POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Computer Engineering

Master Degree Thesis

# A Web-Based Mobile Platform for Continuous Monitoring and Assessment of Classroom Comfort

**Supervisors**
Prof. Antonio Servetti

**Candidate**
Longsheng Zhao

Academic Year 2024-2025

# Acknowledgements

The candidate would like to express his sincerest gratitude to:

His family, for keeping patience through all the occurrences during the procedure.

His supervisor, for being the assisting force when facing obstacles.

And all those who have been around.

**Abstract**

In modern societies, Indoor Environmental Quality (IEQ) has been proved to be a significant index for classrooms and workplaces, as it indicates quantities that directly relate to comfort and efficiency. A user interface with higher readability and accessibility can play a significant role throughout the feedback loop for such a platform.

PROMET&O is a multi-sensor system that excels in monitoring IEQ data that are strongly connected to human comfort in a crowded open environment (e.g. classroom) and processing the feedback from users.

During this thesis, works have been put into the development of a user-friendly interface on mobile devices; through it, the system can include an effective and efficient procedure for users to view data streams transmitted from the system, provide feedback, and maintain a profile on mobile devices such as phones and tablets.

The thesis also explores the possibilities of introducing a more explicit network structure to the platform by utilizing Apache APISIX® for improved maintainability, with its features that provide easily configurable routes for API calls and a library of plugins; it can prove itself to be a viable upgrade for the system.

# Contents


**1 Introduction** 5

**I Reassuring responsiveness: a development of user interface for PROMET&O on mobile devices** **7**

**2 Background** 9
- 2.1 Motivations . . . 9
- 2.2 Visual hierarchy . . . 9
- 2.3 Useful features from CSS . . . 10
- 2.4 SASS: a CSS pre-processor . . . 10
- 2.5 Typical user interfaces on mobile devices . . . 11
- 2.6 Determination of the device type . . . 12
- 2.7 Utilizing dynamic importing in React.js . . . 13
- 2.8 Differentiating display modes . . . 13
- 2.9 Avoiding overflow . . . 14
- 2.10 "Immutable" content and maintainability . . . 15

**3 Homepage** 17
- 3.1 Welcome page for anonymous users . . . 17
- 3.2 Login and homepage . . . 19

**4 IEQ questionnaire** 21
- 4.1 Overall Comfort . . . 21
- 4.2 Classification of Issues . . . 25
- 4.3 Thermal comfort questionnaire . . . 26
- 4.4 Acoustic comfort questionnaire . . . 30
- 4.5 Visual comfort questionnaire . . . 33
- 4.6 Air quality questionnaire . . . 36
- 4.7 Final comments . . . 38
- 4.8 Notification on completion . . . 40

**5 Profile page** 41
- 5.1 Profile page . . . 41
  - 5.1.1 Showcase . . . 41

5.1.2 Style sheets . . . 42

**6 Personal questionnaire** 45
6.1 General introduction . . . 45
6.2 Questionnaire . . . 46
6.3 An inspection in detail into the style sheets . . . 49
6.3.1 Navigation . . . 49
6.3.2 Main container . . . 50
6.3.3 Question rows . . . 52
6.3.4 Buttons . . . 55

**7 Dashboard** 57
7.1 Overview . . . 57
7.1.1 Dividing the devices . . . 57
7.1.2 The display of data . . . 58
7.1.3 Navigation . . . 59
7.2 Gauge view . . . 59
7.2.1 Notification on scrolling . . . 64
7.3 Off-canvas sidebar . . . 66
7.4 Graph display . . . 69
7.5 Graph comparison . . . 70
7.5.1 The initial view . . . 70
7.5.2 The filter sidebar . . . 72
7.5.3 Graph comparison view . . . 75

**II Reorganizing the back-end: integrating Apache APISIX®** **77**

**8 Background** 79
8.1 A parallel in real life . . . 79
8.2 API gateway . . . 80
8.2.1 Apache APISIX® . . . 81
8.3 Current state of the system . . . 82

**9 The new architecture** 85
9.1 An overview on the new architecture . . . 85
9.2 Integrating the API gateway . . . 87
9.3 The authentication service . . . 89
9.4 The Grafana routes . . . 91
9.5 Review . . . 93

**10 Centralizing the authentication service** 95
10.1 Introducing realms in Keycloak . . . 95
10.2 The indented workflow . . . 96
10.3 The login flow . . . 97
10.3.1 GET /auth/login . . . 97

10.3.2 GET /auth/login/callback . . . . . 98
10.3.3 GET /userInfo . . . . . 101
10.4 The logout flow . . . . . 103

**11 Review on the new architecture** 107
11.1 The objectives achieved . . . . . 107
11.2 Limits and reasoning . . . . . 109
11.3 A perception to further development . . . . . 111

**12 Conclusion** 113

**13 Appendices** 115
13.1 List of the SASS items . . . . . 115

**Bibliography** 119

# Chapter 1

# Introduction

In modern societies, Indoor Environmental Quality (IEQ) has been shown to be a significant index for classrooms and workplaces, as it indicates quantities that directly relate to comfort and efficiency. A user interface with higher readability and accessibility can play a significant role throughout the feedback loop for such a platform, as it has the potential to help the user accurately express their feelings and needs.

PROMET&O is a Internet of Things (`IoT`) system with the purpose of assessing the quality of the indoor environment. It consists of both hardware and software parts; the former includes a series of sensors on perimeter that are able to collect environmental data such as room temperature, humidity, noise level, and others from their surroundings and periodically upload records to a database, and the latter part consists of a front-end web application in charge of user experience and back-end for data exchange. The front-end, acting as the portal for users through a web browser, will display relevant data on a Single-Page Web Application, and the user will be able to provide feedback messages back to the system.

