OULU UNIVERSITY OF
APPLIED SCIENCES

Trung Anh Nguyen

WORKING DIARY THESIS AS A WEB DEVELOPER

WORKING DIARY THESIS AS A WEB DEVELOPER

Trung Anh Nguyen
Bachelor Thesis
Autumn 2020
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of

Author: Trung Anh Nguyen
Title of Bachelor´s thesis: Working Diary Thesis As A Web Developer
Supervisor: Veijo Väisänen
Term and year of completion: Autumn 2020 Number of pages: 52

The subject of this bachelor's thesis was to report the result of my working days in a Smilee company as a web developer. In this thesis, I will discuss my daily works and tasks. Also, I will try to list out some of the difficulties I have faced and my solutions for it.

During my working time at Smilee, I worked on the agent website of the company. I was part of the development team that is working to improve the website's user experience and user interface, develop a new website's feature, and fix bugs. When I started the thesis, I had already worked in the company for 3 months. The thesis is a look at my personal view of development as a developer. It will show the progress of improvement I have gained during the time working for the thesis report.

As a result of this thesis, I was able to look back on my progression as a developer. I can surely say that I have grown as a developer. My skills in designing a User Interface for the web, coding skills, and how to approach and solve problems have been improved.

Keywords: report, daily, web developer, front-end, back-end

CONTENTS

1 INTRODUCTION

This diary thesis is made based on the working process of me in the Smilee company from 15.2.2020 to 30.4.2020, 10 weeks of working full-time. In this diary report, I will discuss my daily activities as a Web Developer. I will analyze my progress as we go further to every task that I have made, my decision toward each problem and how the communication in the developer team of the company has helped me to solve the task and how far have I gone in a professional field of working in the IT industry.

1.1 Background

My daily tasks as a developer are development, user interface design, bug fixing, and writing test files for designed features. Our development team consists of four people: The lead-developer, the back-end developer, and two front-end developers. I am one of two front-end developers who work mostly in design and develop new features for the website of the company. Other tasks can be finding and fixing bugs or sometimes writing a test file for designed features. The application is a website we use for our agents to chat and manage conversations with a client and to modify settings for the conversations, tracking data for the chat. My work is mostly in the front-end field which is designing and making a designed feature into reality but my team does not separate Back-end or Front-End individually and clearly so sometimes I can work on a server-side or database task depending on the backlog and task goal.

Our web is run on the React framework and most of our User Interface is made with HTML5 and CSS3 and the most important syntax are ES6 and JavaScript. The developing environment is run based on Linux OS and using a browser, such as Chrome, Firefox, Microsoft Edge for UI view results. Our Back end is based on GraphQL and Apollo.

For me to handle all the required tasks in the team, I had previous experiences on JavaScript, ES6, and HTML5, CSS3 before so when I got into the position, what I had to learn throughout the time is GraphQL to deal with basic database problems. Then I must learn how things are running together to create a complete website so I can know more when fixing bugs. It is crucial to know how the process step by step works and how modules work with each other to do the tasks well.

1.2 Skills

Joining the team means that I must have some skills to finish my work probably and I think they are coding skills, information utilization, communication, and problem-solving. There are reasons why those skills are chosen to suit for being a developer. Coding skills are essential for being a developer, I cannot program without having a knowledge of programming skills. Information utilization is the speed of how I can process and get what I need from the source when I need so I can ensure the speed of task processing. Communication is also needed for connecting between every member of the developing team so the fastest and best outcome can be achieved. Teamwork is always better than handling problems alone. Problem-solving is how I can process every time when facing a problem. I think that I have improved mine a lot since I work for the company. In that way approaching and dealing with tasks is more useful and bright.

1.3 Working Environment and Goals

My office is near my house and my team has 4 members. The language of communication is English. I am having a daily task which is assigned based on the customer's demand, bugs appear, feedback from the test team, and schedule for new features launch. I got instructions from the lead developer to work with those tasks and being helpful when I need it.

For the goals, I will evaluate my learning and professional development during 10 weeks of working. Looking further into my works and analyzing them. In the end, I think I want to have a good overall result in the view of a developer. I can be able to evaluate the growth of my knowledge and skills.

2 OVERALL VIEW

My team is a developing team consisting of 4 people: lead developer, one back-end main developer, 2 front-end main developers. I am one front-end main developer and my task is mostly relevant to user interface and design but sometimes I also have tasks for the back-end. We are a dynamic team with flexible skills to handle all kinds of tasks, using Jira as a method to manage and build our tasks. I usually have tasks assigned to Jira and if the task is included with a new design feature, I will have to design and search for the lead developer's approval and show a demo for the lead developer after the work is done.

2.1 Daily work tasks

I started the day with my team having the daily SCRUM where every member tells about these things:

- What did I do yesterday?
- What will I do today?
- Are there any impediments?
- Any knowledge in the field you do not know that others might help with?

After the daily SCRUM, we have our duties for the day and start to work. I will have a mini-meeting with the lead developer to discuss the design task, asking for the desired prototype and giving my opinion on it, and finally, we come out for the final result. Then I will get back to the work by designing the prototype with the Figma sketch. We organize and manage tasks with our backlog where we have all our tasks. Tasks can be divided into a new feature and a bug fix. A new feature task is to develop a new feature and a bug fix is fixing a task for occurred bugs in the web app. Solving tasks really depends on the difficulty of the task. I usually develop a new feature for a couple of days including design and coding. A bug fix task can usually be a couple of hours but there are few cases that I solved in 3,4 days when it is hard to deal with. Also, writing a test file is my duty when new features are implementing, old test file is not correct anymore and I have to write new test files using jest. Also, the requirement for each test file is that the coverage must be 70% and above.

3 DIARY REPORT

This part will list out my diaries for 10 weeks during several big projects and the different tasks I made. Here I will analyze my works and comments about it.

3.1 Week 1: Remaking Archive List and adding more information for chat

3.1.1 Remaking Archive List

During my first week at work, my first task was to recreate the old design from using the style by CSS flexbox based on using CSS grid-based. So, my first move was to have a conversation with the lead developer about the task. The Archive List page based on what he said to me is that they used to style that page with the CSS flexbox quite a long time ago. Now when the demand for multiple platforms and the large variation of resolution from a mobile to a desktop screen has increased a lot so the user interface might break and look bad on some resolutions. So, the lead developer decided to change from the CSS flexbox to the CSS grid in order to keep the interface from broken down. The lead developer sent me a link to the CSS grid tutorial and I spent most of the day learning the CSS grid and testing its syntax.

Then I know how the CSS grid and the CSS flexbox work more probably. The basic difference between the CSS Grid Layout and the CSS Flexbox Layout is that the flexbox was designed for the layout in one dimension - either a row or a column. The grid was designed for a two-dimensional layout - rows, and columns at the same time. (1.)

Here are some things where Grid is specifically better at than Flexbox:

- Making whole page layouts. It is better for that even just considering layout performance reasons.
- Making literal grids. Like X columns with Y gap between them homegrown framework stuff. grid-gap is wonderful, as gutters are the main pain point of grid systems.
- Requiring fewer media query intervention with powerful functionality like auto layout, min-max(), repeat(), and auto-fill. Here is a neat idea. (2.)

Next, when I am able to know the CSS grid syntax properly, I start to look further into the component's code of the Archive List component. Looking through its style code and determining where is the flexbox style for the Archive List to remove it and replace it with the CSS grid. During the day I was able to replace the CSS flexbox code to the CSS grid. And here is some code replacement I have made. (FIGURE 1)

```
const ChatArchiveRoot = styled.div`
  display: grid;
  grid-template-columns: [first] 250px [second] 250px [third] 250px [fourth] 1fr
  grid-template-rows: [navigation] 50px [content] 1fr [end];
  align-items: center;
  grid-row-gap: 15px;
`
```

FIGURE 1. CSS grid code for Archive List page

The next day is some adjustments I made to sort the user interface back to normal, styling the page so it is fitted with the design. Here are some images of the page after being re-style with the CSS grid. (FIGURE 2, FIGURE 3)

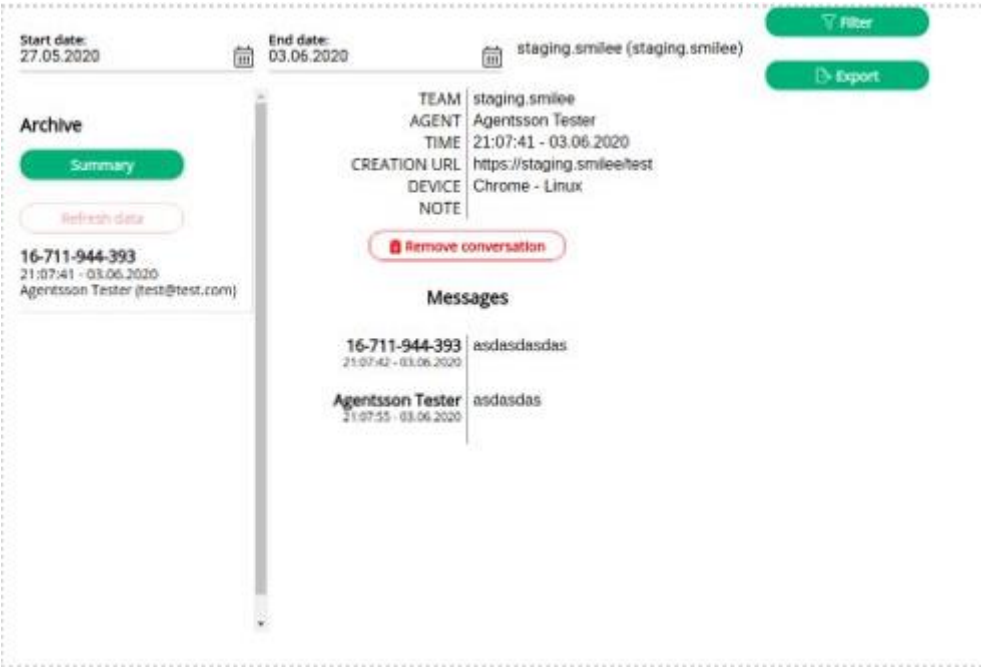

FIGURE 2. Archive List page after styling with CSS grid

.

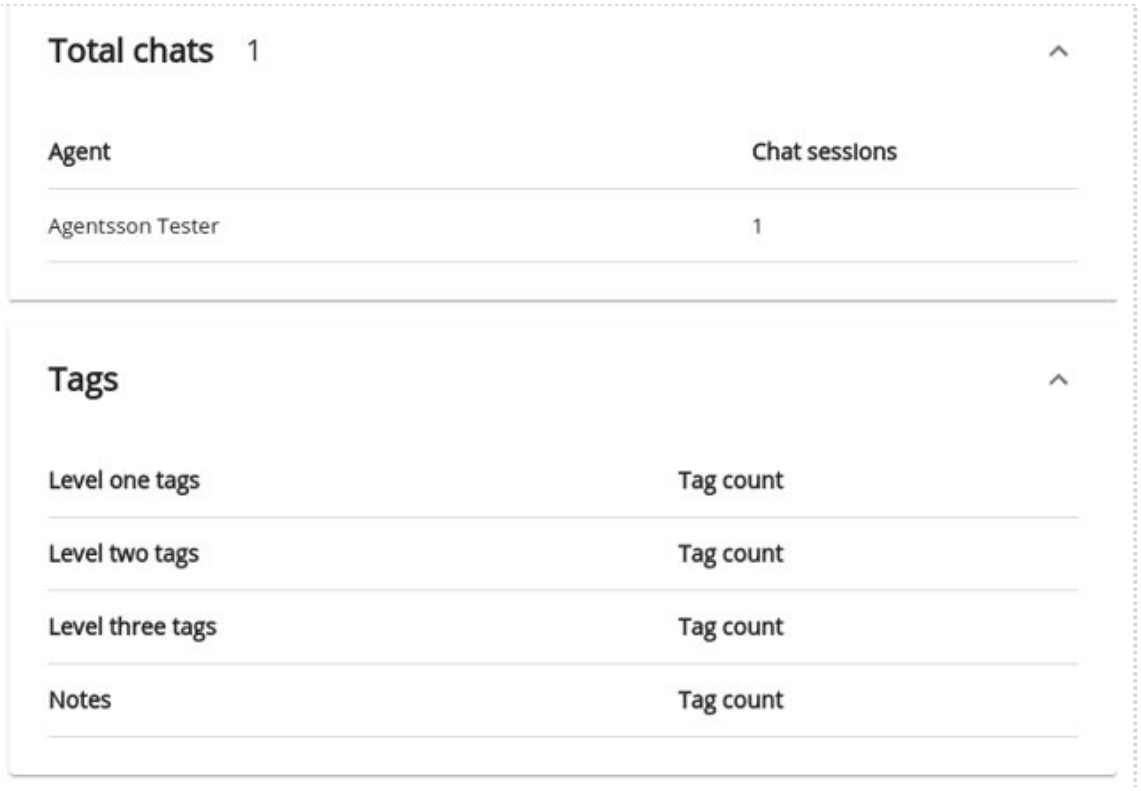

FIGURE 3. Archive List page in view session with CSS grid

This first task took me 3 days to complete in every step so I can produce a new user interface using the CSS grid and the most important thing is that now, the user interface of the Archive List page will not break in every resolution and any platform like mobile, tablet or desktop.

