Subject Name: Enterprise Web Software Development

Subject Code: COMP1640

Tutor: Dzung Lai Manh

Project name: eTutoring

Team Members:

Phan Ngoc Hoang: 001126464

Nguyen Ngoc Hai: 001126325

Dinh Ngoc Tram: 001126093

Ngo Quang Huy: 001126329

Tran Quoc Viet: 001126103

Tran Dinh Thu: 001127405

Table of Contents

I. Introduction:

Based on the design requirements management system for a university, our team has been established to develop and build eTutor system. The project was developed on a request that included three main users: staff, student and personal tutor.

Role	Member
Product Owner	Phan Ngoc Hoang
Database Design	Phan Ngoc Hoang, Nguyen Ngoc Hai, Tran Dinh Thu, Ngo Quang Huy
Web Design	Front-end: Ngo Quang Huy, Tran Quoc Viet Back-end: Phan Ngoc Hoang, Nguyen Ngoc Hai, Tran Dinh Thu
Tester	Dinh Ngoc Tram, Tran Quoc Viet
Scrum master	Dinh Ngoc Tram

URL Source code of website:

https://drive.google.com/drive/folders/1_s94-Sak96bWJBury9LRqwwKTBXwFnm-?usp=sharing

URL Presentation: https://drive.google.com/drive/folders/1JPSexTmA_MwjK8WBpkjfAPmdWX-jnPDq?fbclid=IwAR1sjMzBPdAlvWZInE3Zu6yTmkDMHO8cvvzb_hYkqG91fu9QHUzWNZlRtDo

URL Functional demo video: https://www.youtube.com/watch?v=xEWysSGTWXw

URL Group repository: https://github.com/PhanNgocHoang/CMS

URL Documentation: https://drive.google.com/open?id=1B9rtlI2VeL9TQeGA0YlUQ7L2SZ5UFzd-

URL Site: https://demoteam.herokuapp.com/

Tools:

Why choose NodeJs:

- Nodejs is a platform built on the Javascript V8 engine be developed by Google and Nodejs is open source, it is completely free. Through the Javascript V8 engine, it helps to compile Javascript functions so that the server can understand and execute. (Romanyuk, 2019)

Reasons to choose NodeJs: (Romanyuk, 2019)

- Nodejs has open-source, it's completely free.
- The user community is large so it can support to solve some difficulties when using.
- There are many packages, this provides many solutions to solve problems thereby creating many development conditions.
- Has good performance due to the use of the Javascript V8 engine developed by Google and continuously improved in terms of performance.

Why choose NoSQL and MongoDB:

- NoSQL database is a database that does not have data-binding constraints such as SQL relational databases, NoSQL allows storing various types of data and allows users to easily expand horizontally. (Seda Unal, Yuchen Zheng, 2017)

Reasons to choose NoSQL: (Seda Unal, Yuchen Zheng, 2017)

- High extensibility due to no strict data constraints and storage structure.
- Good query performance because ignoring data constraints
- Ability to store large data
- Reasons to choose MongoDB: (Hooda, n.d.)
- MongoDB is capable of handling large volumes of data well.
- High query speed
- Easy to manage and use

II. Database:

1. Database Design:

Unlike the SQL database management systems that save data to tables, Mongodb saves data into the "Collection" and in each "Collection" that has the document. Each of these documents can have a different number of data fields, and their size can be different, and they are saved as JSON. (MongoDB, 2020). We use the One-To-Many Design Model and combine embedded relationships and refer, this model where Collections will use references to other Collections and in a Collection can be subcollections.

For the One-To-Many Design Model with reference relationship we use to connect Collections to ensure data interact with each other via _id generated by MongoDB automatically when adding new Documents and avoiding repeat the information.

One-To-Many design model with the embedded relationship we use in storing information exchanged between personal tutor and student.

Combining both relationships and embedded types will help the database improve querying capabilities, but it will also make the size of the documents too large.

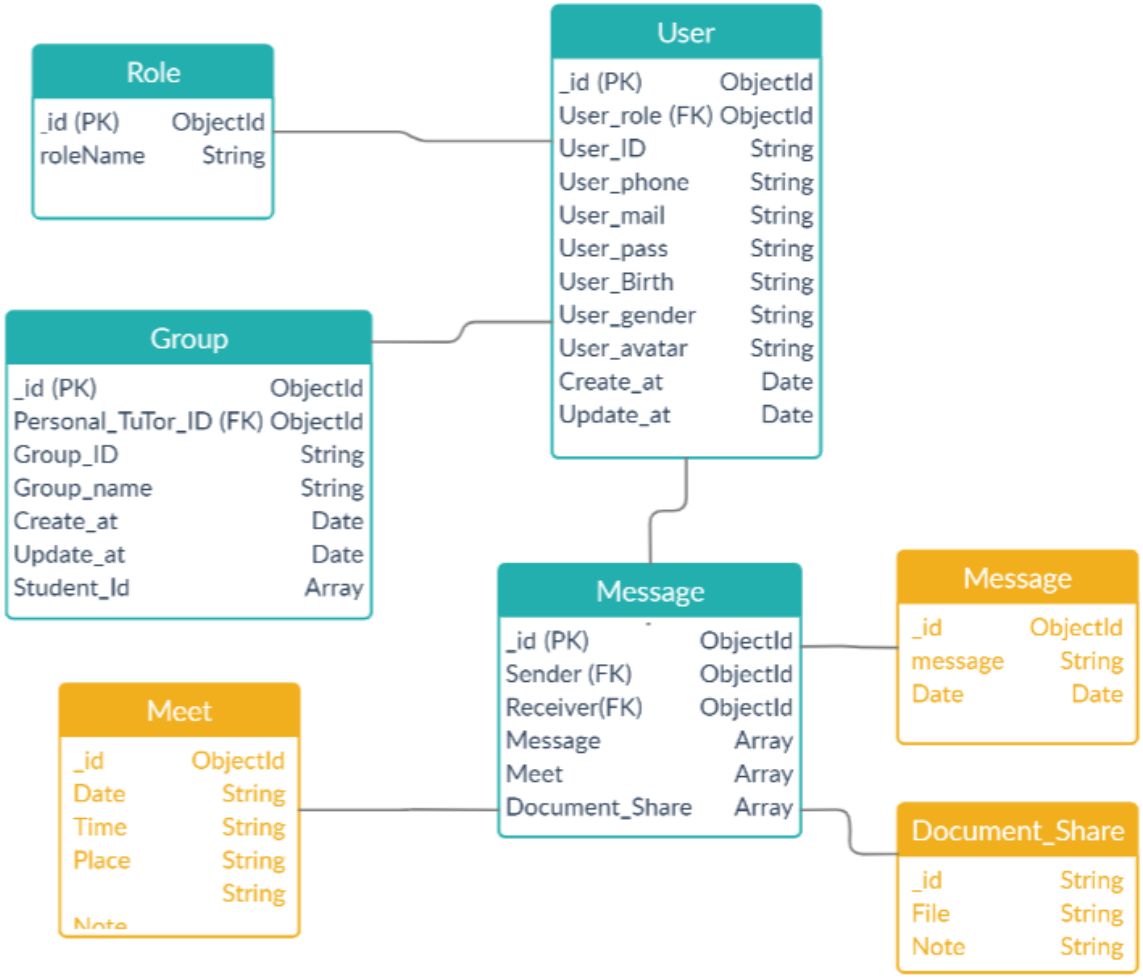

Figure 1 Design Database

In the database of this project we designed a database that consists of 4 collections:

- Collection Role: Used to store information about the role of each user, in the Role Collection there are 2 data fields that are _id and roleName. The _id field is saved as an ObjectId created by MongoDB and automatically incremented by adding a new Document in the Collection Role, the roleName field is used to store the name of the role with the String data type.
- User Collection: Used to save information of users, User_ID, User_phone, User_mail, User_pass, User_birth, User_gender, User_avatar fields to store ID, phone, mail, password, day data of birth, gender, user avatar. The two fields Create_at and Update_at are used to

store information about creation and update times of users. The _id field is saved as an ObjectId created by MongoDB and automatically incremented by adding a new Document in the User Collection. The User_role field is used to store the _id of the roles that are referenced to the Collection Role.

- Collection Group: Used to store information related to the list of students of each individual tutor-supported. Like the two collections above, the _id data field is saved as an ObjectId created by MongoDB and automatically increases when adding a new Document in Collection Group, two fields of Group_ID and Group_name are used to store the IDs of each list and the name of each list. Book set by Staff for easy management, the Personal_Tutor_ID field is used to store the _id of each Personal tutor and the data type is ObjectId and is referenced to the Collection User, the Student_Id field will store a sequence of _id of the Student as String.

- Collection Message: Used to store information exchanged between personal tutor and Student. The Sender and Receiver fields are used to store the _id of the sender and the receiver and are saved as ObjectId.

    - Message field is used to store sender's messages, Message field will save messages as array, each element in array will be stored as Object and each Object will contain _id components in ObjectId form and it will automatically increment to distinguish the objects, the message is saved as a string and it is used to store the content of the message and finally the Date is saved as a Date to store the sending time of the message.
    - Meet data field is used to store appointment information and it will store appointments in array and as object, each object will include _id, Date components to save the time of appointment, it includes including day, month and year, Time is used to save information related to the specific time of the appointment, Place is used to save the location of the appointment, and finally Note is used to save notes about the appointment.

- The Document_Share data field is used to store documents exchanged between personal tutors and students similar to the Meet and Message fields, it will save the related documents exchanged in array and in object form, each object will be including _id elements saved as ObjectId, File with a data type of String used to store file names, Note saved as String and will store comments related to the file.

2. Security data:

We use cookies and token to encrypt the information used to authenticate users.

```
const user = {
  user_id: docs._id,
  user_mail: docs.User_mail,
  user_role: docs.User_role,
};
let token = jwt.sign({ user: user }, "Team2DevelopmentCms", {
  algorithm: "HS256",
  expiresIn: "3h",
});
```

Figure 2 Convert user information into token code

The code above we use to convert user information into a token code to protect user information. To convert user information into token code, the data used for encryption includes "_id", "User_mail", "User_role", they are used to authenticate users and delegate rights to users, Signatures. of token code is "Team2DevelopmentCms", the algorithm used to code is "HS256", the survival time is 3 hours.

```
res.cookie("user", token, { maxAge: 10800000, signed: true, secure: true, httpOnly: true, domain: "demoteam.herokuapp.com"});
```

Figure 3 Send token to user via cookies

After creating the token, we use cookies to send the token to the server.

When creating the cookies, they pass the name of the cookie ("user") of the cookie data which is the generated token code. But sending information via cookies can be stolen through fake websites so we have set a number of options for our cookies to increase the safety of cookies, the "maxAge" option is time-consuming. The cookies are now in milliseconds and 108,000,000

mini seconds change to 3 hours, the "signed" option is to use cookies with the signature and the cookie signature for us is "123@123 @!T".

```
app.use(cookieParser("123@123@!T"))
```

Figure 4 Signature of cookies

The "secure: true" option is to ensure cookies are only sent to https addresses, the "httpOnly: true" option is to minimize XSS attacks. We also use the "domain" option to protect users 'cookies, using the "domain" option will help ensure that users' cookies will only be sent to the system domain to minimize the possibility of cookies by the user. Stolen users.

To increase the security after cookies are sent to the server, we check the existence of cookies, decode the token stored in cookies, and check the token information from the token with the user information stored in the cookie. database. In case the token cannot be decoded or the information in the token does not exist in the database, we send the sender information about the login page of the system, otherwise, the token is decoded and the information stored in it exists. In the database, we will let them through.

```
async function checkAth(req, res, next) {
  let token = req.signedCookies.user;
  if (!token) {
    return res.redirect("/login");
  }
  Complexity is 9 It's time to do something...
  let userId = jwt.verify(token, "Team2DevelopmentCms", (err, decode) => {
    if (err) return res.redirect("/login");
    global.user_ifo = decode.user;
    Complexity is 6 It's time to do something...
    Models.UserModel.find({ _id: decode.user.user_id }).exec((err, docs) => {
      if(err) return res.redirect("/login");
      if (docs == null) {
        return res.redirect("/login");
      }
      res.locals.user = docs;
      global.User_id = docs._id
      return next();
    });
  });
}
```

Figure 5 Check cookies

After passing the cookies test, we continue to identify the sender's role so that we can take them to the website pages and the features they can use on the system. This check ensures each user is using the right features for Role.

```
function CheckStaff(req, res, next) {
  Complexity is 4 Everything is cool!
  Models.RoleModel.findById({ _id: user_ifo.user_role }).exec((err, role) => {
    if (role.roleName === "Staff") {
      return next();
    }
    return res.redirect("/login");
  });
}
Complexity is 7 It's time to do something...
function CheckTutorAndStudent(req, res, next) {
  Complexity is 6 It's time to do something...
  Models.RoleModel.findById({ _id: user_ifo.user_role }).exec((err, role) => {
    if (role.roleName === "Personal Tutor") {
      res.locals.role = role.roleName;
      return next();
    }
    if(role.roleName === "Student")
    {
      res.locals.role = role.roleName;
      return next();
    }
     return res.redirect("/login");
  });
}
```

Figure 6 Check user role

To prevent users from entering the wrong data type for each field, we validated the entire data properly for each type. For example:

Users will not be allowed to enter special characters and numbers for the Full Name field:

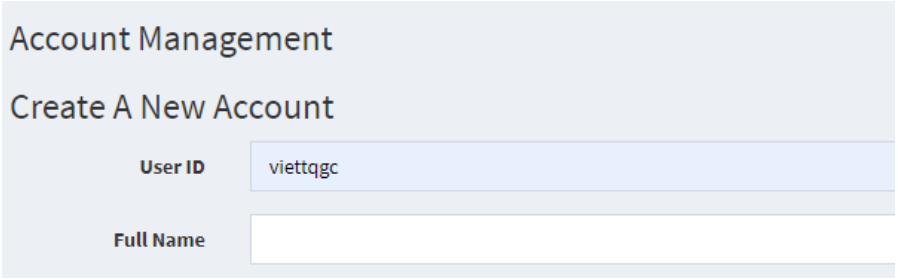

Figure 7 Full Name Validated

For passwords, the data will be fully encrypted and the password when entered must include uppercase letters, at least 1 special character and 1 numeric character.

Figure 8 Password Validated

We use Regular Expression (Regex), which is also known as the regular expression used to handle advanced strings through its own expressions. These expressions have their own rules. We have used the "pattern" attribute in combination with the regex to perform a validate form on the input tags. In addition, we also use the max length attribute to specify the number of characters that a user is allowed to enter in some special fields. In some special fields like this, the user must enter data without emptying compression, we apply the available attribute which is "required".

```
<div class="form-group">
    <label class="col-md-2 col-sm-3 col-xs-12 control-label">Password</label>
    <div class="col-md-10 col-sm-9 col-xs-12">
        <input type="password" class="form-control" name="User_pass" required pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}"
        title="Password must be at least 8 characters, at least 1 uppercase letter and 1 lowercase letter!">
    </div>
</div>
```

Figure 9 Regex Validate Password

This is the password field: "(? =. * \ D) (? =. * [Az]) (? =. * [AZ]). {8,}" => This is a regex expression we have to use to validate the password. In addition, we also use the "title" attribute to notify the correct format of the password that the user should enter when entering the wrong password.

```
<div class="form-group">
    <label class="col-md-2 col-sm-3 col-xs-12 control-label">Phone</label>
    <div class="col-md-10 col-sm-9 col-xss-12">
        <input type="text" class="form-control" name="User_phone" pattern="(\+84|0)\d{9,10}"
        title="Please follow vietnamese phone number, max 12 character: +84..../ 0...." required>
    </div>
</div>
```

Figure 10 Regex validate phone

This is the field to enter the phone number: "(\ +84 | 0) \ d {9,10}" => This is a regex expression that only allows users to enter the phone number with the first +84 or 0. If If the user enters the wrong format, the system will notify "Please follow Vietnamese phone number, max 12 character: + 84 ... / 0 ...", displayed by the "title" attribute. In this field, we also use the required attribute to not allow the user to leave it blank.

The above are some of the most practical examples of user data security that we have used and developed.

II. Website Design:

1. Student and Personal Tutor Interface:

Home Page

First, we will draw the homepage as required. For each different user, we will have different interfaces for them. According to the initial requirements, the system must be accessible on many different devices such as phones, laptops (PC), and tablets. We have drawn a ready-made interface for different screen types with different position changes and interface sizes. The site has been optimized for the responsive interface.