Since its first edition, which is based on a serverless solution and relies on multiple external web services, PROMET&O has been benefiting from multiple updates that came afterward. Looking back at its path of evolution, it now has a solid construction including a relational database using MySQL, a middle-ware that serves as a reverse proxy utilizing Keycloak for user authentications, and a front-end user interface designed and implemented using `React.JS`. It has gained more independence in data storage, higher security, and user management functionality with less operating expense. All the previous works have laid a solid foundation for the effort that is being demonstrated in later chapters.

To further experiment on possible solutions, and take a step forward on the user-experience aspect of this project, the author is honored to take this opportunity and make several modifications on the software system, specifically web development part, to achieve a more user-friendly and more functional product in the end.

In the first part of the thesis "Reassuring responsiveness: a development of user interface for PROMET&O on mobile devices", the candidate will experiment on a various

of alternative solutions and different sets on configurations, in attempt to design and implement a brand new user interface for displaying and analyzing data, providing user feedback and maintaining user profiles.

In the second part of the thesis "Reorganizing the back-end: integrating Apache APISIX®", the candidate will experiment on improving the network architecture of PROMET&O by integrating Apache APISIX® as an API gateway, in pursuit for a more light-weight and flexible structure. Moreover, the Apache APISIX® has the possibility of introducing brand new features that will benefit the current system in numerous aspects, such as efficiency and security.

# Part I

# Reassuring responsiveness: a development of user interface for PROMET&O on mobile devices

# Chapter 2

# Background

## 2.1 Motivations

The front-end of the project, which is designed and implemented during previous work, has been overall stable in terms of performance and user interface/user experience aspects. However, it is necessary to point out its lack of adequate responsiveness.

After switching the context from workplaces to classrooms, PROMET&O has majority of its user base on mobile devices such as mobile phones and tablets. With such devices, operators should be able to collect environmental data from the dashboard, and regular users can express their personal experiences according to their own perspectives efficiently.

The inadequacy of responsiveness means sacrificing functionality and efficiency during user interactions, which could potentially introduce biases into user feedback. For example, when a user feels exhausted during the interaction, that user may provide more negative feedback. Biases like this are obviously not in the favor of inspectors. For this reason, the candidate recognizes the necessity of a rework towards the layout.

## 2.2 Visual hierarchy

In terms of user interface design, visual hierarchy is a key aspect worth highlighting.

Visual hierarchy, according to Gestalt psychology, is a pattern in the visual field wherein some elements tend to "stand out," or attract attention, more strongly than other elements, suggesting a hierarchy of importance. [5]

There are several ways to form visual hierarchy during design, namely:

- Color
  - Contrasting colors could lead users to identity and in turn focus on specific parts on the interface.
- Size
  - Like color, contrasting sizes can also guide users to identity and classify information.

- Alignment
  - Alignment is the basis of layout, it helps users to quickly find the order of the information.
- Character
  - Different shapes can help building a hierarchy of information, providing that complex shapes tend to attract more attention from the user.

Ensuring the visual hierarchy inside user interface is an important mission during the front-end design and one of the candidate's most focused part, since the user interface of PROMET&O has a significant requirement on efficiency: to the operators, direct and clear data display should be the priority, while to users it should be friendly and time-saving to ensure minimum impact on user biases. the candidate constructs such a hierarchy by using available methods and functions native to CSS and front-end frameworks (namely React.JS and Bootstrap).

## 2.3 Useful features from CSS

CSS, or Cascading Style Sheets, is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript. [4]

CSS has a wide range of features that can be used to enhance the user interface of a web application. One of such features is the `vw` and `vh`. These units are relative to the size of the view, which is the visible portion of a web page. The `vw` is equal to 1% of the width of the view, while the `vh` is equal to 1% of the height of the view. These units are useful as they allow the developer to create layouts that adapt to the size of the view.

Also, `em` and `rem` are also useful in CSS. The `em` is relative to the font size of the element, while the `rem` unit is relative to the font size of the root element. They allow the developer to create scalable text that adapts to the size of the view.

Moreover, `@media` is a feature of CSS that allows the developer to apply different styles based on different conditions, such as the size of the view or the orientation of the device. This feature enables the developer to create layouts that adapt to the size and orientation of the view.

Those are merely a fraction of the features that CSS provides.With those features, the candidate has been able to create a responsive and efficient user interface that meets the requirements of the project.

## 2.4 SASS: a CSS pre-processor

SASS(short for syntactically awesome style sheets) is a CSS preprocessor that grants developers the access to more flexible and maintainable style sheets on their web applications. SASS supports custom variables, nested rules, and functions. Those are commonly used features that provide a more efficient way to write style sheets. Its compatibility to

native CSS syntax significantly eased the learning curve. React.js has native support for SASS format thus decreasing the complexity of the integration.

In the development on the new front-end, the candidate has been trying to embrace this new format, as it truly presents a balance between readability and efficiency without impacting the current technology stack.

For example, the following is a pair of functions that are constantly used throughout the development:

```
// Define the default font size
$default-size-font: 16

// Returns the relative font size based on the default font size
@function getRelativeFontSize($num)
    @return calc($num / $default-size-font)

// Returns the relative font size for landscape orientation
@function getRelativeLandFontSize($num)
    @return clamp(#{$num}px, #{getRelativeFontSize($num)}em, #{$num * 2}px)
```

The first is a declaration and definition of a SASS variable value, it sets the default font size to `16px`.

The function right below returns the relative font size based on the default font size, basically transforms the value into em value. This is to adapt to the high-level prototype of the user interface designed by external staff where all the values are defined using pixels.

The second function uses clamp to define a vague range of font size that is being used under landscape mode. This can help to ensure the proper display of font sizes.

As shown above, the SASS is presented an efficient and effective tool for the development of the new front-end.

## 2.5 Typical user interfaces on mobile devices

The most common cases of a user interface on mobile devices consist of a screen with a 9:16 resolution aspect on "portrait" mode, and a 16:9 aspect on "landscape" mode which is similar to a regular PC monitor.

A regular user, specifically a participant of the survey, is typically using the "portrait" mode. In our user story, a user gets to the survey form by scanning a QR code attached to a classroom, users tend to use portrait mode to do so.

