232.         /// </summary>
233.         /// <remarks>
234.         /// Admin can Delete the user by entering the NIC of the user
235.         /// </remarks>
236.         /// <response code="200">Returns success message</response>
237.         /// <response code="403">User don't have permission to access this
    endpoint</response>
238.         /// <response code="404">User not found</response>
239.         /// <response code="500">Internal server error</response>
240.         [Authorize(Roles = "Admin")]
241.         [HttpDelete("{nic}")]
242.         public async Task<ActionResult<ResponseModel>> DeleteUser([FromRoute] string nic)
243.         {
244.             using (var transaction = await _context.Database.BeginTransactionAsync())
245.             {
246.                 try
247.                 {
248.                     var accessToken = await HttpContext.GetTokenAsync("access_token");
249.                     var token = new JwtSecurityTokenHandler().ReadJwtToken(accessToken) as
    JwtSecurityToken;
250.                     var loggedInUsername = token.Claims.First(claim => claim.Type ==
    "unique_name").Value;
251.
252.
253.                     var findUser = await _context.Users.FirstOrDefaultAsync(x => x.NIC ==
    nic);
254.                     var NICInfo = _authService.GetNICInfo(nic);
255.
256.                     if (findUser == null)
257.                     {
258.                         return NotFound(new ResponseModel { Status = "Error", Message =
    "User not found!" });
259.                     }
260.
261.                     findUser.DeleteStatus = true;
262.                     var result = await _userManager.UpdateAsync(findUser);
263.
264.                     if (result.Succeeded)
265.                     {
266.
267.                         _logger.LogInformation(string.Format("{0} is deleted user: {1}.",
    loggedInUsername, findUser.UserName));
268.                         var qualifications = await _context.Qualifications.Where(x =>
    x.NIC == nic).ToListAsync();
269.                         qualifications.ForEach(x => x.IsDelete = true);
270.
271.                         var afiliations = await _context.AffiliationData.Where(x => x.NIC
    == nic).ToListAsync();
272.                         afiliations.ForEach(x => x.IsDelete = true);
273.
274.                         var educationData = await _context.EducationData.Where(x => x.NIC
    == nic).ToListAsync();
275.                         educationData.ForEach(x => x.IsDelete = true);
276.
277.                         await _context.SaveChangesAsync();
278.                         transaction.Commit();
```

```
279.                         return Ok(new ResponseModel { Status = "Success", Message = "User
    data deleating successfull!" });
280.                     }
281.                     else
282.                     {
283.                         return StatusCode(StatusCodes.Status500InternalServerError, new
    ResponseModel { Status = "Error", Message = "User data deleating failed!" });
284.                     }
285.                 }
286.                 catch (Exception ex)
287.                 {
288.                     await transaction.RollbackAsync();
289.                     _logger.LogError("Error occcured in DELETE: user/id", ex.Message);
290.                     throw;
291.                 }
292.             }
293.         }
294.
295.         /// <summary>
296.         /// Verify documents
297.         /// </summary>
298.         /// <remarks>
299.         /// Staff members can Verify the uploaded documents if all the required documents
    are uploaded to the system
300.         /// </remarks>
301.         /// <response code="200">Returns success message</response>
302.         /// <response code="403">User don't have permission to access this
    endpoint</response>
303.         /// <response code="404">Job seekers data not found for entered nic or selected
    file not uploaded yet.</response>
304.         [Authorize(Roles = "Officer")]
305.         [HttpPut("verify-document/{nic}/{type}")]
306.         public async Task<ActionResult<ResponseModel>> VerifyDocuments([FromRoute]
    FileCategories type, [FromRoute] string nic, [FromQuery] bool isApproved = false)
307.         {
308.
309.             try
310.             {
311.                 var data = await _context.JobSeekerData.Where(x => x.NIC ==
    nic).FirstOrDefaultAsync();
312.                 if (data == null)
313.                     return NotFound(new ResponseModel { Status = "Error", Message = "Job
    seeker data not found." });
314.
315.                 switch (type)
316.                 {
317.                     case FileCategories.CV:
318.                         if (data.CvFileName == null)
319.                             return NotFound(new ResponseModel { Status = "Error", Message
    = "This user doesn't uploaded birth certificate yet." });
320.                         data.IsCvValidated = isApproved ? FileVerificationStatus.approved
    : FileVerificationStatus.rejected;
321.                         break;
322.                     case FileCategories.BirthCertificate:
323.                         if (data.BirthCertificateFileName == null)
324.                             return NotFound(new ResponseModel { Status = "Error", Message
    = "This user doesn't uploaded birth certificate yet." });
325.                         data.IsBirthCertificateValidated = isApproved ?
    FileVerificationStatus.approved : FileVerificationStatus.rejected;
326.                         break;
327.                     case FileCategories.Passtport:
328.                         if (data.PassportFileName == null)
329.                             return NotFound(new ResponseModel { Status = "Error", Message
    = "This user doesn't uploaded birth certificate yet." });
```