Laptop, PC:

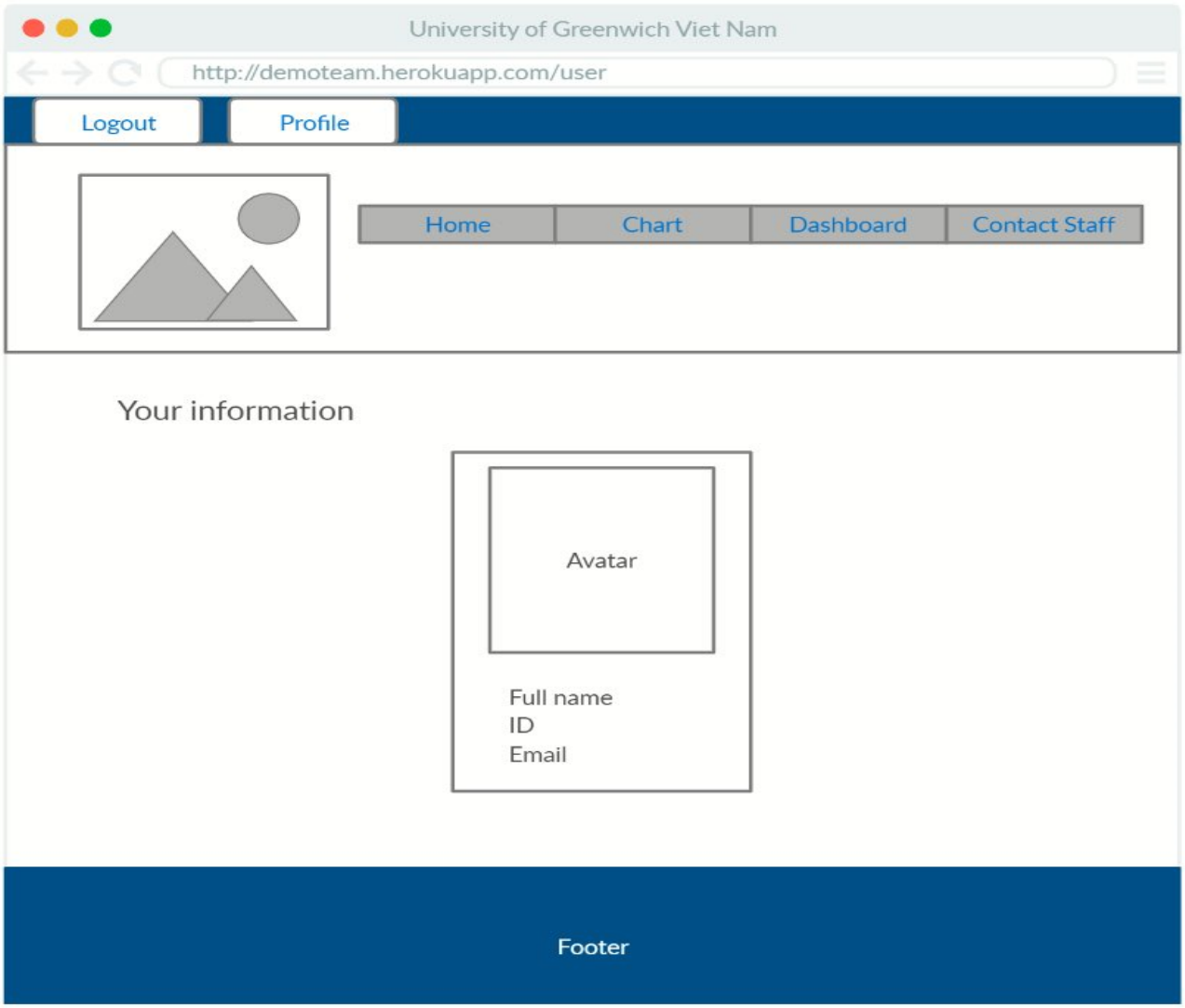

Figure 11 Wireframe Home Page of Student and Personal Tutor on Laptop

Tablet and Mobile:

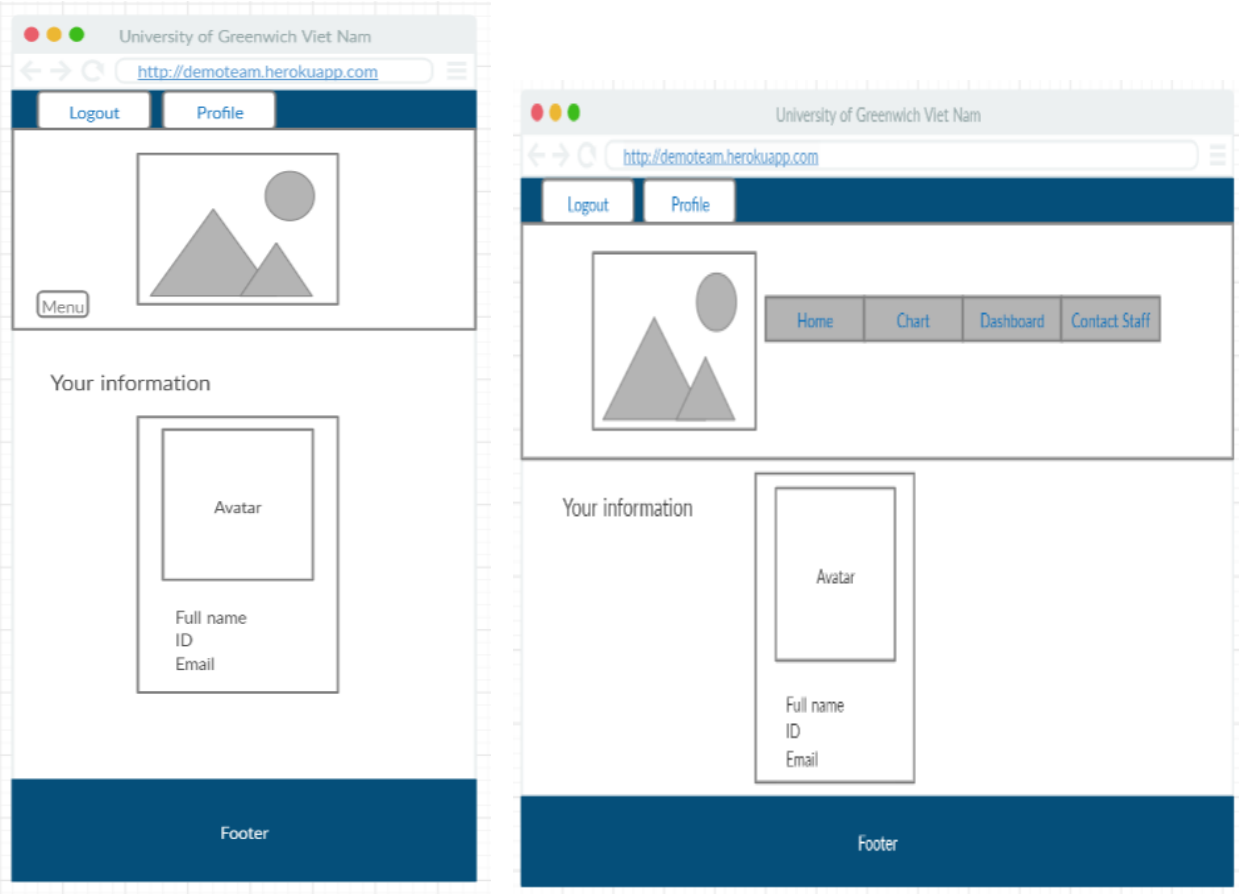

Figure 12 Wireframe Home Page of Student and Personal Tutor on Tablet, Mobile

Message Page

This is a drawing of the interface of the messaging function as required by the customer. The interface of this page is used for students and personal tutors. According to customer requirements, any interface must be used on many types of devices such as phones, laptops (PCs), tablets. So, this interface we have also designed so that it can meet different types of devices. They will vary in size and position of the components on the website according to the screen size of different devices.

Laptop, PC:

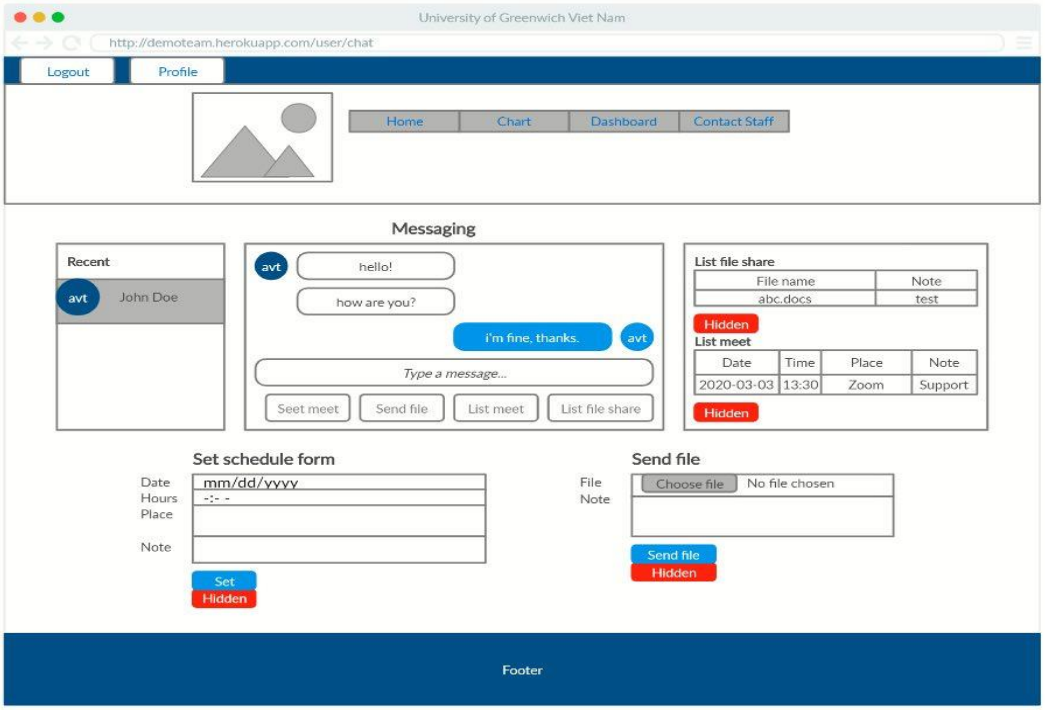

Figure 13 Wireframe Message Page of Student and Personal Tutor on Laptop

Tablet and Mobile:

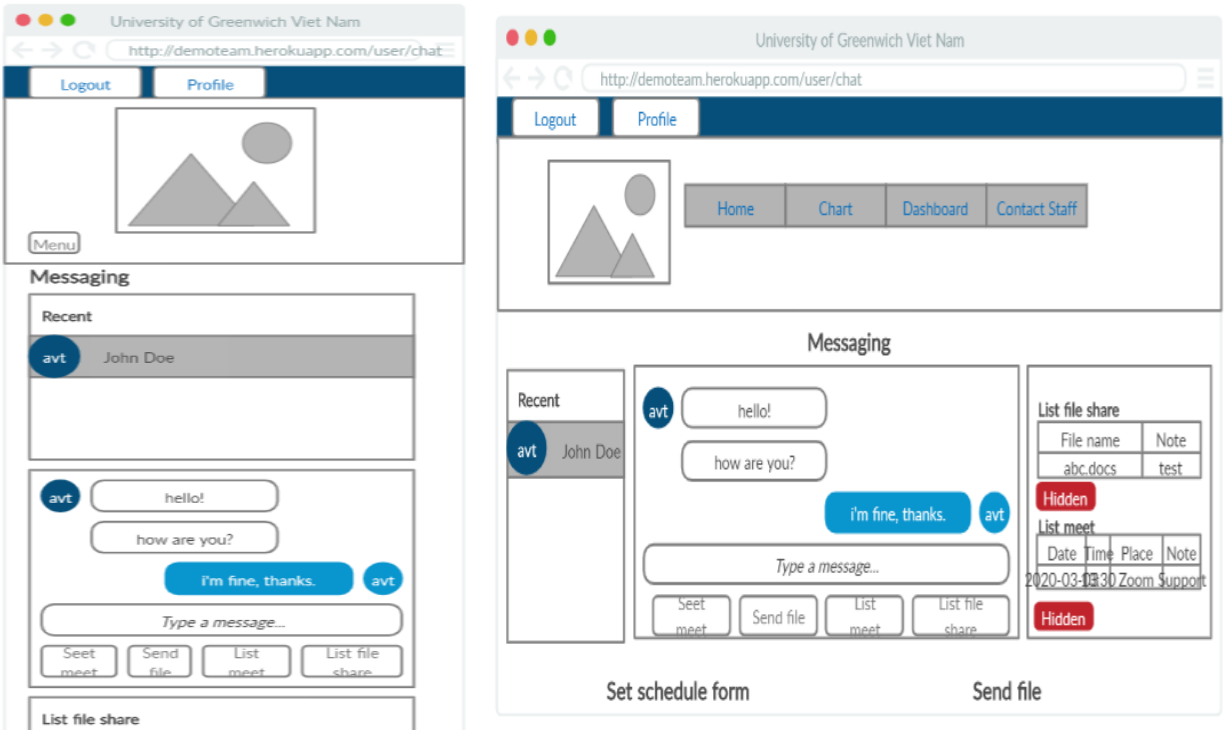

Figure 14 Wireframe Message Page of Student and Personal Tutor on Tablet, Mobile

For the interface of tablets and phones, the position of the components on the website will change to accommodate the device screen size. So, there are some interfaces we draw that do not show the footer but the fact that the footer has been pushed down below. When the user drags down, the footer will automatically show up.

2. Staff Interface:

We will design a number of main interfaces according to customer requirements, along with the interface displayed on the different screen types of each function.

Home Page

For the staff interface, we also designed the first homepage interface. Here users can see the functions that they can use. We set it up so that users can easily understand. According to customer requirements, we have designed the interface of the application to run on different types of devices like the interface of student and personal tutor. The devices that are supported are laptop (PC), phone, and tablet.

Laptop, PC:

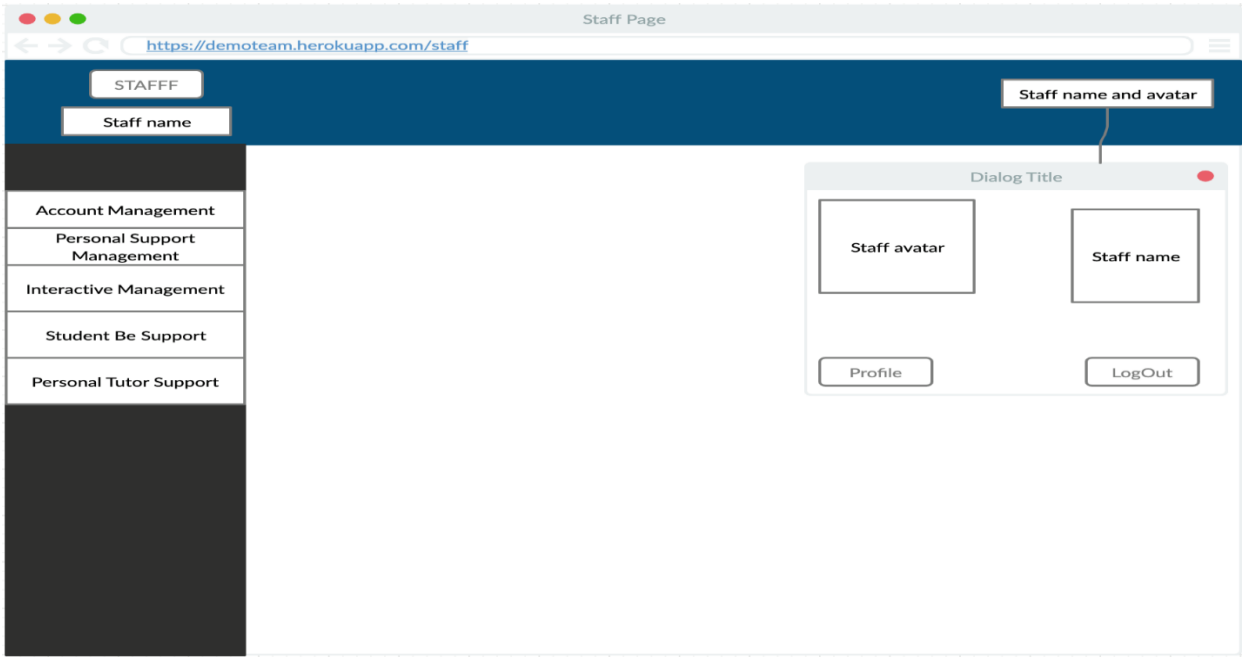

Figure 15 Wireframe Home Page of Staff on Laptop

Tablet and Mobile:

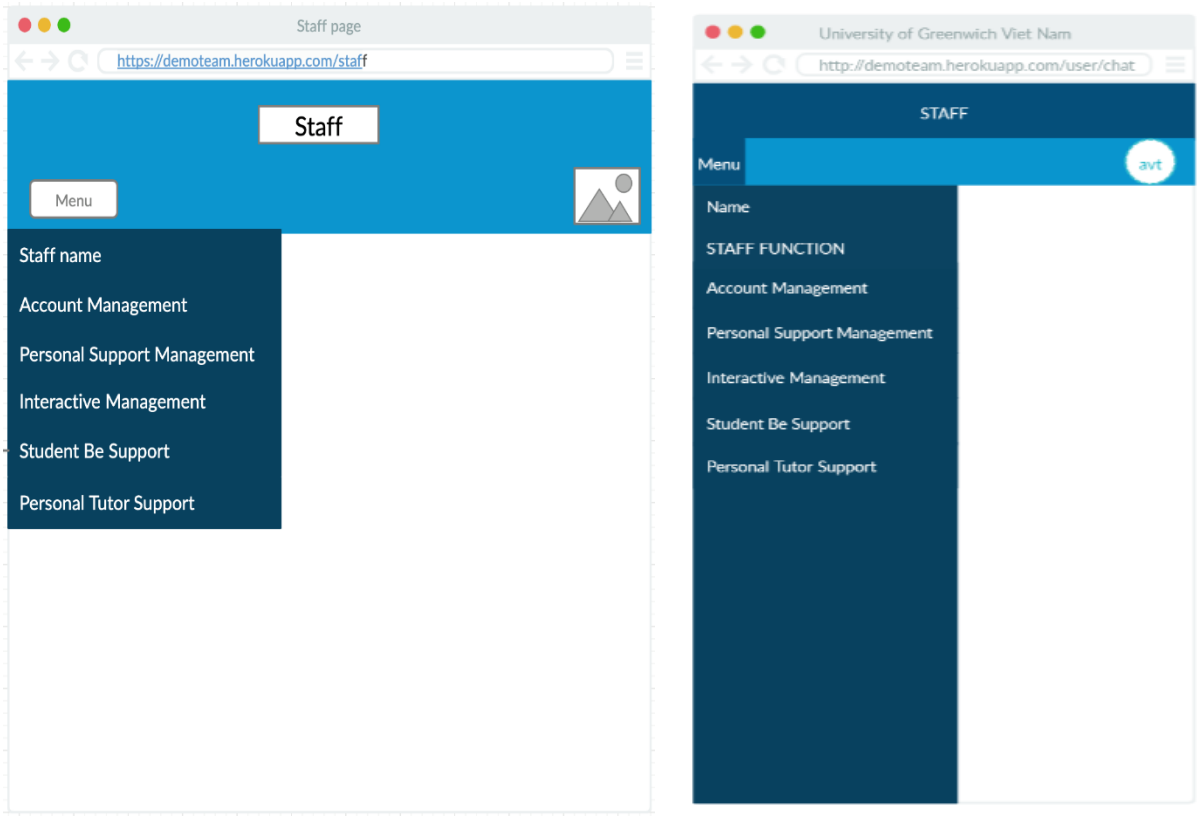

Figure 16 Wireframe Home Page of Staff on Tablet, Mobile

Group Management (Personal Support Management)

At the request of the customer, we must design a page to delegate the personal tutor to manage our students. This page has the same interface as the account management interface. It includes function buttons such as Create, Detail, Updates, Delete. Therefore, I only demo the wireframe interface of the group management page according to customer requirements. Along with that is the interface displayed on different types of devices such as a laptop (PC), mobile, tablet.