As for operators, that may not always be the case – in fact, operators or inspectors have multiple contexts that vary from sitting before a desk to being on the move. In this case, the display mode is unpredictable – but since the user interface in landscape mode (on PC monitors) is already finished and tested before our work here, the candidate can focus on the design of portrait mode.

One aspect that may not always be visible to the user is that, the user interface is almost entirely designed to be adapting to different resolutions. Despite applying to the two-mode display, the resolutions of different devices are almost all nonidentical to each

other. Indeed, the actual resolution of a mobile device is a result of multiple factors including screen size, frame design, configuration and others. With this in mind, the candidate chooses the margin sizes, font size and other values in units adapting to the current view in pursuit of a higher level of responsiveness across different devices.

## 2.6 Determination of the device type

The process of determining the type of device is surprisingly not very straightforward, as nowadays mobile and desktop devices are becoming more and more similar to each other; one may find mobile devices being simply smaller in terms of screen size and that is all the difference one can name. The narrowing gap between those two types has made the determination not so reliable at first glance.

However, in logic, the size of the device is directly connected to the size of the screen, which can be the sole reason why we are engaging the development in the first place. After this realization, the candidate can safely say that differentiating devices solely based on their screen size is a reliable method. Such example can be found in our project.

```
@media(min-height: 901px) {
    #prometeoSmallLogo {
        display: none;
    }
}
```

It shows where we need to enforce certain properties based on the height of the view, a more detailed explanation on a similar example can be found in the snippet 2.8.

Another feasible method can be by processing the user agent. The user agent is a string that can be fetched from browsers representing various aspects about the user context, such as the operating system and browser that the user is currently using. One disadvantage is that it is not secured as it can be modified at ease, but since the user interface has very limited functionality regarding security, it can be a worthy trade.

One particular issue is that, the project have a dedicated type of device for specific use cases, which will be used by specific staff members. And the project has an original version of user interface specifically optimized for that usage, so naturally the candidate has no intention altering the user experiences on that device.

After evaluating all alternative methods, the candidate has made the decision to simply use a single line in vanilla JavaScript to determine the device type. It is not in any case a perfect solution but a trade-off between maintainability, reliability and efficiency. The line of code is shown below:

```
if (/Android|iPhone|iPad/i.test(navigator.userAgent) &&
↪  !/SM-X200/i.test(navigator.userAgent)) {
    document.body.classList.add("mobile-view");
}
```

The JavaScript line is tagging a specific class name to the class list of `body`. Since the front-end is a Single-Page Application, the `body` will maintain the class list throughout

the whole life cycle of the application, regardless of the navigating happening across views. This makes a simple way to allow the style sheet to identify the correct condition to apply corresponding style sheets.

## 2.7 Utilizing dynamic importing in React.js

The React.js framework supports conditional importing, which is a feature that enables more flexible import of resources based on current context or state. This feature is particularly helpful in the case as it can save computing resources by only importing certain style sheets (SASS files in our case) after the user interface has determined the device type, rather than importing useless style sheets when desktop users never need. The following code snippet illustrates the only usage of such feature in the project.

```
useEffect(() => {
    if (isDesktop) {
        import("./CSS/Hello_A8.css");
        import("./CSS/PrivacyNotice.css");
        import("./CSS/Thanks.css");
        import("./CSS/App.css");
        import("./CSS/SurveyJS_A8.css");
        import("./CSS/dashboard.css");
        import("./CSS/Profile.css");
    } else {
        import("./CSS/sass/App.sass");
    }
}, []);
```

In this case, the corresponding style sheets are imported based on the device the user is using (e.g., user agent). Such feature also helps the future maintenance.

## 2.8 Differentiating display modes

In the previous section, the candidate has discussed the determination of the device type. However, the question still remains as the display mode also needs evaluation.

The most common display modes are "portrait" and "landscape", and the most distinct way of telling which one the user is using is by inspecting the width and height of the view, in other words, the aspect ratio. The aspect ratio is calculated by dividing the width of the view by the height of the view. If the aspect ratio is greater than 1, then the user is using the landscape mode, otherwise the portrait mode.

Vanilla CSS already has the feature of `@media`, which can be used to apply different styles based on different conditions. Naturally, `@media` is supported by SASS as well, and is the main method used by the candidate to apply different style sheets based on display modes.

```
@media screen and (max-aspect-ratio: 1/1)
    .row-nav p
```

```
        padding-top: getRelativeHeight(20)
        padding-bottom: getRelativeHeight(20)
        margin: 0
```

The lines shown above set the `padding` and `margin` of the navigation bar based on the aspect ratio of the view. `max-aspect-ratio` is a feature of `@media` that can be used to set the maximum aspect ratio.

In this thesis, the candidate will highlight the situations where significant differences exist between the results under portrait and landscape mode, and how the candidate handles them. Normally, the candidate will demonstrate only the snippet applied to portrait mode, as it is the main design focus during the development.

## 2.9 Avoiding overflow

Overflow, being a very commonly found "feature" that can be found in almost every web or not web application, is a situation where the content of a container exceeds the size of the container itself. The default behavior for handling the issue by common browsers is by making the user scroll to look for hidden content. In the candidate's point of view, overflow is troublesome in terms of both aesthetics and user experience.

The problem of overflow is not only about the aesthetics of the user interface, but also about the experience; taking an example where users slide from one side to another, they will either get frustrated along the way, or only to find out there are merely things that they imagined to be there in the first place, or what's worse, nothing.

The worst overflow occurs when it reveals nothing valuable in the end. Those situations are typically due to improper `padding`, `margin` value, or lack of suitable implementation in the style sheets. In any case, overflow is not a desirable outcome, and getting rid of it has been a mission for the candidate throughout development. A typical example of preventing overflow could be a simple line like:

```
.some-random-class {
    width: 95vw;
}
```

By limiting the maximum width of `.container`, this line tries to prevent possible overflow within this container when it attempts to adapt to its content. Sometimes, content exceeds the container anyway; indeed, if the component is able to overflow even the screen size, surely a mere "container" will not contain it. In conclusion, the candidate understands that he needs to be careful when overruling the layout, and to be constantly aware of the content size.