```
330.                         data.IsPassportValidated = isApproved ?
    FileVerificationStatus.approved : FileVerificationStatus.rejected;
331.                         break;
332.                 }
333.                 _context.Entry(data).State = EntityState.Modified;
334.                 await _context.SaveChangesAsync();
335.
336.                 return Ok(new ResponseModel { Status = "Success", Message = "Status update
    successful." });
337.             }
338.             catch (Exception ex)
339.             {
340.                 _logger.LogError("Error occcured in PUT: user/veriify-document/nic/type",
    ex.Message);
341.                 throw;
342.             }
343.         }
344.
345.
346.         /// <summary>
347.         /// Get users own profile.
348.         /// </summary>
349.         /// <remarks>
350.         /// Returns data  according to the user's authorization token(JSON Web Token).
351.         /// </remarks>
352.         /// <response code="200">Returns users profile</response>
353.         /// <response code="404">User not found</response>
354.         [HttpGet("profile")]
355.         public async Task<ActionResult<UserViewModel>> GetProfile()
356.         {
357.             try
358.             {
359.                 var accessToken = await HttpContext.GetTokenAsync("access_token");
360.                 var token = new JwtSecurityTokenHandler().ReadJwtToken(accessToken) as
    JwtSecurityToken;
361.                 var username = token.Claims.First(claim => claim.Type ==
    "unique_name").Value;
362.                 var nic = token.Claims.First(claim => claim.Type == "nameid").Value;
363.                 var role = token.Claims.First(claim => claim.Type ==
    ClaimTypes.Role).Value;
364.
365.                 var user = await _userManager.FindByNameAsync(username);
366.                 if (user == null)
367.                 {
368.                     return NotFound(new ResponseModel { Status = "Error", Message = "User
    not found!" });
369.                 }
370.
371.                 var userJobSeekerDataViewModel = new UserJobSeekerDataViewModel();
372.
373.                 if (role != UserRoles.Admin.ToString() && role !=
    UserRoles.Officer.ToString())
374.                     userJobSeekerDataViewModel = await _userService.GetJobSeekerData(nic);
375.
376.                 var userPersonalInfo = new UserPersonalInfoViewModel
377.                 {
378.                     NIC = nic,
379.                     Address = user.Address,
380.                     Email = user.Email,
381.                     FirstName = user.FirstName,
382.                     LastName = user.LastName,
383.                     PhoneNumber = user.PhoneNumber,
384.                     Username = user.UserName,
385.                     Birthday = user.Birthday,
386.                     Gender = user.Gender
```

```
387.                 };
388.
389.                 var userData = new UserViewModel
390.                 {
391.                     UserInfo = userPersonalInfo,
392.                     JobSeekerData = userJobSeekerDataViewModel
393.                 };
394.
395.
396.                 return Ok(userData);
397.
398.             }
399.             catch (Exception ex)
400.             {
401.                 _logger.LogError("Error occcured in GET: user/profile", ex.Message);
402.                 throw;
403.             }
404.         }
405.
406.         /// <summary>
407.         /// Get users list according to userRole. [Access: Admins and Staff only]
408.         /// </summary>
409.         /// <remarks>
410.         /// Admins and Staff members can Get the list of users according to the list of
    qualifications entered and the items of the list should be seperated by a comma(',').
411.         /// </remarks>
412.         /// <response code="200">Returns users list according to selected role</response>
413.         /// <response code="403">Users don't have permission to access this end-
    point</response>
414.         /// <response code="404">No users found</response>
415.         [Authorize(Roles = "Admin, Officer")]
416.         [HttpGet("{qualifications}")]
417.         public async Task<ActionResult<IEnumerable<UserViewModel>>>
    GetUsersForQualifications([FromRoute] string qualifications)
418.         {
419.             try
420.             {
421.
422.                 var qualificationsList = qualifications.Split(',').ToList<string>();
423.                 qualificationsList = qualificationsList.ConvertAll(d => d.ToLower());
424.                 var userQualifications = _context.Qualifications.Where(x => x.IsDelete !=
    true).Select(x => new { x.NIC, x.Qualification }).ToList();
425.                 var nicList = userQualifications.Select(x => x.NIC).Distinct().ToList();
426.                 var filteredUserNICsList = new List<string>();
427.                 foreach (var nic in nicList)
428.                 {
429.                     var userQualificationFormatted = userQualifications.Where(x => x.NIC
    == nic).Select(x => x.Qualification.ToLower()).ToList();
430.                     if (qualificationsList.All(userQualificationFormatted.Contains))
431.                     {
432.                         filteredUserNICsList.Add(nic);
433.                     }
434.                 }
435.
436.                 var usersInfoList = new List<UserViewModel>();
437.
438.                 foreach (var nic in filteredUserNICsList)
439.                 {
440.                     var userData =
    _userManager.GetUsersInRoleAsync(UserRoles.User.ToString()).Result
441.                         .Where(x => x.NIC.ToLower() == nic.ToLower()).Select(user => new
    UserPersonalInfoViewModel
442.                         {
443.                             NIC = user.NIC,
444.                             Address = user.Address,
```

```
                            Email = user.Email,
                            FirstName = user.FirstName,
                            LastName = user.LastName,
                            PhoneNumber = user.PhoneNumber,
                            Username = user.UserName,
                            Birthday = user.Birthday,
                            Gender = user.Gender,
                        }).FirstOrDefault();
                        var itemJobSeekerData = await _userService.GetJobSeekerData(nic);

                        if (userData != null)
                        {
                            var itemData = new UserViewModel
                            {
                                UserInfo = userData,
                                JobSeekerData = itemJobSeekerData
                            };

                            usersInfoList.Add(itemData);
                        }
                    }

                    if (usersInfoList == null)
                        return NotFound(new ResponseModel { Status = "Error", Message = "No
    users found!" });

                    return Ok(usersInfoList);

                }
                catch (Exception ex)
                {
                    _logger.LogError("Error occcured in GET: user/qualifications",
    ex.Message);
                    throw;
                }
            }
        }
    }

```

COMPLAINTS CONTROLLER

