Checked by Department staff

Internship Program Outcome Report

Title

Full-stack development intern

Employment Information

Name of Institution	Wooksung Media Co., Ltd.		Employed Department	R&D
Employment period	Start date 2020/12/21	End date 2021/ 02/12	40 days (8 weeks)	40 hr. work week (5 days a week, 8 hours/day)

Graduation Research Substitution

☑ Recognized (3 credits) ☐ Unrecognized

※ 1 credit for 4 weeks or more, 2 credits for 6 weeks or more, 3 credits for 8 weeks or more.

※ For students who have earned 1 or 2 credits for graduation study, 1 or 2 credits for individual studies will be replaced with 3 credits for graduation studies upon receipt of the graduation study as an internship.

I hereby submit the following Outcome Report
as I have participated in the Internship Program as above

2021 (Year) 02 (Month) 22 (Day)

Dept.: _**School of Computing**_

Student No.: _**20170751**_

Name: _**Qilman Beytullazada** _[illegible]_(Signature)

Attachment: Internship Program Outcome Report

Summary

(Brief summary by practice schedule, Within 3 sheets of A4 paper)

During this winter vacation, I attended the CUOP internship program in Wooksung Media Co., Ltd. as a full-stack developer. Starting from December 21, 2020, I had full-time remote work which ended on February 12, 2021. The main task that I was required to and managed to accomplish can be summarized as the development of both client and server-side for a web service which must allow checking the usage statistics of the multi multi-party video conferencing servers. It can further be broken down into multiple subtasks as follows:

- Developing business logic for secure user registration and deletion per the employment status:
    - Administrators should be able to register users of any status: other administrators, business managers, instructors, and students as well as delete or see the statistical information of their video conference attendance.
    - Managers must be able to register instructors and students of the same division as well as delete or see the statistical information of their video conference attendance.
    - Instructors should be able to register students of the same division as well as delete or see the statistical information of their video conference attendance.
- Accessing and interpreting the information data of the multi-party video conferencing server operated by the headquarters which is transmitted as an HTTP POST request
- Parse the data and store it in the database following the access hierarchy among the user types
- Designing algorithm for checking the information stored in the database by daily, and monthly usage.
- Development of an algorithm for checking the usage time of a specific user from the information stored in the database.
- Management the authority to access the webserver.
- Development of functions to register and delete multi multi-party video conferencing servers as well as displaying the information about rooms and the users currently present in the server.

Upon negotiation with the mentor, I was allowed to use any development frameworks and database management systems. Though it was initially planned to develop the service using Vanilla Javascript, PHP, and store it in MySQL, I decided to use more modern tools and therefore chose React.js with Material-UI for the frontend development and Node.js with Express.js for the server-side. For DBMS, MongoDB seemed appropriate, and thus Mongoose.js was needed for its object modeling.

Among the above-mentioned tools, I had only had experience with React.js, and, therefore, my first two weeks of the internship were dedicated to familiarizing myself with the new environment and instruments to be used.

As the scale of the project was big, and to provide the adequate required functionality I choose to use Material-UI components for the frontend, which promises the absence of any errors that a developer would experience if implemented the frontend using purely HTML and CSS. Besides, it offers better modularity, and, as our service had a somewhat “administrator panel” conceptual design, it was not very difficult to maneuver it for finding appropriate Material-UI components.

During the development process, a lot of effort was spent on learning to implement and actual implementation of the secure registration and login. For this matter, I implemented password creation and encryption by JSON Web Signature and JSON Web Encryption using JSON Web Token.

Another challenge was building a well-structured hierarchy of both users, their connections to the web-server, and relationship with others following the rooms and servers to attend. It became especially challenging to test such relations between different parties, for which I decided to use visual clues, and mnemonic rules, which will be described further in the report.

Outcome Report

(More than 10 sheets of A4 paper)

In this section, I will describe the daily progress that I was making and by that demonstrate the steps of analyzing the issues, increasing knowledge, development, testing, debugging, and preparation for deployment.

12/21/2020

Listed all features requested by the client and tried to predict what kind of technologies will be needed to develop those features both in the backend and frontend. After listing all features it was decided to research other similar applications and what stack they have built with. Read several articles about this topic and choose the MERN stack which stands for Mongo DB, Express JS, React, and Node JS. The main reasons why this stack was chosen to use is because they have many libraries which go well with each other which is great to decrease development time and increase quality.

12/22/2020

During the first day many problems were foreseen which were not experienced before. That is why it was decided to find some resources and explore stack documentation where it is possible to find an answer to the problems:

1. How to store data received from various sources in one location?
2. How to handle authentication of the users that were not logged in?

12/23/2020

Compared different libraries to use, for example, reactstrap, React-Bootstrap, and Material-UI for the frontend side and finally it was decided to choose Material-UI for working on this project. Even Google’s design is based on Material-UI. Started familiarizing myself with Material-UI components by reading and analyzing its documentation.

12/24/2020

Continued reading Material-UI documentation, researched the “Drawer” for implementing the navigation bar and “Grid” for the layout as it offers great responsiveness. Analyzed their Component APIs to understand the limits of manipulations by props.

12/25/2020

To fulfill the company’s requirement the application needed to communicate the client's previous server and store the data which comes from there. Since this was the important feature of the new application it was decided to try this communication before starting to write the application itself. Some developer tools like Postman were used to try this communication and the data sent by other servers was successfully received and parsed.

12/28/2020

Started the day by creating some example data to use in my function. That gave information about what kind of data the function is going to receive and return. Such identification helps to speed up the process.

12/29/2020

Set up the development environment on the local machine and wrote the necessary scripts to run the server. Connected to the remote server using the SSH key which was provided by the company. Installed Mongo DB on Ubuntu 16.04 server and Node.js. Also, using the Remote SSH extension, connected Visual Studio Code to the company’s server.

12/30/2020

Read about best practices in Node.js and some development patterns like MVC. Downloaded Mongo DB server and Atlas on the local machine. Connected to the remote Mongo DB server, then started adding example data and familiarized myself with the interface of Mongo DB Atlas.

12/31/2020

Listed all route types and endpoints inside them. The names of routes are the following:

- authentication
- users
- entrances
- room routes
- server routes
- connections

Researched many authentication libraries in Node.js, decided to use JSON Web Token.

01/01/2021

Created the necessary files to connect to the database from the server and listen to the requests. Installed Express.js, imported the module and created my HTTP server. Created a .env file to store my environmental variables. Because my node server and database server were running on the same remote server I used a local server connection and connected to the server. I decided to use ODM called Mongoose.js, Downloaded and imported that on my codebase. It was even easier to connect the database over the Mongoose.js it was decided to continue with the Mongoose.js.

01/04/2021

Mongoose.js uses a special schema-based solution, which means it was needed to create all the document schema, and based on those schemas the documents would be created.
Started with User schema because authentication is an important part of the application and needed to store user information to authenticate the user for the future visits. Decided to collect all of the routes under "api/v1/", and it continued with the controller name which would handle this request. For example, in “auth” routes it is in the following form: "api/v1/auth". Created my first endpoint which is the POST request handler. This request is responsible to authenticate a user and

send the token for the later usage.
Express.js is a middleware-based framework, which means the main logic is done using middlewares. All middlewares get access to the "request", "response" and "next" object. The “request” object contains information about the request itself, and “response” object is for sending the response or, in other words, to finish the request. Next is how express works. Wrote several functions for only authentication purposes, these functions are expected to be used by the request handler which is responsible to handle authentication. The main function sends coming data to another helper function and gets created token in return and sends it back to the user in case of success, otherwise, it returns with an error.

01/05/2021

Implemented a new route to handle registering. As with all other controller functions this function has a different endpoint. When the user sends a request to this endpoint the function gets called and handles the user's request. The user is responsible to send the data with the:

- name
- email
- status
- division
- userId.

Password is created randomly and sent to the user's email. Created several users by storing his or her password in a plain test. However, the best practice is to store the password in hashed version. After researching, found there are great methods to do it in Node.js modules. Used the built-in module “crypto” and “bcryptjs” external library.

01/06/2021

Created new function called "sendEmail" to send all the emails. After application is scaled many kinds of emails are expected to be sent. For example, when user forgets his password or when admin creates a new user, etc. That is why new separate function was created which is only responsible for sending the emails. Did a research and choose to use an external package called Nodemailer. Nodemailer has many built-in services to send emails from services like Gmail, Twillo, SendGrid, etc. The decision was made to use Gmail service. Nodemailer needs a transporter to send emails.

01/07/2021

One of the main features of application is to see all of the users and the data related to them. For that a new route and controller were created which were responsible to get all the users from the database and send them to the frontend side.

01/08/2021

Created React app by using create-react-app npm package which prevents developer from many time-consuming processes. Downloaded npm package and imported into my react app. Material-UI library is component based, after quick research in Material-UI documentation I noticed that there are many components which I can use.

01/11/2021

I imported react-router to handle client-side routing and “axios” library to send requests to the backend. I also need to store user data in client side. Researched about the state management in React. There are 2 main option to use, Redux and React Context API. Chose React Context API since it was possible to implement all the needed features using this form of state management and plus, it is a built-in React feature.

01/12/2021

Created new login page and wrote a new function to send requests to backend and this function stored JSON Web Token in Context API. Context API is a global state for react applications, it means that every part of the application in React app has an access to the state and can consume data by using special React function. This is an important feature because other parts of the application like displaying users needed to get the data from the server, and, in order to get that data, client side needs the token to be authenticated by the server.

01/13/2021

Created “create server” user page from end side, this is basically form of the page where the user can fill input fields with the data of the user that he wants to create. After providing the data, the form should be submitted and the server creates a new user document in the database and emails his or her password to the user.

01/14/2021

Created the view of the “Profile” page. Because there are already several pages after authentication happens in front side. I needed some kind of navigation panel on the client side to enable the users to navigate from one page to another. That is why a new component was created only for this purpose. In the final version of the application there are 4 types of users: administrator, manager, instructor and student. Based on their type, they are expected to see different pages. In order to have this functionality, the navigation panel is supposed to be flexible. Implemented the navigation panel to accept a prop, and based on that prop it displayed some routes. Theses props should contain the user's type. Adjusted the login page to get the user's type from backend and stored in the global state. It is passed as a prop to the navigation. After passing the prop the navigation component is responsible to display the appropriate navigation menu items.