However, in numerous situations overflow is inevitable, or is a result of trade-off for something of higher priority. In those cases, the candidate will try to at least erase the scroll bar and use other components to indicate users that there could be something hiding for aesthetic reasons.

## 2.10 "Immutable" content and maintainability

As the candidate introduced in the chapter 1, PROMET&O did not start in the hand of the candidate. Out of the trust and respect to previous developers, the candidate has been trying to keep the original content as much as possible. The reasons are mainly:

- The original content has been tested and proven to be functional.
- The candidate is not an expert on visual designs and questionnaire designs, so he will not try to undermine the work of more experienced person who came before.

In this section, the candidate will borrow the concept of "immutability" from the programming world to refer to a higher "resistance" to changes than amendments, or a mindset that can be constituted into a phrase "make modifications only when it is absolutely necessary", or a more commonly saying "if it works, do not fix it."

Despite the subjective argue given previously, a more objective reason is, some of the static resources, especially from SurveyJS, are previously compiled and generated before the user interface development. This also provides an explanation to the fact that, the main workload throughout the development of the candidate is making modifications on style sheets rather than the content or components themselves.

Also, decoupling and separating different generations of user interface design is eventually beneficial to the candidates afterwards by providing higher maintainability. Dynamic importing the style sheets allows other candidates to easily add or remove style sheets without touching the previous iteration, benefiting the testing.

On the other hand, there are situations where a modification on source code could save a remarkable amount of resources, or the code itself needs revision. Under those circumstances, the candidate will not hesitate to make necessary changes.

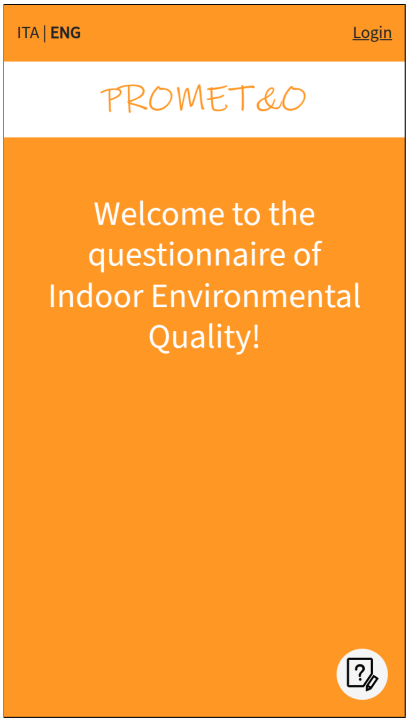
# Chapter 3

# Homepage

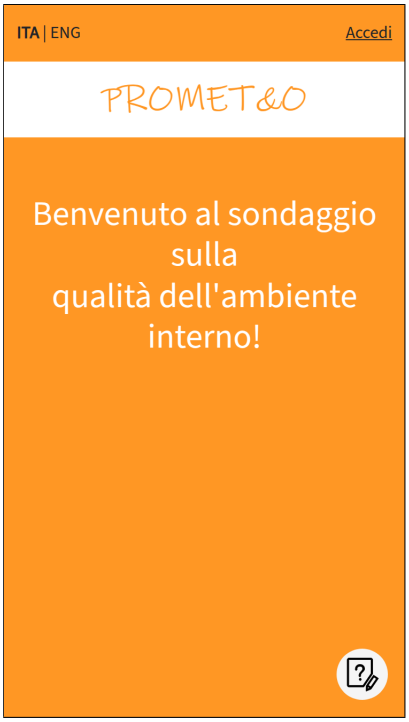
## 3.1 Welcome page for anonymous users

Upon accessing the domain without a cookie, the user is welcomed by the view for anonymous users. The default language is in English, the user can change the language setting by tapping the language option on top of the screen.

The main focus is the logo text which is rendered in a different font family than other text among the view. Aside from aesthetic reasons, the title also stands in between two other major parts of this view, namely the navigation bar and the interactive area, in order to help the user differentiate from each other.




(a) English welcome page




(b) Italian welcome page

As previously stated, the user interface is designed with respect to the actual view size of the user's device, so all the views will adapt to various screen sizes. For convenience of demonstration, every view under landscape mode or on another device will not be shown.

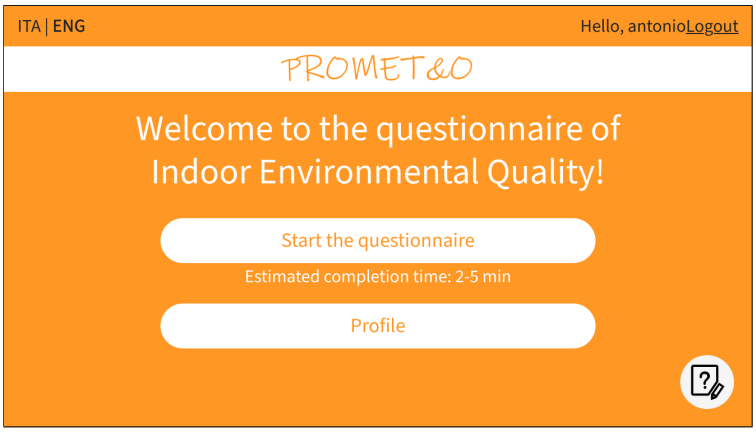



Figure 3.2: welcome page under landscape mode

On the top there is a navigation header, it serves the purpose of helping users to log in and change the current language. The interface uses an underlined style on the "Login" button to highlight it, as it is the most important functionality of this view nonetheless.