```
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Net.Mime;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using SLBFEMS.Enums;
using SLBFEMS.Models;
using SLBFEMS.ViewModels.Complaint;

namespace SLBFEMS.Controllers
{
```

```
19.     [Authorize]
20.     [Route("complaints")]
21.     [ApiController]
22.     [Produces(MediaTypeNames.Application.Json, MediaTypeNames.Application.Xml)]
23.     [Consumes(MediaTypeNames.Application.Json, MediaTypeNames.Application.Xml)]
24.     public class ComplaintController : Controller
25.     {
26.
27.         private readonly ApplicationDbContext _context;
28.         private readonly ILogger<UsersController> _logger;
29.
30.         public ComplaintController(ApplicationDbContext context, ILogger<UsersController>
    logger)
31.         {
32.             _context = context;
33.             _logger = logger;
34.         }
35.
36.         /// <summary>
37.         /// Get complaints
38.         /// </summary>
39.         /// <remarks>
40.         /// You can Get message included in complaint and the NIC of the complaint sender by
    passing the Id of the complaint. If no ID is passed, all the comaplints are returning
41.         /// </remarks>
42.         /// <response code="200">Returns complaint</response>
43.         [HttpGet]
44.         [Authorize(Roles = "User, Admin")]
45.         public async Task<ActionResult<IEnumerable<ComplaintViewModel>>>
    GetComplaint([FromQuery] int? id = null)
46.         {
47.             try
48.             {
49.                 var accessToken = await HttpContext.GetTokenAsync("access_token");
50.                 var token = new JwtSecurityTokenHandler().ReadJwtToken(accessToken) as
    JwtSecurityToken;
51.                 var username = token.Claims.First(claim => claim.Type ==
    "unique_name").Value;
52.                 var nic = token.Claims.First(claim => claim.Type == "nameid").Value;
53.                 var role = token.Claims.First(claim => claim.Type == ClaimTypes.Role).Value;
54.
55.                 var complaints = new List<ComplaintViewModel>();
56.
57.                 if (role == UserRoles.Admin.ToString())
58.                 {
59.                     if (id == null)
60.                     {
61.                         var complaintsList = await _context.Complaints.ToListAsync();
62.                         foreach (var complaint in complaintsList)
63.                         {
64.                             complaints.Add(new ComplaintViewModel
65.                             {
66.                                 ComplaintStatus = complaint,
67.                                 MessageThread = await _context.ComplaintMessages.Where(x =>
    x.ComplaintId == complaint.Id).Select(x => new ComplaintMessagesViewModel
68.                                 {
69.                                     Complaint = x.Message,
70.                                     Nic = x.Nic,
71.                                     TimeStamp = x.TimeStamp,
72.                                     Name = _context.Users.Where(y => y.NIC ==
    x.Nic).Select(x => x.UserName).FirstOrDefault()
73.                                 }).ToListAsync()
74.                             });
75.                         }
76.
```

```
77.                         return Ok(complaints);
78.                     }
79.                     else
80.                     {
81.                         return Ok(new ComplaintViewModel
82.                         {
83.                             ComplaintStatus = await
    _context.Complaints.FirstOrDefaultAsync(x => x.Id == id),
84.                             MessageThread = await _context.ComplaintMessages.Where(x =>
    x.ComplaintId == id).Select(x => new ComplaintMessagesViewModel
85.                             {
86.                                 Complaint = x.Message,
87.                                 Nic = x.Nic,
88.                                 TimeStamp = x.TimeStamp,
89.                                 Name = _context.Users.Where(y => y.NIC == x.Nic).Select(x =>
    x.UserName).FirstOrDefault()
90.                             }).ToListAsync()
91.                         });
92.                     }
93.                 }
94.                 else
95.                 {
96.                     if (id == null)
97.                     {
98.                         var complaintsList = await _context.Complaints.Where(x =>
    x.CreatedBy == nic).ToListAsync();
99.                         foreach (var complaint in complaintsList)
100.                        {
101.                            complaints.Add(new ComplaintViewModel
102.                            {
103.                                ComplaintStatus = complaint,
104.                                MessageThread = await _context.ComplaintMessages.Where(x
    => x.ComplaintId == complaint.Id).Select(x => new ComplaintMessagesViewModel
105.                                {
106.                                    Complaint = x.Message,
107.                                    Nic = x.Nic,
108.                                    TimeStamp = x.TimeStamp,
109.                                    Name = _context.Users.Where(y => y.NIC ==
    x.Nic).Select(x => x.UserName).FirstOrDefault()
110.                                }).ToListAsync()
111.                            });
112.                        }
113.
114.                        return Ok(complaints);
115.                    }
116.                    else
117.                    {
118.                        var complaint = await _context.Complaints.Where(x => x.CreatedBy
    == nic).FirstOrDefaultAsync(x => x.Id == id);
119.                        return Ok(new ComplaintViewModel
120.                        {
121.                            ComplaintStatus = complaint,
122.                            MessageThread = await _context.ComplaintMessages.Where(x =>
    x.ComplaintId == id).Select(x => new ComplaintMessagesViewModel
123.                            {
124.                                Complaint = x.Message,
125.                                Nic = x.Nic,
126.                                TimeStamp = x.TimeStamp,
127.                                Name = _context.Users.Where(y => y.NIC == x.Nic).Select(x
    => x.UserName).FirstOrDefault()
128.                            }).ToListAsync()
129.                        });
130.                    }
131.                }
132.            }
```