3.1.2 Adding more information to chat history

The Archive List page is a page where you can see an old conversation chat between the client and the agent. It lists the details of each conversation. Since the lead developer wants to show out more information about the talk beside the client's name, the date, the time. I was receiving a task to add more information to chat history. The object that we are aiming to add is the user's email because to identify the customer by name can sometimes be mistaken but with additional information, such as the user's email, we will be able to clarify each person clearly without any mistake.

First, I have made a research on the code and how the chat data is being fetched to the table and displayed out to the user interface. Then I take a further look into the database schema and find what information we need to add more to the chat information display and their properties so it can be called out in the function to be fetched out and displayed on the screen.

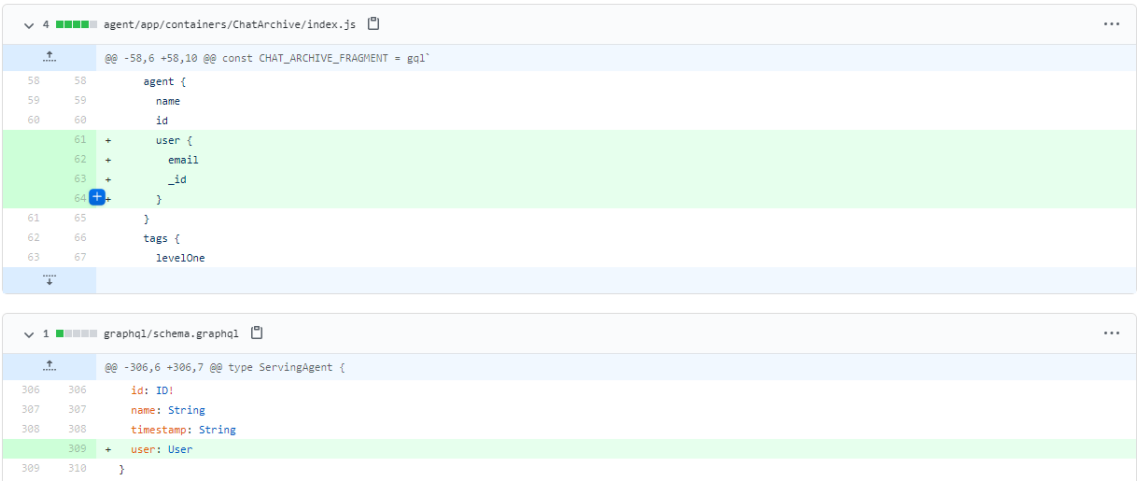

FIGURE 4. Adding user email to GraphQL Schema

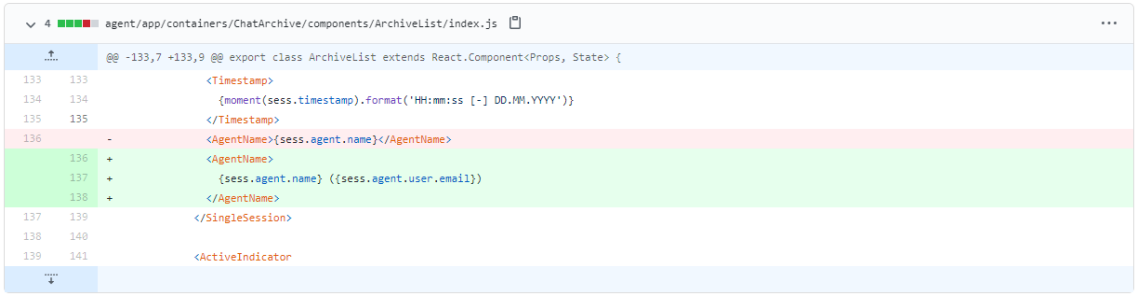

FIGURE 5 Adding email along with the user's name

It takes me one day to learn the code of how to chat information fetching and one day to add more user email to backend GraphQL(FIGURE 4) and fetch it to the screen to show user email. And the result is the email has been fetched and display along with the user's name to identify more clearly for agent. (FIGURE 5, FIGURE 6)

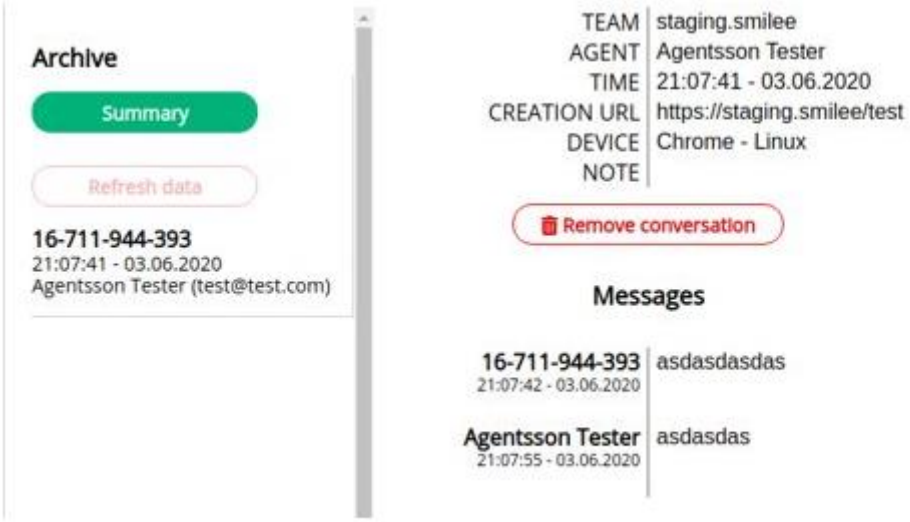

FIGURE 6. User's name displays along with email on the left column

3.1.3 Weekly Analyzing

For this week, I am dealing with two tasks and the difficulty is normal in my opinion and my working speed is fast based on all the steps I take in order to work probably. The struggle I have faced this week is learning the new syntax for the CSS grid and understand the difference between the CSS grid and the CSS flexbox. So, I can be able to replace the CSS flexbox's code with the grid to have the same page UI and the page is not broken anymore. The next task is learning the GraphQL so I can add a new field to the database and being able to fetch out to the screen. I think I did a great job this week without any struggle and I want to work better next week.

3.2 Week 2: Making an expansion panel for Tab Customize and adjusting width layout

3.2.1 Making an expansion panel for Tab Customize and adjusting width layout

This week, I am assigned a task is to move all content of the Tab Customize page into an expansion panel and adjusting the layout of Tab Customize when the resolution is decreasing. The Tab Customize page is where the agent can customize our settings such as icon and icon position. My first move is to look at the expansion panel of material-UI. Material-UI is a front-end framework our company uses for the web user interface. I must check through the syntax of the expansion panel API in the material-UI framework. The old design of Tab Customize is to list out each part of the

settings, now I am grouping each content into the group and put each group into an expansion panel. (FIGURE 7)

Here is the result after I group up Tab Customize:

FIGURE 7. Tab Customize group in expansion panel

Here is an example of one specific expansion panel being open: (FIGURE 8)

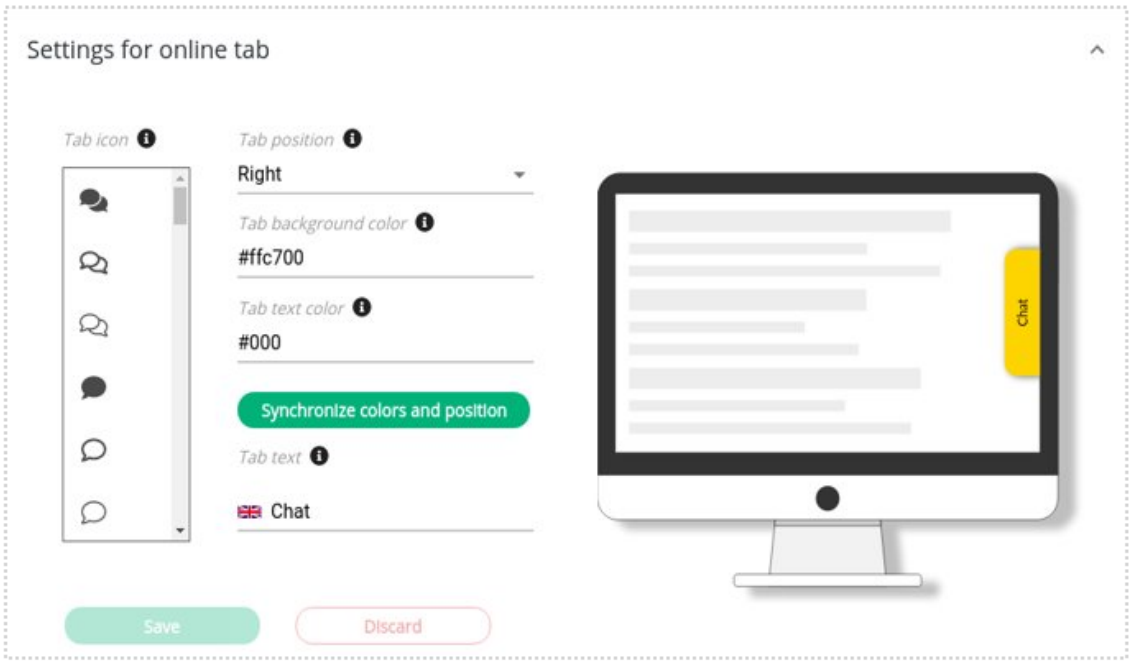

FIGURE 8. Example of an open panel

After every content of the Tab Customize is group into a panel. I made some changes in responsive design so the layout can be rearranged when the width is shrunken down. (FIGURE 9)

FIGURE 9. Coding for width layout of Tab Customize page

As I take the layout with variable smaller than window length is 1000 pixel, which is when the resolution is shrunk down to phone and tablet size. The conditional style below called ColumnLayout which will be trigger styling the flex-direction in CSS when the window length is smaller than 1000-pixel flex-direction will be changed into a column. It will make all the content stack up in a column and when the width is larger than 1000 pixels, flex-direction will be a row and the content will be lineup from left to right now stacking from top to bottom like in mobile resolution.

3.2.2 Weekly Analyzing

For this week, I was taking on those tasks that are quite easy to do, for the first task all I have to do is to read the article about the Expansion Panel of Material UI framework. After that is the changes and replacements for code adding the expansion panel code to wrap around the old code of the Tab Customize page then I made some adjustments with the styles to complete.

3.3 Week 3 Adding features for the domain in the homepage

3.3.1 Styling domains in the homepage for expired domains:

For this week the goal for this task is to style domains whenever it is expired so I am looking for conditional styling in styled-components to do it.

The styled-component is a package of React framework, it is very popular in user interface styling.

styled-components is the result of wondering how we could enhance CSS for styling React component systems**.** By focusing on a single use case we managed to optimize the experience for developers as well as the output for end users.

Apart from the improved experience for developers, styled-components provides:

- **Automatic critical CSS**: styled-components keep track of which components are rendered on a page and injects their styles and nothing else, fully automatically. Combined with code splitting, this means your users load the least amount of code necessary.
- **No class name bugs**: styled-components generates unique class names for your styles. You never have to worry about duplication, overlap, or misspellings.
- **Easier deletion of CSS**: it can be hard to know whether a class name is used somewhere in your codebase. styled-components make it obvious, as every bit of styling is tied to a specific component. If the component is unused (which tooling can detect) and gets deleted, all its styles get deleted with it.
- **Simple dynamic styling**: adapting the styling of a component based on its props or a global theme is simple and intuitive without having to manually manage dozens of classes.
- **Painless maintenance**: you never have to hunt across different files to find the styling affecting your component, so maintenance is a piece of cake no matter how big your codebase is.
- **Automatic vendor prefixing**: write your CSS to the current standard and let styled-components handle the rest.