01/15/2021

Adjusted the authentication controller and made it send to the frontend the user's status. This is important on the front side, because what the user sees on the pages depends on it.

01/18/2021

After the navigation panel, worked on the navbar to enable the user to logout. Put the logout button to the navbar which was available in all pages. And this enabled the user to logout whenever he/she desired. When the user clicks on the logout button, the “logout” function gets called which cleans the token and other user related data in the global state. As the global state changes, all application is rendered again, and by that, the user is redirected to the login page. If the user wants to navigate to the user panel, user should provide credentials in the login page, only after successful login request the user can be redirected to the user panel.

01/19/2021

Started to work on “Server Management” page. This page consists of some navigation elements and tables. The table used on this page can also be used on the “User Management”, “Statistics” pages. Decided to work on this table first, before implementing the page’s related functionalities. Created the component called "Table" which accepted several props. One of these props is called "rows" and rows are the data the page is supposed d to display. For example, if it is the server page, it means the "rows" will contain the data about the servers. After the server page passes the data to the table, the table is responsible to display the data in a table format. If the user is an administrator, the user is also able to delete or update the server’s name.

01/20/2021

Created a new model called “Server” to store the server related data.

01/21/2021

Created servers’ page which sends the request to the backend and when it gets the data it passes that data to the table component. In case an error occurs, Material UI component called "snackbar" is imported and “Servers” page passes error message to this component for displaying the kind of occurred error. It is decided to use this component throughout all application to handle the errors.

Worked on servers' related functions, for which I created "serverRoutes" file to store endpoints and "serverController" file to store the functions which would accept the requests. One of the controller function is called "getAllServers", which is called when the root server endpoint gets a request, and thus gets all the servers from database and sends them back to where the server sends – for this case, this is the client side of our application, but it is also reusable enough to be converted in the future for different usage.

01/22/2021

Created new page to display users. This page is almost the same with "servers" page with the only differences that this page sends request to the different endpoint. Error and Table function are the same with the "servers" page.
Added new function to the "userController" to get all the users from the database.

01/25/2021

Until this point I used the servers which were entered by me manually using Mongo DB atlas, but in our application there is a requirement to generate them automatically when our root URL gets a POST request from another server. That is why I created a new controller and routes function called "connectionController". The "connectionController" gets the data from the other server and creates a new server document based on the data. The data coming from the MCU server are in the following form:
userid: String, mcuid: String, roomid: String, time: String, type: String.

01/26/2021

The tables representing the servers were unnecessarily repeated. After clarification the user access it was decided to refactor the table to make it modular and combine the necessary features of both tables into one, thus making it more universal and at the same time keeping its necessary functionality. Namely the page and the tab "Server Management" was removed.

01/27/2021

One of the features of the application is to show statistics about the servers as: which server contains how many students currently, which rooms are the students in are etc. In order to display these data, I created a new page called "Statistics" which contains the table and input (to specify which day's statistics we want to see). Statistics page gets the data from the server and passes it to the table and the table is responsible for displaying the data.

One problem is that until this point all of the models created by the servers were removeable, for example the user created could be deleted by an administrator after some point. But statistics feature is not the same, even after the user is deleted, the user's statistics should be persistent. That is why I decided to use a different data model here called 'Entrance' and this model will not be deleted when the user is deleted or the server is deleted. By such approach I was able to display the server data correctly even when some of the users were deleted later. I collected all this logic in the "connectionController" which is called when the different servers send a request to the root (”/”) endpoint.

01/28/2021

In addition to the main statistics page, two other pages related to the statistics must be built on the same menu tab. They are called "MonthlyStatistics" and "DailyStatistics". These pages are for displaying specific data for months and days respectively. I decided to build the "monthlyStatistics" and "DailyStatistics" logic on the frontend side. When I send a request to the backend to get specific data, I send it with the day query parameter called "day". This parameter sends the date the data of which the user wants to see.

Created a new controller called "entranceController" and routes file called "entranceRoutes". "entranceController"’s functions get called when the server gets a request. There is one important function called getEntrances, which gets all of the documents from the Database and sends them over the request. "getEntrances" function itself expects query parameter called "day" and parses it. After passing the date, the server gets all the data specifically from that day from the database. I specifically used the database layer for this application, because I knew it would yield better performance than the server, because Mongo DB is designed to get this kind of queries and uses the fastest algorithms to retrieve the data. I encountered a problem which is: although I used "$gte" and "$lte" methods to specify the date it didn't work. I got no successful result that day, and left the problem unsolved on that day.

01/29/2021

Created new "pop-up" component to get user's confirmation before deleting any data. Thought this is important, otherwise the user can delete the server or user accidentally. Used Mongo DB "Dialog" component for that, I designed it in a way that would satisfy the necessary need. The component itself contains only 2 buttons called "Cancel", "Submit", a heading and a message. The heading shows the user what action he or she intends to execute, the message gives additional information. For example, the name of the server which the user wants to delete.

Read the documentation about "$gte" and "$lte" in the official documentation of Mongoose.js, Mongo DB. It didn't work as I expected, all the solutions I found didn't help me to solve my problem. I decided to move on and do other tasks, and come back later.

02/01/2021

The previous day gave me confidence for continuing to solve the problem which I couldn't solve previously. This time I started with videos, watched some videos about Mongo DB on YouTube. While watching the video I noticed how the date data shown in the video differed from mine in Mongo DB Atlas and this gave me clue that I may have been storing the date in the wrong format. I checked my "Entrance" model and noticed that I didn't specify the type right. I changed "startTime" property's type to "Date" and erased the DB. Then I entered new data and tried the way I did before, this time it worked as expected.

02/02/2021

One of the needs of client is to add and delete the divisions. And these divisions would be displayed as choices in a select box, to assign them to the user when creating a user. Despite initial version, I thought the best place to enable such functionality is not on a separate page, but on the user registration page. A popup would allow to create and delete the divisions and the select box division list would automatically be updated. The administrator needs it only when he/she wants to create a new user.

I created a new controller called "divisionController" which contains methods to get, create and delete divisions. One the functions is called "getAllDivisions" which is responsible for getting all the divisions from DB, and sending them to the frontend, for user to complete the user registration. The second one is called "createDivision", which expects the data in "req.body", and data it expects is called "divisionName". And it generates a new division based on that name. The last one is responsible to delete the division and is called "deleteDivision". This delete function expects "divsionId" as the query parameter.

02/03/2021

Started the day reading an article about how to embed the data to option tag element in HTML. It was simple enough and that is why I implemented sending the request by "useEffect" function. The "useEffect" function is a special function which is called one time when the page is initially rendered. This allows us to get the data from server one time. The "useEffect" also takes second parameter which is for calling the function inside of useEffect in a later point. This allowed me to bring the divisions from the backend and display them inside of option input. If the user wants to add a new division, the user can click the icon next to the input field and this will trigger the dialog to open. User can update all divisions and save division at once in that dialog.

02/04/2021

The input fields are mainly handled by Material-UI, their active, required states are manipulated using the predefined props such as "active", "required". At the same time input values that are traditionally in other more advanced we services pass as correct input value might not pass if condition such as the following is not met: despite the fact that input value has spaces before or after the first and last characters correspondingly, in general it has to pass correctly. Therefore, I located all the input fields and utilized Javascript's str.trim() function on all of them to fix this bug.

02/05/2021

During the testing process a lot of the logging into console is needed. It helps to print out the outcome to see the flow and detect the possibly occurring bugs. At the same time for deployment, such console logs must be cleared out, so I did everything to keep the necessary logs and removed the unnecessary ones. At the same time, some functions were actual in the beginning of the development, but turned obsolete due to the improved implementation style. Therefore, I started finding and getting rid of such functions as well.

02/08/2021

After creating all of the routes and controllers, I started to bring some kind of restriction to the routes. Up until now, all the users could send any request to the backend and get any kind of data. But this is a security vulnerability, it means even a user that has no connection this company can get the data about all the user information, including server details and etc. After doing a quick research about it turned out that as all other functionalities in Express.js, this is also done by utilizing middlewares. I created a new middleware function called "protect", which is called in all protected routes before main function. The protect function checks if the user has a token or not. If not, it sends an error message to the backend, otherwise request continues.

02/09/2021

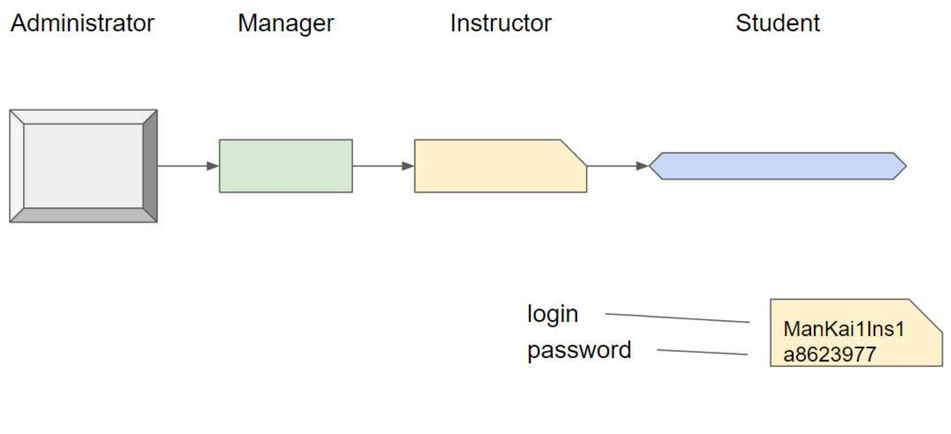
After implementing all these routes I created the main error handler controller called "errorHandlerController", and global error class called AppError which extended built-in "Error" class. Main error controller imports AppError class and throws AppError whenever needed. Because error controller is the first function to get called when error happens, it has access to all the messages sent by server, and based on that message it handles the errors differently. For example, if the Mongoose.js throws an error, it checks error and logs it in console but doesn't send an error back to the frontend directly. Instead, it send the message "something happened in server". This is important, because we don't give the unnecessary information to the user about the errors happening in our server.