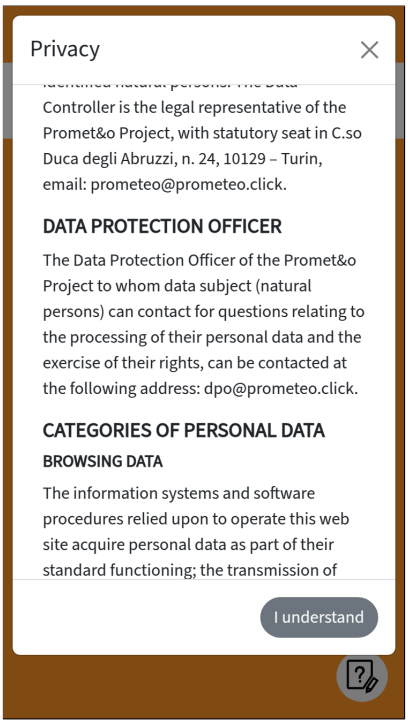



Figure 3.3: Privacy dialogue modal

A button for information on privacy is located on the bottom, once clicked it will communicate our principles of handling sensitive data to the user. The candidate used the following SASS code to prevent the dialog modal from overflowing to the external view.

```
body.mobile-view
    .modal-content
        max-height: 90vh
        max-width: 100vw
        overflow: hidden
```

## 3.2 Login and homepage

If the user possesses an account or needs to sign up, he can tap on the button. From there, the client will send a request to the authentication service to perform inspection on cookies and credentials. If the user does not possess any necessary information, the authentication process is now handed to the authentication service through a form.

If the user succeeds in logging into the system, they will be redirected back to the home page. The view now displays two buttons that indicate users they are about to have a choice of whether to participate in a feedback process or go to the personal profile page.

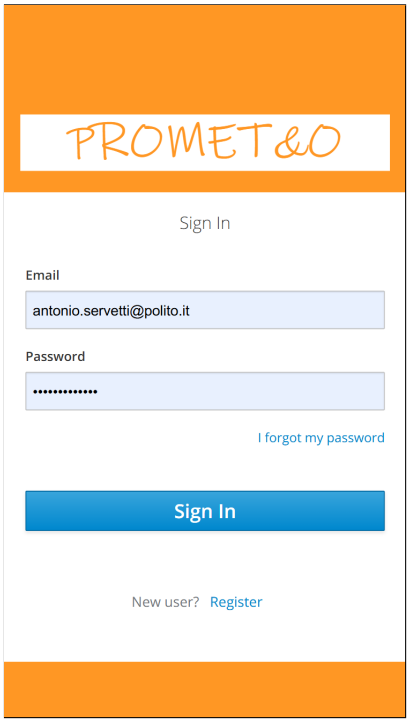



Figure 3.4: Login page

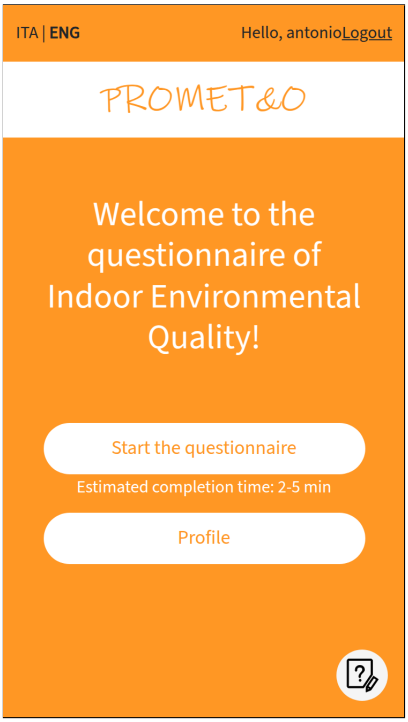



Figure 3.5: Logged page

# Chapter 4

# IEQ questionnaire

## 4.1 Overall Comfort

In order to capture the most direct and authentic feeling, the user is asked to first provide an overall rating about the room comfort.

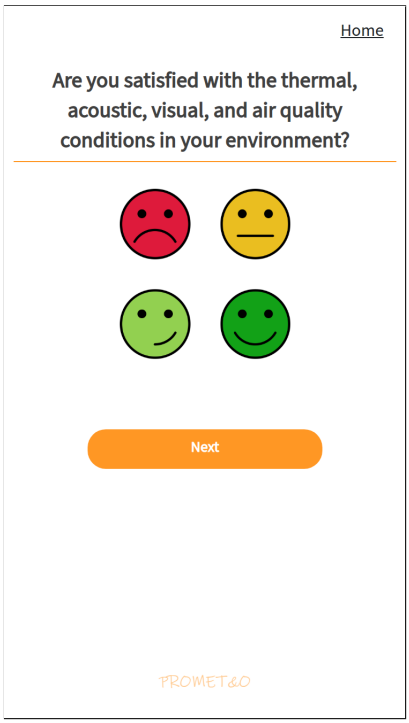



Figure 4.1: Overall feedback

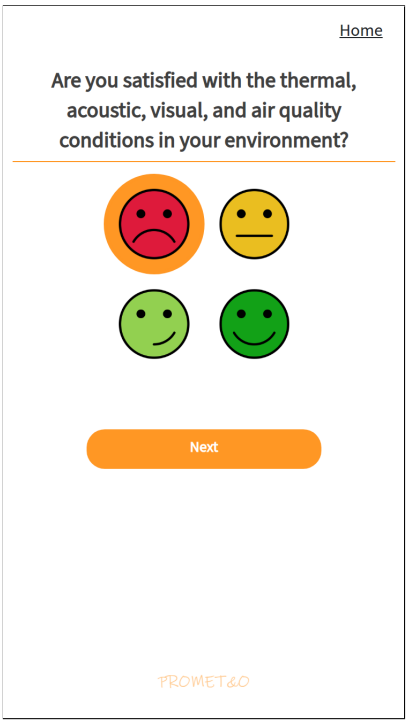



Figure 4.2: Overall feedback (picked)

The "HOME" button on the top right corner is the single point for navigation throughout the whole questionnaire. Users can tap on it to go back to the homepage, or to the profile page if they are logged in. The style sheets for the button are shown below:

```
#container-survey #button-home
    text-align:         end
    font-size:          getRelativeFontSize(16) + em
    padding-top:        getRelativeHeight(16)
    padding-right:      getRelativeHeight(16)
    text-decoration:    underline
```

In the application, the candidate will present several different style of navigation buttons, in adapting to external requirements and high-fidelity prototype, meanwhile maintaining the user experience.