You get all of these benefits while still writing the CSS you know and love, just bound to individual components.(.3)

After reading styled-component, I made a Boolean in the react file to validate the domains if it is expired or not. (FIGURE 10)

```
let isExpired
let d1 = new Date()
const team = teams.find(team => team._id === teamId)

if (d1.getTime() < Date.parse(team.domain.validUntil)) {
  isExpired = false
} else {
  isExpired = true
}
```

FIGURE 10. Sorting function to indicate expired domains

Then I attached the Boolean variables that indicate the domain's expiration to the row of the domain that needs to style. The Boolean that indicates from the function above isExpired is linked with the variable validDateExpired so I can style the UI feature based on the domain is expired or not. (FIGURE 11)

```
<TableRowStyle
  validDateExpired={isExpired}
  onMouseEnter={() =>
    isExpired && this.onMouseEnter(teamId)
  }
  onMouseLeave={
    isExpired ? this.onMouseLeave : undefined
  }
>
```

FIGURE 11. Link the Boolean with domain UI feature

Then come to the styled-components part where I style the UI TableRowStyle based on the status of the domain if it is expired or not. (FIGURE 12)

```
const TableRowStyle = styled(TableRow)`
  display: flex;
  height: 54px;
  :nth-child(2n) {
    background: #f0f0f0;
  }
  opacity: ${props => (props.validDateExpired ? 0.5 : 1)};
  cursor: ${props => (props.validDateExpired ? 'not-allowed' : 'default')};
`
```

FIGURE 12. Styling code for the domain, based on expired Boolean

The style makes the domain row go blur and not available to click in. You can see the expired date on the hover.

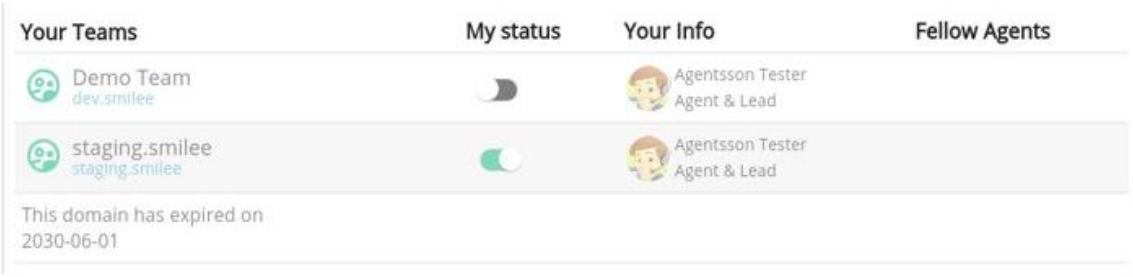

FIGURE 13. Domain rows when expired

The style for domains also paints grey the background for the even number row so you can easily notice the domains without confusing.

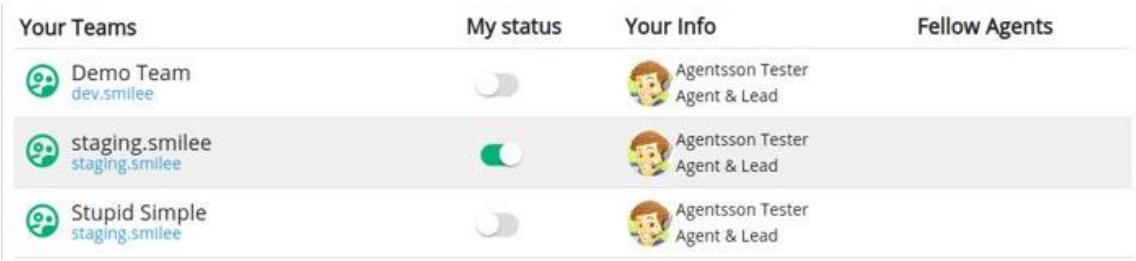

FIGURE 14. Domain row styling for easy indicating

3.3.2 Fix color rendering row bug:

After finishing the styling and expire feature for domains in the dashboard, there is one bug that appears when there are multiple expired domains. When we hover one expired row, other rows are automatically re-render. The color background which makes the screen very unappealing to the eyes, so I have to fix the bug.

```
  const TableRowStyle = styled(TableRow)`
@@ -84,12 +84,8 @@ const TableRowStyle = styled(TableRow)`
     :nth-child(2n) {
       background: #f0f0f0;
     }
-    max-width: 65%;
     opacity: ${props => (props.validDateExpired ? 0.5 : 1)};
     cursor: ${props => (props.validDateExpired ? 'not-allowed' : 'pointer')};
-    :hover {
-      opacity: ${props => (props.validDateExpired ? 0 : 1)};
-    }
   `
```

FIGURE 15. Fix color re-render bug in domain rows

The solution for the bug is not using hover for element row but using props validDateExpired to manipulate all the style of the row which means it is only change based on expired Boolean and not re-render when hover. (FIGURE 15)

3.3.3 Week Analyzing

For this week I successfully made a new feature and re-style the domain user interface on the homepage and have a small problem with render style bugs CSS. But after 2 days I finally manage to fix it. This week of working help me learning in-depth about props styling with CSS and styled component.

3.4 Week 4 Remake UI for settings tab, making new features for Contact Form

3.4.1 Remake UI for setting tab

Week 4 and I was assigned a task to remake the user interface for General Settings in Setting Tab. I was planning to use the framework material-UI for remaking these features on the General Settings Tab because it is a good and easy framework to use with a nice user interface.

Material-UI is an open-source project that features React components that implement Google's Material Design. It kick-started in 2014, not long after React came out to the public, and has grown

in popularity ever since. With over 35,000 stars on GitHub, Material-UI is one of the top user interface libraries for React out there. (. 4)

Not only does it address the problems inherent with LESS, but it also unlocks a ton of features, including:

- dynamic styles generated at runtime
- nested themes with intuitive overrides
- reduced load time with code splitting (. 4)

For such an incredible features material provide, I can easily use and manipulate the user-interface and styling it to follow my will so I can archive the best user interface.

Here is the result of the General Settings Tab after remaking it: (FIGURE 16)

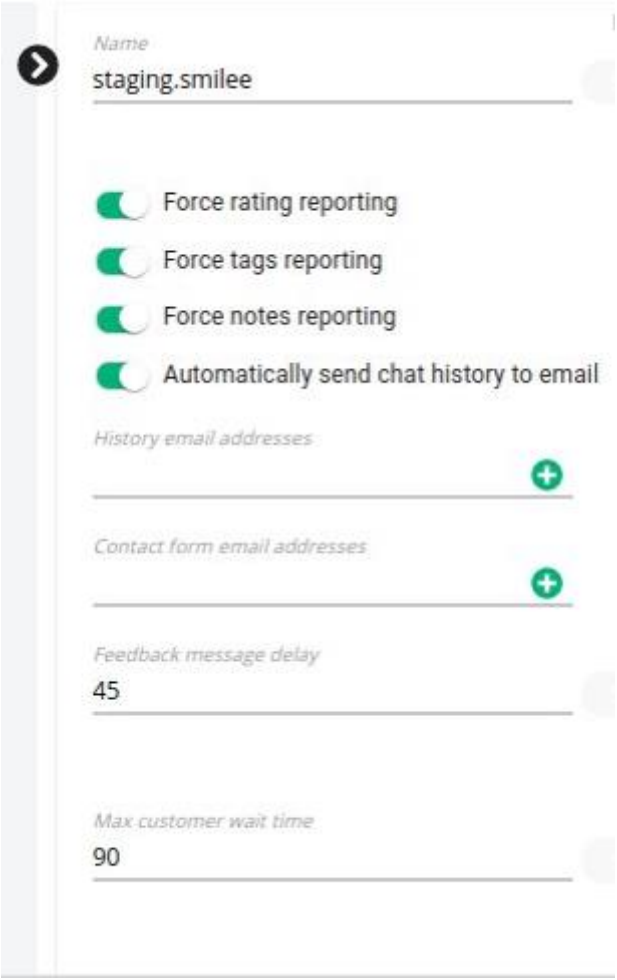

FIGURE 16. General Settings Tab after a remake

After using material-UI on the website, the default color and shape of those user interface features are not as expected so the interesting things about the material-UI framework are you will be able to override the default style and styling it follow your will. There are many ways you can override the style of a feature:

- Specific variation for the one-time situation: overriding style with class names, class, global class names
- Dynamic variation for the one-time situation: Dynamic CSS, Class name branch, CSS variables, inline-styles, Theme nesting
- A specific variation of a component
- Material Design variations
- Global theme variation: Theme variables, Global CSS override, Global theme override

I choose to use the Global theme variation- Global theme override because it is easy to use and you can see the name tag of feature you need to override in the debug tool.
For example, these code lines are how I am using Global theme variation to the style text field of material-UI.

```
import { MuiThemeProvider, createMuiTheme } from '@material-ui/core/styles'

const timePickerOverrideTheme = createMuiTheme({
  overrides: {
    palette: {
      primary: { main: '#02A676' },
      secondary: { main: '#D82B03' }
    },
    MuiInputLabel: {
      root: {
        '&$focused': {
          color: '#02A676'
        }
      }
    },
    MuiPickersToolbar: {
      toolbar: {
        backgroundColor: '#02A676'
      }
    },
    MuiPickersClock: {
      pin: {
        backgroundColor: '#02A676'
      }
    },
```

FIGURE 17. Examples of styling override theme

In these codes, I am changing the color from the default color to the desired color in the design. (FIGURE 17) Such as MuiInputLabel is the variable that indicates the color of the input text field, I am changing it to another color by calling the feature name and address new color to it.

3.4.2 Adding a new feature to the contact form

The next task I was working on is to add a new category named “Instruction Text” to the field of a contact form. The task is to add a feature that can add a new field “Instruction Text” onClick to contact form. Both in User Interface and in GraphQL back-end.

First, I add a new row in the schema of GraphQL back-end named INSTRUCTION to store the given variable we save into the field “Instruction Text” later. (FIGURE 18) The schema below is the schema for Contact Form.

1 graphql/src/schema/typeDefs/team.graphql

```
@@ -16,6 +16,7 @@ enum FormFieldType {
    PHONE
    SELECT
    SELECT_MULTIPLE
+   INSTRUCTION
  }

  enum AssistantContentLanguage {
```

Figure 18. Adding new filed of data to Contact Form’s schema

Next, I code a new feature name InstructionPanel that can be added into the contact form. (FIGURE 19)

```
+ export default function InstructionPanel(props) {
+   let item = props.item
+   let fieldList = props.fieldList
+   let updateFieldList = props.updateFieldList
+   return (
+     <MuiThemeProvider>
+       <ExpansionPanelDetailsStyle>
+         <InstructionComponent
+           fieldList={fieldList}
+           updateFieldList={updateFieldList}
+           item={item}
+         />
+       </ExpansionPanelDetailsStyle>
+     </MuiThemeProvider>
+   )
+ }
```

FIGURE 19. New Instruction Panel component to render

All the data save in InstructionPanel will be saved into the variable fieldList and it will be updated when we press the save button.

Next, I import the Instruction Panel component I just made into the contact form so we can use it and also styling the component. (FIGURE 20)

```
+                <MenuItem
+                  onClick={() =>
+                    updateFieldList({
+                      action: 'add',
+                      label: '',
+                      required: false,
+                      type: 'INSTRUCTION',
+                      values: [],
+                      _id: generateTemporaryId()
+                    })
+                  }
+                >
+                  Instruction
+                </MenuItem>
```

FIGURE 20. onClick function render Instruction Panel component

The instruction panel is now imported into the Menu so you can select it. Here is the result in the user interface:

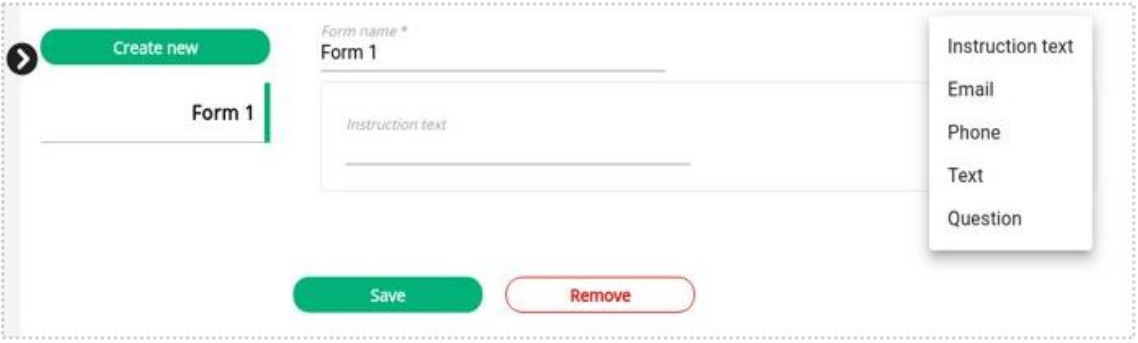

FIGURE 21. Instruction Panel component on the web

The Add field button is the menu we imported the InstructionPanel in, so now we can add a new Instruction Panel into the contact form and use it. It is also can be saved into the database by adding the variable INSTRUCTION into schema earlier.