02/10/2021

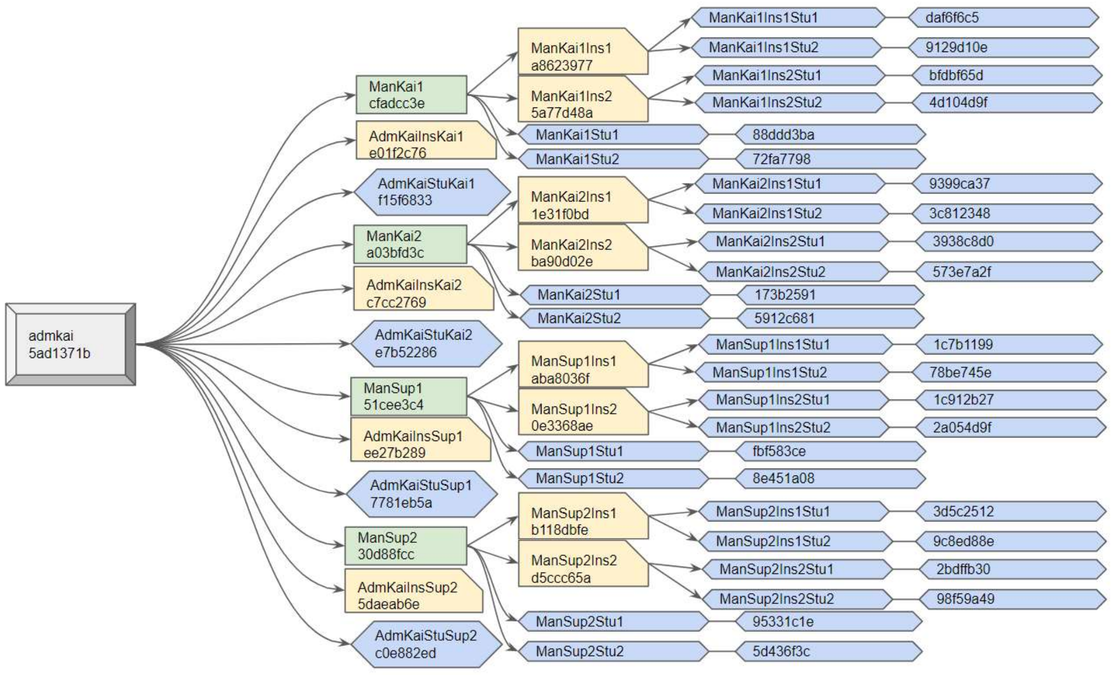
I started the day to test all functionality of all type of users. Some unexpected error happened during testing. For example, table components complained about the "key", which is important for React to detect which part of application has changed to guarantee the update of only that part.

In order to test the correctness of hierarchical access I used the following pattern. There must be mnemonics and logic in forming the names of all the combinations of the users with their corresponding divisions, types and accesses. Otherwise there would be gaps which would result in missing crucial erroneous outcomes.

I designed them visually by designating the colors and storing the related logins and passwords.



Note: Tail of the arrow denotes the "creator" of the user to which the tip of the arrow points



This way, I could enter the credentials and test all the relations, accesses and functionalities step by step, without forgetting any kind of important credentials.

The result is as follows:

1. LOGIN

1. All the users are able to login using user ID and password
2. Alert is displayed in case of login failure due to:
    1. incorrect userID or password input
    2. non-registered email input for password recovery
    3. missing the adequate input data for required fields
3. Users can reset their password by enterin the email their registered email

2. STATUS

1. Only administrators can see this page and it allows them to:
    1. see the table containing registered servers and the appropriate information related to them. The detailed contents of the table are:
        1. Server Name
        2. Server ID
        3. Number of users that are currently in the server.

    2. see detailed information about rooms registered in a particular server by clicking on the Server ID that shows. The detailed contents of the table are:
        1. Room ID
        2. Start time (date and time, when the first user entered the room)
        3. Number of users (currently in the room)
    3. register/delete server
    4. update server name by double clicking on it

3. <u>**USER MANAGEMENT**</u>

1. All the registered users receive the password by email.
2. Administrators are able to:
    1. see all the users of all divisions and the required information associated with them
    2. register/delete users of any division and type, namely, administrator, instructor, manager and student
    3. register/delete divisions
3. Managers are able to:
    1. see all the instructors and students of only the same division (includes the instructors and students registered by another manager and administrator)
    2. register/delete instructors and students of only the same division(includes the instructors and students registered by another manager and administrator)
4. Instructors are able to:
    1. see all the students registered only by himself/herself
    2. register students of the same division and delete students registered only by himself/herself

4. <u>**STATISTICS**</u>

1. Users of all types can see the statistics information of the users that are displayed to them in USER MANAGEMENT (students can see only themselves).
2. Statistics tab shows daily and monthly statistics of each user in hours and minutes.
3. To avoid incompleteness, the initial day in daily statistics is set to display the information related to 1 day before the current date.
4. Month and exact date can be chosen from the calendar.

5. “Monthly Statistics” table displays participation duration for each day that the user was present in the class and in total. The detailed information is:
    1. Date
    2. Usage in minutes
6. Clicking on a particular date cell leads to the table displaying corresponding day’s “Daily Statistics” table.
7. “Daily Statistics” table displays participation duration regarding each “enter - leave” timestamp pair and in total during the chosen day. The detailed information is:
    1. Start Time - exact time the user entered the classroom
    2. End Time - exact time the user left the classroom
    3. Server ID - ID of the server containing the room
    4. Room ID - ID of the classroom being entered

5. PROFILE

1. All the users are able to:
    1. see their User ID, Division, Type
    2. Update email address
    3. Reset password, as the initial password sent to email is generated randomly

The following screenshots display all the pages and available features. The labels of the screenshots correspond with the appropriate sections mentioned above. For example, **Fig. 1.1** refers to above mentioned 1.1 point (*All the users are able to login using user ID and password)*.

LOGIN PAGE

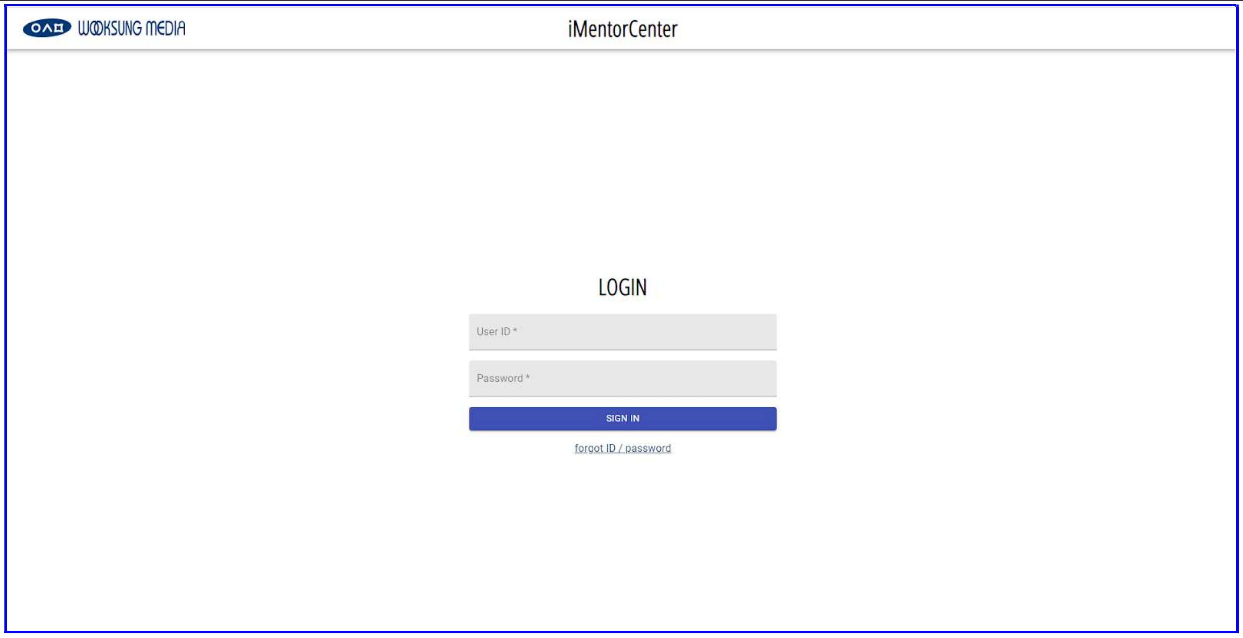
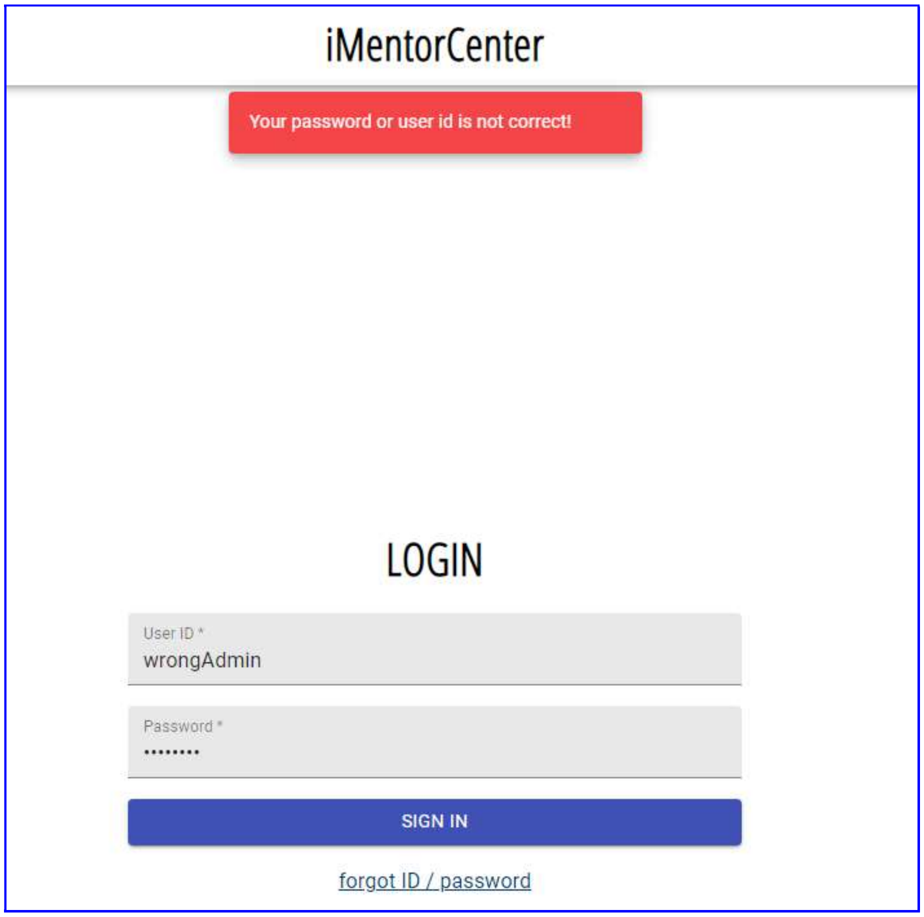


Fig. 1.1.