Due to restriction of Single-Page Application and SurveyJS, the user cannot have the option to go back and forth during the questionnaire, it can sometimes cause confusion. The candidate is fully aware about this.

The main content of the view is obviously the image picker, manipulating them is not always easy to achieve. The original plan was to pursuit the most responsive behavior: the image should be centered and scaled to fit the view, based on the view width the image should be displayed either in a single line, or a two-by-two layout. However, the candidate found that the image picker provided by SurveyJS is not as flexible as expected. After considering the balance between comprehensiveness and feasibility, the candidate decided to use the uniformed two-by-two layout for the image picker.

```
@media screen and (max-aspect-ratio: 1/1)
    #container-survey [id^="surveyModel"] .sv-question__content > .
    ↪ sv-selectbase:has(.sv-imagepicker__item)
        display:                grid
        justify-content:        center
        grid-template-columns:  repeat(2, 1fr) // Two columns of equal width
        max-width:              fit-content
```

The significant part should be the `grid-template-columns: repeat(2, 1fr)` line, it enforces the said layout. The `max-width: fit-content` is used to prevent the image picker from overflowing to the external view.

Under landscape mode, the candidate has made the decision to take the advantage of the extended width from the mode. The image picker will be displayed in a single line, as shown below:

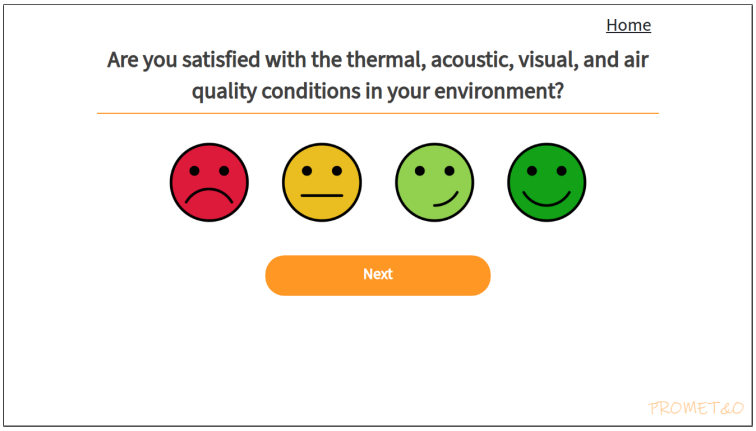



Figure 4.3: Overall feedback (landscape)

This is achieved by removing the `grid-template-columns: repeat(2, 1fr)` from the style sheets from portrait mode, thus reverting back to the default behavior of `display` value. Also, notice the `display` here has been changed to `flex`.

```
@media screen and (min-aspect-ratio: 1/1)
    #container-survey [id^="surveyModel"] .sv-question__content >
    ↪  .sv-selectbase:has(.sv-imagepicker__item)
        display:            flex
        justify-content:    center
        max-width:          fit-content
```

And it is required to adapt the size and scale of the button to fit this new layout.

```
@media screen and (min-aspect-ratio: 1/1)
    #container-survey [id^="surveyModel"] .sv-action > .sv-action__content >
    ↪  .sv-btn
        text-align: center
        height:     clamp(25px, getRelativeHeight(90), 50px)
        width:      clamp(25vw, getRelativeWidth(130), 50vw)
```

Here the candidate uses the `clamp` function to set a upper, optimal, and lower bound for the button size. `clamp` will prove itself very useful throughout this whole project.

The comparison between the portrait and landscape mode has been one of the major topic during the development of this project. By default, the user interface is designed with respect to portrait mode, and the landscape mode is considered as an extension of the portrait mode. But there could be situations where major differences happen, the candidate will hight light those differences when it is necessary.

Another interesting component of this view is actually the PROMET&O logo underneath the main content; under portrait mode, its relevant style sheets are shown below:

```
#prometeoLogo {
    position:       fixed;  // Fixed at the bottom of the view
```

```
    bottom:         1vh;    // 1% of view height higher than the bottom
    width:          100%;   // Takes the full width of the container (body)
    text-align:     center;
    pointer-events: none;
    z-index:        -1;
}

#prometeoLogo p {
    margin-bottom: 0;
}
```

But under landscape mode, the logo will adjust itself to the right corner of the view, avoiding it being overshadowed by the main content, specifically the button. This is a hard requirement from external.

As shown above, the logo is fixed at bottom of the view with one percent of view height higher than the bottom. The `pointer-events` is set to none to prevent the logo from being clicked, `z-index` is set to -1 to ensure the logo is always behind the main content. Also, in order to precisely control the margin of the paragraph, the candidate has set the `margin-bottom` to zero.

```
@media screen and (min-aspect-ratio: 1/1)
    #container-survey + #prometeoLogo
        text-align: end
        right: 1vw
```

By using the plus sign `+` in the selector, the CSS selector will only choose the logo that is accompanied by the `#container-survey`, which is the main content of the view. The `text-align: end` will align the text to the right, and the `right: 1vw` will set the logo to one percent of the view width from the right side of the view.

Throughout the questionnaire, if the user did not fill in any mandatory field, they will be notified by built-in notification mechanisms.