3.4.3 Week Analyzing:

For this week I was taking on 2 big tasks both heavy on the front-end side and it is not so difficult to make it. The struggle that I am facing is adding the new title “INSTRUCTION” into the contact form’s back-end schema and how to deal with the input, process, and save it into the database. But I manage to finish it with the help of my lead developer and my colleague.

3.5 Week 5 Sketching and coding session tab in mobile view

3.5.1 Design and coding for session tab

This week's 5 task is a design and making a user-interface for the session tab in the mobile view. It takes me 2 days to sketch the design on Figma and have a conversation with the lead developer of the team to discuss the content layout of the tabs and color theme. The old content layout of session tabs used to be horizontal and exceed the width of the mobile view. So, I change the order into a vertical layout so we can see it better.

The color theme for the website has always been light green. I also adding even-odd styling background for each chat session so we can identify each session easily with the background color interleaving color white and grey. The code below is styling interleaving with the background color on odd session chat and white background color on even session chat. (FIGURE 22, FIGURE 23)

```
+ const TableWrapper = styled.div`
+   :nth-child(2n + 1) {
+     background: #f0f0f0;
+   }
+ `
```

FIGURE 22. Styling interleaving for session chats

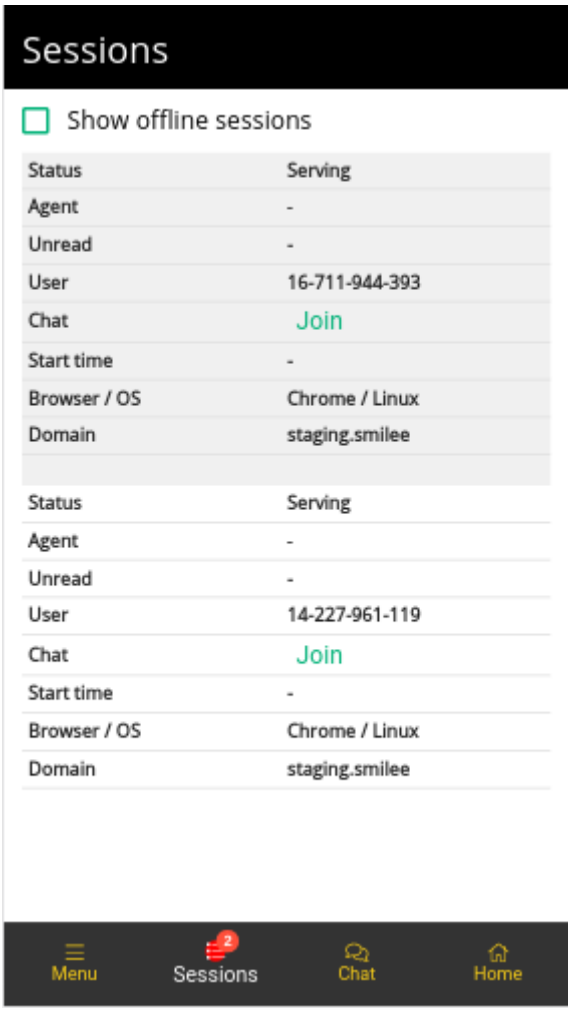

FIGURE 23. Styling background interleaving

Using the map method for sessions object I manage to create a new list of the session that can be fetched each into the table and is identified by the id. (FIGURE 24) When successfully fetch to the table, each property of object sessions will be displayed in each individual cell of the table vertically<HeadCell> and the variable of the properties will be displayed to the right of it also vertically<BodyCell>. (FIGURE 25)

```
+         {sessions.length > 0 && (
+           <ContentWrapper>
+             {sessions.map(session => {
+               const unreadMessageCount =
+                 (isOwnSession(session, self) && chat?.unreadMessages?.length) ||
+                 0
+               const status = getStatus(session, locales)
+               const userAgent = parseUserAgent(session.browserUa)
+               const agentName = getAgentName(session, self)
+               const startTime = session.roomStartTime
+                 ? moment(session.roomStartTime).format('HH:mm:ss DD/MM/YYYY')
+                 : '-'
+
+               const canJoinChat =
+                 isUpcomingSession(session) || isMySession(session, self)
+               const chat = ongoingChats[sessions._id]
+               return (
+                 <TableWrapper key={session._id}>
+                   <Table
+                     chat={ongoingChats[sessions._id]}
+                     key={session._id}
+                     data-testid="sessions-table"
+                     padding="checkbox"
+                     className={classes.table}
+                   >
```

FIGURE 24. Fetching data of session list to the table

```
<TableBody data-testid="session-table-row">
  <>
    <TableRow>
      <HeadCell>
        <CellText>{locales.session_list_status}</CellText>
      </HeadCell>
      <BodyCell>
        <CellText>{status}</CellText>
      </BodyCell>
    </TableRow>
```

FIGURE 25. Fetching properties and its variables

And here is the final result: (FIGURE 26)

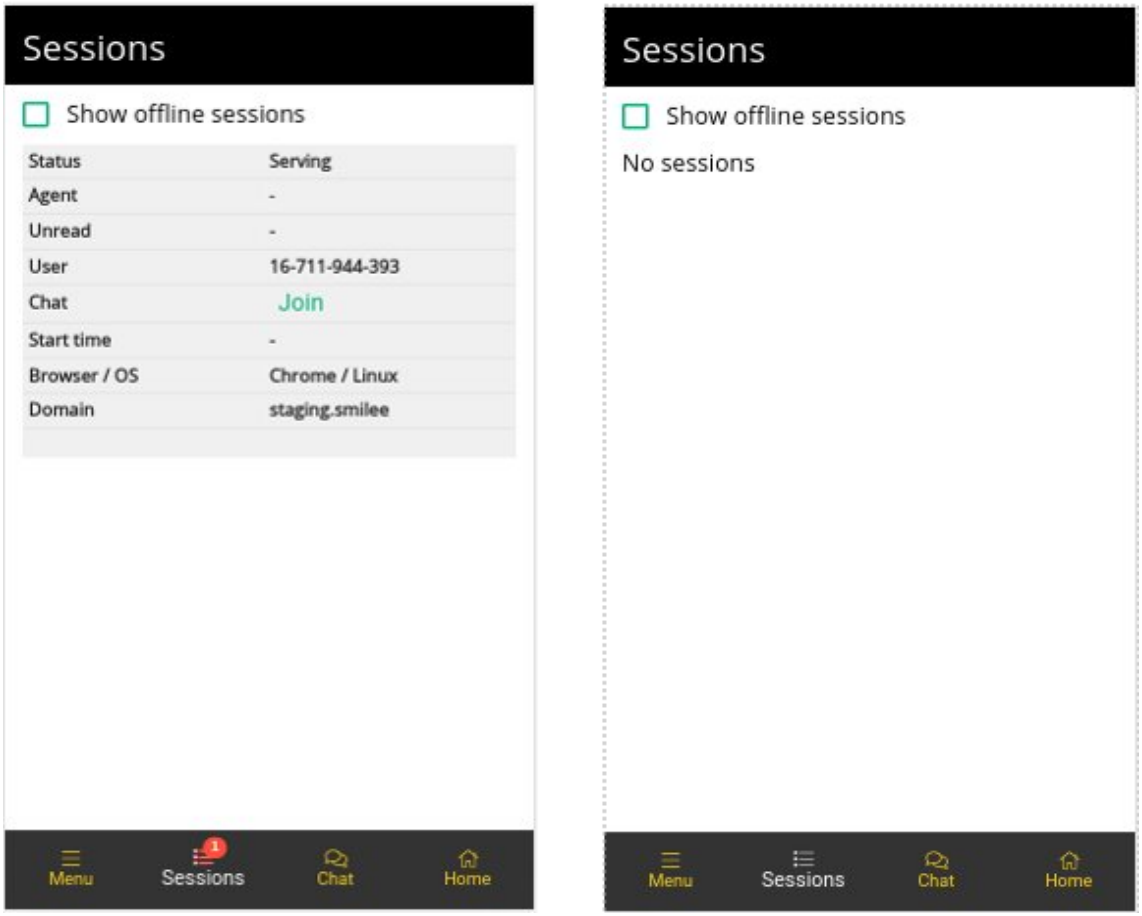

FIGURE 26. New user interface for session tab

3.5.2 Week Analyzing

This week's task is not difficult in terms of a new thing but very precise in terms of design and coding front-end features. It takes a lot of time to code the user interface from the design into real life because you must be pixel correction. But overall, it is not so difficult technically and I finally finished it on time as expected.

3.6 Week 6 Adding email validation for Chat Bot and fix some minor bugs

3.6.1 Adding email validation for Chat Bot

This week's task is adding validation for input that user input into Chat Bot. The email that login must be in the right form of an email. For this task, I have studied regular expression or RegEx to learn about the form pattern for email.

A regular expression (shortened as regex or regexp also referred to as rational expression) is a sequence of characters that define a search pattern. Usually, such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. It is a technique developed in theoretical computer science and formal language theory. Regular expressions are used in search engines, search and replace dialogs of word processors and text editors, in text processing utilities such as sed and AWK, and in lexical analysis. Many programming languages provide regex capabilities either built-in or via libraries. (. 5)

The regular expression for email in JavaScript is " /^\w+([\.-]?\w+)@\w+([\.-]?\w+)(\.\w{2,3})+$/ " which contains all the rules to form a valid email and those are:

- mysite.ourearth.com [@ is not present]
- mysite@.com.my [Top-Level domain cannot start with dot "."]
- @you.me.net [No character before @]
- mysite123@gmail.b [".b" is not a valid Top-Level domain]
- mysite@.org.org [Top-Level domain cannot start with dot "."]
- .mysite@mysite.org [an email should not be start with "."]
- mysite()*@gmail.com [here the regular expression only allows character, digit, underscore, and dash]
- mysite..1234@yahoo.com [double dots are not allowed](. 6)

After getting the regular expression for email, I take the validation and trying to apply it to the code base for our input. (FIGURE 27)

```
+         } else if (
+           variant === 'EMAIL' &&
+           /^\w+([\.-]?\w+)@\w+([\.-]?\w+)(\.\w{2,3})+$/.test(node.send) ===
+             false
+         ) {
+           this.onValidationError(node, 'Invalid email address')
+           return false
          }
          break
        case 'button':
@@ -483,6 +490,15 @@ export class AssistantEditor extends React.Component<Props, State> {
            return false
          }
          break
+       case 'send':
+         if (
+           /^\w+([\.-]?\w+)@\w+([\.-]?\w+)(\.\w{2,3})+$/.test(node.send) ===
+           false
+         ) {
+           this.onValidationError(node, 'Invalid email address')
+           return false
+         }
+         break
```

FIGURE 27. Apply validation for an email in the codebase

On the code base above, I apply the regular expression of email to the codebase. With the if statement to start the validate if the input (node.send) will be tested through the regex pattern and the result will return a Boolean true or false. If the input did not qualify then the Boolean will be true and the code in if code block will be executed, an error notification will show up and the result for saving data will be false then we cannot save the data that have error email form.