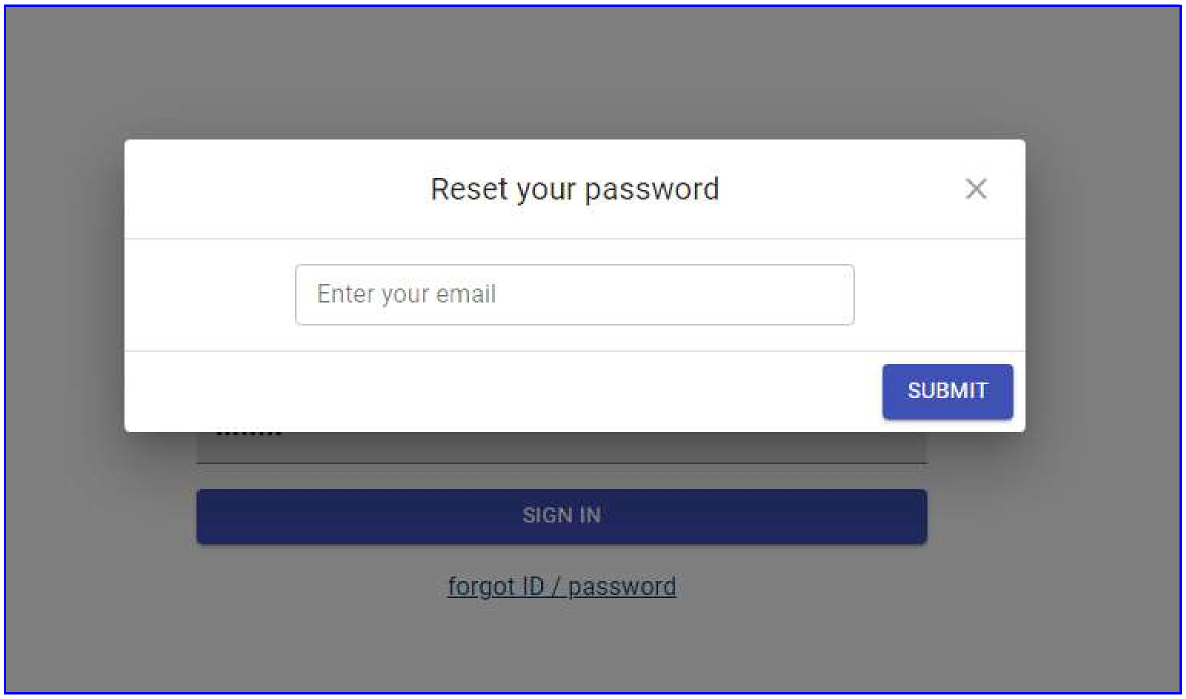


Your password or user id is not correct!

User ID * wrongAdmin

Password * ••••••••

Fig. 1.2.


Reset your password

Enter your email

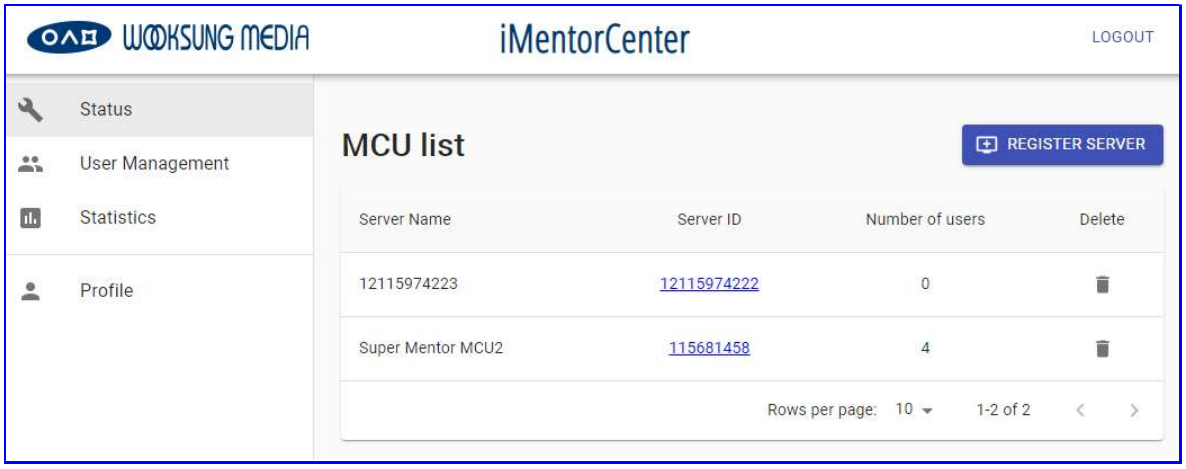
SUBMIT

SIGN IN

forgot ID / password

Fig. 1.3.

STATUS PAGE


WOOKSUNG MEDIA | iMentorCenter | LOGOUT

Status
User Management
Statistics
Profile

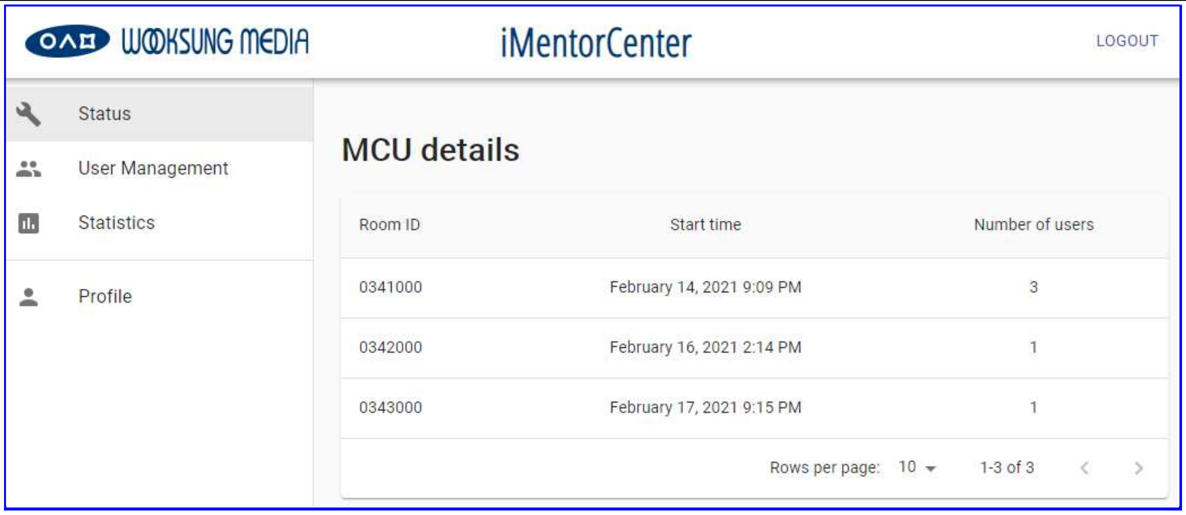
MCU list

REGISTER SERVER

Server Name	Server ID	Number of users	Delete
12115974223	12115974222	0	🗑
Super Mentor MCU2	115681458	4	🗑

Rows per page: 10 1-2 of 2

Fig 2.1.1.

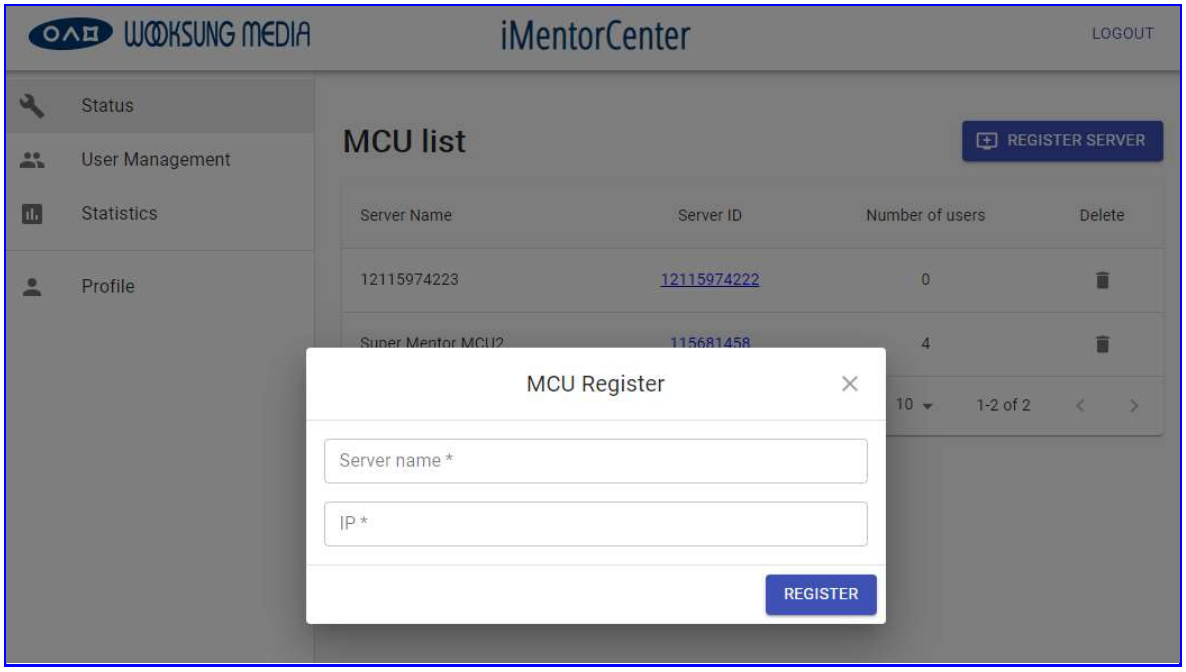


MCU details

Room ID	Start time	Number of users
0341000	February 14, 2021 9:09 PM	3
0342000	February 16, 2021 2:14 PM	1
0343000	February 17, 2021 9:15 PM	1

Rows per page: 10 1-3 of 3

Fig 2.1.2.