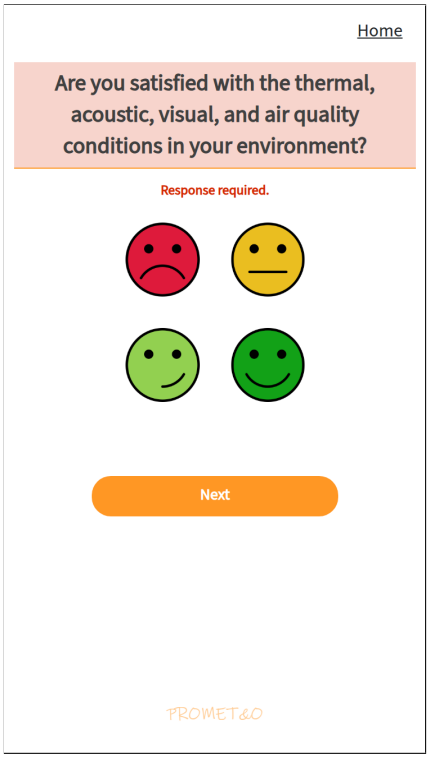



Figure 4.4: Overall unfilled

## 4.2 Classification of Issues

After the system acquired the overall comfort level from the user, it will attempt to further classify the issues that the user is experiencing. The user can choose multiple choices from a list of predefined issues that is monitored by the platform.

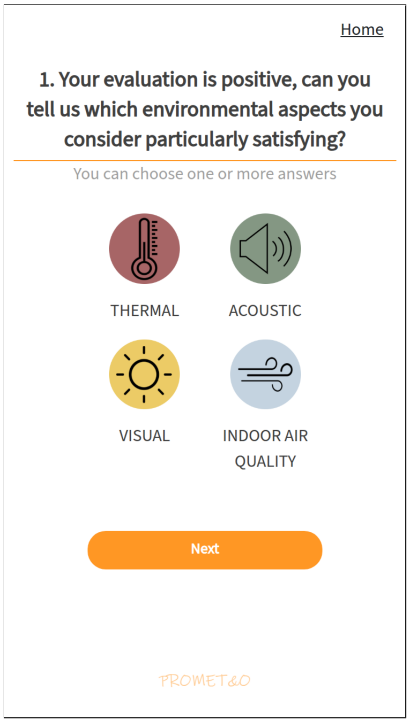



Figure 4.5: Overall feedback (positive)

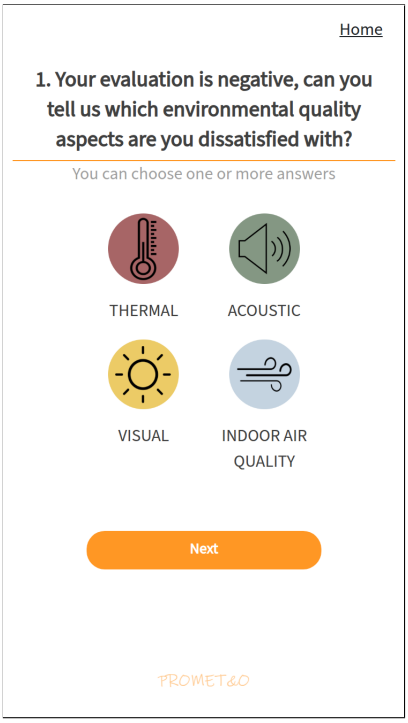



Figure 4.6: Overall feedback (negative)

One difference, being not so significant, is the help message that will be displayed underneath the question text if exists. There are situations where additional messages or blocks like this will effect the overall layout in the next views, major or not.

Another issue about this view is that, as one can see in the picture, the forth icon which has the longest caption text and thus taking the most width of the view, is affecting the overall layout. This can be avoided by limiting the maximum width value of the component, combining with the `word-break` to properly wrap the word, as shown below. The value of `max-width` is chosen subjectively.

```
#container-survey [id^="surveyModel"] .sv-question__content > .sv-imagepicker >
↪  .sv-imagepicker__item .sv-string-viewer
    font-size:  getRelativeFontSize(14) + em
    max-width:  30vw    // Limit the width of the caption
    display:    inline-block
    word-break: break-word
```

## 4.3 Thermal comfort questionnaire

If the user has selected a negative overall comfort level about the thermal comfort in the section 4.1, the system will display this view to gather further information on the feedback.

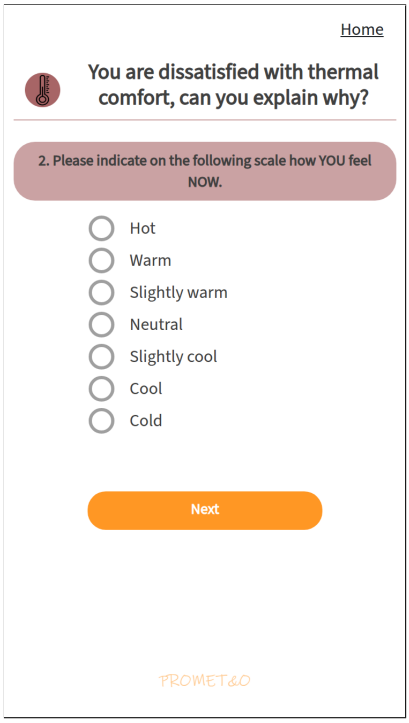



Figure 4.7: Thermal comfort (1)

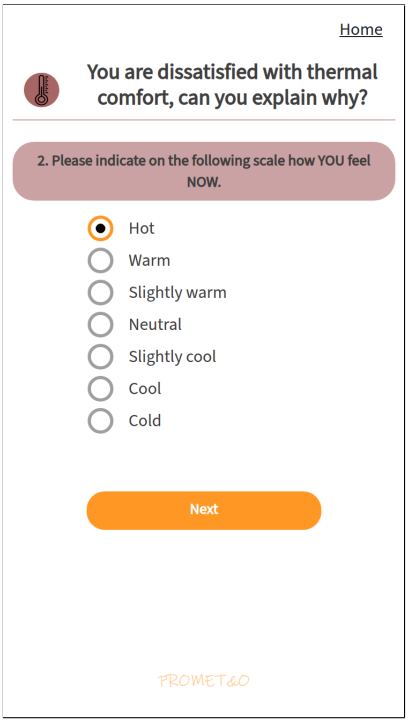



Figure 4.8: Thermal comfort (2)