And the website will be shown as below for the result when you enter any kind of the wrong email follow the rules above of regular expression for email. (FIGURE 28)

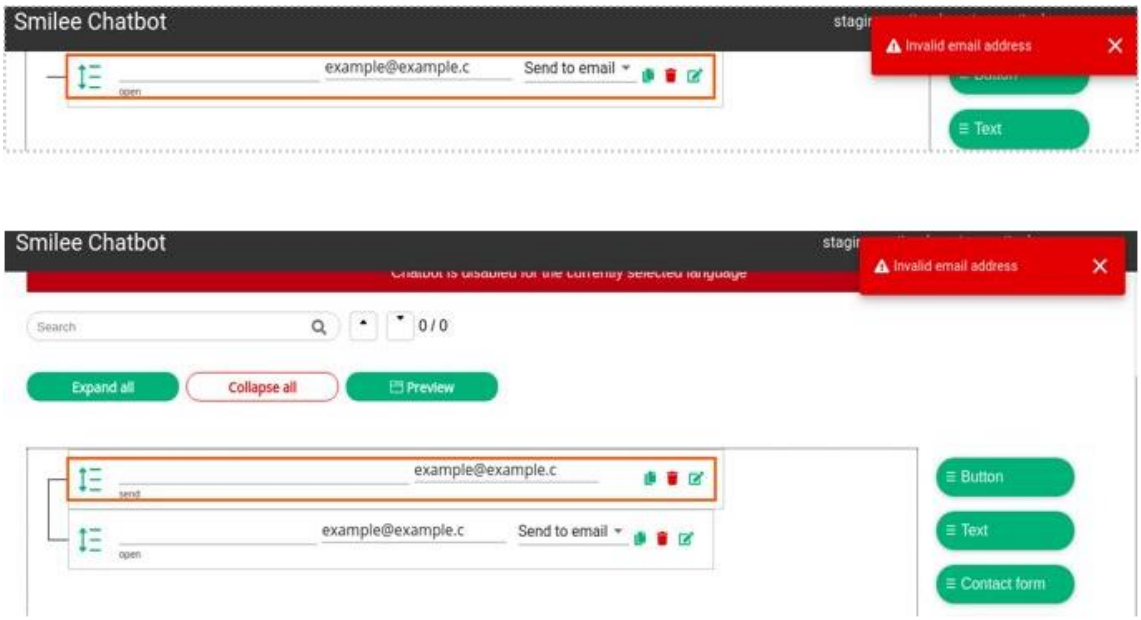

FIGURE 28. Error notification on save

3.6.2 Fix minor bugs

The next task for me is to fix the color bugs of the team selector every time we choose a new team or refresh. The color of the text will be the same as the color of the background so it is hard to identify so I take a look through the team selector component to search for the styling of the team selector so I can fix it. To fix the problem I put the input value will be displayed in placeholder position and then re-style the placeholder text when being focused. (FIGURE 29) And since the value we input into the text field has been transferred into the placeholder, the old text of the team selector component will be removed and I made some CSS re-style to make it better looking. (FIGURE 30)

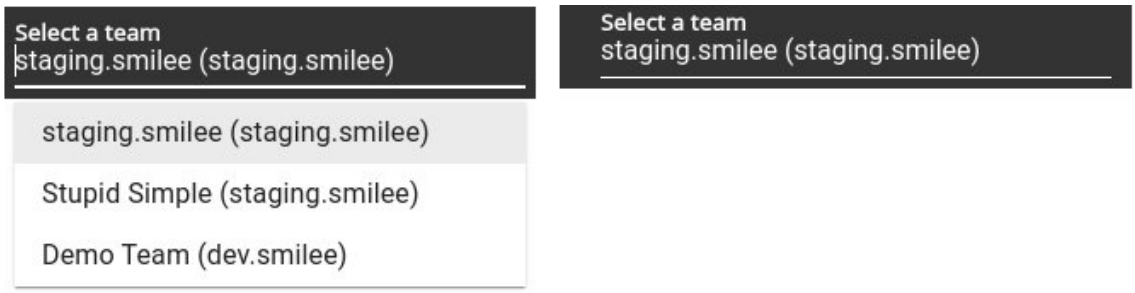

FIGURE 29. Team selector color on focused and after choosing

```
+         MuiFormLabel: {
+           root: {
+             '&$focused': {
+               // change label color on focus
+               color: 'white'
+             }
+           }
           }
         }
       })

@@ -232,6 +240,7 @@ const TeamSelector = ({
               }

               const { onBlur, onFocus, onClick, ...inputProps } = getInputProps({
+                value: placeholderText,
                 placeholder: placeholderText,
                 'data-testid': 'team-selector-input',
                 onFocus: openMenuAndClearSelection,
```

FIGURE 30. Styling code for placeholder text

3.6.3 Week Analyzing

For this week I was very happy to complete all the task and learning new thing for regular expression so I could use for later. This week feels like a normal week with an easy task and things go on well without any pressure.

3.7 Week 7 Adding date validation for domains mobile, making onClick render component for Contact Form

3.7.1 Adding date validation for domains mobile

This week's task was another validation task for domains in the dashboard but this time is for mobile since I only made a feature that is similar for desktop. The task is very easy and straight forward, but it takes a lot of steps and effort to make it done. The first thing will always be a meeting and discussion with the group about the user interface of the feature, when it comes to the final idea then I would process to code it.

```
+                let isExpired
+                let d1 = new Date()
+                const team = teams.find(team => team._id === teamId)
+                if (d1.getTime() < Date.parse(team.domain.validUntil)) {
+                  isExpired = true
+                } else {
+                  isExpired = false
+                }
```

FIGURE 31. The function that sort expired domains

The function above is to sort the expired domains and pass the Boolean isExpired into the domain row that is expired so I can use it to style it and make it different from the non-expired domains. (FIGURE 31) Next is adding an alert icon for expired domains since it is mobile so it does not have a hover effect like the desktop to identify because I use hover effect to display expired date for expired domains in desktop. (FIGURE 32)

```
+                        {isExpired ? (
+                          <ExpiredAlert>
+                            <StyleButton>
+                              <FontAwesomeIcon
+                                size="2x"
+                                icon="exclamation-circle"
+                                color="#D82B03"
+                              />
+                            </StyleButton>
+                            <ExpiredContent>
+                              <p>
+                                {locales.domain_expired_on}{' '}
+                                {this.getDomainInfo(teamId, 'date')}
+                              </p>
+                            </ExpiredContent>
+                          </ExpiredAlert>
```

FIGURE 32. Expired alert notification

The expired domains will have the ExpiredAlert component to add on to the user on mobile can notice. The last thing to do is styling all the user interface that I just made according to plan and design.

Here is the result: (FIGURE 33)

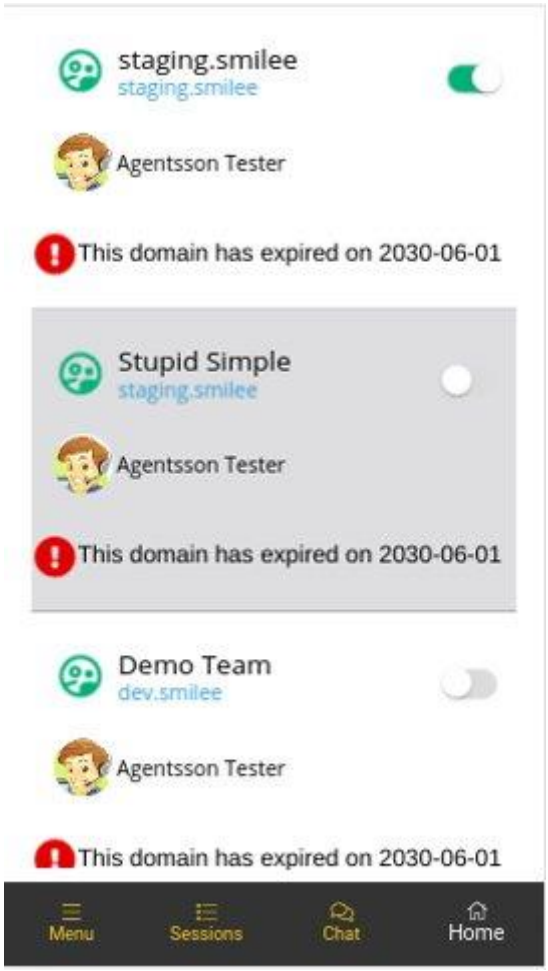

FIGURE 33. Expired alert result

3.7.2 Making onClick render component for Contact Form

The next task was hard for me in the first place and took quite a lot of time to understand and make it. The original form of Contact Form was auto rendering all 4 categories: email, phone, text, and question when we first create a new form. So, the lead developer wants me to create a blank contact form so the user can freely add categories if they want and in their order of choice. After 2 days of learning the code and meeting, asking for advice I was able to start doing it and have a plan. I make a button that contains all 4 categories and render them onClick if the user wants to add multiple text and question, email, and phone category are limited at one. So, I remove all the old codes of the rendering component and write a new one.

```
+                <MenuItem
+                  onClick={() =>
+                    updateFieldList({
+                      action: 'add',
+                      label: '',
+                      required: false,
+                      type: 'EMAIL',
+                      values: [],
+                      _id: generateTemporaryId()
+                    })
+                  }
+                >
+                  Email
+                </MenuItem>
```

FIGURE 34. Adding component Email onClick

The contact form fetches data from the back end so whenever we update the back-end data, the category will appear, the picture above is an example of an Email component and click to render out the component. (FIGURE 34) Below is the example component Email Panel: (FIGURE 35)

```
export const EmailPanel = ({ fieldList, updateFieldList, email, setEmail }) => {
  const classes = useStyles()

  const [emailInput, changeEmailInput] = React.useState(
    fieldList.find(findEmailField) ? fieldList.find(findEmailField).label : ''
  )
  const { locales } = useLocales()
  const notification = useNotification()

  function findEmailField(entry) {
    return entry.type === 'EMAIL'
  }
```

FIGURE 35. Email component

Like the Email component, every category of the contact form will be re-coded in a new file as an export component so it can be called out when clicking the button. The button adds category will add a blank category to the contact form so we can modify and save it to the database. All the data will be saved into object fieldList then will be saved into the database.

Here is the result for click render component:

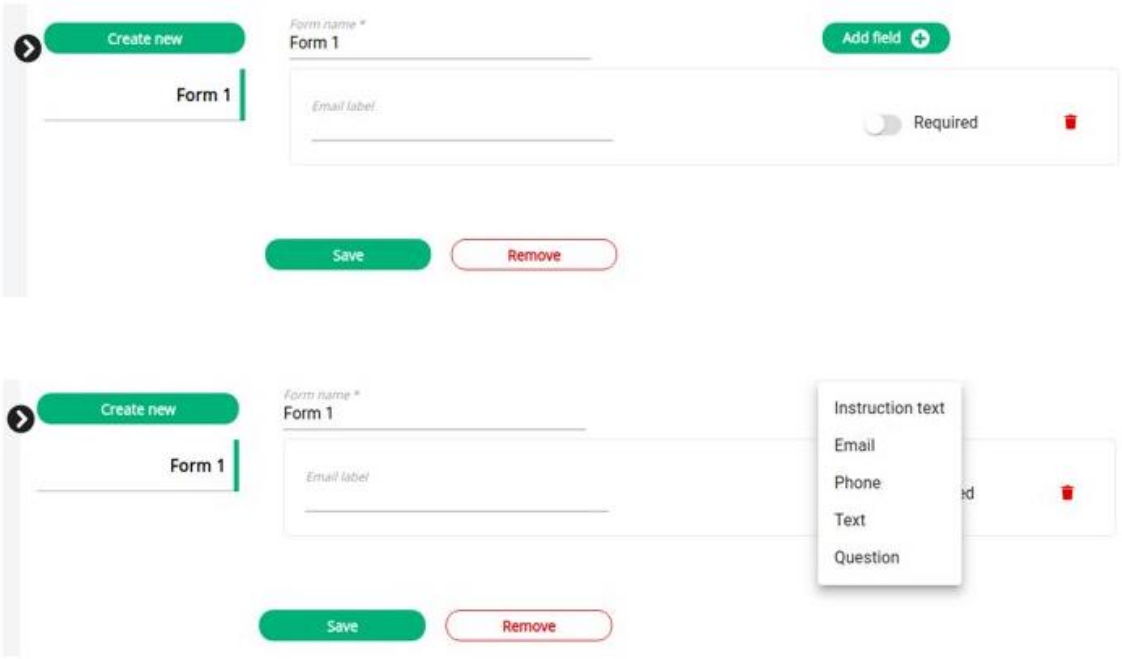

FIGURE 36. Email component render onClick

The contact form will be blank when we first created and then the Add field button will be used to add a category to the form and you can customize the order and number of categories, it is not strictly formed like it is used to be anymore. (FIGURE 36)

3.7.3 Week Analyzing

For this week, the tasks were overload for both the amount of code and technical knowledge. Lots of hard work and research to learn how to code some parts. But I finally manage to make it done, 4 days of research, and coding new features was finally delivered to the client. Tough week for me so far and I learn a lot during this week of working.