Laptop, PC:

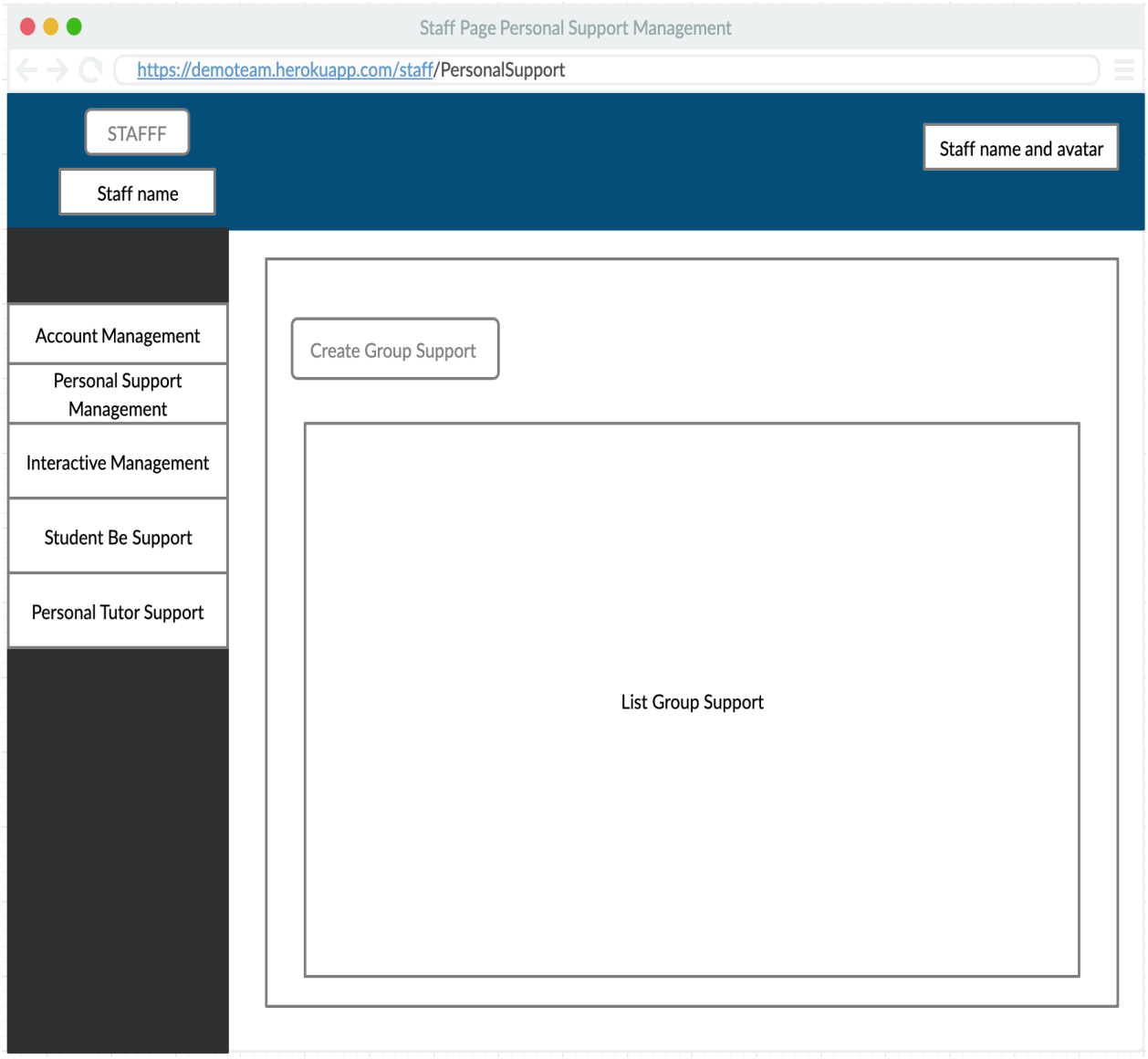

Figure 17 Wireframe of Manage Group Support Page of Staff on Laptop

Tablet and Mobile:

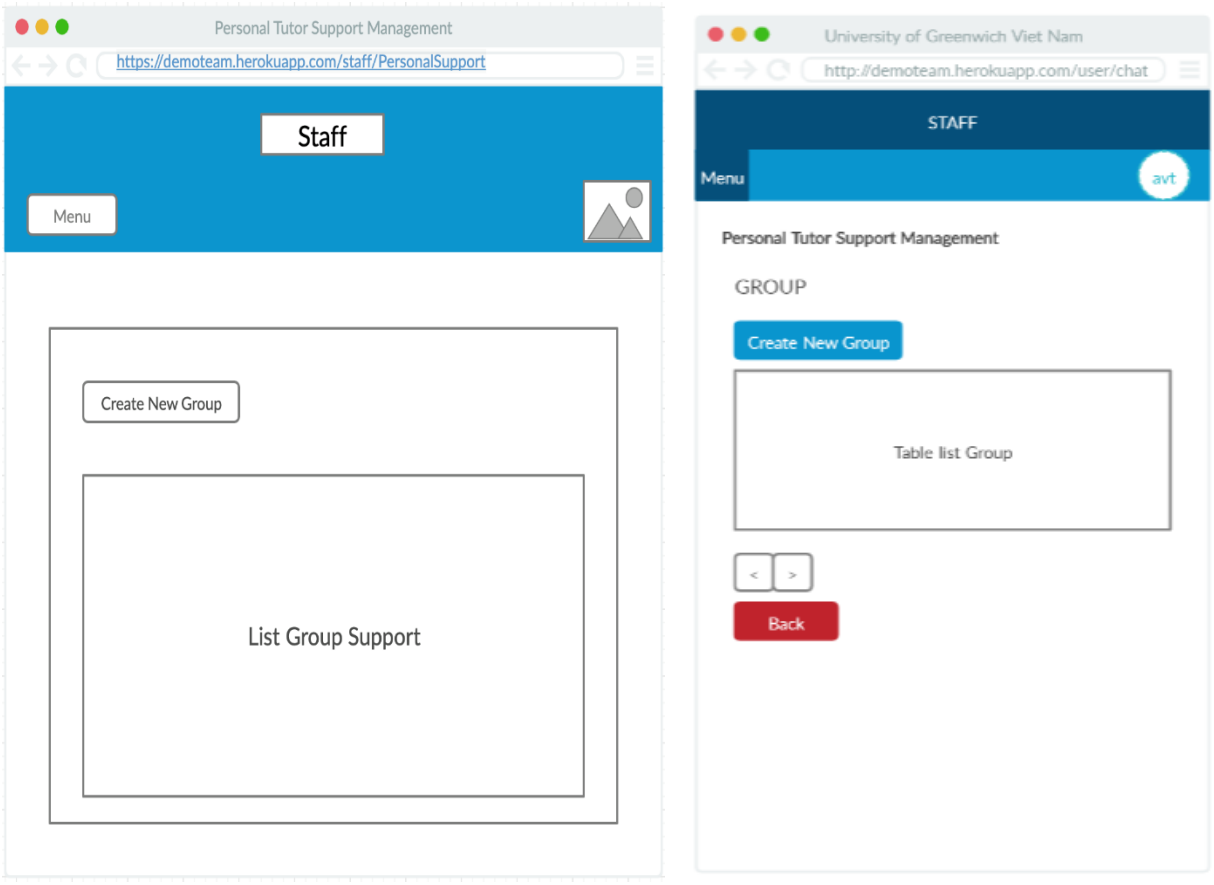

Figure 18 Wireframe Personal Support Management Page of Staff on Tablet, Mobile

3. Testing in different screen resolution

This is the interface that has been designed and put into use by us. The interface can run on different device types as introduced above. Depending on the size of the device screen, the components on the interface will be reduced in length and length. To a certain size allowed, the parts will be pushed down and displayed vertically. Users can drag the screen down to view. Here we will show some of the main interfaces of each user, the remaining interfaces are similar.

Message Page Interface for Student and Personal Tutor

Laptop, PC:

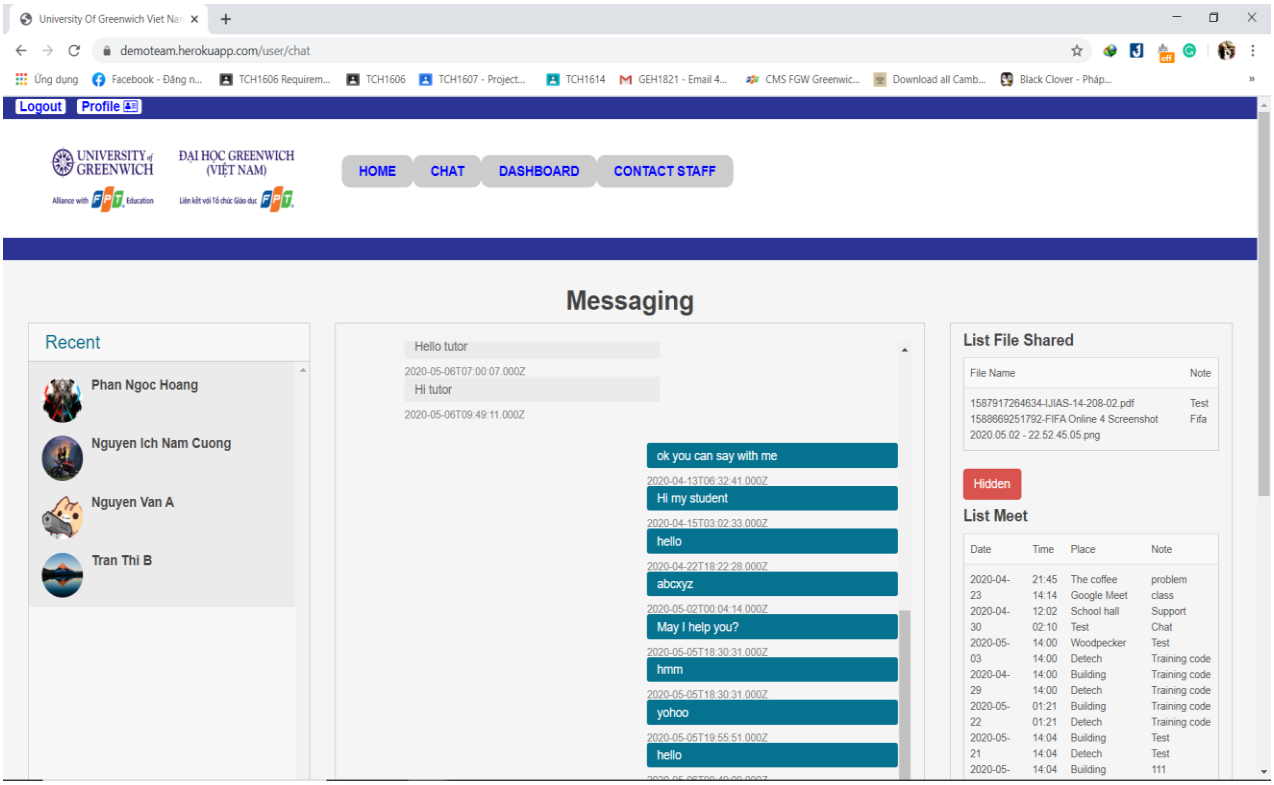

Figure 19 Interface after Design of Message Page on Laptop

Tablet and Mobile:

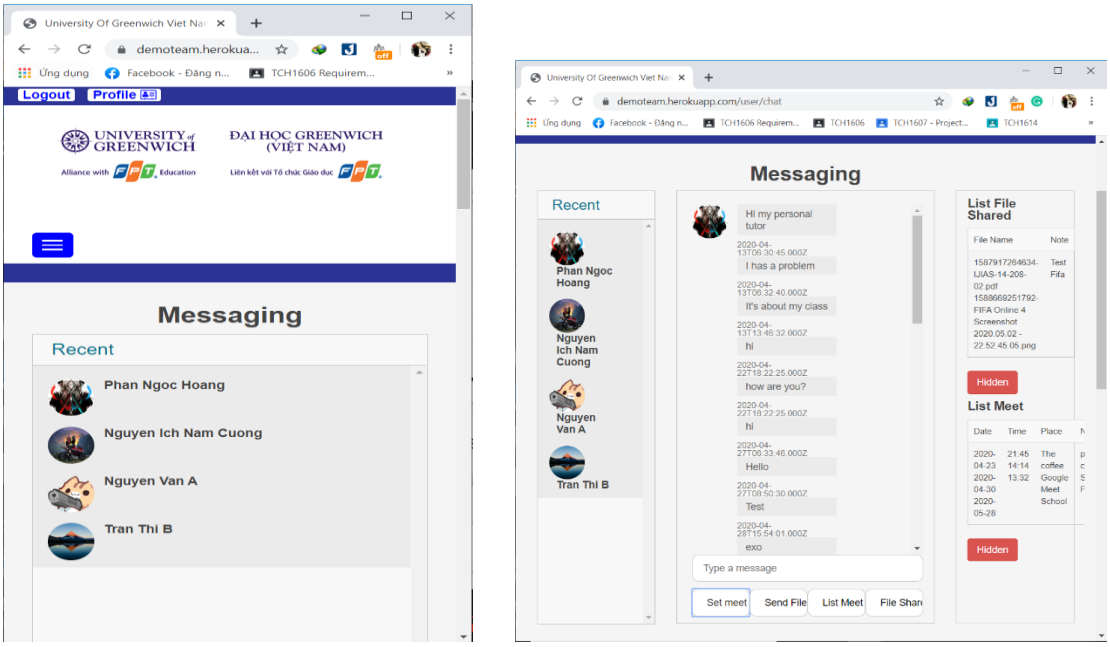

Figure 20 Interface after Design of Message Page on Tablet, Mobile

For the tablet interface, the parts in the interface will be scaled down by the length and width to fit the screen size. But the hidden part, users can drag the screen down to see.

For the phone interface, the parts of the interface are displayed vertically. Users scroll down to view and use the functions.

Manage Group Page for Staff

The interface will shrink vertically and horizontally. For the left sidebar, it will shrink to icons representing each function that the user can use. For group management panel, the system will display as a horizontal pull bar for users if using smaller screens than laptops and pc screens.

Laptop, PC:

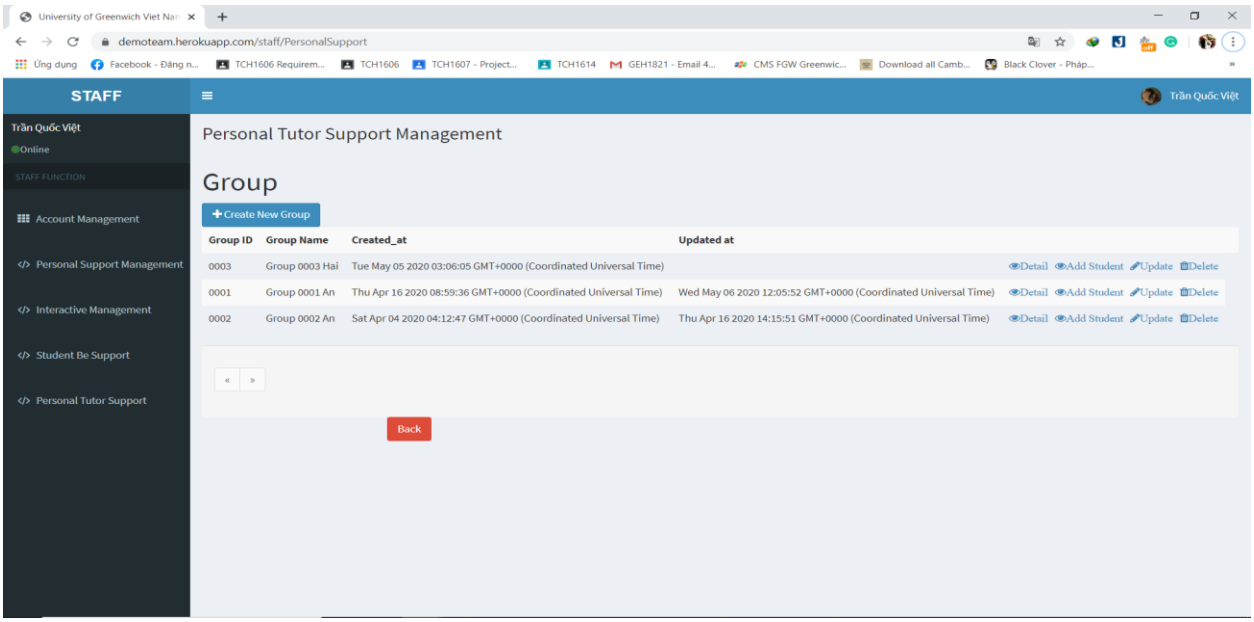

Figure 21 Interface after Design of Home Page of Staff on Laptop

Tablet and Mobile:

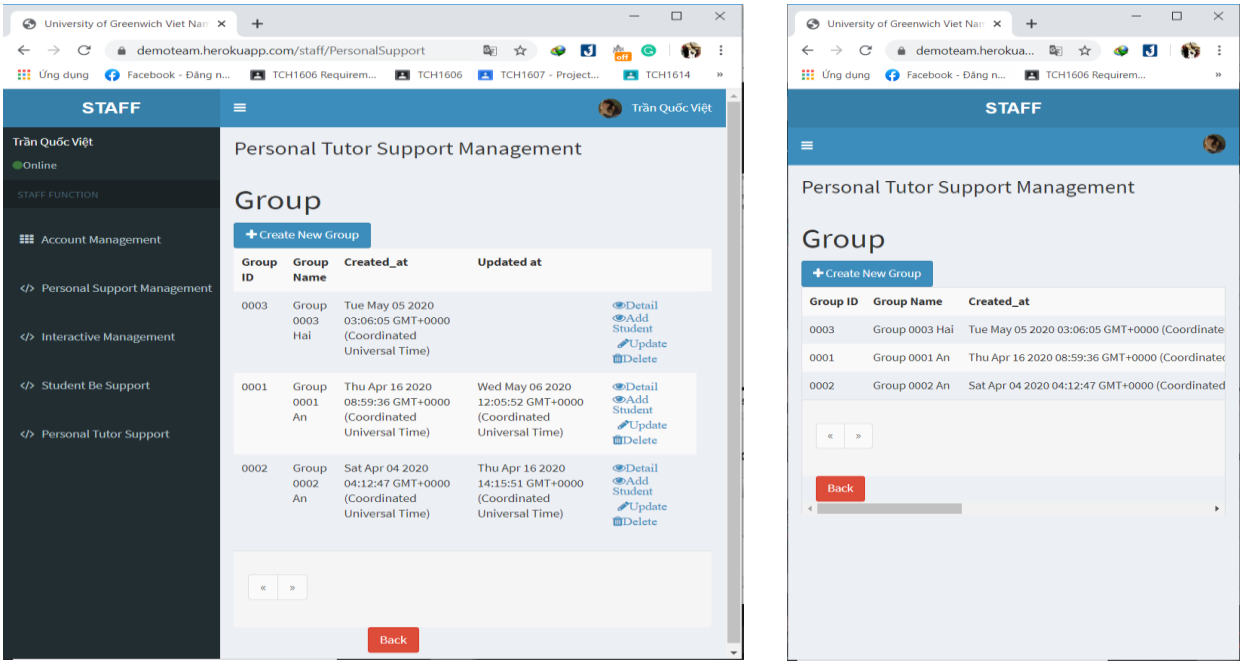

Figure 22 Interface after Design of Home Page of Staff on Tablet, Mobile

4. Usability:

Usability is part of the user experience. The extent to which an assigned user uses the product to achieve specific goals with efficiency, effectiveness, and satisfaction in the context of specified use.

Our website has an easy to use and recognizable interface for users

- Easy to get acquainted with the functions in the website
- Easily achieve your goals
- Easy to recall the user interface and how to use it on subsequent visits

5. Accessibility:

It is a way to enable everyone, including people with disabilities, to have full access to the web and the internet. Web and Internet are increasingly playing an important role in our life, providing many resources and capabilities, from work, knowledge to entertainment, and access to public

services of the government. This has led to the equal rights of all in cyberspace. A website that provides functionality for people with disabilities is a good idea to avoid this discrimination.

There are 3 levels to show this: A - Required, AA - Need to complete AAA - may need to

Our website meets the required level A:

- Image tags all have the alt tag attribute
- Users can use the keyboard to navigate and use features on the website
- Comply with the basic rules so that people with disabilities can understand the content on our website

For example:

- Button tag is in the form tag
- Use a variety of heading tags to define titles for each function. The title tags are clearly displayed with bold text and large fonts.