```
133.             catch (Exception ex)
134.             {
135.                 _logger.LogError("Error occcured in GET: complaint.", ex.Message);
136.                 throw;
137.             }
138.         }
139.
140.         /// <summary>
141.         /// Make complaint
142.         /// </summary>
143.         /// <remarks>
144.         /// Sample request:
145.         ///
146.         ///     POST /complaint
147.         ///     {
148.         ///         "title": "Test Complaint",
149.         ///         "complaint": "your complaint"
150.         ///     }
151.         ///
152.         /// You can Make Complaint to the system by entering the message. NIC of the
    logged in user is getting by the system.
153.         /// </remarks>
154.         /// <response code="200">Returns complaint</response>
155.         /// <response code="404">User not found</response>
156.         [HttpPost]
157.         [Authorize(Roles = "User")]
158.         public async Task<ActionResult<ComplaintViewModel>> PostComplaint([FromBody]
    ComplaintCreateViewModel model)
159.         {
160.             using (var transaction = await _context.Database.BeginTransactionAsync())
161.             {
162.                 try
163.                 {
164.                     var accessToken = await HttpContext.GetTokenAsync("access_token");
165.                     var token = new JwtSecurityTokenHandler().ReadJwtToken(accessToken) as
    JwtSecurityToken;
166.                     var loggedInUserNic = token.Claims.First(claim => claim.Type ==
    "nameid").Value;
167.                     var username = token.Claims.First(claim => claim.Type ==
    "unique_name").Value;
168.
169.                     var complaint = new ComplaintDataModel
170.                     {
171.                         Status = ComplaintStatus.New,
172.                         CreatedBy = loggedInUserNic,
173.                         Title = model.Title
174.                     };
175.
176.                     await _context.Complaints.AddAsync(complaint);
177.
178.                     await _context.SaveChangesAsync();
179.
180.                     await _context.ComplaintMessages.AddAsync(new ComplaintMessageModel
181.                     {
182.                         ComplaintId = complaint.Id,
183.                         Message = model.Complaint,
184.                         Nic = loggedInUserNic,
185.                         TimeStamp = DateTime.Now
186.                     });
187.
188.                     await _context.SaveChangesAsync();
189.                     await transaction.CommitAsync();
190.                     return Ok(new ComplaintViewModel
191.                     {
192.                         ComplaintStatus = complaint,
```

```
                        MessageThread = new List<ComplaintMessagesViewModel>()
                        {
                            new ComplaintMessagesViewModel() {
                                Complaint = model.Complaint,
                                Nic = loggedInUserNic,
                                TimeStamp = DateTime.Now,
                                Name = username
                            }
                        }
                    });

                }
                catch (Exception ex)
                {
                    await transaction.RollbackAsync();
                    _logger.LogError("Error occcured in POST: complint.", ex.Message);
                    throw;
                }
            }
        }

        /// <summary>
        /// Update complaint
        /// </summary>
        /// <remarks>
        /// Sample request:
        ///
        ///     PUT /complaint/1
        ///     {
        ///         "complaint": "your reply"
        ///         "isComplete": true
        ///     }
        ///
        /// If you want to Edit the complaint which you sent, you can edit the message you sent
        /// </remarks>
        /// <response code="200">Returns success message</response>
        /// <response code="404">Complaint or User not found</response>
        [HttpPut("{id}")]
        public async Task<ActionResult<ResponseModel>> PutComplaint([FromRoute] int id, [FromBody] ComplaintUpdateViewModel model)
        {
            using (var transaction = await _context.Database.BeginTransactionAsync())
            {
                try
                {
                    var complaint = await _context.Complaints.FirstOrDefaultAsync(x => x.Id == id);

                    if (complaint == null)
                        return NotFound();

                    var accessToken = await HttpContext.GetTokenAsync("access_token");
                    var token = new JwtSecurityTokenHandler().ReadJwtToken(accessToken) as JwtSecurityToken;
                    var loggedInUserNic = token.Claims.First(claim => claim.Type == "nameid").Value;

                    if (model.IsComplete)
                    {
                        complaint.Status = ComplaintStatus.Resolved;
                        complaint.CompletedAt = DateTime.Now;
                    }
                    else
                    {
```

```
                        complaint.Status = ComplaintStatus.Replied;
                    }
                    _context.Entry(complaint).State = EntityState.Modified;

                    await _context.SaveChangesAsync();

                    await _context.ComplaintMessages.AddAsync(new ComplaintMessageModel
                    {
                        ComplaintId = complaint.Id,
                        Message = model.Complaint,
                        Nic = loggedInUserNic
                    });

                    await _context.SaveChangesAsync();
                    await transaction.CommitAsync();
                    return Ok(new ResponseModel { Status = "Success", Message = "Complaint update successful" });

                }
                catch (Exception ex)
                {
                    await transaction.RollbackAsync();
                    _logger.LogError("Error occcured in PUT complaint/id", ex.Message);
                    throw;
                }
            }
        }
    }
}
```

CV PARSER CONTROLLER

```
using System;
using System.IO;
using System.Net.Mime;
using System.Threading.Tasks;
using FileUploadTest.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using SLBFEMS.Enums;
using SLBFEMS.Interfaces;
using SLBFEMS.Models;
using SLBFEMS.ViewModels.CVParser;

namespace SLBFEMS.Controllers
{
    [Route("cv-prase")]
    [ApiController]
    [Produces(MediaTypeNames.Application.Json, MediaTypeNames.Application.Xml)]
    public class CVParserController : ControllerBase
    {
        private readonly IFileManager _fileManager;
        private readonly ICVParserService _cvParser;
        private readonly ILogger _logger;

        public CVParserController(IFileManager fileManager, ICVParserService cvParser, ILogger<CVParserController> logger)
        {
```