3.8 Week 8 Fix tutorial links and adding Microsoft Edge to allowed browser, adding validations for the contact form

3.8.1 Fix tutorial links and adding Microsoft Edge to allowed browser

This week I was doing a bug fixing task and the goal was to fix the old tutorial links that have been outdated and created error for the user. These tutorial links were made for users if they did not

know any feature, it can lead the user to the page and performed an action for a tutorial. For example, if I wanted to change the password and did not know how to do it, I just needed to click on the tutorial icons for help and click on the change password. The website will automatically lead us to the site where we can change the password and show a notification to help people learn how to do it. (FIGURE 37)

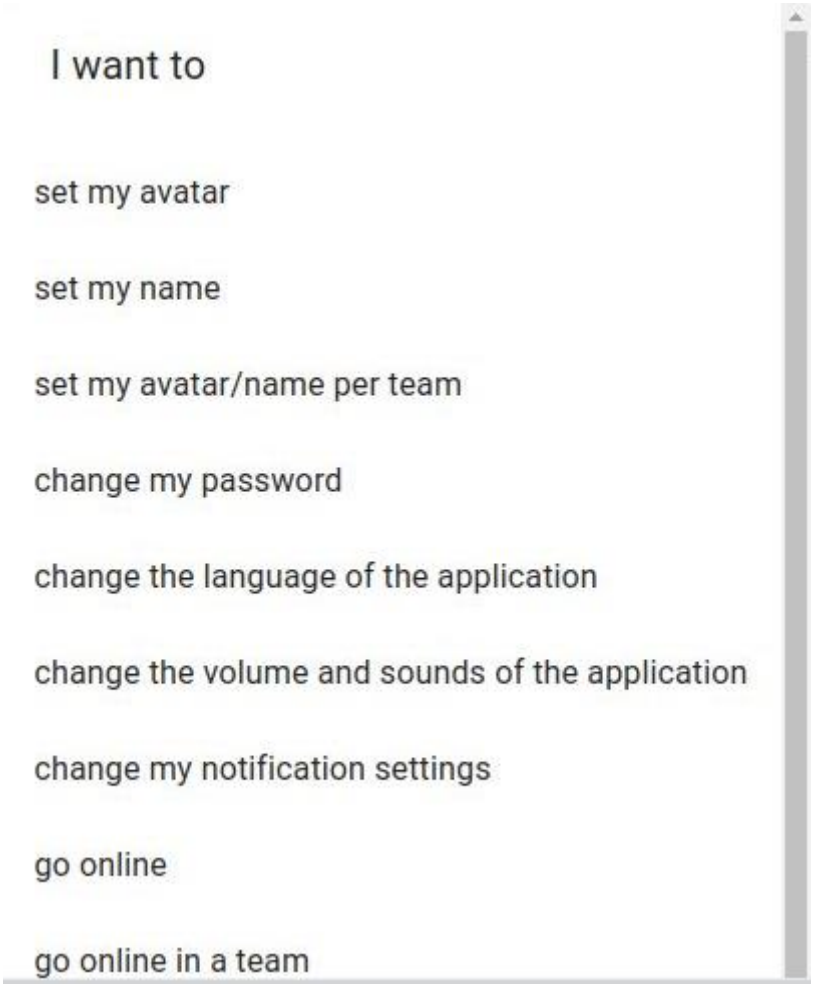

FIGURE 37. Tutorial links user interface

So, these tutorial links are currently bugged out for the change in our system component and path so I took a day to learn about the tutorial links component and then fixed it. (FIGURE 38)

```
-         if (document.location.pathname !== `/`) {
-           await store.dispatch(push(`/`))
+         if (document.location.pathname !== `/settings`) {
+           await store.dispatch(push(`/settings`))
```

FIGURE 38. Fixing the navigation path

First thing is to fix the path that leads to the page of the website since the structure of the web has been changed so the path of react-router is not the same anymore. (FIGURE 39)

```
-     target: '[data-testid="agentname-text"]',
+     target: '[data-testid="default-name"]',
      placement: 'bottom',
      event: 'hover'
    }
@@ -48,7 +48,7 @@ export const setPrimaryAvatar = locales => {
      _id: 4,
      title: `${locales.tutorial_primary_avatar_title}`,
      content: `${locales.tutorial_primary_avatar_content}`,
-     target: '#primary-avatar',
+     target: '[data-testid="default-avatar"]',
```

FIGURE 39. Target fix for tutorial

Next is to fix the target of the tutorial since it was changed in some folder components so the feature might not be the same and also the target id so I have to track to the correct component and find the correct id for the target so help notification will display out to guide the user. For example change password dialog guide: (FIGURE 40)

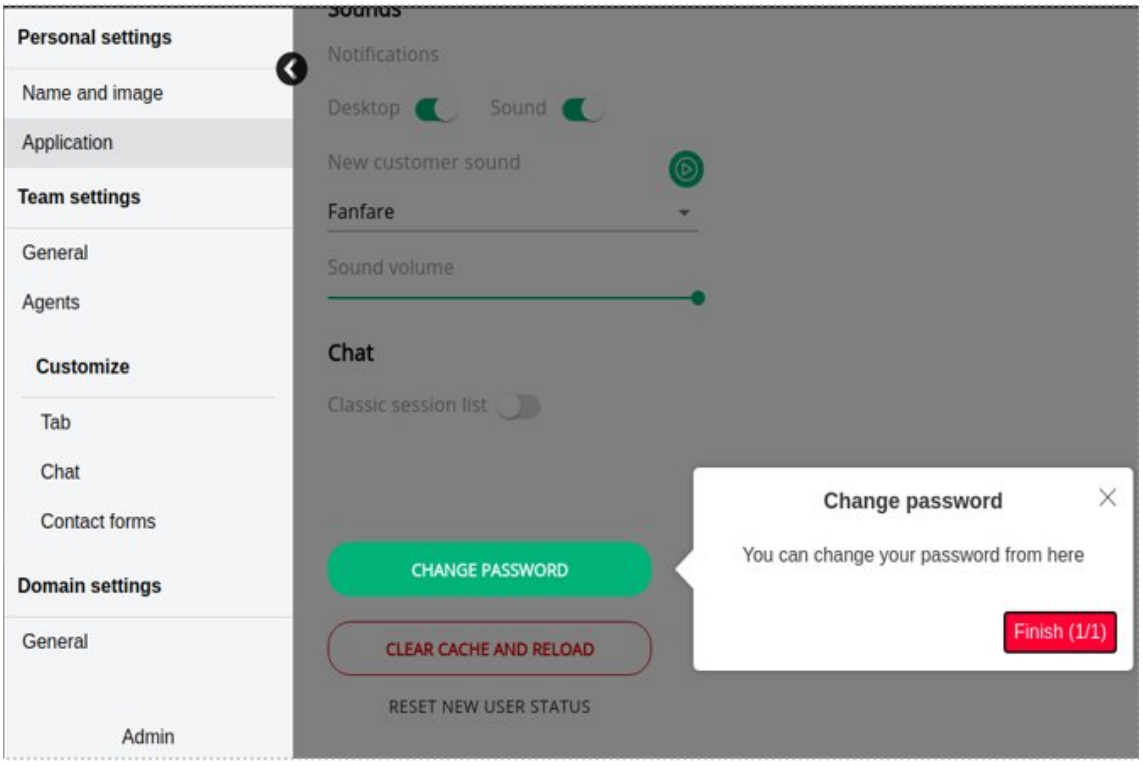

FIGURE 40. Change password dialog guide

The next task is a small task to add Microsoft Edge to allow browser list since the popularity of its gain recently, so our company decided to support Microsoft Edge to use our agent website and adding design, features for Microsoft Edge. Microsoft Edge is a web browser developed by Microsoft. It was first released for Windows 10 and Xbox One in 2015, then for Android and iOS in 2017,[9][10] and for macOS in 2019. Edge includes integration with Cortana and has extensions hosted on the Microsoft Store. Unlike Internet Explorer, Edge does not support the legacy ActiveX and BHO technologies. Originally built with Microsoft's own proprietary browser engine Edge HTML and their Chakra JavaScript engine, Edge was rebuilt as a Chromium-based browser in 2019,[12][13] using the Blink and V8 engines. (. 7)

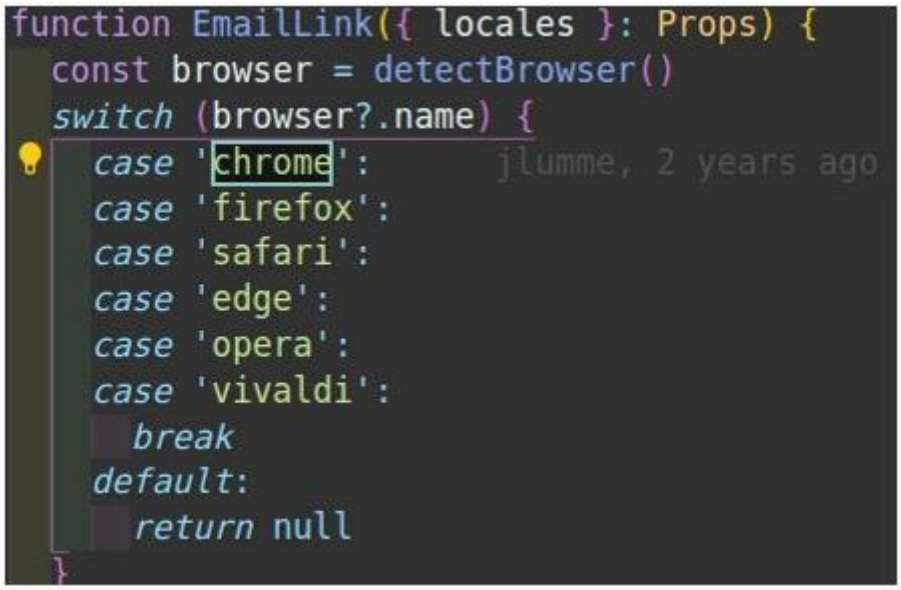

FIGURE 41. Adding Edge for allow browser list

After adding Edge for browser list, (FIGURE 41) I code the front end for the Alert Browser component when the browser is not in the allowed list. (FIGURE 42)

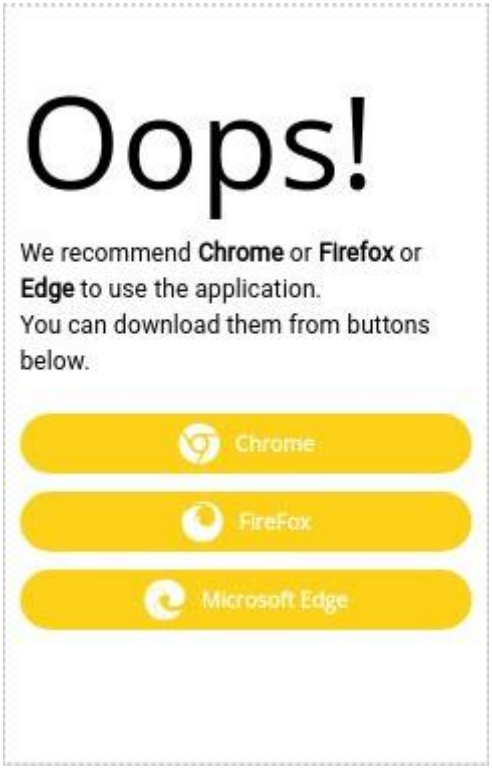

FIGURE 42. Adding Edge for recommend list

3.8.2 Adding validation for the input contact form

After click renders component task complete for the contact form, I started to make input validation for the contact form since it needs some validation for the input to the database back to end otherwise the form will start to crash. So, I add validation in two cases for the contact form which are form cannot be empty- zero categories and question category list cannot be empty. The codebase is too long to capture with the screen so here is the small part of it validating the question category for an empty list. (FIGURE 43)

```
let fieldQuestionArr = fieldList.filter(
  field => field.type === 'SELECT' || field.type === 'SELECT_MULTIPLE'
)
fieldQuestionArr.forEach(field => {
  if (field.values.length !== 0) {
    onSave(contactForm._id, newName, formattedFields)
  } else {
    notification.show({
      message: 'Options list can not be empty',
      error: true
    })
  }
})
```

FIGURE 43. Validating question category for an empty list

And the final result for the task is validating if the form is empty or the question category is empty. (FIGURE 44)