Figure 23 Example of Heading

- There are clear error messages that let users see if they enter the wrong data
- Accompanying the error message is the message about the correct format of the data that users should enter

IV. Functionality:

1. Use case Diagram:

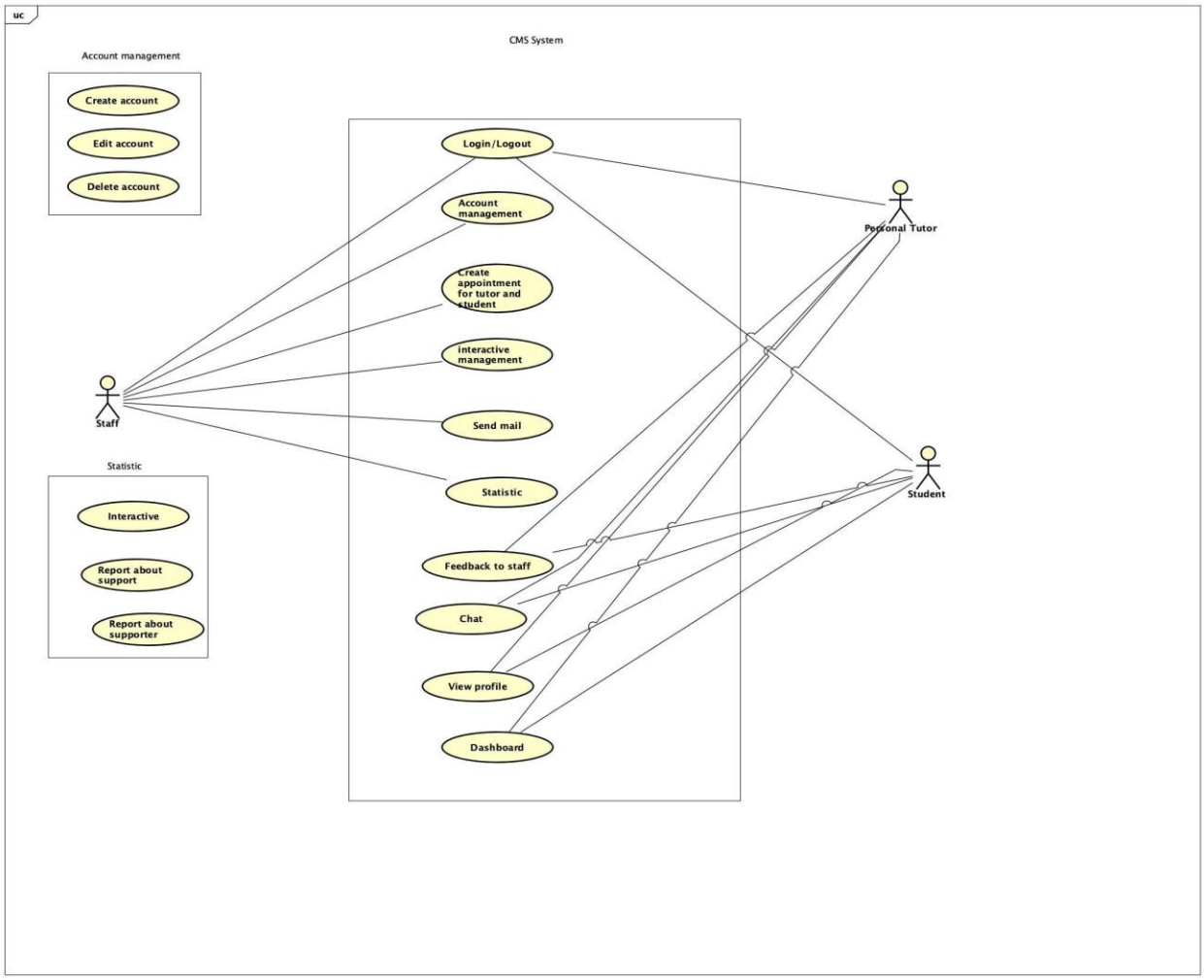

Figure 24 Use-case Diagram

- The staff has the ability to allocate student and personal tutor based on the requirement that a student will have a tutor and a tutor can charge many students at the same time (10 maximum students).
- Staff will send you an email to notify the student and tutor about which tutor will take charge of the student.
- Authorized staff can participate in the virtual meeting, as well as be able to view the conversations exchanged between the student and the tutor.

- Chat software between the student and the tutor.
- Students and tutors can arrange meetings between two people when problems are resolved and the meeting will be recorded regardless of the actual meeting or the virtual meeting.
- Students and professors can post important documents such as matters that need to be exchanged or a record of meeting details between a student and a tutor.
- Each student will have their own dashboard to statistical the interaction between students and their personal tutor.
- Personal tutor will also have their own control panel for proper sorting and filtering.
- eTutoring system is required to be compatible with all devices (phones, computers, and tablets)

2. Activity Diagram:

Student/Tutor:

Chat:

- Select "Chat", the System Returns a list of dialogs.
- The user selects the conversation; the system will pay out the sent messages.
- The user needs to enter a message to send.
- If the message has not been entered, the system will not send anything, do not store information on Dataset.
- If the user enters the correct message and sends it, the message will be sent simultaneously, the database and the message content, the recipient ID and the time of the message.

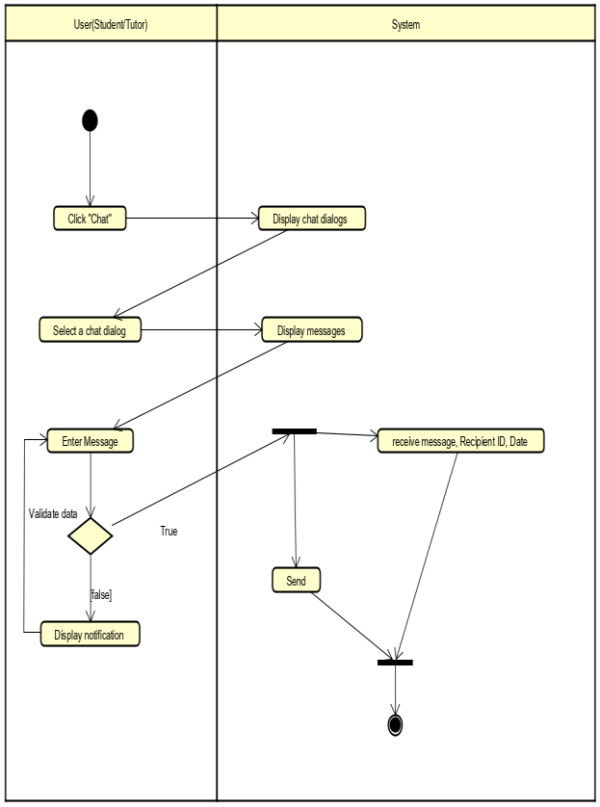

Figure 26 Use-case Diagram

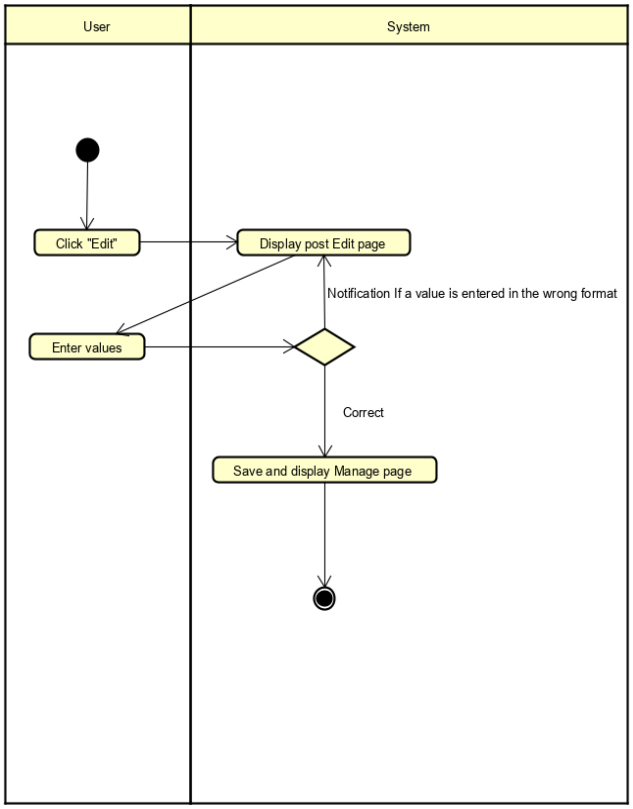

Figure 25 Edit

Send File:

- Select "Send file" in the Chat section
- The user will select the file to send in the machine.
- The system will check the file quota. If the file is over 4MB, the oath will notify the file invalid, too space. Return to the Select file step.
- If the File is valid, less than 4MB, the file will simultaneously be saved on the database and display the file in the list. Display files for senders and recipients

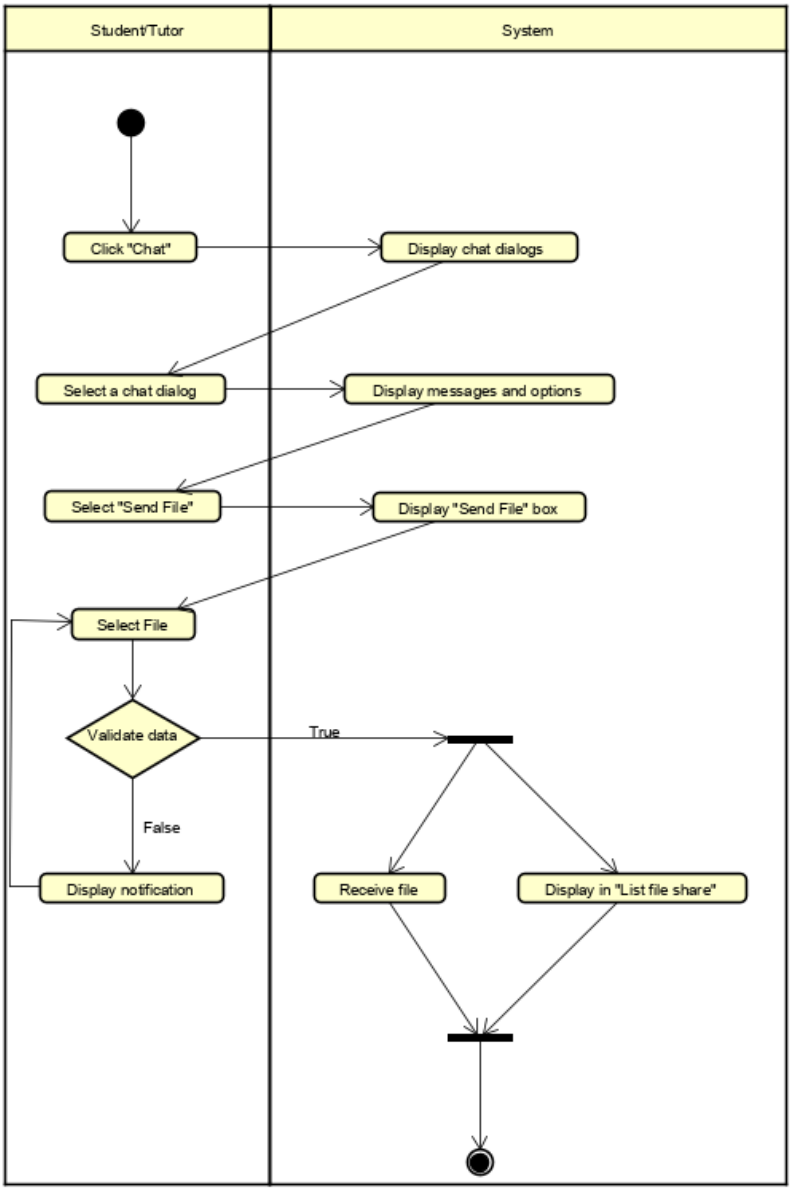

Figure 27 Send File

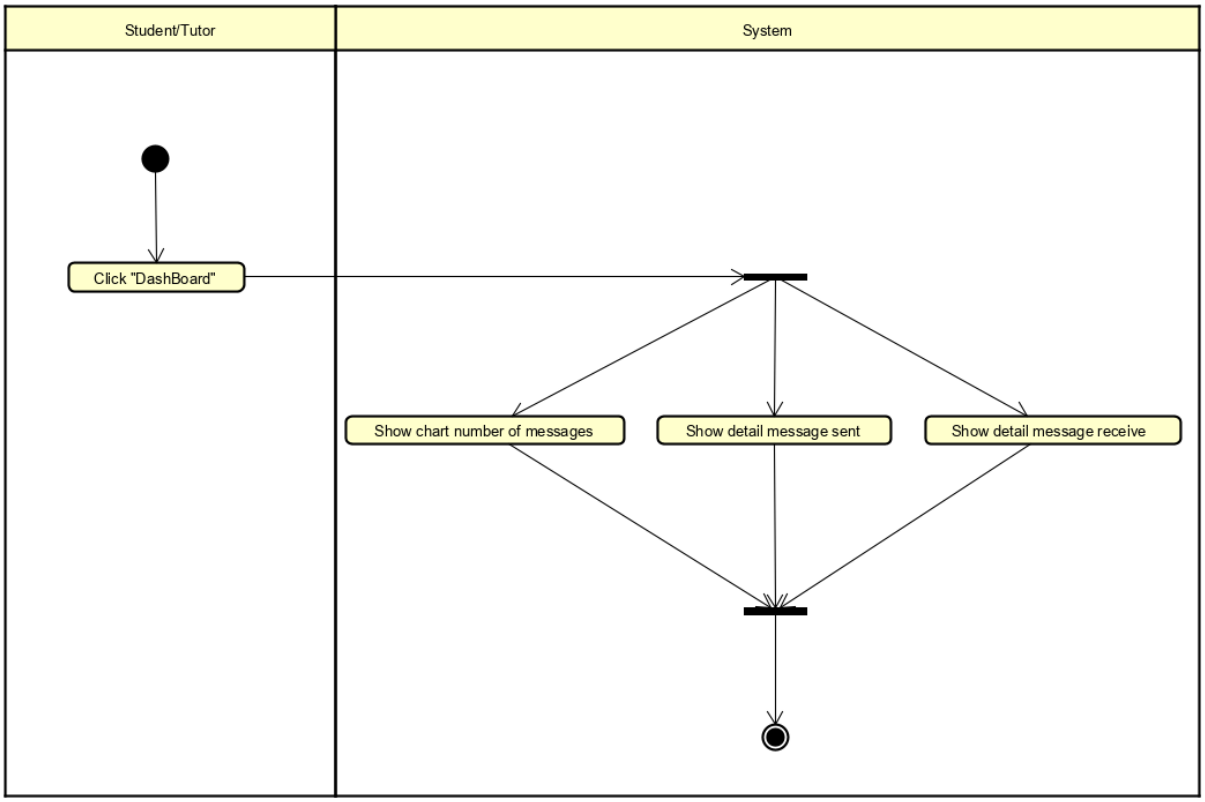

Figure 28 Dashboard

Send Mail:

- The user selects "Contact staff", the system shows the list staff and their mail address the same frame to enter the mail content.
- Users need to enter enough information: Staff mail, name, ID, mail, content mail.
- If the user enters the missing contents, the system when the test will return the error message to the user and return to the Enter email information step.
- If the system check correct content will send mail successfully

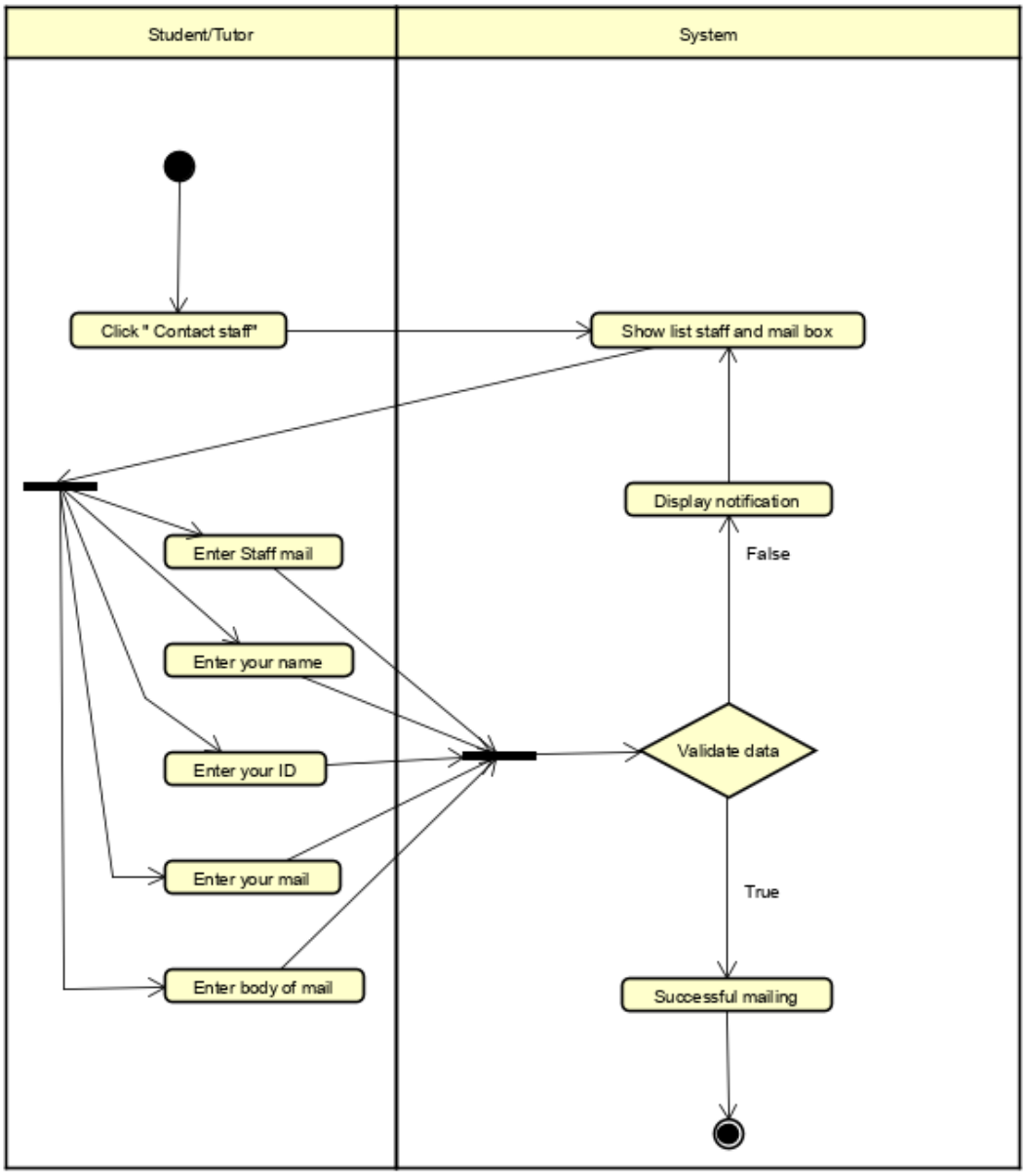

Figure 29 Send mail

Set Meet:

- User selects "Set meet"
- The system returns the dictionary information set meet.
- User enters meeting information
- Create a successful Meet the system will save the meeting information on the database and display it in the meet list for student and tutor.