MCU list

Server Name	Server ID	Number of users	Delete
12115974223	12115974222	0	
Super Mentor MCU2	115681458	4	

10 1-2 of 2

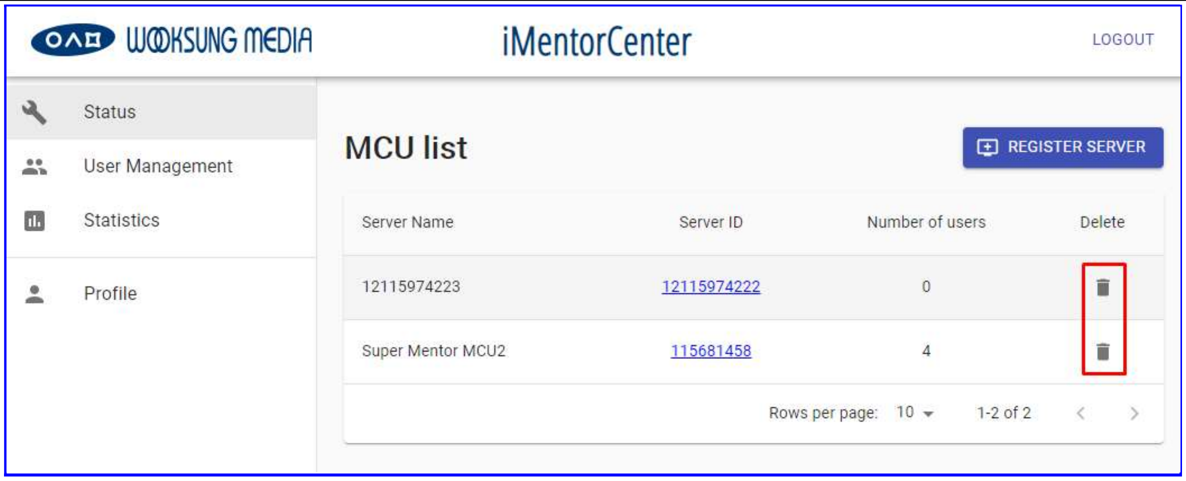
MCU Register

Server name *

IP *

REGISTER

Fig 2.1.3.(register)

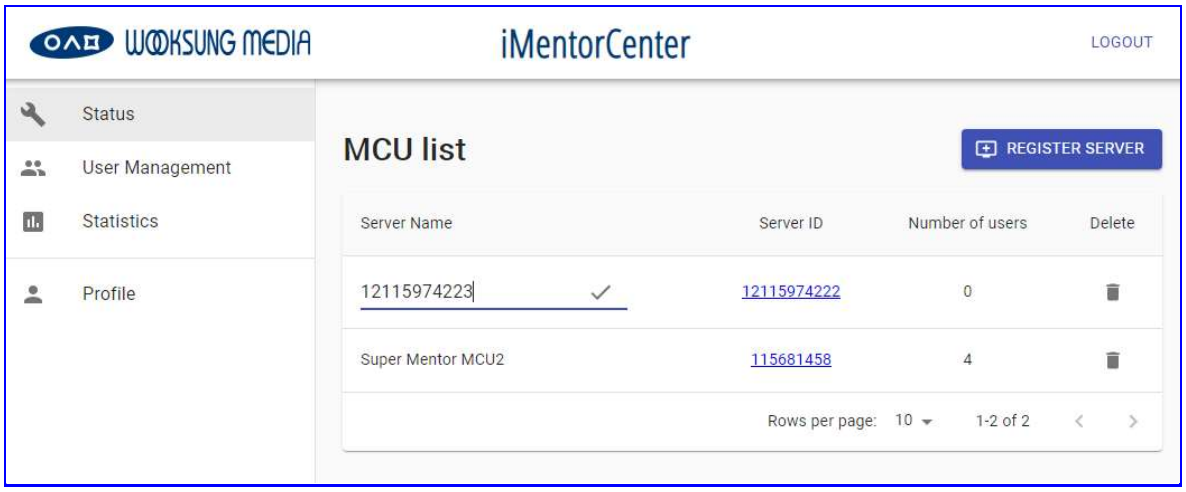


MCU list

Server Name	Server ID	Number of users	Delete
12115974223	12115974222	0	🗑
Super Mentor MCU2	115681458	4	🗑

Rows per page: 10 1-2 of 2

Fig 2.1.3.(delete)



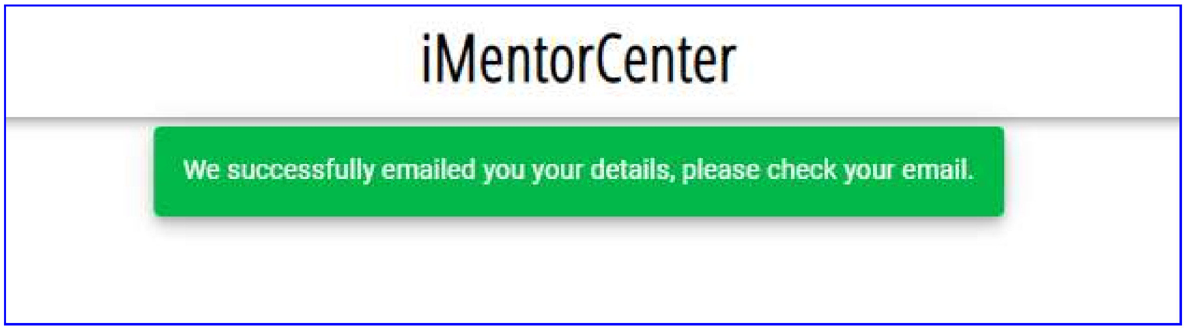
MCU list

Server Name	Server ID	Number of users	Delete
12115974223 ✓	12115974222	0	🗑
Super Mentor MCU2	115681458	4	🗑

Rows per page: 10 1-2 of 2

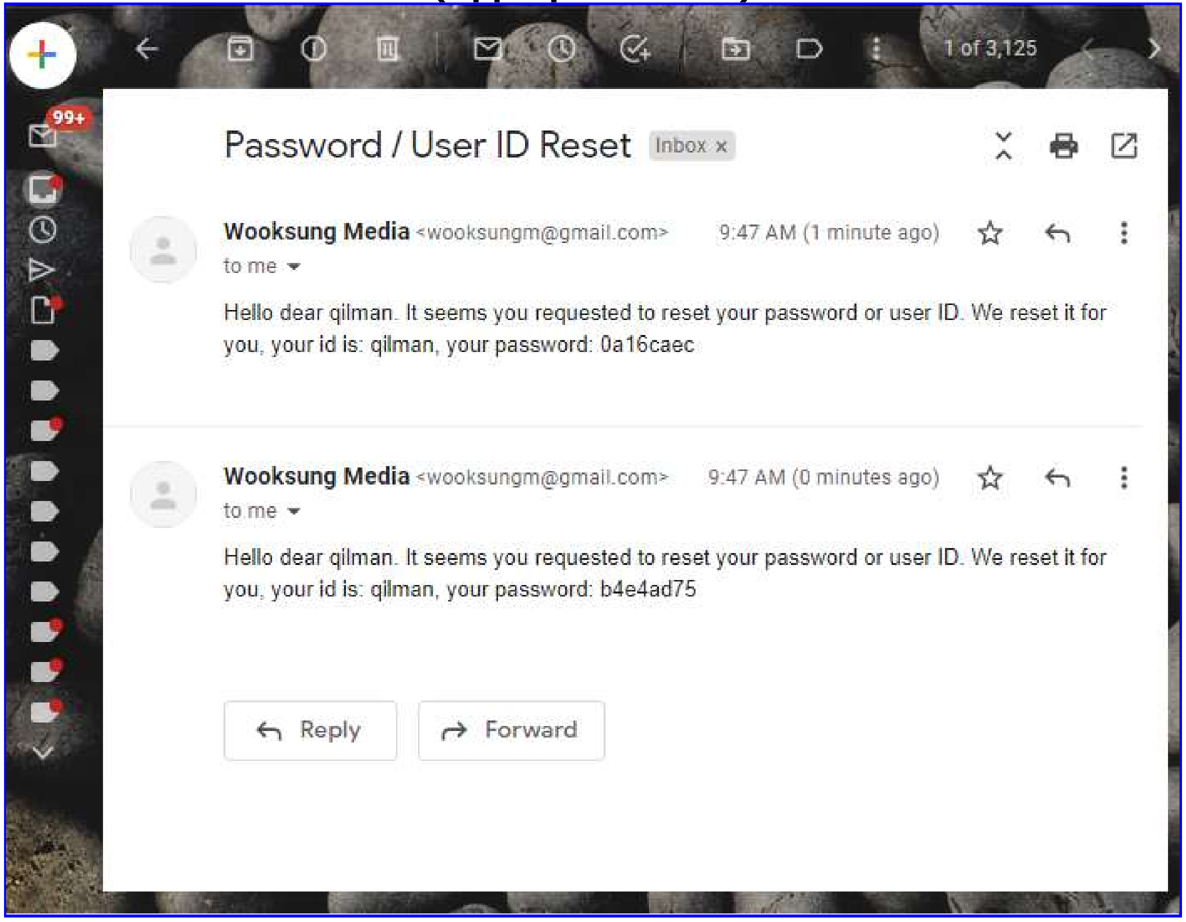
Fig 2.1.4.