```
27.             _fileManager = fileManager;
28.             _cvParser = cvParser;
29.             _logger = logger;
30.         }
31.
32.         /// <summary>
33.         /// Upload and parse CV
34.         /// </summary>
35.         /// <remarks>
36.         /// Accepted formats : docx / pdf
37.         ///
38.         /// You can Upload your CV to the system which is in a valid format( .docx or .pdf)
39.         /// </remarks>
40.         /// <response code="200">Returns cv data with uploaded file name</response>
41.         /// <response code="400">Unsupported file format</response>
42.         /// <response code="500">Something went wrong during cv prasing</response>
43.         [HttpPost]
44.         public async Task<ActionResult<CVParserResponseViewModel>> Upload([FromForm]
    FileModel model)
45.         {
46.             try
47.             {
48.                 if(Request.Form.Files[0] != null)
49.                 {
50.                     model.File = Request.Form.Files[0];
51.                 }
52.                 if (model.File != null)
53.                 {
54.                     var fileFormat = Path.GetExtension(model.File.FileName);
55.                     if (fileFormat == ".docx" || fileFormat == ".pdf")
56.                     {
57.                         var response = await _fileManager.Upload(model, FileCategories.CV);
58.                         var cvResponse = await _cvParser.GetCvData(response.URL);
59.                         if (cvResponse.IsSuccessful)
60.                         {
61.                             return Ok(new CVParserResponseViewModel
62.                             {
63.                                 CVData = cvResponse.CvData,
64.                                 FileName = response.FileName
65.                             });
66.                         }
67.
68.                         return StatusCode(StatusCodes.Status500InternalServerError, new
    ResponseModel { Status = "Error", Message = "Something went wrong during cv prasing" });
69.                     }
70.                 }
71.                 return BadRequest(new ResponseModel { Status = "Error", Message =
    "Unsupported file format" });
72.             }
73.             catch (Exception ex)
74.             {
75.                 _logger.LogError("Error occcured in POST: cv-parser.", ex.Message);
76.                 throw;
77.             }
78.         }
79.     }
80. }
81.
```

FILE MANAGER CONTROLLER

```
1.  using System.IO;
2.  using System.Threading.Tasks;
3.  using Microsoft.AspNetCore.Authorization;
4.  using Microsoft.AspNetCore.Http;
5.  using Microsoft.AspNetCore.Mvc;
6.  using SLBFEMS.Enums;
7.  using SLBFEMS.Interfaces;
8.  using SLBFEMS.Models;
9.
10. namespace SLBFEMS.Controllers
11. {
12.     [Authorize]
13.     [Route("file-manager")]
14.     [ApiController]
15.     public class FileManagerController : ControllerBase
16.     {
17.         private readonly IFileManager _fileManager;
18.
19.         public FileManagerController(IFileManager fileManager)
20.         {
21.             _fileManager = fileManager;
22.         }
23.
24.         /// <summary>
25.         /// Get file
26.         /// </summary>
27.         /// <remarks>
28.         /// You can retrive the file by entering the category and the file name.
29.         /// </remarks>
30.         /// <response code="200">Returns file</response>
31.         /// <response code="404">File not found</response>
32.         [HttpGet("{type}")]
33.         public async Task<IActionResult> Get([FromRoute] FileCategories type, [FromQuery]
    string fileName)
34.         {
35.             var fileFormat = Path.GetExtension(fileName);
36.             if (fileFormat == ".docx" || fileFormat == ".pdf" || fileFormat == ".png" ||
    fileFormat == ".jpeg" || fileFormat == ".jpg")
37.             {
38.                 var file = await _fileManager.Get(fileName, type);
39.                 if (file != null)
40.                 {
41.
42.                     switch (fileFormat)
43.                     {
44.                         case ".docx":
45.                             return File(file, "application/vnd.openxmlformats-
    officedocument.wordprocessingml.document");
46.                         case ".pdf":
47.                             return File(file, "application/pdf");
48.                         case ".png":
49.                             return File(file, "image/png");
50.                         case ".jpeg":
51.                             return File(file, "image/jpeg");
52.                         case ".jpg":
53.                             return File(file, "image/jpg");
54.                     }
55.                 }
56.             }
57.             return NotFound(new ResponseModel { Status = "Success", Message = "File not
    found" });
```

```
58.
59.         }
60.
61.         /// <summary>
62.         /// Get file url
63.         /// </summary>
64.         /// <remarks>
65.         /// According to the file category can get the file url
66.         /// </remarks>
67.         /// <response code="200">Returns file URL</response>
68.         /// <response code="404">File not found</response>
69.         [HttpGet("url/{type}")]
70.         public ActionResult GetUrl([FromRoute] FileCategories type, [FromQuery] string
    fileName)
71.         {
72.             var file = _fileManager.GetFileUrl(fileName, type);
73.             if (file == null)
74.             {
75.                 return NotFound(new ResponseModel { Status = "Success", Message = "File not
    found" });
76.             }
77.             return Ok(new { url = file });
78.
79.         }
80.
81.         /// <summary>
82.         /// Delete file
83.         /// </summary>
84.         /// <remarks>
85.         /// You can Delete the file by entering the type of the file and name of the file
86.         /// </remarks>
87.         /// <response code="202">Returns success message</response>
88.         /// <response code="404">File not found</response>
89.         [HttpDelete("{type}")]
90.         public async Task<IActionResult> Delete([FromRoute] FileCategories type, [FromQuery]
    string fileName)
91.         {
92.             var file = await _fileManager.Delete(fileName, type);
93.             if (file)
94.                 return Ok( new ResponseModel { Status = "Success", Message = "File deleted"
    });
95.             return NotFound(new ResponseModel { Status = "Success", Message = "File not
    found" });
96.         }
97.     }
98. }
99.
```

JOBSEEKERS DATA CONTROLLER