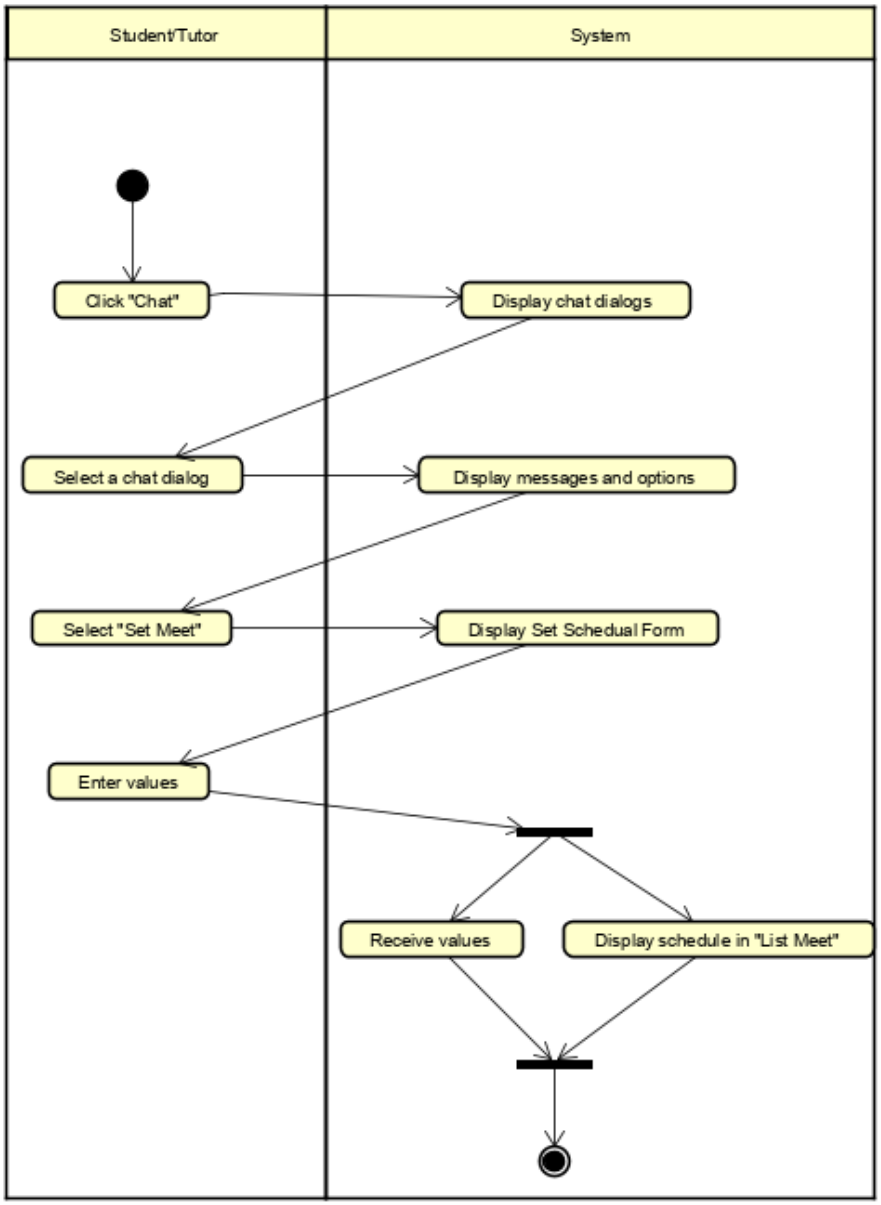

Figure 30 Set Meet

Staff:

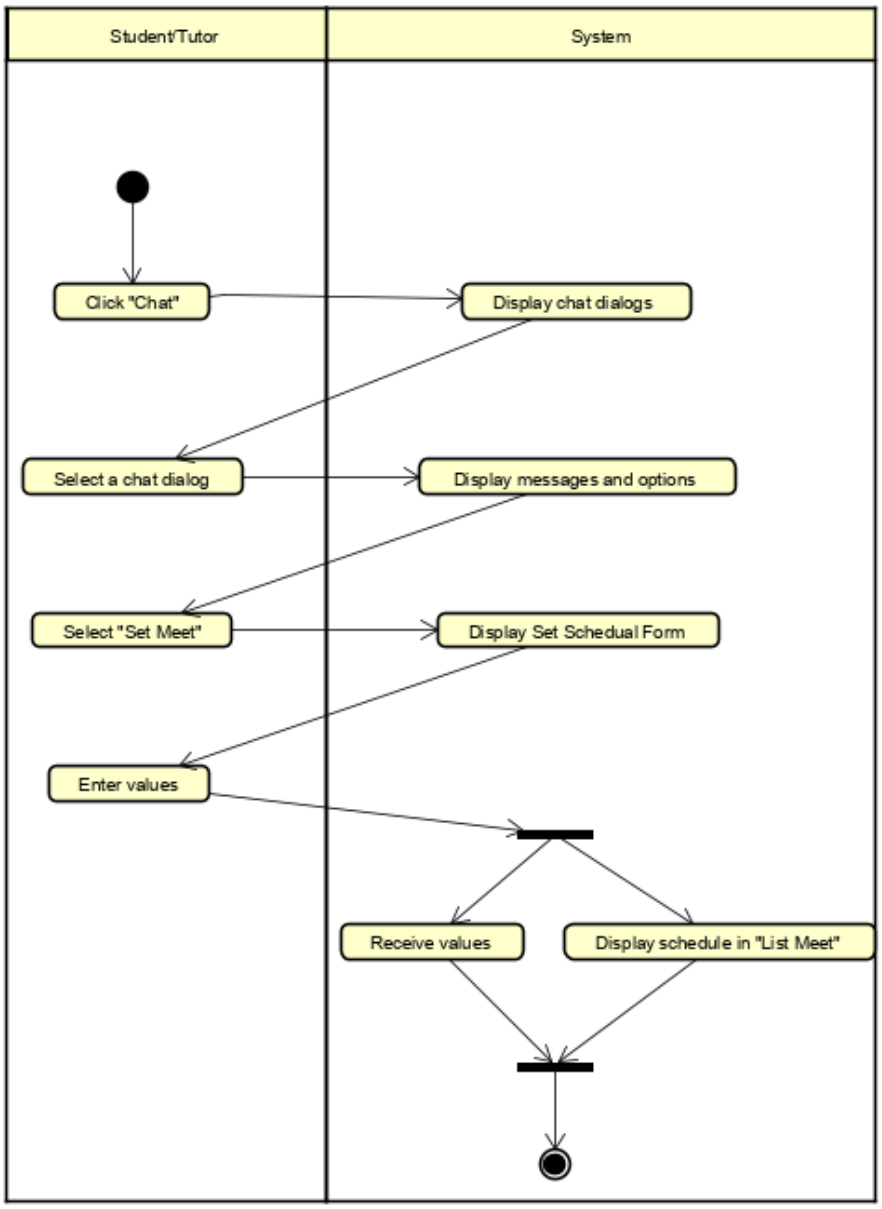

Figure 31 Staff create account

Create Group Chat:

- Log in staff
- Using the "Personal Support Management" feature, the system returns, displays the group list.
- Users who want to create new group screen will display the information pane for users to enter new data including "Group ID", "Group Name", Selected Tutor for group.
- The user data type satisfies, database saves group information and notify the user of the successful group creation.

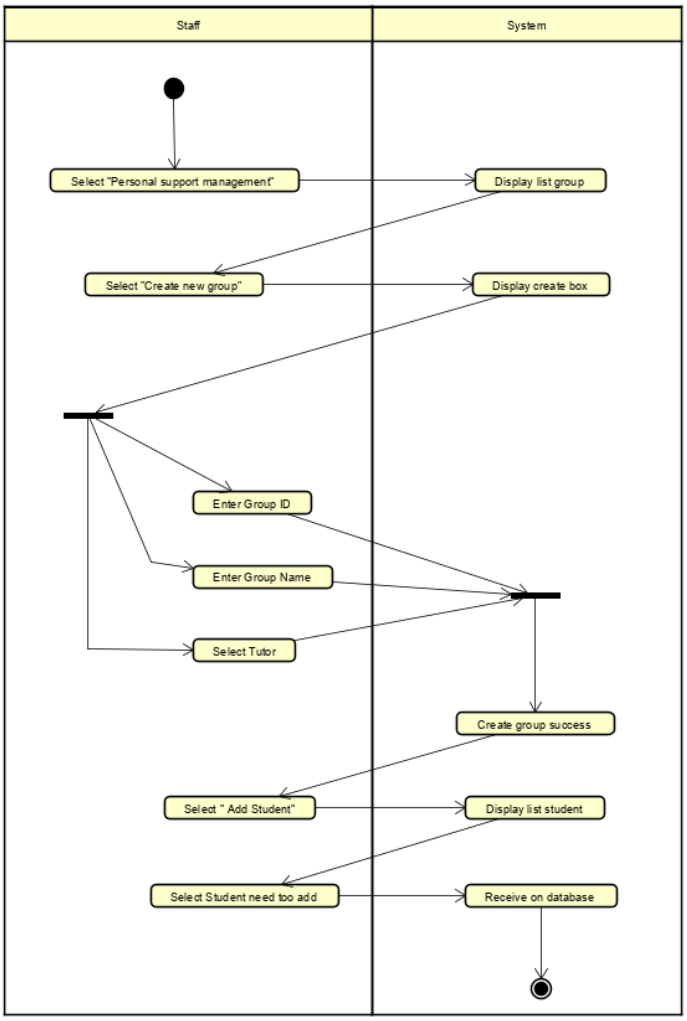

Figure 32 Staff create group chat

Send mail:

- Select "Account management", select the object that you want to send mail: Staff, Tutor, Student.
- Select Send Mail, the system will return to compose mail.
- Users need to enter the content in the frame as Email address, Subject, Content.
- The system will check if the user satisfies the information, the system will send mail content. If it is wrong, it will return to the account list screen.

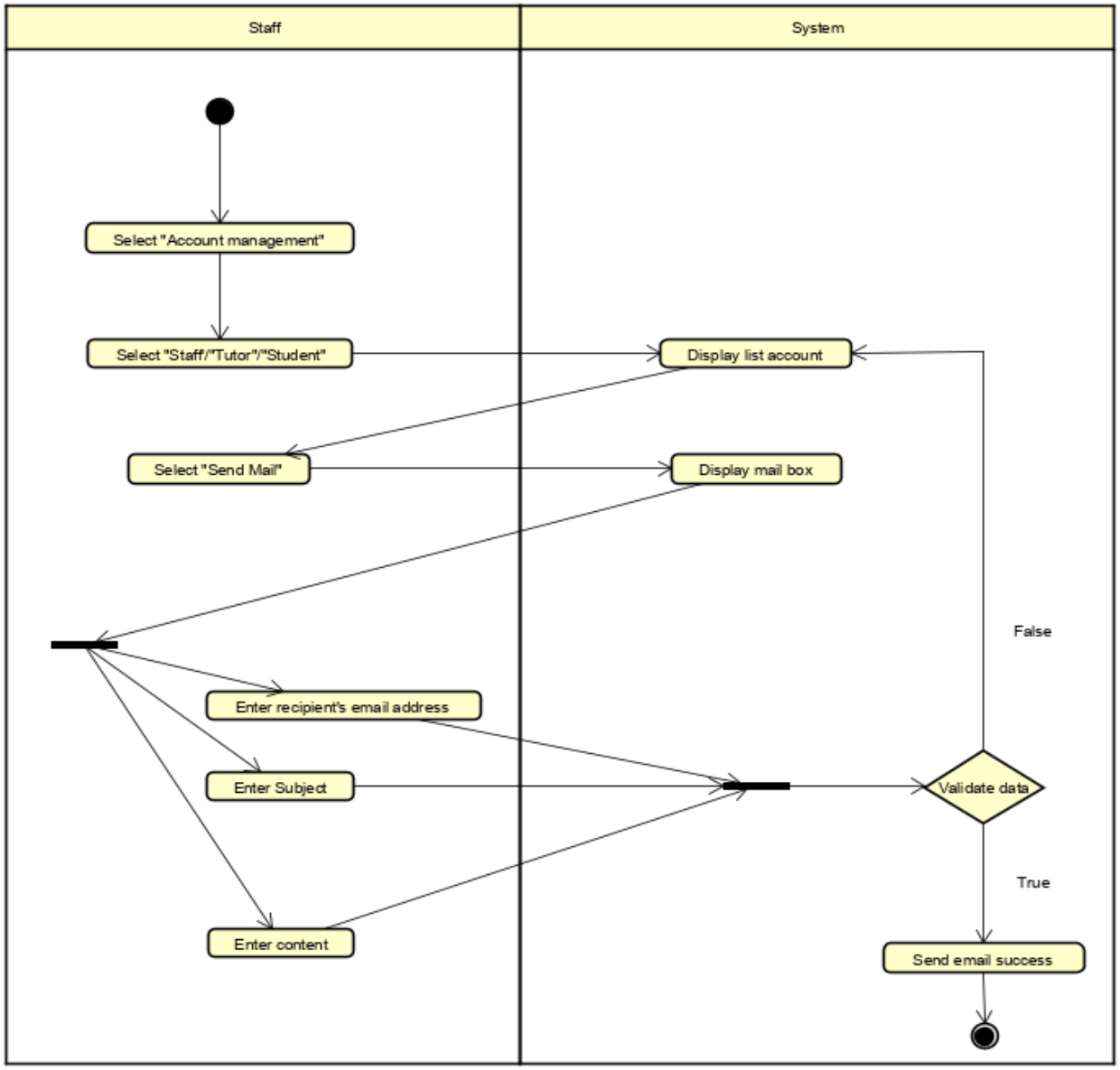

Figure 33 Staff sends mail

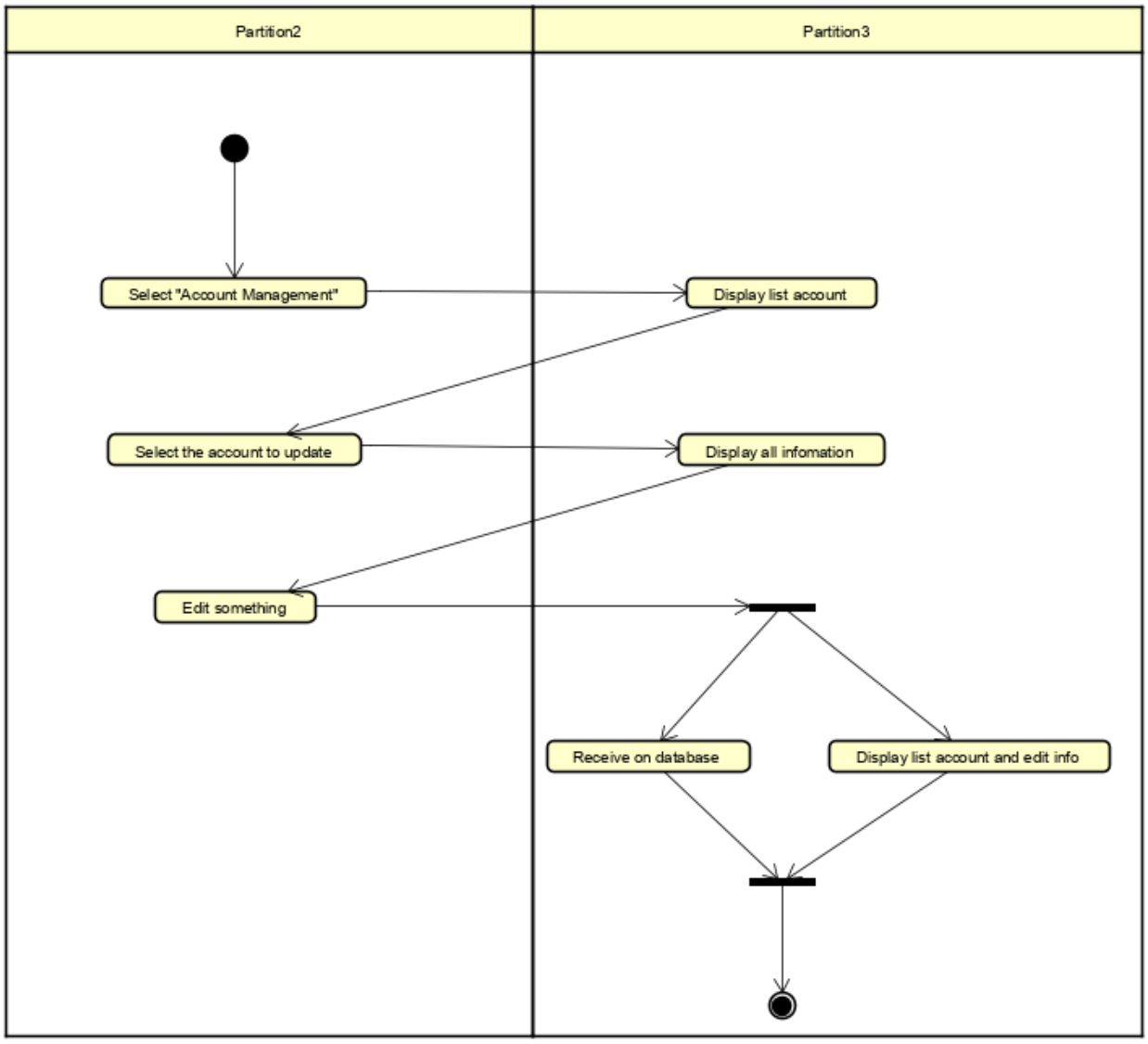

Figure 34 Staff update account

3. Functions:

The Back-end code can be present in the *app.js* file in the project under the following link:

https://github.com/PhanNgocHoang/CMS/app.js

3.1. Login

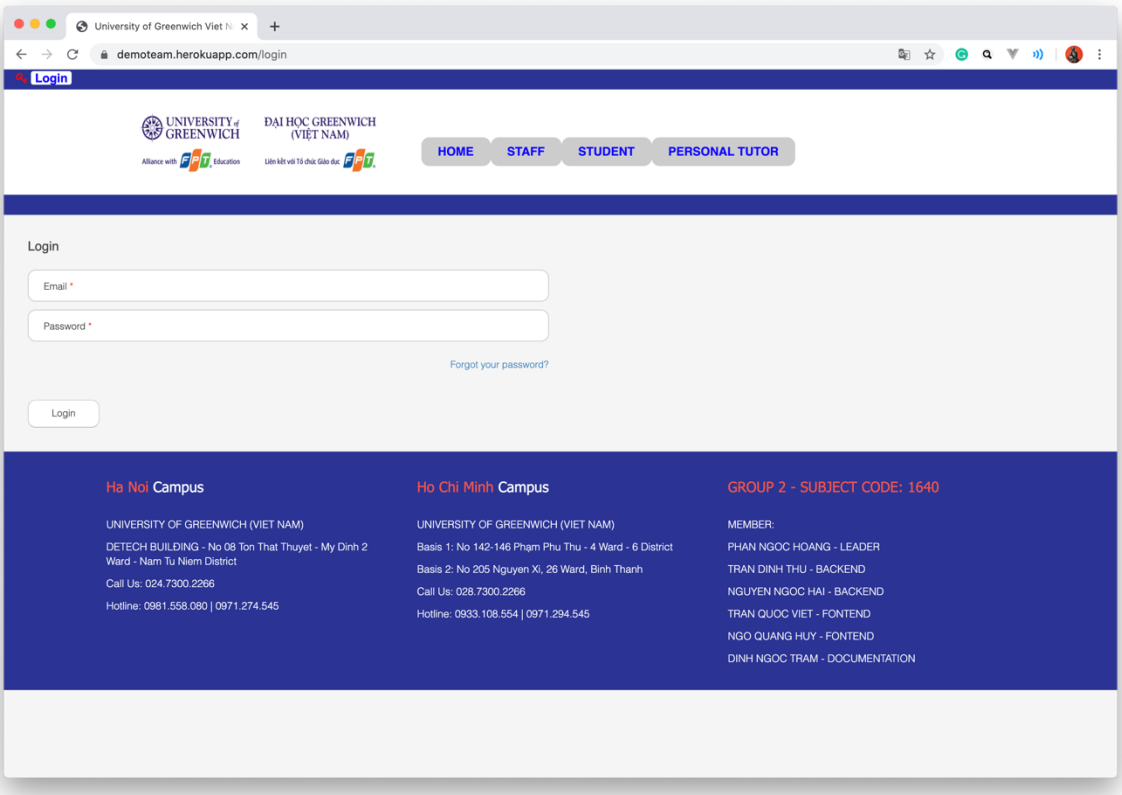

Figure 35 Interface of Login function

The server receives the user's email and password, and then checks the database if the user's email address or password does not exist, and sends a notification to the user. As for the user's email and password field, the server will pack: _id, the user's email and role with the token, the token used to authenticate the user and delegate the user, the token signature is "Team2DevelopmentCms", the algorithm used to code is "HS256", the survival time is 3 hours.

```
async function PostLogin(req, res) {
  let email = req.body.email;
  let password = req.body.password;
  Complexity is 12 You must be kidding
  Models.UserModel.findOne({ User_mail: email }).exec((err, docs) => {
    if (docs === null) {
      let error = "Wrong Email or Password";
      res.render("HomePage/login", { data: { error: error } });
      return;
    }
    if (docs.User_pass !== password) {
      let error = "Wrong Email or Password";
      res.render("HomePage/login", { data: { error: error } });
      return;
    } else {
      const user = {
        user_id: docs._id,
        user_mail: docs.User_mail,
        user_role: docs.User_role,
      };
      let token = jwt.sign({ user: user }, "Team2DevelopmentCms", {
        algorithm: "HS256",
        expiresIn: "3h",
      });
      res.cookie("user", token, {
        maxAge: 10800000,
        signed: true,
        secure: true,
        httpOnly: true,
        domain: "demoteam.herokuapp.com",
      });
      //set domain, secure to cookie only send to a domain and https
      Complexity is 6 It's time to do something...
      Models.RoleModel.findById({ _id: docs.User_role }).exec((err, role) => {
        if (role.roleName === "Staff") {
          return res.redirect("/staff");
        }
        if (role.roleName === "Student" || role.roleName === "Personal Tutor") {
          return res.redirect("/user");
        }
      });
    }
  });
```

Figure 36 Code Login Function

After generating the token, we use cookies to send the token to the server. When creating cookies, they pass in the name of the cookie ("user") the cookie's data is the token code generated. But sending information via cookies can be stolen through fake websites so we have set a number of options for our cookies to increase the safety of cookies, the "maxAge" option is time-consuming. The cookies are now in milliseconds and 108,000,000 mini seconds change to 3 hours, the "signed" option is to use cookies with the signature and the cookie signature for us is "123 @ 123 @! T".