USER MANAGEMENT


iMentorCenter

We successfully emailed you your details, please check your email.

(Appropriate alert)



Password / User ID Reset

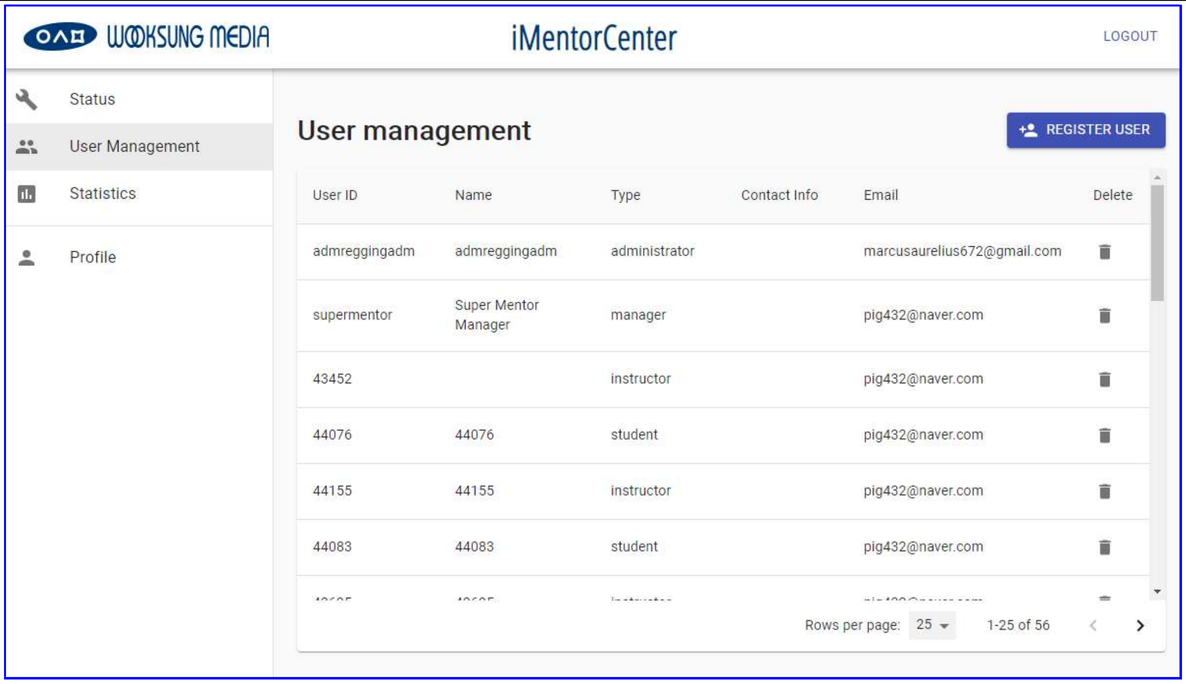
Wooksung Media <wooksungm@gmail.com> 9:47 AM (1 minute ago)
to me

Hello dear qilman. It seems you requested to reset your password or user ID. We reset it for you, your id is: qilman, your password: 0a16caec

Wooksung Media <wooksungm@gmail.com> 9:47 AM (0 minutes ago)
to me

Hello dear qilman. It seems you requested to reset your password or user ID. We reset it for you, your id is: qilman, your password: b4e4ad75

Fig. 3.1. (Gmail inbox)

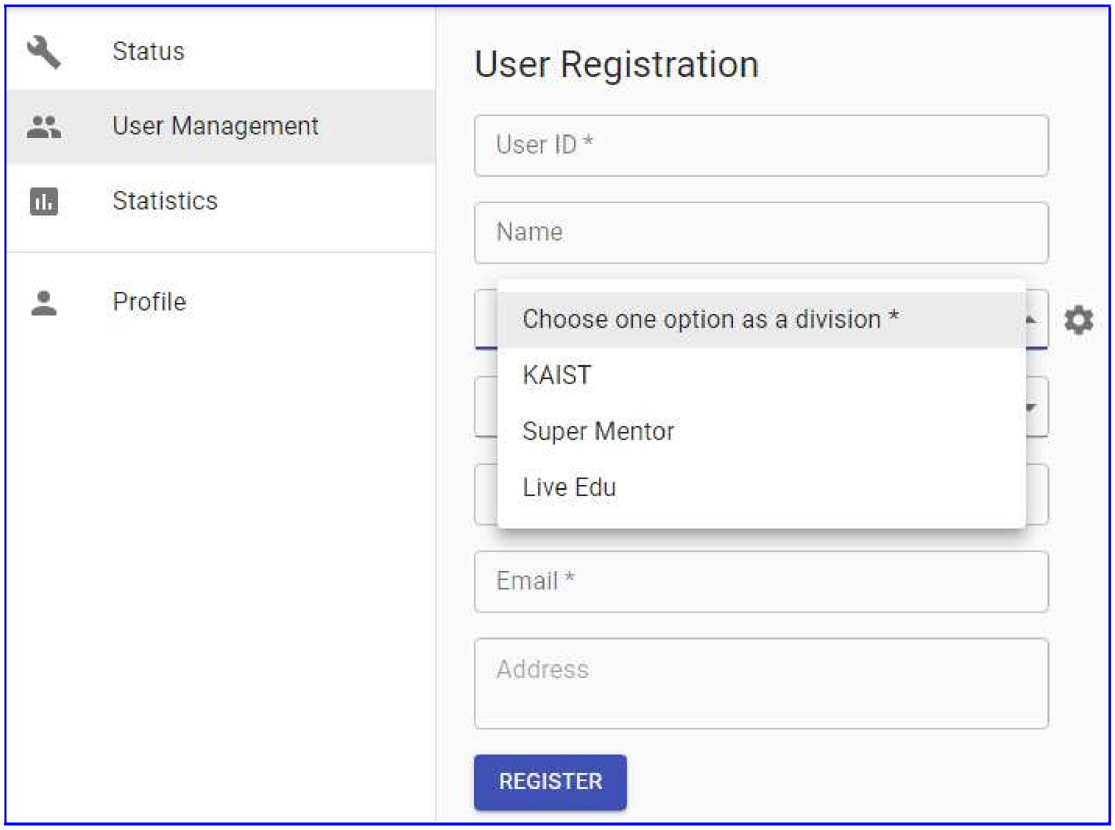


User management

User ID	Name	Type	Contact Info	Email	Delete
admreggingadm	admreggingadm	administrator		marcusaurelius672@gmail.com	
supermentor	Super Mentor Manager	manager		pig432@naver.com	
43452		instructor		pig432@naver.com	
44076	44076	student		pig432@naver.com	
44155	44155	instructor		pig432@naver.com	
44083	44083	student		pig432@naver.com	

Rows per page: 25 1-25 of 56

Fig. 3.2.



User Registration

User ID *

Name

Choose one option as a division *

KAIST

Super Mentor

Live Edu

Email *

Address

Fig.3.2.2.(any division)

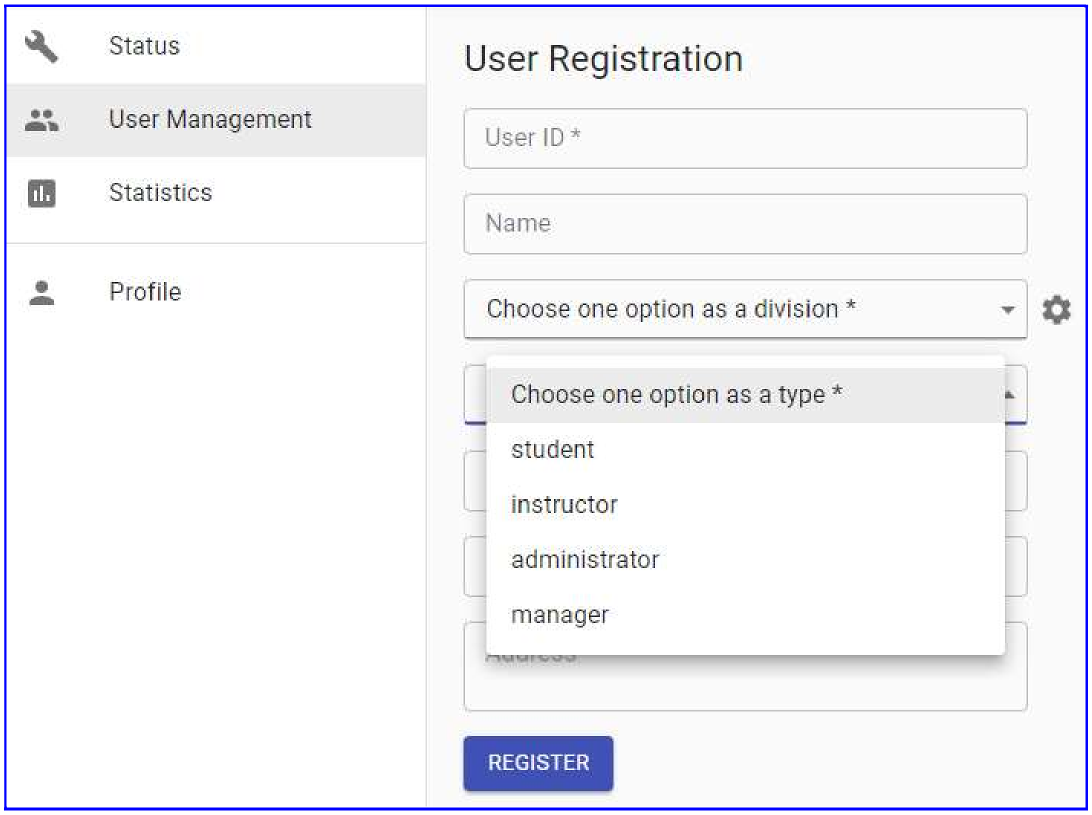


Fig.3.2.2.(any type)

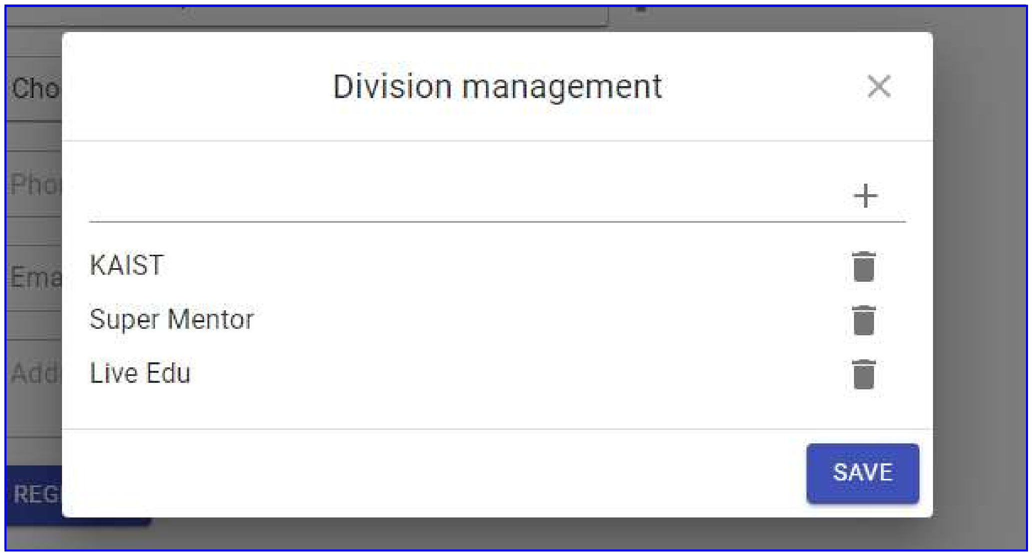


Fig.3.2.3.