```
1.  using System;
2.  using System.IdentityModel.Tokens.Jwt;
3.  using System.Linq;
4.  using System.Net.Mime;
5.  using System.Threading.Tasks;
6.  using FileUploadTest.Models;
7.  using Microsoft.AspNetCore.Authentication;
8.  using Microsoft.AspNetCore.Authorization;
9.  using Microsoft.AspNetCore.Http;
10. using Microsoft.AspNetCore.Identity;
```

```
11. using Microsoft.AspNetCore.Mvc;
12. using Microsoft.EntityFrameworkCore;
13. using Microsoft.Extensions.Logging;
14. using SLBFEMS.Enums;
15. using SLBFEMS.Interfaces;
16. using SLBFEMS.Models;
17. using SLBFEMS.ViewModels.Authentication;
18. using SLBFEMS.ViewModels.FileManager;
19. using SLBFEMS.ViewModels.User;
20.
21. namespace SLBFEMS.Controllers
22. {
23.     [Authorize]
24.     [Route("job-seekers")]
25.     [ApiController]
26.     [Produces(MediaTypeNames.Application.Json, MediaTypeNames.Application.Xml)]
27.     public class JobSeekerDataController : ControllerBase
28.     {
29.         private readonly UserManager<ApplicationUserModel> _userManager;
30.         private readonly ApplicationDbContext _context;
31.         private readonly IUserService _userService;
32.         private readonly ILogger<JobSeekerDataController> _logger;
33.         private readonly IFileManager _fileManager;
34.
35.         public JobSeekerDataController(UserManager<ApplicationUserModel> userManager,
    ApplicationDbContext context, IUserService userService, ILogger<JobSeekerDataController>
    logger, IFileManager fileManager)
36.         {
37.             _userManager = userManager;
38.             _context = context;
39.             _logger = logger;
40.             _userService = userService;
41.             _fileManager = fileManager;
42.         }
43.
44.         /// <summary>
45.         /// Register as a new user
46.         /// </summary>
47.         /// <remarks>
48.         /// Sample request:
49.         ///
50.         ///     PUT /job-speeker/993581088v
51.         ///     {
52.         ///         "nic": "993581088v"
53.         ///         "currentLat": "1.232323",
54.         ///         "currentLong": "2.32323",
55.         ///         "profession": "Software Engineer",
56.         ///         "affiliations": [
57.         ///           {
58.         ///             "start": "jan 2021",
59.         ///             "end": "feb 2021",
60.         ///             "location": "kandy",
61.         ///             "organization": "built apps",
62.         ///             "title": "Associate Software Engineer"
63.         ///           }
64.         ///          ],
65.         ///          "education": [
66.         ///           {
67.         ///             "start": "jan 2021",
68.         ///             "end": "feb 2021",
69.         ///             "name": "Dharmaraja college"
70.         ///           }
71.         ///          ],
72.         ///         "qualifications": [
73.         ///           "java",
```

```
74.         ///           "Angular",
75.         ///           "SQL"
76.         ///         ]
77.         ///      }
78.         ///
79.         /// You can update NIC, Address, Email, FirstName, LastName, PhoneNumber, Username
    by entering the current NIC in the system
80.         /// </remarks>
81.         /// <response code="200">Returns updated data</response>
82.         /// <response code="400">Provided nic's are not matching</response>
83.         /// <response code="403">Can't update others data</response>
84.         [HttpPut("{nic}")]
85.         [Consumes(MediaTypeNames.Application.Json, MediaTypeNames.Application.Xml)]
86.         public async Task<ActionResult<UserJobSeekerDataViewModel>>
    PutJobSeekerData([FromRoute] string nic, [FromBody] JobSeekerDataUpdateViewModel model)
87.         {
88.             using (var transaction = await _context.Database.BeginTransactionAsync())
89.             {
90.                 try
91.                 {
92.                     if (nic != model.NIC)
93.                     {
94.                         return BadRequest(new ResponseModel { Status = "Error", Message =
    "Something went wrong!" });
95.                     }
96.
97.                     var accessToken = await HttpContext.GetTokenAsync("access_token");
98.                     var token = new JwtSecurityTokenHandler().ReadJwtToken(accessToken) as
    JwtSecurityToken;
99.                     var loggedInUsername = token.Claims.First(claim => claim.Type ==
    "unique_name").Value;
100.                    var loggedInUserNic = token.Claims.First(claim => claim.Type ==
    "nameid").Value;
101.
102.                    if (loggedInUserNic != nic)
103.                    {
104.                        _logger.LogWarning(string.Format("{0} tried to update {1}'s user
    informatino ", loggedInUsername, nic));
105.                        return StatusCode(StatusCodes.Status403Forbidden, new
    ResponseModel { Status = "Error", Message = "Can't update others data." });
106.                    }
107.
108.                    var data = await _context.JobSeekerData.FirstOrDefaultAsync(x => x.NIC
    == nic);
109.
110.                    data.CurrentLong = model.CurrentLong;
111.                    data.CurrentLat = model.CurrentLat;
112.                    data.Profession = model.Profession;
113.
114.                    _context.Entry(data).State = EntityState.Modified;
115.
116.                    var qualifications = await _context.Qualifications.Where(x => x.NIC ==
    nic).ToListAsync();
117.                    var qualificationsFormatted = qualifications.Select(x =>
    x.Qualification).ToList();
118.
119.
120.                    if (!(qualificationsFormatted.All(model.Qualifications.Contains) &&
    qualificationsFormatted.Count == model.Qualifications.Count))
121.                    {
122.                        foreach (var qualification in qualifications)
123.                        {
124.                            _context.Qualifications.Remove(qualification);
125.                        }
126.
```