Figure 37 Signature of cookies

The "secure: true" option is to ensure cookies are only sent to https addresses, the "httpOnly: true" option is to minimize XSS attacks. We also use the "domain" option to protect users 'cookies, using the "domain" option will help ensure that users' cookies will only be sent to the system domain to minimize the possibility of users' cookies being stolen.

In order to increase the security after cookies are sent to the server, we check the existence of cookies, decode the token stored in cookies and check the token information from the token with the user information stored in the cookie. database. In case the token cannot be decoded or the information in the token does not exist in the database, we send the sender information about the login page of the system, otherwise tokens are decoded and the information stored in them if they exist in the database, we will let them through.

```
async function checkAth(req, res, next) {
  let token = req.signedCookies.user;
  if (!token) {
    return res.redirect("/login");
  }
  Complexity is 9 It's time to do something...
  let userId = jwt.verify(token, "Team2DevelopmentCms", (err, decode) => {
    if (err) return res.redirect("/login");
    global.user_ifo = decode.user;
    Complexity is 6 It's time to do something...
    Models.UserModel.find({ _id: decode.user.user_id }).exec((err, docs) => {
      if(err) return res.redirect("/login");
      if (docs == null) {
        return res.redirect("/login");
      }
      res.locals.user = docs;
      global.User_id = docs._id
      return next();
    });
  });
}
```

Figure 39 Check cookies

```
function CheckStaff(req, res, next) {
  Complexity is 4 Everything is cool!
  Models.RoleModel.findById({ _id: user_ifo.user_role }).exec((err, role) => {
    if (role.roleName === "Staff") {
      return next();
    }
    return res.redirect("/login");
  });
}
Complexity is 7 It's time to do something...
function CheckTutorAndStudent(req, res, next) {
  Complexity is 6 It's time to do something...
  Models.RoleModel.findById({ _id: user_ifo.user_role }).exec((err, role) => {
    if (role.roleName === "Personal Tutor") {
      res.locals.role = role.roleName;
      return next();
    }
    if(role.roleName === "Student")
    {
      res.locals.role = role.roleName;
      return next();
    }
    return res.redirect("/login");
  });
}
```

Figure 38 Check user role

After passing the cookies test, we continue to identify the sender's role so that we can take them to the website pages and the features they can use on the system. This check ensures each user is using the right features for his role.

3.2 Create new group for Staff

```
async function Get_Create_Group(req, res) {
  let role = await Models.RoleModel.findOne({ roleName: "Personal Tutor" });
  global.tutor = await Models.UserModel.find({ User_role: role._id });
  return res.render("StaffPage/groups/create", {
    data: { tutor: tutor },
  });
}
```

Figure 40 Code create new group support

We retrieved the users whose personal tutor role. Before data is transferred to the server, we validate it on the client-side. Retrieving Personal information and rendering the user interface, the user will see the names of the personal tutors for easy selection but the saved data will be _id for the selected personal tutor.

```
async function Post_Create_Group(req, res) {
  let group_id = req.body.group_id;
  let group_name = req.body.group_name;
  let tutor_id = req.body.tutor_id;
  let New_Group = new Models.GroupModel({
    Group_ID: group_id,
    Group_name: group_name,
    Tutor_id: tutor_id,
    Create_at: date,
    Update_at: "",
  });
  Complexity is 4 Everything is cool!
  New_Group.save((err) => {
    if (err) {
      let error = "Group ID already exist";
      return res.render("StaffPage/groups/create", {
        data: { error: error, tutor: tutor },
      });
    }
    return res.redirect("/staff/PersonalSupport");
  });
}
```

Figure 41 Save Personal tutor to group

We add data to the database in addition to the Group_ID we make sure it is unique so we have added a unique: true property to ensure this. When creating a new support group with an existing Group_ID, we notify this to the user.

3.3. Send mail

To send any notifications to members of a support staff group, click the detail icon of a support group and the Staff will be redirected to that group's detail page. Staff click on the send mail button, the staff will see all the mail of the support team members and can enter the contents of the mail. To implement the tomorrow shipping feature, we have used the Nodemailer package.

We use google gmail translation, we set up the information to send mail. These include the service, the port, the gmail account used to send it, and the content of the email will include the sender, recipient list, subject and email content.

```
function Post_Send_Mail_For_Group(req, res) {
  let student_email = req.body.student_mail;
  let tutor_mail = req.body.tutor_mail;
  let subject = req.body.subject;
  let content = req.body.content;
  let group_id = req.params.group_id;
  let arr_mail = new Array()
  arr_mail = arr_mail.concat(student_email, tutor_mail)
  let transporter = nodemailer.createTransport({
    host: "smtp.gmail.com",
    port: 465,
    secure: true,
    auth: {
      user: "Hoangpnc123@gmail.com",
      pass: "Hoang123@",
    },
  });
  let mailOptions = {
    from: '"Admin" <Hoangpnc123@gmail.com>',
    to: arr_mail,
    subject: subject,
    text: content,
  };
  Complexity is 3 Everything is cool!
  transporter.sendMail(mailOptions, (err) => {
    if (err) console.log(err)
    return res.redirect("/staff/PersonalSupport/GroupDetail/" + group_id);
  });
}
```

Figure 42 Send Mail

In addition, we also developed two features related to sending mail: a feature that allows staff to send separate emails to one person and forget the password. In terms of setup, it is similar to sending mail to the support group.

3.4. Set Meet

```
$(document).on("click", "#set", () => {
  let date = $("#Date").val();
  let hours = $("#hours").val();
  let place = $("#place").val();
  let note = $("#note").val();
  if(date === "" || hours==="" || place === "" || note ===="")
  {
    alert("Please complete all the meeting related information")
  }
  else{
  let date_meet = "<li>" + date + "</li>";
  let time_meet = "<li>" + hours + "</li>";
  let place_meet = "<li>" + place + "</li>";
  let note_meet = "<li>" + note + "</li>";
  $("#date_meet").append(date_meet);
  $("#time_meet").append(time_meet);
  $("#place_meet").append(place_meet);
  $("#note_meet").append(note_meet);
  $("#Date").val("");
  $("#hours").val("");
  $("#place").val("");
  $("#note").val("");
  socket.emit("Set-meet", date, hours, place, note, id);
  $("#set_met").hide();
  }
});
```

Figure 43 The person who set the appointment

Personal tutor and student can set appointments and appointment information such as time, location, notes of the meeting will be sent to the server via the "Set-meet" event. From this event, the server will save the appointment information to the database and transfer the information of the appointment to the right person selected through the "P-set-meet" event.

```
socket.on("P-set-meet", (data) => {
  let date = "<li>" + data.date + "</li>";
  let time = "<li>" + data.hours + "</li>";
  let place = "<li>" + data.place + "</li>";
  let note = "<li>" + data.note + "</li>";
  $("#date_meet").append(date);
  $("#time_meet").append(time);
  $("#place_meet").append(place);
  $("#note_meet").append(note);
  bt = setInterval(() => {
    $("#chat-icon").css("background-color", "blue");
  }, 50);
});
```

Figure 44 Recipient of the appointment

3.5. Send File

Personal tutors and students can send files to each other. When the user clicks the send button, the client checks the size of the file and renames the file. If the file is larger than 4mb, there will be a notice to the user. And the file's information will be used via the "send-file" event.

```
$(document).on("click", "#send", () => {
  let my_file = $("#fileinput").prop("files")[0];
  let note = $("#note-File").val();
  let size_file = my_file.size;
  let name = `${Date.now()}-${my_file.name}`;
  if (size_file > 4194304) {
    alert("File is too large");
  } else {
    $("#fileinput").val("");
    $("#note-File").val("");
    socket.emit("send-file", my_file, name, id, note);
```

The server receives the information of the file through the "send-file" event, after which the server writes the file to the storage directory and saves the file name to the database. The server sends the file's information to the recipient through the "P-send-file" event.

```
socket.on("P-send-file", (my_filename, note) => {
  let l_file =
    "<li><a href='/static/Files/" +
    file.File +
    "' download='/static/Files/" +
    file.File +
    "'>" +
    file.File +
    "</a></li>";
  let l_note = "<li>" + note + "</li>";
  $("#l_files").append(l_file);
  $("#note_file").append(l_note);
  bt = setInterval(() => {
    $("#chat-icon").css("background-color", "blue");
  }, 50);
});
```

Figure 45 Recipient of the appointment

The client receives the file's information through the "P-send-file" event, then notifies and displays the file's information to the recipient.

3.6. Forgot password

With the forgotten password feature, we check the existence of the email entered by the user. If such email exists, we will send password information via email. If the email does not exist, we will notify the user.

```
function Post_Forgot_Password(req, res) {
  let mail = req.body.email;
  Complexity is 8 It's time to do something...
  Models.UserModel.findOne({ User_mail: mail }).exec((err, user) => {
    if (err) {
      let error = "Email does not exist";
      return res.render("HomePage/Forgotpass", { data: { error: error } });
    } else {
      let transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
          user: "Hoangpnc123@gmail.com",
          pass: "Hoang123@",
        },
      });
      let mailOptions = {
        from: '"Admin" <hoangpn2201@gmail.com>',
        to: user.User_mail,
        subject: "Your password of University",
        text:
          "Your Password of email : " +
          user.User_mail +
          " is : " +
          user.User_pass,
      };
      Complexity is 4 Everything is cool!
      transporter.sendMail(mailOptions, (err, info) => {
        if (err) return console.log(err);
        return res.redirect("/login");
      });
    }
  });
}
```

3.7. Interactive report

```
let role_id = $("#role_id").val();
socket.emit("RoleId", role_id);
Complexity is 4 Everything is cool!
socket.on("Sender_Receiver", (sender, receiver, User) => {
  User.forEach((user) => {
    let user_full =
      "<td>" +
      user.User_full +
      "| <a href='/staff/DetailMessage/" +
      user._id +
      "' title='Detail'><i class='fa fa-eye'>Detail</i></a></th>";
    $("#user_full").append(user_full);
  });
  sender.forEach((element) => {
    let num_sender = "<td>" + element + "</td>";
    $("#sender").append(num_sender);
  });
  receiver.forEach((receiver) => {
    let num_receiver = "<td>" + receiver + "</td>";
    $("#receiver").append(num_receiver);
  });
});
let user_id = $("#user_id").val();
socket.emit("detail-mess", user_id);
Complexity is 3 Everything is cool!
socket.on("message_detail", (arr_mess_sent, arr_mess_receiver) => {
  $("#sent").append(arr_mess_sent.length);
  $("#receive").append(arr_mess_receiver.length);
  arr_mess_sent.forEach((message) => {
    let list = "<li>" + message.Date + "</li>";
    $("#h_sent").append(list);
  });
  arr_mess_receiver.forEach((message) => {
    let list = "<li>" + message.Date + "</li>";
    $("#h_receive").append(list);
  });
});
```

Figure 46 Client Side

For statistics tables, we send information about the role and _id of the person in the statistics to the server and listen for server responses related to client responses. For interactive statistics we process the data retrieved by the server, we display the full names of the people in the statistics

and their total interactions. As for the detailed interaction report, we only show interaction time to ensure privacy.

```
socket.on("RoleId", async (data) => {
  let User = await Models.UserModel.find({ User_role: data });
  let arr_sender = [];
  let arr_receiver = [];
  for (let i = 0; i <= User.length - 1; i++) {
    let sender = await Models.MessageModel.find({ Sender: User[i]._id });
    arr_sender.push(sender.length);
    let receiver = await Models.MessageModel.find({ Receiver: User[i]._id });
    arr_receiver.push(receiver.length);
  }
  socket.emit("Sender_Receiver", arr_sender, arr_receiver, User);
});
Complexity is 5 Everything is cool!
socket.on("detail-mess", async (data) => {
  let arr_mess_sent = new Array();
  let arr_mess_receiver = new Array();
  let Mess_send = await Models.MessageModel.find({ Sender: data });
  let Mess_receive = await Models.MessageModel.find({ Receiver: data });
  Mess_send.forEach((message) => {
    for (i = 0; i <= message.Message.length - 1; i++) {
      arr_mess_sent.push(message.Message[i]);
    }
  });
  Mess_receive.forEach((message) => {
    for (i = 0; i <= message.Message.length - 1; i++) {
      arr_mess_receiver.push(message.Message[i]);
    }
  });
  socket.emit("message_detail", arr_mess_sent, arr_mess_receiver);
});
```

Figure 47 Server Side

On the server side, we listen to events sent by the client. For interactive statistical events, we receive "User_role" from which we look for interactions of people with "User_role" that match "User_role" in the Collection Message from a client, which we then send to. The information clients of the people we find are also their total interaction.

```
socket.on('message_detail', (arr_mess_sent, arr_mess_receiver) => {
    let user_name = document.getElementById('user_name').value
    var ctx = document.getElementById("myChart").getContext("2d");

    var data = {
        labels: [user_name],
        datasets: [
            {
                label: "Sent",
                backgroundColor: "#80aaff",
                data: [arr_mess_sent.length]
            },
            {
                label: "Received",
                backgroundColor: "#ff6666",
                data: [arr_mess_receiver.length]
            },
        ]
    };

    var myBarChart = new Chart(ctx, {
        type: 'bar',
        data: data,
        options: {
            title:{
                display: true,
                text: "Chart the number of messages"
            },
            barValueSpacing: 20,
            scales: {
                yAxes: [{
                    ticks: {
                        min: 0,
                    }
                }]
            }
        }
    });
})
```

Figure 48 Interactive statistical chart

The client sends the event "student_support" and when the server receives the event "student_support" then proceeds to search for a list of students who have personal tutor support

and a list of all students and send them back to the client through the event " list_student”

```
socket.emit("student_support");
```

```
socket.on("student_support", async () => {
  let student = await Models.RoleModel.findOne({ roleName: "Student" });
  let Student = await Models.UserModel.find({ User_role: student._id });
  let Group = await Models.GroupModel.find({});
  let Student_support = [];
  let Student_not_Support = [];
  for (i = 0; i <= Group.length - 1; i++) {
    Student_support = Student_support.concat(Group[i].Student_id);
  }
  for (i = 0; i <= Student.length - 1; i++) {
    Student_not_Support.push(Student[i]._id);
  }
  socket.emit("list_student", Student_support, Student_not_Support);
```

Figure 49 Server handling event "student_support”

When the client receives the "list_student" event, it will filter students who have personal tutors and do not have personal tutors to support. After the client has finished filtering, dump the results on the chart and send _id of the student without personal tutor to support to search for information and display on the screen through the event "Student_notSupport".