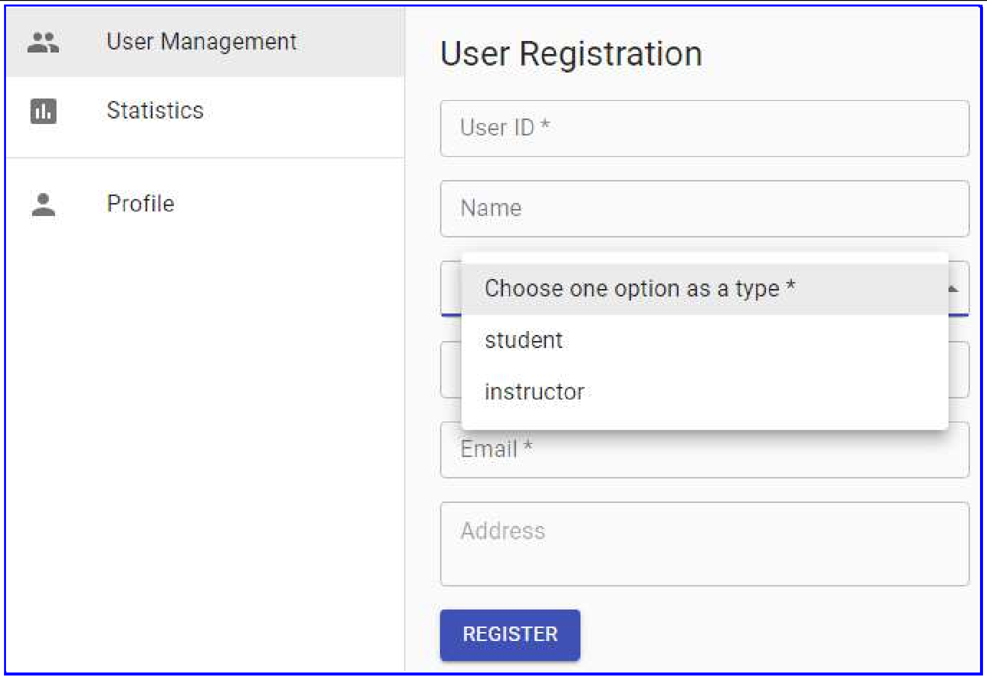


Fig.3.3.2.(the division is already determined and only instructors and students can be registered by managers)

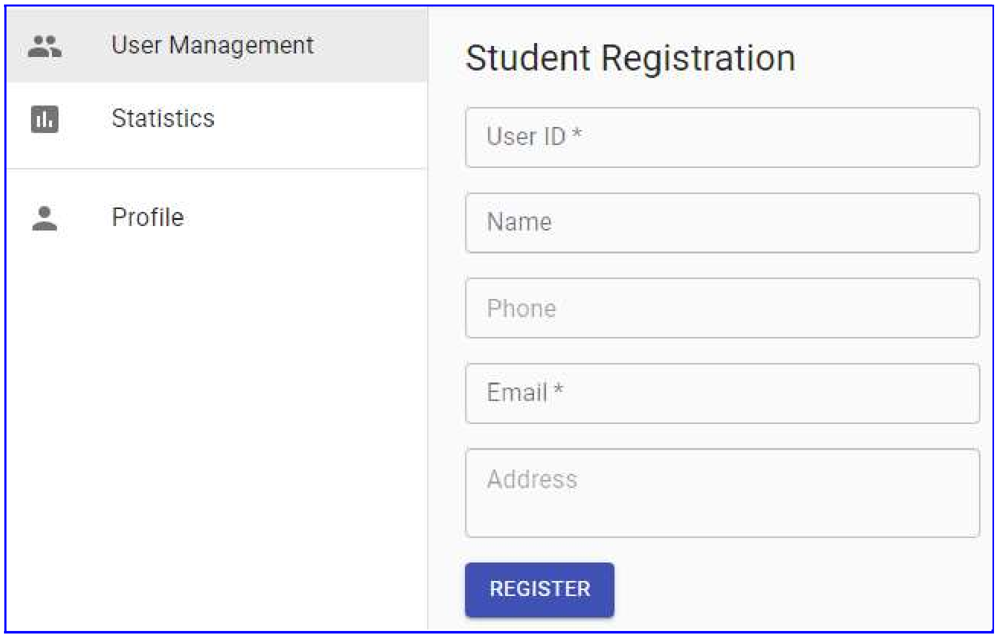
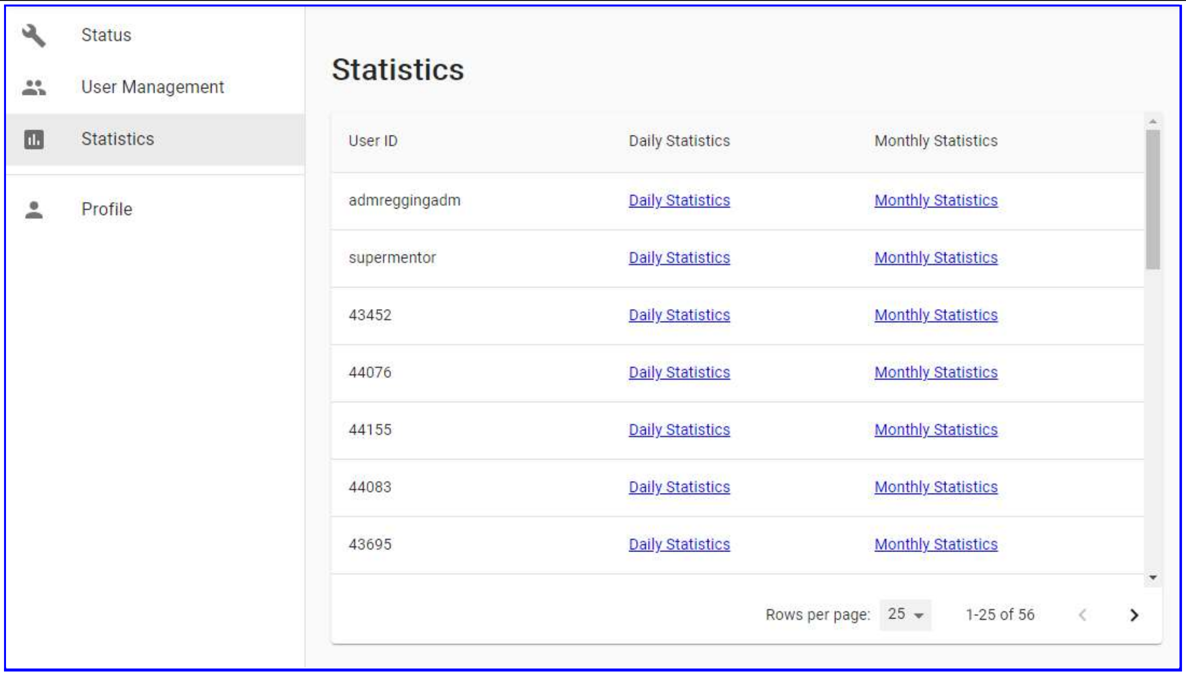


Fig.3.4.2.(both division and type is already determined for instructors)

<u>STATISTICS</u>

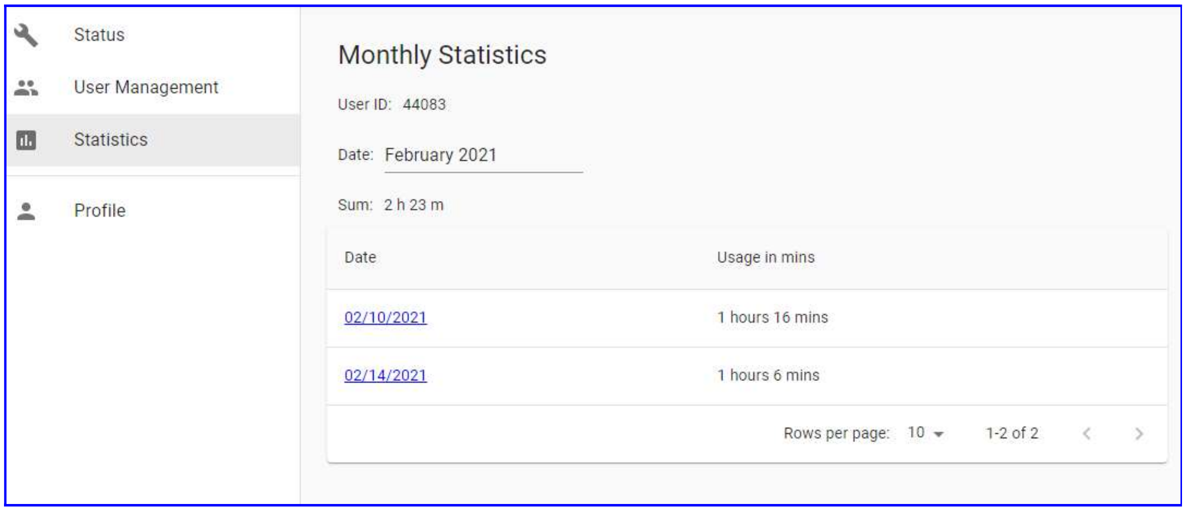


Statistics

User ID	Daily Statistics	Monthly Statistics
admreggingadm	Daily Statistics	Monthly Statistics
supermentor	Daily Statistics	Monthly Statistics
43452	Daily Statistics	Monthly Statistics
44076	Daily Statistics	Monthly Statistics
44155	Daily Statistics	Monthly Statistics
44083	Daily Statistics	Monthly Statistics
43695	Daily Statistics	Monthly Statistics

Rows per page: 25 1-25 of 56

Fig.4.2.