```
127.                         if (model.Qualifications.Count > 0)
128.                         {
129.                             foreach (var qualification in model.Qualifications)
130.                             {
131.                                 await _context.Qualifications.AddAsync(new
    QualificationsModel
132.                                 {
133.                                     NIC = nic,
134.                                     Qualification = qualification,
135.                                 });
136.                             }
137.                         }
138.                     }
139.
140.                     var affiliationData = await _context.AffiliationData.Where(x => x.NIC
    == nic).ToListAsync();
141.                     var affiliationDataFormatted = affiliationData.Select(x => new
    AffiliationDataViewModel
142.                     {
143.                         End = x.End,
144.                         Location = x.Location,
145.                         Organization = x.Organization,
146.                         Start = x.Start,
147.                         Title = x.Title
148.                     }).ToList();
149.
150.
151.                     if (!(affiliationDataFormatted.All(model.Affiliations.Contains) &&
    affiliationDataFormatted.Count == model.Affiliations.Count))
152.                     {
153.                         foreach (var affiliation in affiliationData)
154.                         {
155.                             _context.AffiliationData.Remove(affiliation);
156.                         }
157.
158.                         if (model.Affiliations.Count > 0)
159.                         {
160.                             foreach (var affilicate in model.Affiliations)
161.                             {
162.                                 await _context.AffiliationData.AddAsync(new
    AffiliationDataModel
163.                                 {
164.                                     NIC = nic,
165.                                     Start = affilicate.Start,
166.                                     End = affilicate.End,
167.                                     Title = affilicate.Title,
168.                                     Location = affilicate.Location,
169.                                     Organization = affilicate.Organization,
170.                                 });
171.                             }
172.                         }
173.                     }
174.
175.                     var educationData = await _context.EducationData.Where(x => x.NIC ==
    nic).ToListAsync();
176.                     var educationDataFormatted = educationData.Select(x => new
    EducationDataViewModel
177.                     {
178.                         End = x.End,
179.                         Start = x.Start,
180.                         Name = x.Name
181.
182.                     }).ToList();
183.
184.
```

```
185.                     if (!(educationDataFormatted.All(model.Education.Contains) &&
    educationDataFormatted.Count == model.Education.Count))
186.                     {
187.                         foreach (var education in educationData)
188.                         {
189.                             _context.EducationData.Remove(education);
190.                         }
191.
192.                         if (model.Education.Count > 0)
193.                         {
194.                             foreach (var education in model.Education)
195.                             {
196.                                 await _context.EducationData.AddAsync(new
    EducationDataModel
197.                                 {
198.                                     NIC = nic,
199.                                     Start = education.Start,
200.                                     End = education.End,
201.                                     Name = education.Name
202.                                 });
203.                             }
204.                         }
205.                     }
206.
207.                     await _context.SaveChangesAsync();
208.                     await transaction.CommitAsync();
209.                     var userJobSeekerDataViewModel = await
    _userService.GetJobSeekerData(nic);
210.                     return Ok(userJobSeekerDataViewModel);
211.
212.
213.                 }
214.                 catch (Exception ex)
215.                 {
216.                     await transaction.RollbackAsync();
217.                     _logger.LogError("Error occcured in PUT: job-seeker/nic", ex.Message);
218.                     throw;
219.                 }
220.             }
221.         }
222.
223.         /// <summary>
224.         /// upload user's documents
225.         /// </summary>
226.         /// <remarks>
227.         /// User can upload file documents after Selecting the type of the uploading file
228.         /// </remarks>
229.         /// <response code="200">Returns new file name</response>
230.         /// <response code="400">Invalid file</response>
231.         /// <response code="404">User not found</response>
232.         [HttpPost("file-upload/{type}")]
233.         public async Task<ActionResult<ResponseModel>> UploadFiles([FromRoute]
    FileCategories type, [FromForm] FileModel model)
234.         {
235.             try
236.             {
237.                 if (Request.Form.Files[0] != null)
238.                 {
239.                     model.File = Request.Form.Files[0];
240.                 }
241.
242.                 var accessToken = await HttpContext.GetTokenAsync("access_token");
243.                 var token = new JwtSecurityTokenHandler().ReadJwtToken(accessToken) as
    JwtSecurityToken;
```