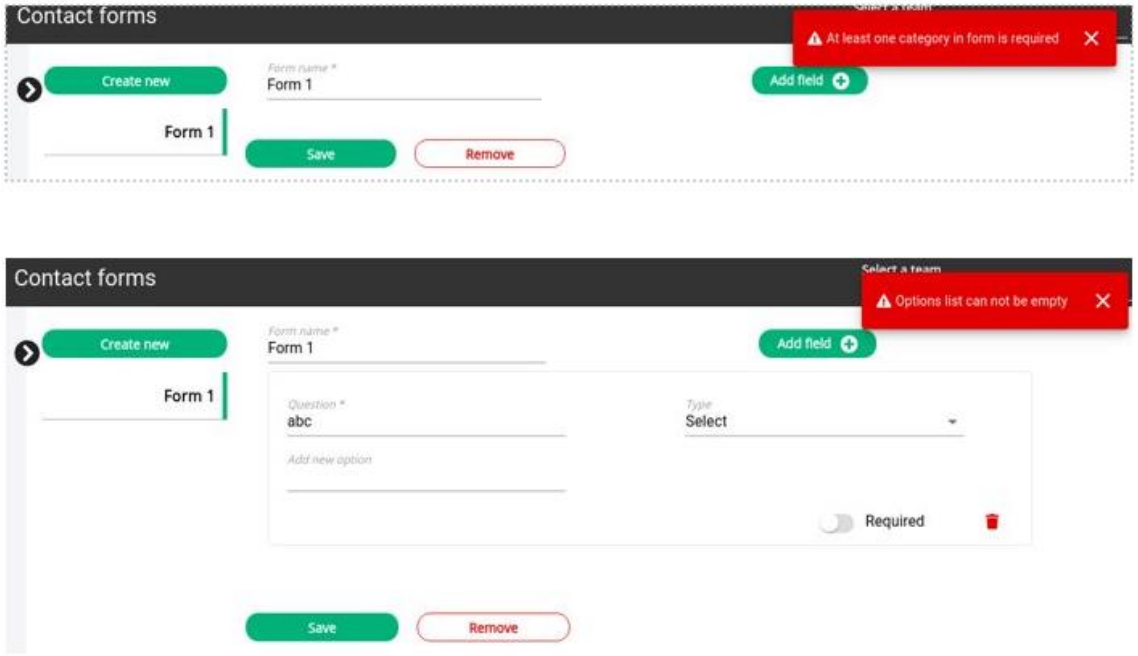

FIGURE 44. Result for validating contact form

3.8.3 Week Analyzing

For this week I was doing it easily and having a meeting with the team to handle the feature on time for our customer. This week task is tricky with the logic when I created the main function for validating contact form since the nested object for the database is very big and difficult to solve it,

I have to test many times to write down the perfect function to cover all of the validation cases in order not to create bugs. Overall, this week is a success for me and the team to archive the goal on time.

3.9 Week 9 Replacing all save icon with button, admin manages team table UI remake and fix bugs scroll. Email validation for the login page

3.9.1 Replacing all save icon with a button

For this week's task, I must replace all save icon buttons in the general setting tab to save button text since the feedback on the save icon button is not positive. Some users and agents did not recognize the save icon button to save data so I have to re-make it into a button with text so it can be visually understandable. Our web system uses a font-awesome package to use for icons and now I have to do all over again to make a new button for the website. In order to re-use the component later, I made a customize button component and can be imported to use everywhere in the code folder. (FIGURE 45)

```
import styled from 'styled-components'

const CustomizeButton = styled.button`
  outline: none;
  border: ${props => (props.primary ? 'none' : '1px solid #d82b03')};
  cursor: pointer;
  font-family: Open Sans, sans-serif;
  font-size: ${props => (props.fontSize ? props.fontSize : '14px')};
  font-weight: bold;
  min-width: 165px;
  height: ${props => (props.height ? props.height : '30px')};
  width: ${props => props.width};
  background-color: ${props => (props.primary ? '#02a676' : '#fff')};
  border-radius: ${props => (props.borderRadius ? props.borderRadius : '14px')};
  color: ${props => (props.primary ? '#fff' : '#d82b03')};
  margin-top: 12px;
  padding: 5px 20px;

  &:hover {
    box-shadow: 0 1px 6px 0 rgba(32, 33, 36, 0.28);
  }
  ${props => props.disabled && 'opacity: 0.2;'}
`

export default CustomizeButton
```

FIGURE 45. Customize Button component

The Customize Button component I made can justify all of the properties we need to customize like height, width, color, and background color. This component can be re-use anywhere we need like a standard form of a button for our website.

Back in the replacing button task, after finish writing the new button component, I started to get back and replace all the icon save button into the customize button I just made. Here is a part of the general settings tab where I replace icon buttons to my button component: (FIGURE 46)

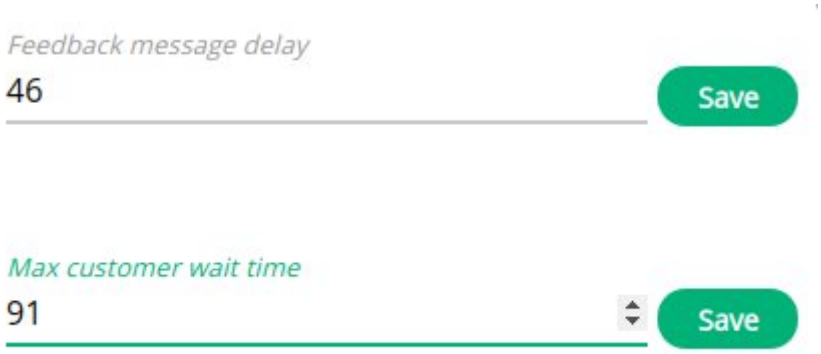

FIGURE 46. Button component that replaced

3.9.2 Admin manage team table remake UI and fix bugs scroll for table

The next task is another user interface upgrade for Admin manage team table and it is also has a bug that is unable to scroll the table when the list is too long.

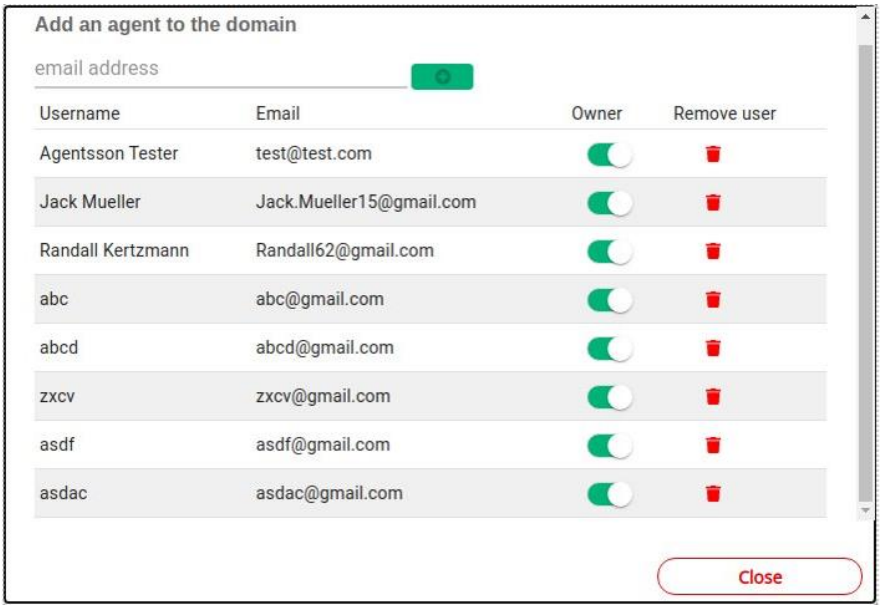

Figure 47. The final result of the Admin manage team table

After a few changes in CSS styles and JavaScript, it is now fixed and has a new user interface with the useable button component I just made last week. This task was easy overall and only took one day to finish. (FIGURE 47)

3.9.3 Adding validation for an email in the Login page

Next to the email validation for the chatbot, this time is validation for the login page.

```
validateEmail(email) {
  const regex = /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/
  return regex.test(email)
}
```

FIGURE 48. Validate the email function

Validation for email is still the same regular expression form I used in the chatbot. (FIGURE 48) I attached the function with an event and in this case, it is an onChange event when we enter an email. The event checks the target value on every change of value in the text field and puts it through the validate function to check for the email form or when we press the login button. The event onClick also was attached with the validate function to evaluate the email input. (FIGURE 49)

```
<LoginActions>
  <CustomButton
    data-testid="loginButton"
    id="loginButton"
    type="submit"
    onClick={this.handleClickButton()}
    disabled={!this.validateEmail(email)}
  >
    {locales.login_button_signin}
  </CustomButton>
```

FIGURE 49. Validate onClick login button

```
<TextField
  data-testid="loginUsername"
  placeholder={
    locales.login_email_placeholder.charAt(0).toUpperCase() +
    locales.login_email_placeholder.slice(1)
  }
  error={this.validateEmail(email) === false}
  helperText={
    this.validateEmail(email) ? '' : locales.login_email_not_valid
  }
  type="email"
  required
  onChange={this.updateLoginProps('email')}
  onKeyUp={this.onKeyUp()}
  autoComplete={(keepSignedIn && 'on') || 'off'}
  value={email}
  disabled={disabled || isModal || inProgress || formDone}
  classes={{
    root: classes.rootInput
  }}
/>
```

FIGURE 50. Validate email on text input

For the validate email onChange text input, the event onChange was attached to the updateLoginProps() function where it processes the input string to validate through the email validation. (FIGURE 50) So it can indicate whether it is true or false so the notification error can pop up when it is false. Here is the result of the validation: (FIGURE 51)

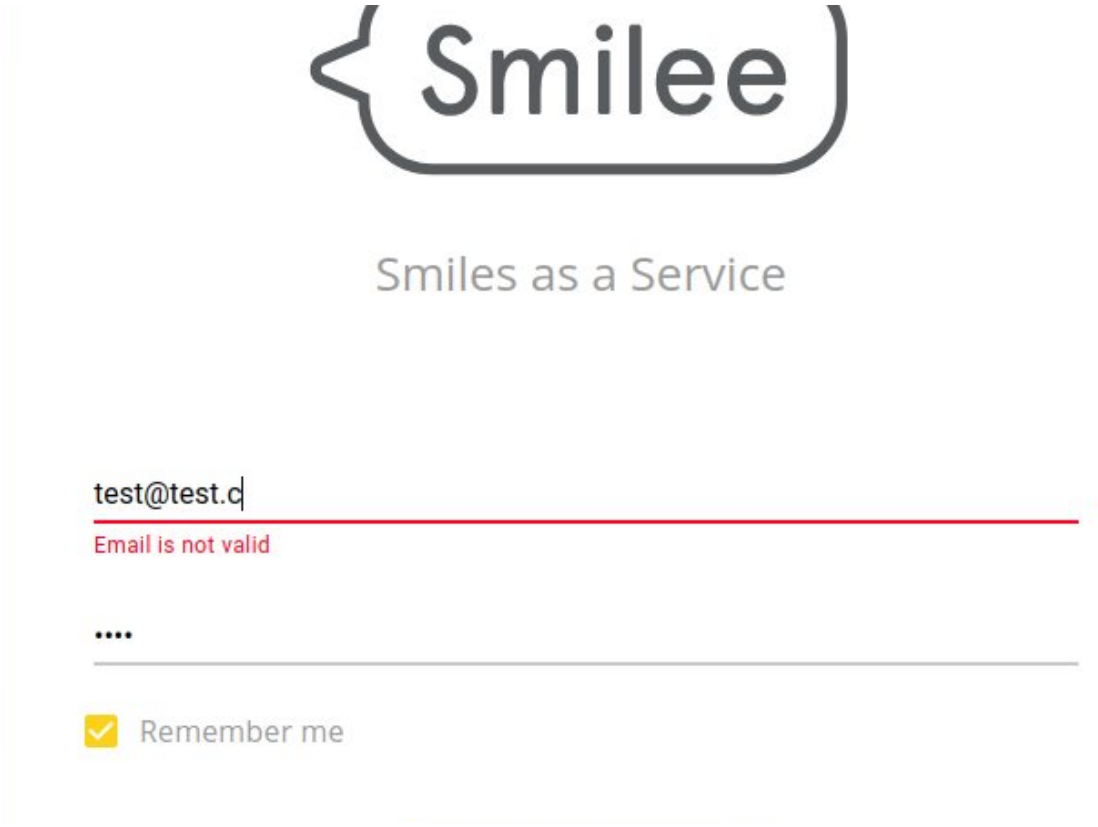

Figure 51. The final result of email validation

3.9.4 Week Analyzing

For this week, the task was pretty easy overall in the first two tasks, it just pure front-end styling for the remake user interface. The last task was a bit harder where I have to learn deep in the code line. Finding the existing code props and event for login and merging the validation code so it can work smoothly and cause no crash and bugs.