In addition to the above charting features, we also have the same handling features: The actual chart of the number of information exchanged by users in the system, the chart of personal tutor including the list Displays the number of students supported by that personal tutor.

```
socket.on("data_dashboard", async (user_id) => {
  let Sent = await Models.MessageModel.find({ Sender: user_id }).populate(
    "Receiver"
  );
  let Receive = await Models.MessageModel.find({
    Receiver: user_id,
  }).populate("Sender");
  socket.emit("Your_data", Sent, Receive);
  let group = await Models.GroupModel.find({ Tutor_id: user_id });
  let list_student = new Array();
  group.forEach((student) => {
    list_student = list_student.concat(student.Student_id);
  });
  let student = await Models.UserModel.find({ _id: list_student });
  socket.emit("list_student", list_student, student);
});
```

Figure 50 Server handling event data dashboard from client

The client sends his _id to the server then the server looks for _id-related information that is sent in the Model Message to retrieve the history of messages and look in the Model Group and User to retrieve the information that _id is sent to support. Where _id is submitted by Student, the

search data in Model Group and User will be empty. After searching, the data will be returned via two events: the event "Your_data" to send data about the history of messages and the "list_student" event to send data to the student support list

3.8. Chat for Student and Personal Tutor

After the user clicks Chat, the user sends an event "user_info" containing _id, role, the full name of the user to the server.

```
socket.emit("user_info", {
  user_id,
  user_role,
  user_full,
});
```

Figure 51 Client sends event "user_info"

The server that receives the "user_info" event will let the user automatically join the room named _id from the user. The server checks the user's role and returns a list of people that the user can chat with.

```
socket.on("user_info", async (data) => {
  socket.name = data.user_full;
  socket.user_room = data.user_id;
  socket.user = data.user_id;
  socket.join(socket.user_room);
  if (data.user_role === "Student") {
    let group = await Models.GroupModel.findOne({ Student_id: data.user_id });
    let tutor = await Models.UserModel.findById({ _id: group.Tutor_id });
    let tutor_id = group.Tutor_id;
    let tutor_full = tutor.User_full;
    let tutor_avatar = tutor.User_avatar;
    socket.emit("chat_tutor", {
      tutor_id,
      tutor_full,
      tutor_avatar,
    });
  }
  if (data.user_role === "Personal") {
    let group = await Models.GroupModel.find({ Tutor_id: data.user_id });
    let student_id = new Array()
    group.forEach(ls_student=>{
      student_id = student_id.concat(ls_student.Student_id)
    })
    let students = await Models.UserModel.find({ _id: student_id });
    socket.emit("chat_student", {
      students,
    });
  }
```

Figure 52 Server event handler "user_info"

The client receives two events that the server returns and displays the names of the people that the user can chat with.

```
socket.on("chat_tutor", (data) => {
  let tutor =
    "<li id='chat-icon'><div Id ='" +
    data.tutor_id +
    "' name ='" +
    data.tutor_full +
    "' class='a-user'><img src='/static/images/" +
    data.tutor_avatar +
    "' alt='sunil' class='chat_img'><div class='chat_title' >" +
    data.tutor_full +
    "</div><br><div class='chat_ib'></div><div class='clearfix'></div></div></li>";
  $(".list_people").append(tutor);
});
socket.on("chat_student", (data) => {
  data.students.forEach((student) => {
    let user =
      "<li id='chat-icon'><div Id ='" +
      student._id +
      "'  name ='" +
      student.User_full +
      "'class='a-user'><img src='/static/images/" +
      student.User_avatar +
      "' alt='sunil' class='chat_img'><div class='chat_title' >" +
      student.User_full +
      "</div><br><div class='chat_ib'></div><div class='clearfix'></div></div></li>";
    $(".list_people").append(user);
  });
});
```

Figure 53 The client processes two events for the chat information from the server

Each time a user clicks on a person in the chat list, they send a "get_mess" event containing the _id and the name of the person they clicked on the server to retrieve the message history.

```
let id = $(this).attr("Id");
let name = $(this).attr("name");
$("#list_mess").show(50);
socket.emit("get_mess", { id, name });
```

Figure 54 Client sends event "get_mess"

The server searches the message history based on the data received from the client through the "get_mess" event and then sends the data related to the message history that the user requests through the "list_message" event. Actions that retrieve information about the

appointment history and shared files also work similarly to retrieving the message history. To get the history of appointments and shared files users just need to click on the "List Meet" and "File Shared" buttons.

```
socket.on("get_mess", async (data) => {
  let sender = await Models.MessageModel.findOne({
    Sender: socket.user,
    Receiver: data.id,
  }).populate("Receiver, Sender");
  let receiver = await Models.MessageModel.findOne({
    Sender: data.id,
    Receiver: socket.user,
  }).populate("Receiver, Sender");
  let name = data.name;
  socket.emit("list_message", sender, receiver, name);
});
```

Figure 55 The server handles the "get_mess" event

The client receives data from the "list_message" event and displays it to the user.

```
socket.on("list_message", (sender, receiver, name)
```

Figure 56 Client handles "list message" events

After clicking on the person you want to chat the user can send a message and the client will validate the user's message to ensure the user cannot send empty messages. And will send the information of the message through the event "send_message".

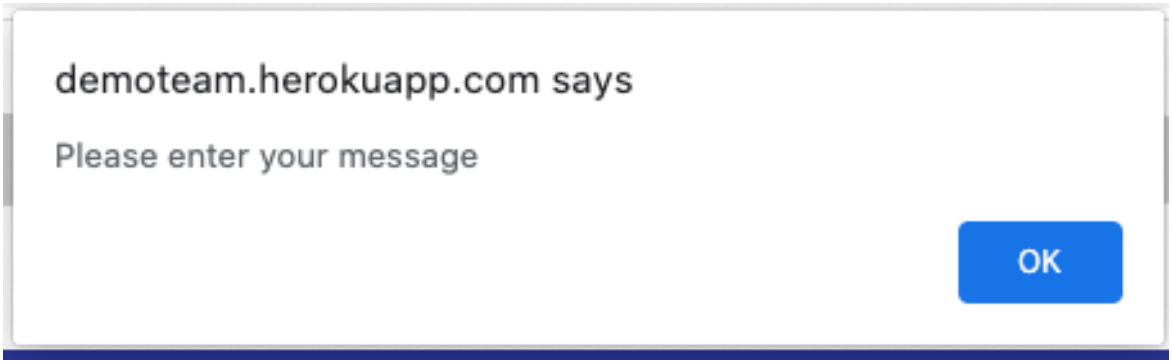

Figure 57 Notifications are displayed on the user interface

```
$("#input-message").on("keyup", (event) => {
  let message = $("#input-message").val();
  if (event.keyCode === 13) {
    if(message != ""){
      let my_msg =
      "<p>" + message + "</p><span class='time_date'>" + date + "</span>";
      $("#my_msg").append(my_msg);
      socket.emit("send_message", {
      id,
      message,
      time: date,
      user_img,
      });
    $("#input-message").val("");
    }
    else{
      alert("Please enter your message");
    }
  }
});
```

Figure 58 The client processes the user's message

The server that receives the "send_message" event will save the message to the database and forward the message to the recipient through the "user-message" event and make sure only the person whose _id matches the _id sent from the client can receive the message.

The server handles the "send_message" event and the client receives the "user-message" event and notifies the recipient.

```
socket.on("user-message", (data) => {
  let msg =
    "<p>" +
    data.message +
    "</p><span class='time_date'>" +
    data.time +
    "</span>";
  $("#msg_come").append(msg);
  bt = setInterval(() => {
    $("#chat-icon").css("background-color", "blue");
  }, 50);
});
```

Figure 59 Client handles "user-message" events

V. Testing:

Each project needs a test plan to make sure the system operates at the best possible level. So, planning a project test is important. It's up to the exact job orientation to do. In this project, the aim is to set up the test plan to describe and make all the testing plans.

1. Test plan:

1.1. Function requirements to perform test:

Frontend: Login, User Rights, Home page, Dashboard, Chat, information, Group support, Interactive statistics, Statistics about tutor and student.

Backend: User Management, User Rights Management, information management, message management, group management, Interactive statistics, management of Statistics about tutor and student

1.2. Environment Implementation test

- Sever test: localhost, Heroku.
- Device: Tablet, smartphone, laptop, desktop.
- Browser: Chrome, CocCoc, Microsoft Edge, Mozilla Firefox, Safari.
- Test plan: Microsoft Word.
    - -Test case: Microsoft Word.
- Image Capture Tool: Window capture, Snipping tool.
- Set up Test report: Microsoft Word.
- Tool Support Test Interface: Multi-Browsers

1.3. Test Strategy:

a. Strategy for building test cases:

- The whole interface will be sketched first and will build according to the manuscript.
- All functions of Frontend and Backend need to set up a testcase.

b. Test browser and devices by the following stages:

- Integration test: The only test on PCs mainly using Safari and Google Chrome.
- System test: Test on PC and other devices.

c. Strategy and resources:

- Tran Quoc Viet: Responsible for testing the Staff part.
- Dinh Ngoc Tram: Responsible for testing the main parts of Tutor and Student.

d. Order of test priority:

- Prioritize the frontend to agree upon the draft
- Staff feature: 12/3/2020
- Feature of Tutor: 4/3/2020
- Student features: 9/3/2020

The risks may occur:

Risk	**Solution**
Testers may lack test skills.	More experienced testers will train skills for the remaining members.
Test schedules can be too rigid and difficult to meet deadlines if unforeseen problems occur.	Priority should be given to the components to be tested.
Lack of interaction between testers can lead to errors during testing.	Testers need to constantly exchange test information for each other to make good judgments and come up with reasonable test methods. Avoid mistakes.

2. Test Case:

Staff:

Test case	Input	Desired results	Actual results	Evaluate	Note
1.1 Login	Login with an account already created: Email: viettqgch17138@fpt.edu.vn Password: 12345678	Login successfully and go to the Staff homepage.	Login successfully and go to the Staff homepage.	Pass	12/3/2020
1.2 Login	Login with an account already created: Email: viettqgch17138@fpt.edu.vn Password: 12345678	Cannot login	Login successfully and go to the Staff homepage.	Fail	12/3/2020 Test on URL: http://demoteam.herokuapp.com/
1.3 Login	Login with an account already created: Email: viettqgch17138@fpt.edu.vn Password: 12345678	Login successfully and go to the Staff homepage.	Login successfully and go to the Staff homepage.	Pass	12/3/2020 Test on URL: https://demoteam.herokuapp.com/
1.4 Login	Login with an account already created: Email: hoangpngch17194@fpt.edu.vn Password: 12345678	Cannot login	Cannot login	Pass	12/3/2020 Wrong email.
1.5 Login	Login with an account already created: Email: viettqgch17138@fpt.edu.vn Password: 88888888	Cannot login	Cannot login	Pass	12/3/2020 Wrong password.

1.6 Login	Login with an account already created: Email: viettqgch17138@fpt.edu.vn Password: 12345678	Login successfully and go to the Staff homepage.	Login fail	Pass	15/3/2020 Test on URL: http://demoteam.herokuapp.com/
1.7 Personal Support Management Create New Group with Group ID has special characters	Group ID: @@123 Group Name: Group Support 1 Personal Tutor: Nguyen Ngoc Hai	Create fail, Group ID cannot use special characters.	Create fail, Group ID cannot use special characters.	Pass	8/4/2020
1.8 Personal Support Management Create New Group with Group Name has special characters	Group ID: GCH147 Group name: ***Hai Personal Tutor: Nguyen Ngoc Hai	Create fail, Group Name cannot use special characters.	Success creates.	Fail	8/4/2020 Group name must not use special characters to set. Need to edit validate data.
1.9 Personal Support Management Create New Group with Group Name has special characters	Group ID: GCH147 Group name: ***Hai Personal Tutor: Nguyen Ngoc Hai	Create fail, Group Name cannot use special characters.	Create fail, Group Name cannot use special characters.	Pass	9/4/2020 Test again after edited.

1.10 Personal Support Management Create New Group with duplicate ID	Group ID: 0001 Group name: Group 01 Personal Tutor: Nguyen Ngoc Hai	Create fail, Notify user ID that has been used.	Create fail, Notify user ID that has been used.	Pass	8/4/2020
1.11 Personal Support Management Create New Group with Valid ID and group name	Group ID: 0003 Group name: Group 0003 Hai Personal Tutor: Nguyen Ngoc Hai	Create successful.	Create successful.	Pass	8/4/2020
1.12 Personal Support Management Add students already in the group	Student name: Phan Ngoc Hoang	Add successful.	Add successful.	Pass	8/4/2020 Notice more successful students. But the list will not be repeated every time.
1.13 Personal Support Management Add new student in the group	Student name: Nguyen Ich Nam Cuong	Add successful.	Add successful.	Pass	8/4/2020
1.14 Interactive Management Detail interactive an user	Click to detail of Tutor “ Tran An An”	Show “Chart the number of messages, The number of messages”. Detail number	Show “Chart the number of messages, The number of messages”. Detail number	Pass	29/4/2020

		of Sent and receive messages and time.	of Sent and receive messages and time.		
1.15 Interactive Management Detail interactive an user	Click to detail of Tutor " Tran An An"	Show "Chart the number of messages, The number of messages". Detail number of Sent and receive messages and time. Responsive.	Show "Chart the number of messages, The number of messages". Detail number of Sent and receive messages and time. Not responsive	Fail	29/4/2020
1.16 Interactive Management Detail interactive a user	Click to detail of Tutor " Tran An An"	Show "Chart the number of messages, The number of messages". Detail number of Sent and receive messages and time. Responsive.	Show "Chart the number of messages, The number of messages". Detail number of Sent and receive messages and time. Responsive	Pass	30/4/2020
1.17 Student Be Support	Click to " Student be support "	Show " The proportion of students have a personal tutor"	Show " The proportion of students have a personal tutor"	Pass	29/4/2020

		and detail info of student.	and detail info of student.		
1.18 Student Be Support	Click to " Student be support "	Show " The proportion of students have a personal tutor" and detail info of student. Responsive	Show " The proportion of students have a personal tutor" and detail info of student. Not responsive	Fail	29/4/2020
1.19 Student Be Support	Click to " Student be support "	Show " The proportion of students have a personal tutor" and detail info of student. Responsive	Show " The proportion of students have a personal tutor" and detail info of student. Responsive	Pass	30/4/2020
1.20 Personal tutor support	Click to detail of Tutor "Tran An An"	Show total number of students in charge and list student support. Responsive	Show total number of students in charge and list student support. Responsive	Pass	29/4/2020
1.21 Personal tutor support	Click to detail of Tutor "Tran An An"	Show total number of students in charge and list student support. Responsive	Show total number of students in charge and list student support. Not Responsive	Fail	29/4/2020

1.22 Personal tutor support	Click to detail of Tutor "Tran An An"	Show total number of students in charge and list student support. Responsive	Show total number of students in charge and list student support. Responsive	Pass	30/4/2020

Personal tutor and student:

Test case	Input	Desired results	Actual results	Evaluate	Note
2.1 Login	Login with an account already created: Email: hoangpngch17194@fpt.edu.vn Password: 12345678	Login successfully and go to the Student's homepage, Student's profile, see the role is Student.	Login successfully and go to the Student's homepage, Student's profile, see the role is Student.	Pass	4/3/2020 **Student**
2.2 Login	Login with an account already created: Email: hoangpngch17194@fpt.edu.vn Password: 12345678	Cannot login	Login successfully and go to the Student's homepage, Student's profile, see the role is Student.	Fail	4/3/2020 Test on URL: http://demoteam.herokuapp.com/
2.3 Login	Login with an account already created: Email: hoangpngch17194@fpt.edu.vn Password: 12345678	Login successfully and go to the Student's homepage, Student's	Login successfully and go to the Student's homepage, Student's	Pass	4/3/2020 Test on URL: https://demoteam.herokuapp.com/

		profile, see the role is Student.	profile, see the role is Student.		
2.4 Login	Login with an account already created: Email: saagdkasgdka@fpt.edu.vn Password: 12345678	Cannot login	Cannot login	Pass	12/3/2020 Wrong email.
2.5 Login	Login: account already created: Email: hoangpngch17194@fpt.edu.vn Password: 88888888	Cannot login	Cannot login	Pass	12/3/2020 Wrong password.
2.6 Login	Login: account already created: Email: hoangpngch17194@fpt.edu.vn Password: 12345678	Cannot login.	Cannot login	Pass	15/3/2020 Test on URL: http://demoteam.herokuapp.com/
2.7 Login	Login: account already created: Email: Antagch17777@fpt.edu.vn Password: 123456789	Login successfully and go to the Personal Tutor's homepage, Personal Tutor's profile, see the role is Personal Tutor.	Login successfully and go to the Personal Tutor's homepage, Personal Tutor's profile, see the role is Personal Tutor.	Pass	9/3/2020 **Personal tutor**
2.8 Login	Login: account already created: Email: Antagch17777@fpt.edu.vn Password:	Cannot login	Login successfully and go to the Personal	Fail	9/3/2020 Test on URL: http://demoteam.herokuapp.com/

	123456789		Tutor's homepage, Student's profile, see the role is Personal Tutor.		
2.9 Login	Login: account already created: Email: Antagch17777@fpt.edu.vn Password: 123456789	Login successfully and go to the Personal Tutor's homepage, Personal Tutor's profile, see the role is Personal Tutor.	Login successfully and go to the Personal Tutor's homepage, Personal Tutor's profile, see the role is Personal Tutor.	Pass	9/3/2020 Test on URL: https://demoteam.herokuapp.com/
2.10 Login	Login: account already created: Email: saagdkasgdka@fpt.edu.vn Password: 123456789	Cannot login	Cannot login	Pass	9/3/2020 Wrong email.
2.11 Login	Login with an account already created: Email: Antagch17777@fpt.edu.vn Password: 888888888	Cannot login	Cannot login	Pass	9/3/2020 Wrong password.
2.12 Login	Login with an account already created: Email: Antagch17777@fpt.edu.vn Password:	Cannot login.	Cannot login	Pass	10/3/2020 Test on URL: http://demoteam.herokuapp.com/

	123456789				
2.13 Dashboard of Student	Mouse over the column of the chart.	Displays information of what column the user hovered over	Displays information of what column the user hovered over	Pass	19/3/2020
2.14 Dashboard of Student	Click " Dashboard" on menu.	Responsive and go to Dashboard's page	Not responsive but still go to Dashboard's page	Fail	19/3/2020
2.15 Dashboard of Student	Click " Dashboard" on menu.	Responsive and go to Dashboard's page	Responsive and go to Dashboard's page	Pass	20/3/2020
2.16 Dashboard of Personal tutor	Mouse over the column of the chart.	Displays information of what column the user hovered over	Displays information of what column the user hovered over	Pass	28/3/2020
2.17 Dashboard of Personal tutor	Click " Dashboard" on menu.	Responsive and go to Dashboard's page	Not responsive but still go to Dashboard's page	Fail	28/3/2020
2.18 Dashboard of Personal tutor	Click " Dashboard" on menu.	Responsive and go to Dashboard's page	Responsive and go to Dashboard's page	Pass	29/3/2020
2.19 Chat	Send massage " Hello"	Show to the screen and the recipient can see.	Show to the screen and the recipient can see.	Pass	9/4/2020

2.20 Chat	Send massage “”	Cannot send massage	Show to the screen and the recipient can see.	Fail	9/4/2020
2.21 Chat	Send massage “”	Cannot send massage	Cannot send massage	Pass	10/4/2020
2.22 Book an appointment	Date: 23/4/2020 Time: 21:45 Place: The coffee Note: Problem	Show to the screen and the recipient can see.	Show to the screen and the recipient can see.	Pass	9/4/2020
2.23 Book an appointment	Do not complete the form.	Cannot set the appointment.	Show to the screen and the recipient can see.	Fail	9/4/2020
2.24 Book an appointment	Do not complete the form.	Cannot set the appointment.	Cannot set the appointment.	Pass	10/4/2020
2.25 Send file	Send kasdlahfa.docx Size: 3MB	Show to the screen and the recipient can see.	Show to the screen and the recipient can see.	Pass	7/4/2020 Student
2.26 Send file	Send kasdlahfa.docx Size: 10MB	Cannot send	Show to the screen and the recipient can see.	Fail	7/4/2020 Student
2.27 Send file	Send kasdlahfa.docx Size: 10MB	Cannot send	Cannot send	Pass	7/4/2020 Student
2.25 Send file	Send kasdlahfa.docx Size: 3MB	Show to the screen and the recipient can see.	Show to the screen and the recipient can see.	Pass	25/4/2020 Student

VI. Agile Methodology:

1. Product Backlog:

Starting with the Product backlog, here is a list of project functions after our team has studied the project requirements. Here are the 19 user stories we have listed; we have divided the user stories into sprint backlogs based on the priority for each item and recorded the progress of the project.