Monthly Statistics

User ID: 44083

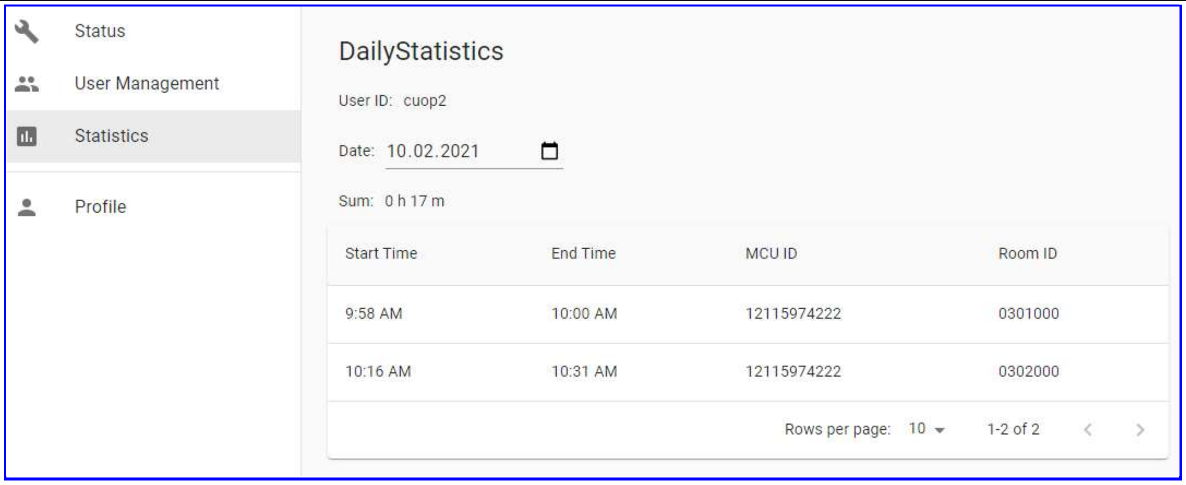
Date: February 2021

Sum: 2 h 23 m

Date	Usage in mins
02/10/2021	1 hours 16 mins
02/14/2021	1 hours 6 mins

Rows per page: 10 1-2 of 2

Fig.4.5.



DailyStatistics

User ID: cuop2

Date: 10.02.2021

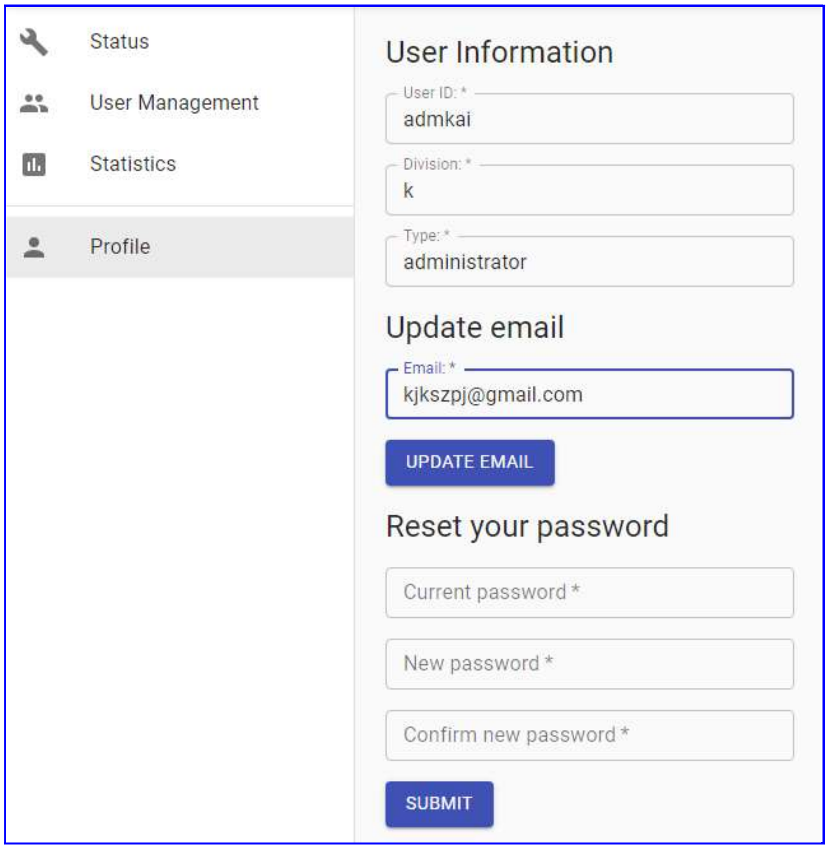
Sum: 0 h 17 m

Start Time	End Time	MCU ID	Room ID
9:58 AM	10:00 AM	12115974222	0301000
10:16 AM	10:31 AM	12115974222	0302000

Rows per page: 10 1-2 of 2

Fig.4.6.

PROFILE



User Information

User ID: * admkai

Division: * k

Type: * administrator

Update email

Email: * kjkszpj@gmail.com

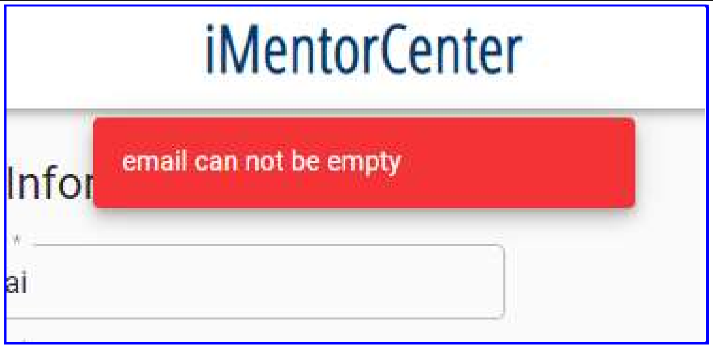
Reset your password

Current password *

New password *

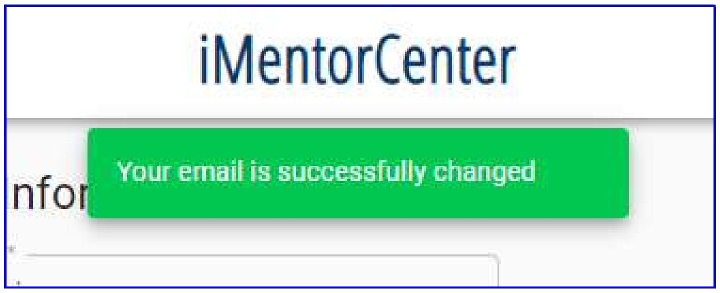
Confirm new password *

Fig.5.1.



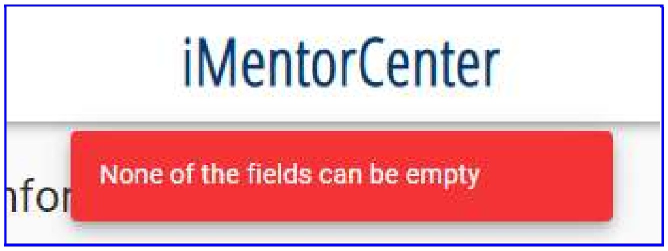
email can not be empty

Fig.5.1.(email update failure alert)



Your email is successfully changed

Fig.5.1.(email update success alert)



None of the fields can be empty

Fig.5.1.(password update failure alert, success alert is omitted)

Review and Suggestion

(More than 1 sheet of A4 paper)

Starting from the application process the internship seemed promising as the requirements included the fundamental knowledge only, and the need for use of only the basic tools. I have never had backend development experience before, and thus, I got a lot of new experience, because it was first of its kind. The flexibility that the mentor provided me was great, I used all the tools I thought were more modern and more popular. Everyday we did a call at 9 a.m. in the morning to discuss my current state, find answers to my questions, which included understanding the requirement specification document.

Although in my communication with my mentor there were language barriers we overcame this problem by dividing the big task in smaller and smaller, at times even tiny subtasks, and that allowed us articulate better. Each task would have specific story board made by him and we would proceed once I achieve the desired result.

Despite the process was going a bit slow during the first half of the internship, the mentor was quite optimistic and motivating and did not put me under pressure. We speeded up in the second half and finished almost all the tasks that were required in the project. On of the reasons why not all of the tasks were finished might be that the difficulties and scale of the tasks were overestimated for such a short-term internship.

I experienced Node.js with Express.js and MongoDB database with Mongoose.js. Before this internship I had some experience with React, but I never utilized Material-UI.js for my design purposes. Experiencing it during this internship brought me skills that would speed up the development process of not only my work, but also school projects. I have had machine learning internship before but I was indecisive as to which field of computer science to pursue, but now I think, the development of web applications is what I would enjoy to do in long period of time. And such combination of tools that I experienced during these 8 weeks is very popular for backend developers nowadays and I think this internship experience will open for me a lot of new doors into the frontend, backend or full-stack developer’s career.

I would definitely suggest my juniors and sophomores to take part in this internship, because it might require them to know just enough to start and experience some real-life work, as was the case with me. It is way better than falling into a project which is so big that one does

not even know how to start.
Also, not only the application process is quite well-organized, but also there is less bureaucratic requirements such as collecting lots of documents with signatures for visiting the immigration office.