3.10 Remake UI for settings tab in mobile, Making draggable, font, chat switch for Chat Customization

3.10.1 Remake UI for setting tab in the mobile

After a successful upgrade in the user interface for the general settings tab in desktop, I was assigned another task about it but this time I got to do the mobile version of the general settings tab for an upgrade in the user interface this time. The works required quite a bit of work on the CSS and media query for an upgrade this time. Media queries are a CSS3 module allowing content

rendering to adapt to conditions such as screen resolution (e.g. smartphone screen vs. computer screen). It became a W3C recommended standard in June 2012 and is a cornerstone technology of responsive web design (RWD). A media query consists of a media type and one or more expressions, involving media features, which resolve to either true or false. The result of the query is true if the media type specified in the media query matches the type of device the document is being displayed on and all expressions in the media query are true. When a media query is true, the corresponding style sheet or style rules are applied, following the normal cascading rules. Media queries use the @media CSS "at-rule". (. 8) In this case, the Boolean mobile will indicate the React component that will be rendered out. (FIGURE 52)

```
+         {mobile ? (
+           <FontAwesomeIcon icon="plus-circle">
+             {state.isAdminOrDomainOwner && (
+               <DeleteTeam
+                 team={team}
+                 locales={locales}
+                 classes={classes}
+                 onDelete={deleteTeam}
+               />
+             )}
+           </FontAwesomeIcon>
+         ) : (
+           <DeleteTeamWrapper>
+             {state.isAdminOrDomainOwner && (
+               <DeleteTeam
+                 team={team}
+                 locales={locales}
+                 classes={classes}
+                 onDelete={deleteTeam}
+               />
+             )}
+           </DeleteTeamWrapper>
+         )}
```

FIGURE 52. Styling for mobile view

And the conditional style based on props mobile to style the component. (FIGURE 53)

```
  const ContentWrapper = styled.div`
-   width: 70%;
+   width: ${props => (props.mobile ? '100%' : '70%')};
  `

  const ButtonWrapper = styled.div`
@@ -116,7 +116,7 @@ const SaveButton = styled(CustomizeButton)`
  const SaveButtonTextField = styled(CustomizeButton)`
    min-width: 70px;
    margin-top: 35px;
-   margin-left: 125px;
+   margin-left: ${props => (props.mobile ? '0px' : '125px')};
    background: ${props => props.disabled && '#e0e0e0'};
  `
```

FIGURE 53. Some block of codes conditional styling for mobile

After a day of coding, the result is very nice and acceptable. (FIGURE 54)

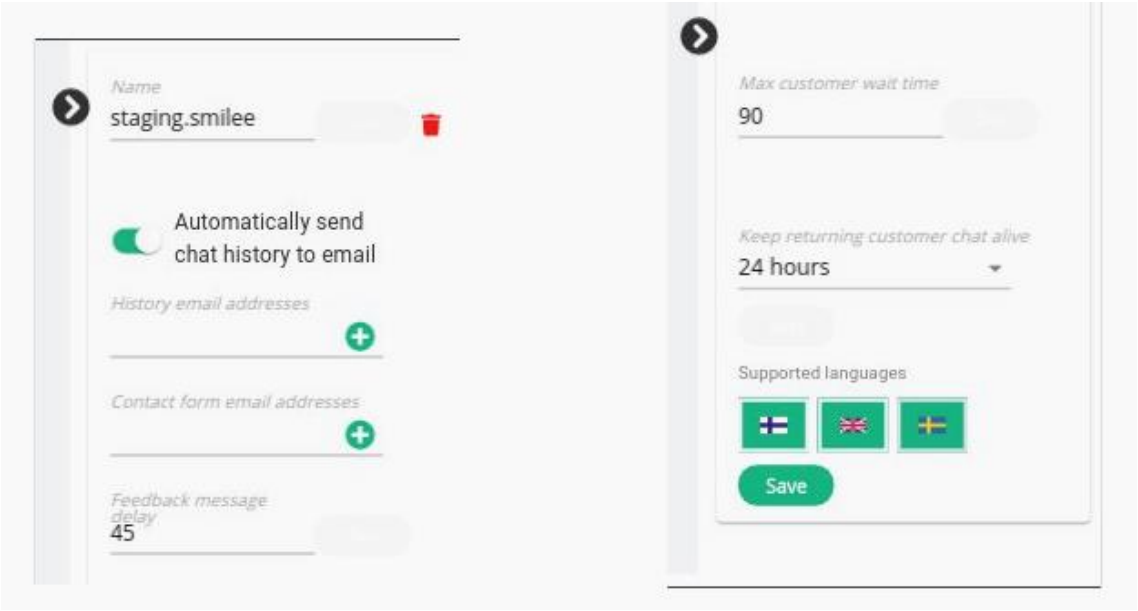

FIGURE 54. Result for the new general settings tab

3.10.2 Making draggable, font, chat switch for Chat Customization

The final task of this week is to make a new category of choices for users. In Chat Customization we can customize our settings for a chat. Now three new settings are planned to add into those settings, they are font, chat and draggable switch allow the user to change settings to the font, chat,

and draggable chat. Those three settings are the Boolean type to toggle features of chat. To make this task I started to learn more about the Chat Customization as well as the back end GraphQL of the Chat Customization tab. All the settings of Chat Customization were under the schema of ToggleTeamSetting in GraphQL. So, the first thing to do is adding new variables to mutation fontResizerEnabled and chatResizerEnabled. (FIGURE 55)

```
const TOGGLE_TEAM_LEAD_SETTING = gql`
  mutation toggleTeamLeadSetting(
    $teamId: ID!
    $field: String!
    $boolean: Boolean
    $string: String
    $number: Int
  ) {
    toggleTeamLeadSetting(
      teamId: $teamId
      field: $field
      boolean: $boolean
      string: $string
      number: $number
    ) {
      _id
      chatCustomization {
        chatResizerEnabled
        fontResizerEnabled
      }
    }
  }
```

FIGURE 55. Adding variables to the schema

```
const TOGGLE_TEAM_LEAD_SETTING = gql`
  mutation toggleTeamLeadSetting(
    $teamId: ID!
    $draggableFieldName: String!
    $fontResizerFieldName: String!
    $chatResizerFieldName: String!
    $draggable: Boolean
    $fontResizerEnabled: Boolean
    $chatResizerEnabled: Boolean
  ) {
```

FIGURE 56. Mutation toggleTeamLeadSetting

After adding variable into the mutation schema, I started to write the save function for mutation. This method is used to call out the mutation toggleTeamLeadSetting and notify us if the function is running or not by show error it had. (FIGURE 56)

```
  const [saveResizerForFontChatMutation] = useMutation(
    TOGGLE_TEAM_LEAD_SETTING,
    {
      onCompleted: () => {
        notification.show({
          message: 'Setting has been updated'
        })
      },
      onError: () => {
        notification.show({
          message: 'Failed update data',
          error: true
        })
      }
    }
  )
```

FIGURE 57. Save Resizer for font and chat function

Next, I use callback for three Boolean values font, chat, and draggable because it will return a memorized version of the callback that only changes if one of the dependencies has changed. (FIGURE 57) This will be useful later on to prevent unnecessary renders because these 3 Booleans will have their variable interact or being changed by the child component's user interface. So, I passed 3 Booleans variables down to the child component and use callback is useful when passing callbacks to optimized child components that rely on reference equality. (FIGURE 58)

```
const saveFontChatResizer = useCallback(
  (choices: {
    fontResizerEnabled: boolean,
    chatResizerEnabled: boolean,
    draggable: boolean
  }) => {
    saveResizerForFontChatMutation({
      variables: {
        teamId: team._id,
        fontResizerFieldName: 'fontResizerEnabled',
        fontResizerEnabled: choices.fontResizerEnabled,
        chatResizerFieldName: 'chatResizerEnabled',
        chatResizerEnabled: choices.chatResizerEnabled,
        draggableFieldName: 'draggable',
        draggable: choices.draggable
      }
    })
  },
  [saveResizerForFontChatMutation, team._id]
)
```

FIGURE 58. use callback for Boolean variable

```
<ChatMessageAppearanceOptions
  options={team.chatLog}
  choices={team.chatCustomization}
  // TODO: preview for chat log settings
  // onUpdate={updateChatLookFrame}
  onUpdate={() => {
    return true
  }}
  onSave={(...args) => {
    saveChatMessageAppearance(...args)
    saveFontChatResizer(...args)
  }}
/>
```

FIGURE 59. Pass variables to child component

Next, I passed those variables to the child component where they are contained in the choices variable and the onSave function is trigger the saveFontChatResizer for Chat Customization. (FIGURE 59) So whenever you save the child component, these variables will be updated and saved into the database of toggleTeamLeadSetting schema.

The final step is making the switch user interface and linking the variable to the switch UI. (FIGURE 60)

```
<SwitchWithTitle
  title="Font"
  tooltip="Toggle font resizer"
  label={state.fontResizerEnabled ? locales.enabled : locales.disabled}
  checked={state.fontResizerEnabled}
  onChange={toggleFontResizer}
/>
```

FIGURE 60. Link the switch UI with a Boolean variable

And the result in website for three new switches: (FIGURE 61)

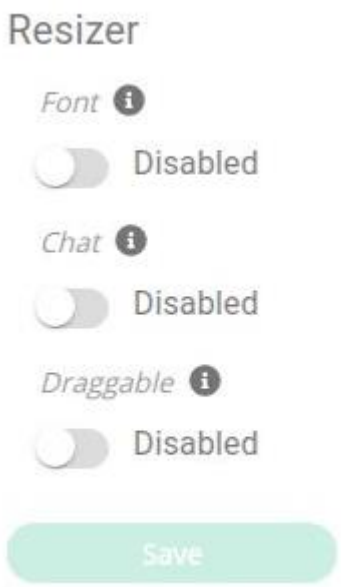

FIGURE 61. Chat Customization switch's user interface

3.10.3 Week Analyzing

For this last week, I have faced the most difficult task for me so far and it is the switch task for Chat Customization. The amount of time I spent on learning the code base is so much and asking for advice from the lead developer and colleague to help me finish this task. For this task I was struggling at first and slowly get back on track to deal with it, it took me more than 6 days till the next week to finish it but luckily it caused no crash or bug so I was very happy about the result.

4 ANALYZING AND CONCLUSION

4.1 Analyzing

During the time of writing this thesis, I have seen the progress of myself and learn so many new things that help me in the future, strengthen the knowledge foundation for my future career. Many useful things are exploited by me or guided by my lead developer. I have learned some new things like redux, props styling, a new cool framework for react development, GraphQL, and many small things more. Learn to approach the issues with a new and efficient mindset, learn to debug efficiently. All are picked up during my work time.

I also realized the importance of communication and teamwork, my works can be great and fast like this thanks to my colleague and meeting where I can discuss the problem of myself and solving the problem together or given many useful tips to use.

4.2 Conclusion

The practical training time is a good learning opportunity for me. Writing this thesis and witness the result of my work, I can see my progress during 10 weeks of working. From school and books to training, it is so much different from what I expected and when facing real trouble. I have known my limitation in knowledge and known I have to work and study harder to complete the work successfully. In real life, trouble is not unexpected and it can be new, I learned to approach and solve the problem not only in technical skills but also in ways of dealing with it and the mindset when solving it. This thesis helps me to see that to succeed I have to learn a lot more in various fields of skills and never rest to become a better developer. I have benefitted a lot from writing down my diary thesis report and it has helped me to progress. After this thesis project, I knew what I capable of and what things I need to study more to improve. Knowing my path from now on and dedicate my time and passion to learn, develop myself, and archive my dream.

REFERENCES

1. Relationship of grid layout to other layout methods. MDN web docs. Cited 24.6.2020 https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Relationship_of_Grid_Layout#:~:text=The%20basic%20difference%20between%20CSS,columns%20at%20the%20same%20time.
2. Does CSS Grid Replace Flexbox. CSS-Tricks. Cited 24.6.2020 https://css-tricks.com/css-grid-replace-flexbox/
3. styled-component documentations. Cited 24.6.2020 https://styled-components.com/docs/basics
4. Meet Material-UI — your new favorite user interface library. Cited 24.6.2020 https://www.freecodecamp.org/news/meet-your-material-ui-your-new-favorite-user-interface-library-6349a1c88a8c/
5. Regular Expression. Cited 25.6.2020 https://en.wikipedia.org/wiki/Regular_expression
6. JavaScript: HTML Form - email validation. Cited 25.6.2020 https://www.w3resource.com/javascript/form/email-validation.php
7. Microsoft Edge. Cited 25.6.2020 https://en.wikipedia.org/wiki/Microsoft_Edge#EdgeHTML
8. Media Queries. Cited 25.6.20 https://en.wikipedia.org/wiki/Media_queries