Product Backlog

Sort Product Backlog

Story ID	Story name	Status	Size	Sprint	Priority	Comments
1	As a student , I want to login so that I can see my information and my work	Done	3	1		
2	As a student , I want to see my information so that I can check what any wrong	Done	3	1		
3	As a student , I want to receive email notification when staff has notification to students	Done	3	1		
4	As a student , I want to see the dashboard so that I can interact with my personal tutor	Done	3	1		
5	As a student , I want to book an appointment with the personal tutor as needed to resolve their problems encountered when not resolved through chat	Done	3	2		
6	As a student , I want to record the contents of the meeting to complete the job better after meeting the personal tutor	Ongoing	3	3		
7	As a student , I want to upload file so that I can ask the personal tutor	Done	3	7		
8	As a personal tutor, I want to login so that I can do my job	Done	3	1		
9	As a personal tutor, I want to see the dashboard so that I can interact with my student	Done	3	2		
10	As a personal tutor, I want to confirm the appointment with student so that I can meet my student	Done	3	3		
11	As a personal tutor, I want to upload a file of the meeting details so that the students can review the notes.	Done	3	3		
12	As a personal tutor, I want to upload important document files so students can download and research	Done	3	1		
13	As a staff, I want to login so that I can do my job	Done	3	1		
14	As a staff, I want to list the student and the personal tutor to allocate required between student and personal tutor	Done	3	2		
15	As a staff, I want to see the dashboard and document between student and personal tutor so that I can assure the quality of exchanges between students and personal tutor	Done	3	2		
16	As a staff, I want to sent mail to studen and personal tutor so that I can notify them of the information	Done	3	1		
17	As a student, I want to see a chart showing the interaction between me and the tutor.	Done	3	3		
18	As a personal tutor, I want to see charts showing the interaction between me and the students.	Done	3	3		

Figure 60 Product Backlog

2. Sprint Backlog & Burndown Chart:

Sprint 1:

Sprint 1 Backlog

Sort Sprint Tasks | Update Task Slips

Sprint implementation days	20			Effort	Remaining on implementation day...																			
Trend calculated based on last	20	Days	Totals	63	58	57	55	51	50	47	44	43	39	34	32	29	27	21	18	20	16	10	4	
Task name	Story ID	Responsible	Status	Est.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Code database "Login" for Student	1	Hoang, Thu, Hai	Done	5	3	2	1	0																
Code interface "Login" for Student	1	Huy, Viet	Done	5	2	2	1	0																
Test "Login" for Student	1	Viet, Tram	Done	3	3	3	3	1	0															
Code database "Login" for Personal Tutor	8	Hoang, Thu, Hai	Done	5	5	5	5	5	5	2	2	1	0											
Code interface "Login" for Personal Tutor	8	Huy, Viet	Done	5	5	5	5	5	5	5	2	2	0											
Test "Login" for Personal Tutor	8	Viet, Tram	Done	3	3	3	3	3	3	3	3	3	2	0										
Code databade "Login" for Staff	15	Hoang, Thu, Hai	Done	5	5	5	5	5	5	5	5	5	5	3	3	1	0							
Code interface "Login" for Staff	15	Huy, Viet	Done	5	5	5	5	5	5	5	5	5	5	4	2	3	1	0						
Test "Login" for Staff	15	Viet, Tram	Done	3	3	3	3	3	3	3	3	3	3	3	3	1	2	0						
Code database "Dashboard" for Student	4	Hoang, Thu, Hai	Done	12	12	12	12	12	12	12	12	12	12	12	12	12	12	9	7	10	8	5	2	0
Code interface "Dashboard" for Student	4	Huy, Viet	Done	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	11	10	8	5	2	0

Figure 61 Sprint 1

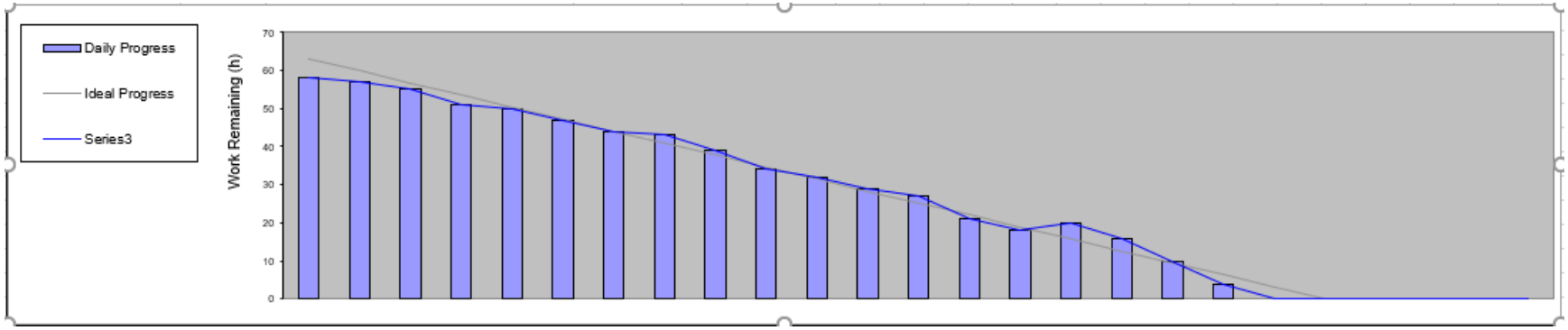

Figure 62 Burndown Chart 1

Sprint 2:

Sprint 2 Backlog

Sort Sprint Tasks | Update Task Slips

Sprint implementation days	20			Effort	Remaining on implementation day…																			
Trend calculated based on last	20	Days	Totals	95	91	87	82	83	79	75	73	66	62	57	52	53	50	42	29	29	20	11	2	
Task name	Story ID	Responsible	Status	Est.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Code database "Dashboard" for Personal Tutor	9	Hoang, Thu, Hai	Done	12	10	8	5	7	5	2	1	0												
Code interface "Dashboard" for Personal Tutor	9	Huy,Viet	Done	12	10	8	6	5	3	2	1	0												
Code database "Dashboard" for Staff	15	Hoang, Thu, Hai	Done	12	12	12	12	12	12	12	12	10	7	5	2	5	3	1	0					
Code interface "Dashboard" for Staff	15	Huy,Viet	Done	12	12	12	12	12	12	12	12	9	8	5	3	1	0							
Code database send mail function	16	Hoang, Thu, Hai,	Done	12	12	12	12	12	12	12	12	12	12	12	12	12	12	8	5	7	3	1	0	
Code interface send mail function	16	Huy,Viet	Done	12	12	12	12	12	12	12	12	12	12	12	12	12	12	10	7	9	5	3	0	
Test send mail function	16	Viet, Tram	Done	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	1	0
Allocate students for personal tutor	14	Hoang,Hai,Thu	Done	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	7	5	2	1	0	
Student books an appointment with pesonal tutor	5	Hoang, Thu, Hai, Huy, Viet, Tram	Done	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	7	5	7	3	1	0

Figure 63 Sprint 2

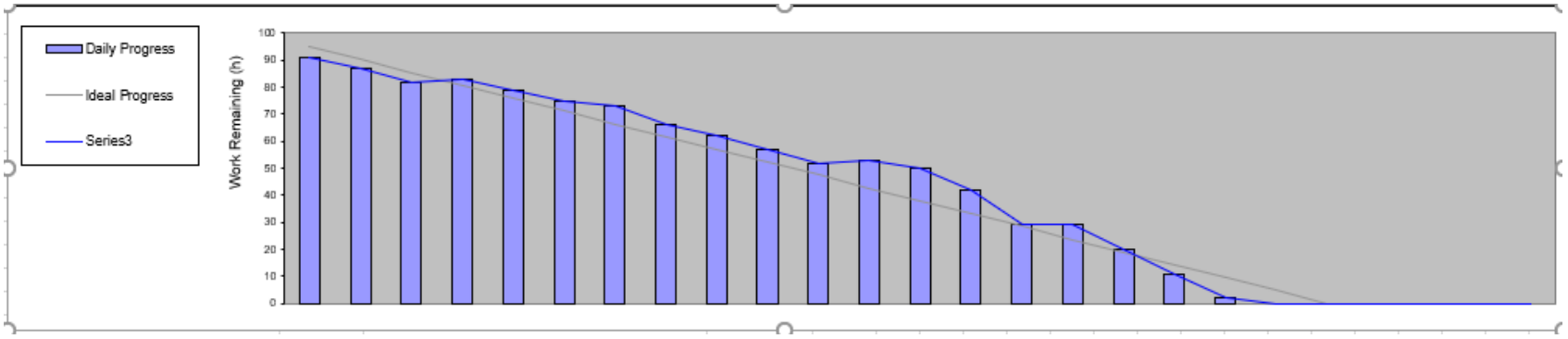

Figure 64 Burndown Chart 2

Sprint 3:

Sprint 3 Backlog

Sort Sprint Tasks | Update Task Slips

Sprint implementation days	20			Effort	Remaining on implementation day...																		
Trend calculated based on last	20	Days	Totals	41	39	39	37	37	35	36	31	32	30	27	26	25	26	20	13	9	6	3	1	
Task name	Story ID	Responsible	Status	Est.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Student uploads files	7	Hoang, Huy, Viet, Hai , Tram, Thu	Done	7	5	5	3	3	1	2	0													
Personal tutor confirm the appointment with student	10	Hoang, Huy, Viet, Hai , Tram, Thu	Done	5	5	5	5	5	5	5	2	3	1	0										
Personal tutor uploads file	12	Hoang, Huy, Viet, Hai , Tram, Thu	Done	5	5	5	5	5	5	5	5	5	5	3	2	1	2	1	0					
Create statistical charts of interaction between tutors and student for student	17	Hoang, Huy, Viet, Hai , Tram, Thu	Done	12	12	12	12	12	12	12	12	12	12	12	12	12	12	7	5	2	0			
Create statistical charts of interaction between tutors and student for personal tu	18	Hoang, Huy, Viet, Hai , Tram, Thu	Done	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	8	7	6	3	1	0

Figure 65 Sprint 3

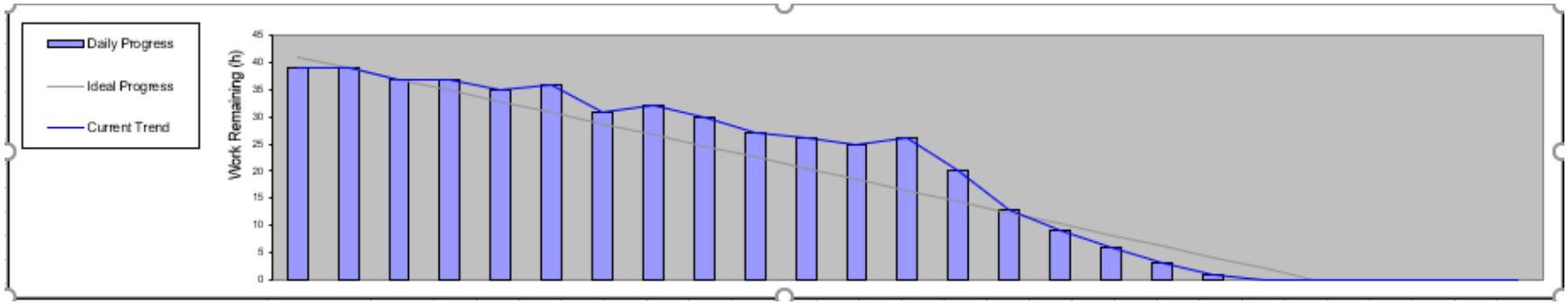

Figure 66 Burndown Chart 3

3. Meetings:

3.1. Sprint meeting:

The first meeting at the start of the project was held on 29/2/2020 at 8:00 am.

- The meeting was organized with the following main ideas:
- The first thing we raised and discussed was the issue of each person's individual competence. Assign work based on the network points and weaknesses of each team member.
- Next, we analyze the project and divide the work by the weeks based on the importance of the feature and the members who will be in charge of the work.
- Define product backlog for weeks.

Choose a language to develop the project.

3.2. Daily Meeting:

Sprint 1 (1/3/2020- 21/3/2020):

We set 3 timelines for sprint 1 to monitor the situation.

The first timeline on March 7, 2020, we met together and divided the work as follows:

- Ngoc Hoang and Quang Huy: List the important features that need to be done first for the project in sprint 1.
- Ngoc Hoang:
- Ngoc Tram: Researching requirements and collecting information to create documents.
- Quoc Viet, Ngoc Hai, Dinh Thu: Research on the language used to develop the project.

The second timeline on March 14, 2020, we have released the results of our work during the 1st milestone of operation and report the work progress will be made in the 2nd timeline:

- Ngoc Hoang and Quang Huy: During the first milestone, Ngoc Hoang researched complete features for the project and MongoDB and NodeJS language training for Quang Huy, Dinh Thu, Ngoc Hai and Quoc Viet. Ngoc Hoang reports that in the next timeline Ngoc Hoang will code the database for the login function for students and tutors.

- Quang Huy: During the first milestone of operation, Quang Huy studied MongoDB and NodeJS. In the second timeline, Quang Huy will code the interface for student and tutor login.
- Ngoc Tram: During the first milestone in operation, Ngoc Tram created a Product backlog. In the second milestone, Ngoc Tram will study and practice Scrum and create test plan for login function.
- Dinh Thu and Ngoc Hai: During the first period, Dinh Thu and Ngoc Hai did research on MongoDB and NodeJS. In the second time, Dinh Thu and Ngoc Hai will research and code database for the login function of students and tutors.
- Quoc Viet: During the first milestone, Quoc Viet researched on MongoDb and NodeJS. During the second time, Quoc Viet will study and code interface functions login of students and create test plan for login function.

The third timeline on March 21, 2020:

- Ngoc Hoang, Dinh Thu, Ngoc Hai: in the second timeline, Ngoc Hoang has completed the login function code for students. In the third milestone, Ngoc Hoang will study and code the Dashboard function for students.
- Quang Huy, Quoc Viet: in the second milestone completed interface code for Staff login function.
- Ngoc Tram: in the third time you will practice Scrum and in the fourth time Ngoc Tram will create a test plan.
- Ngoc Tram and Quoc Viet: After the code team finished logging in for Staff, Quoc Viet and Ngoc Tram performed a number of test cases.

Sprint 2 (22/3/2020- 10/4/2020):

The first timeline on 28/3/2020:

- Ngoc Hoang: The third timeline of Sprint 1, Ngoc Hoang completed the work and researched the code database for Dashboard function for tutors. In the first milestone of sprint 2, Ngoc Hoang will code for Dashboard function for tutors.

- Quang Huy: 3rd time of sprint 1, Quang Huy has completed the interface code for the dashboard function for staff. During the 1st milestone of sprint 2, Quang Huy will code for the tutor's dashboard feature.
- Ngoc Tram: in the 3rd milestone of sprint 1, Ngoc Tram tested several cases for the login function of staff, students, and personal tutors. In the first milestone of sprint 2, Ngoc Tram will start creating reports for the dashboard creation function for the staff and complete the sprint 1.
- Dinh Thu and Ngoc Hai: in the 3rd timeline of sprint 1, Ngoc Hai and Dinh Thu has completed the code dashboard for students.
- Quoc Viet: in the 3rd milestone of sprint 1, Quoc Viet has completed the interface code for the student dashboard function and tested some cases for the login function of staff, student, and tutor. During the 1st milestone of sprint 1, Quoc Viet will research and code functions for tutors.

Second timeline on 4/4/2020:

- Ngoc Hoang, Dinh Thu, Ngoc Hai: code database for Staff dashboard function.
- Quang Huy and Quoc Viet: interface code for the dashboard function of Staff.
- Ngoc Tram: Deploy report creation for Staff dashboard function.

The third timeline on 10/4/2020:

- Ngoc Hoang, Dinh Thu, Ngoc Hai: code databasse for sending mail, assigning students to staff's personal tutors, and student appointment scheduling function with tutor.
- Quang Huy and Quoc Viet: interface code for sending mail function.
- Ngoc Tram: Deploying the work report for the function of asending mail, assigning students to tutors, and the function of scheduling students with tutors.
- Ngoc Tram and Quoc Viet: planning and testing send email function.

Sprint 3 (11/4/2020-30/4/2020):

The first timeline on April 17, 2020:

- Ngoc Hoang, Dinh Thu, Ngoc Hai: by the end of sprint 2, the code has finished sending emails, allocating students with tutors, and scheduling function between student and tutor. During the first sprint 3 timeline, Ngoc Hoang, Dinh Thu, and Ngoc Hai will code the database for function upload file of student's role.
- Quang Huy and Quoc Viet coded the interface of the staff's mailing form. During the first timeline of sprint 3, Quang Huy and Quoc Viet will code the interface of function upload file of student's role.
- Ngoc Tram added a function to send mail, allocate students to tutors and a student's scheduling function with teachers in last timeline of Sprint 2. In this timeline, Ngoc Tram will add function upload file of student's role.
- Tram and Quoc Viet tested the send mail function of staff, the scheduling function between students and tutors during the 3rd milestone of sprint 2. The first timeline of sprint 1, Ngoc Tram and Quoc Viet will test the function upload file of student's role.

The second timeline on 24/4/2020:

- Ngoc Hoang, Dinh Thu, Ngoc Hai: For the first time, 3 people have finished coding the database for student file upload function. At the second time, 3 people will code the database for appointment confirmation function and file upload function of personal tutor.
- Quang Huy and Quoc Viet coded the interface to upload students' files. At the second time, two people will code the interface for the appointment confirmation function and the file upload function of personal tutor.
- Ngoc Tram added the student file upload function report in a timeline. In the second milestone, Ngoc Tram will add a report to the personal confirmation and file upload function of personal tutor.

- Tram and Quoc Viet tested the student file upload function in the first timeline. In the second timeline, Ngoc Tram and Quoc Viet will test the file upload function of personal tutor.

Third timeline on 30/4/2020:

- Ngoc Hoang, Dinh Thu, Ngoc Hai: on the second timeline, 3 people have finished coding the database for appointment confirmation function and file upload function of personal tutor. At the third time, 3 people will code for the chart creation function to show the interaction information between students and tutors for students and tutors.
- Quang Huy and Quoc Viet have coded the interface of the appointment confirmation and file upload function of the personal tutor in the second timeline. At the third time, two people will code the interface for creating a chart to show interaction information between students and tutors for students and tutors.
- Ngoc Tram has added a report to confirm the appointment function and the file upload function of personal tutor in a timeline. In the third milestone, Ngoc Tram will add a function to create chart to show interaction information between students and tutors for students and tutors.
- Tram and Quoc Viet tested the uploading of personal tutor files within the second timeline. In the third time, Ngoc Tram and Quoc Viet will test the chart function of students and students.

VII. Conclusion

The above is a summary of all the processes that we have completed this project. We have launched a website system that fully meets the requirements of customers. Even though it was a little difficult in the process, we finished the assigned work on time. Applying Scrum to the project has helped us save a lot of time and solve a lot of problems that arise in the process of working. In the future, we will try to develop the site with even more optimized features.

Bibliography

Hooda, P., n.d. *MongoDB vs MySQL - GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/mongodb-vs-mysql/
[Accessed 2020 April 29].

MongoDB, 2020. *What Is MongoDB?.* [Online]
Available at: https://www.mongodb.com/what-is-mongodb
[Accessed 29 April 2020].

Romanyuk, O., 2019. *Node.js is a great runtime environment - and here's why you should use it.* [Online]
Available at: https://www.freecodecamp.org/news/what-are-the-advantages-of-node-js/
[Accessed 29 April 2020].

Seda Unal, Yuchen Zheng, 2017. *Distributed Databases: SQL vs NoSQL.* [Online]
Available at:
https://www.cs.rochester.edu/courses/261/spring2017/termpaper/09/Unal_Yuchen_Paper.pdf
[Accessed 2020 April 29].