```
                    var username = token.Claims.First(claim => claim.Type ==
    "unique_name").Value;
                    var nic = token.Claims.First(claim => claim.Type == "nameid").Value;

                    var user = await _userManager.FindByNameAsync(username);
                    var jobSeekerData = await _context.JobSeekerData.FirstOrDefaultAsync(x =>
    x.NIC == nic);
                    if (user == null || jobSeekerData == null)
                    {
                        return NotFound(new ResponseModel { Status = "Error", Message = "User
    not found!" });
                    }

                    var fileUploadResponse = new FileUploadResponseViewModel();
                    if (model.File != null)
                    {
                        fileUploadResponse = await _fileManager.Upload(model, type);
                    }
                    else
                    {
                        return BadRequest(new ResponseModel { Status = "Error", Message =
    "File not valid." });
                    }


                    switch (type)
                    {
                        case FileCategories.CV:
                            jobSeekerData.CvFileName = fileUploadResponse.FileName;
                            jobSeekerData.IsCvValidated = FileVerificationStatus.pending;
                            break;
                        case FileCategories.BirthCertificate:
                            jobSeekerData.BirthCertificateFileName =
    fileUploadResponse.FileName;
                            jobSeekerData.IsBirthCertificateValidated =
    FileVerificationStatus.pending;
                            break;
                        case FileCategories.Passtport:
                            jobSeekerData.PassportFileName = fileUploadResponse.FileName;
                            jobSeekerData.IsPassportValidated =
    FileVerificationStatus.pending;
                            break;
                    }

                    _context.Entry(jobSeekerData).State = EntityState.Modified;
                    await _context.SaveChangesAsync();
                    return Ok(fileUploadResponse.FileName);
                }
                catch (Exception ex)
                {
                    _logger.LogError("Error occcured in POST: job-seeker/file-upload/type",
    ex.Message);
                    throw;
                }
            }

        }
    }

```

LOGS CONTROLLER

```
1.  using System;
2.  using System.Linq;
3.  using System.Net.Mime;
4.  using System.Threading.Tasks;
5.  using Microsoft.AspNetCore.Authorization;
6.  using Microsoft.AspNetCore.Mvc;
7.  using Microsoft.EntityFrameworkCore;
8.  using Microsoft.Extensions.Logging;
9.  using SLBFEMS.Models;
10. using SLBFEMS.ViewModels.Logs;
11.
12. namespace SLBFEMS.Controllers
13. {
14.     [Authorize(Roles = "Admin")]
15.     [Route("logs")]
16.     [ApiController]
17.     [Produces(MediaTypeNames.Application.Json)]
18.     [Consumes(MediaTypeNames.Application.Json)]
19.     public class LogsController : ControllerBase
20.     {
21.         private readonly ApplicationDbContext _context;
22.         private readonly ILogger<LogsController> _logger;
23.
24.         public LogsController(ApplicationDbContext context, ILogger<LogsController> logger)
25.         {
26.             _context = context;
27.             _logger = logger;
28.         }
29.
30.         /// <summary>
31.         /// Get informatoin logs list. [Access: Admins only]
32.         /// </summary>
33.         /// <response code="200">Returns information logs list</response>
34.         /// <response code="403">Forbidden</response>
35.         [HttpGet("information")]
36.         public async Task<ActionResult<LogViewModel>> GetInformations()
37.         {
38.             try
39.             {
40.
41.                 var logs = await _context.Logs.Where(x => x.Level ==
    "Information").OrderByDescending(x => x.TimeStamp).
42.                         Select(x => new LogViewModel
43.                         {
44.                             Message = x.Message,
45.                             Exception = x.Exception,
46.                             TimeStamp = x.TimeStamp.ToString("HH:mm:ss dd/MM/yyyy"),
47.                         }).ToListAsync();
48.
49.                 return Ok(logs);
50.             }
51.             catch (Exception ex)
52.             {
53.                 _logger.LogError(ex.Message, ex);
54.                 throw;
55.             }
56.         }
57.
58.         /// <summary>
59.         /// Get warning logs list. [Access: Admins only]
60.         /// </summary>
```

```
61.         /// <response code="200">Returns warning logs list</response>
62.         /// <response code="403">Forbidden</response>
63.         [HttpGet("warning")]
64.         public async Task<ActionResult<LogViewModel>> GetWarnings()
65.         {
66.             try
67.             {
68.                 var logs = await _context.Logs.Where(x => x.Level ==
    "Warning").OrderByDescending(x => x.TimeStamp).
69.                         Select(x => new LogViewModel
70.                         {
71.                             Message = x.Message,
72.                             Exception = x.Exception,
73.                             TimeStamp = x.TimeStamp.ToString("HH:mm:ss dd/MM/yyyy"),
74.                         }).ToListAsync();
75.                 return Ok(logs);
76.             }
77.             catch (Exception ex)
78.             {
79.                 _logger.LogError(ex.Message, ex);
80.                 throw;
81.             }
82.         }
83.
84.         /// <summary>
85.         /// Get error logs list. [Access: Admins only]
86.         /// </summary>
87.         /// <response code="200">Returns error logs list</response>
88.         /// <response code="403">Forbidden</response>
89.         [HttpGet("error")]
90.         public async Task<ActionResult<LogViewModel>> GetErrors()
91.         {
92.             try
93.             {
94.                 var logs = await _context.Logs.Where(x => x.Level ==
    "Error").OrderByDescending(x => x.TimeStamp).
95.                         Select(x => new LogViewModel
96.                         {
97.                             Message = x.Message,
98.                             Exception = x.Exception,
99.                             TimeStamp = x.TimeStamp.ToString("HH:mm:ss dd/MM/yyyy"),
100.                          }).ToListAsync();
101.
102.                 return Ok(logs);
103.             }
104.             catch (Exception ex)
105.             {
106.                 _logger.LogError(ex.Message, ex);
107.                 throw;
108.             }
109.         }
110.     }
111. }
112.
```