It is obvious that the candidate has introduced a bias upon the horizontal position, it can be argued that such bias maintains the visual hierarchy and also reserves the aesthetic of the view. The following SASS snippet is used to construct such layout:

```
@media screen and (max-aspect-ratio: 1/1)
    #container-survey [id^="surveyModel"] .sv-question__content >
    ↪  .sv-selectbase:has(.sv-radio)
        display:            grid
        justify-content:    center
        padding-right:      90px    // The horizontal bias
```

Major differences between portrait and landscape mode are presented in this view, since the height of the radio selector under portrait mode takes too much space vertically and will introduce major overflow issues under landscape mode. To resolve this, the candidate uses `grid-template-columns: repeat(2, 1fr)` like in previous sections. Additionally, the style sheet removed the margin from the home button to save horizontal space.

```
@media screen and (min-aspect-ratio: 1/1)
    #container-survey #button-home
        margin-bottom:          zero()

    #container-survey [id^="surveyModel"] .sv-question__content >
    ↪  .sv-selectbase:has(.sv-radio)
```

```
        display:                grid
        // Removed the horizontal bias
        // padding-right:       90px
        justify-content:        center
        grid-template-columns:  repeat(2, 1fr)
```

Eventually the view under landscape mode is presented like below.

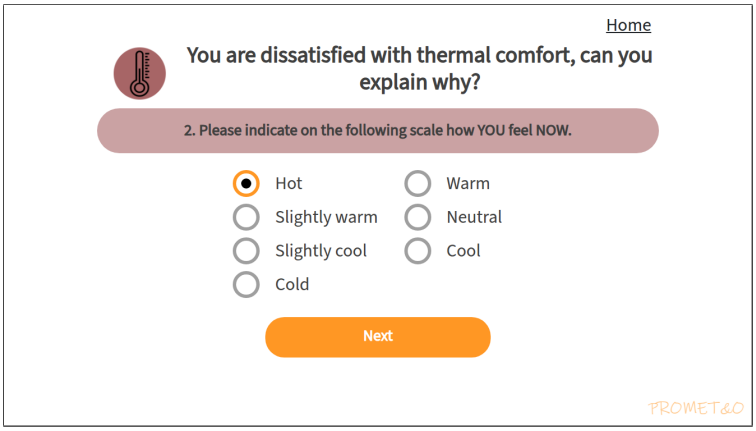



Figure 4.9: Thermal radio landscape (1)

The focus of the user is naturally guided to the title and the help message, which is the desired behavior. One little detail in questionnaire view under landscape mode is the disappearance of the line divider and the gap between the question text and the help message, to save the vertical space as the margin between the background of the help message and the question text is enough to separate them.

```
@media screen and (min-aspect-ratio: 1/1)
    #container-survey [id^="surveyModel"] .sv-question__content > div >
    ↪ div[style]
        display: none
```

This component is a little tricky to select using SASS; if the style sheet choose here `.sv-question__content` it is enough to do the trick currently. However, looking at the class name one may find it being very "abstract" or "general", which could potentially in turn select unintended components. Considering risks like this is very damaging for maintainability, the candidate has decided to use a more specific pseudo-selector `div[style]`, which stands for a `div` with an attribute `style`, no matter what the `style` is. Firstly it is more specific, and secondly this greatly narrows down the possibility of unintended side effects.

After the system acquired the feedback on user comfort about room temperature, it will further ask the user about air velocity as it acts as a major index around the topic. The help message that is highlighted using `BackgroundColor` has its relevant style of the following:

```
#container-survey [id^="surveyModel"] .sv-page .sv-row:not(:first-child) > div
↪  h5
    font-size:      getRelativeFontSize(14) + em !important
    padding:        2%    !important
    border-radius:  20px  !important
```

The font size of this message is slightly less than `1em`, which is to build visual hierarchy as the candidate prefers it not exceeding the title. The padding and border-radius are used to make the message more appealing on visual. Note that the candidate did not use `em` as the unit for `border-radius` as the candidate believes it should be a fixed value and is independent to the screen size.

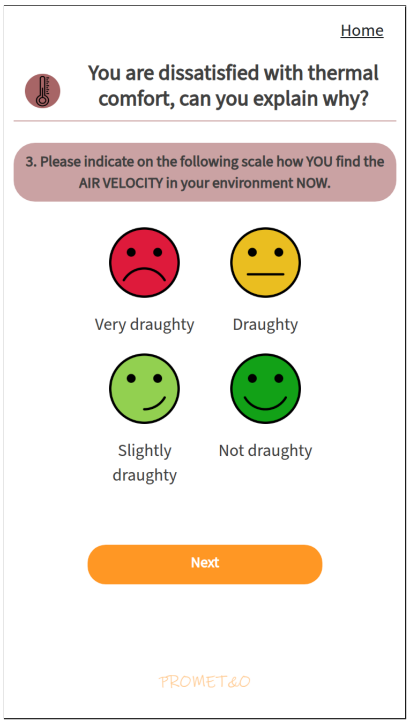



Figure 4.10: Thermal comfort picker (1)


Figure 4.11: Thermal comfort picker (2)

Similar to the previous view, the candidate constructs the layout with a single line display on landscape mode.

A point that was not mentioned in the previous view is the change of the icon size under portrait and landscape mode. The following shows the relevant SASS lines:

```
// Portrait
@media screen and (max-aspect-ratio: 1/1)
    #container-survey [id^="surveyModel"] .sv-question__content >
    ↪  .sv-imagepicker > div > .sv-imagepicker__label .sv-imagepicker__image
        width:  20vw
        height: 20vw
```

```
// Landscape
@media screen and (min-aspect-ratio: 1/1)
    #container-survey [id^="surveyModel"] .sv-question__content >
    ↪  .sv-imagepicker > div > .sv-imagepicker__label .sv-imagepicker__image
        width:  15vw
        height: 15vw
```

Looking at the values it is easy to see the icon size is 33% percent larger under portrait mode than landscape. This better utilizes the vertical space and makes the icons more prominent, since the picker is supposed to be the main focus inside this view.

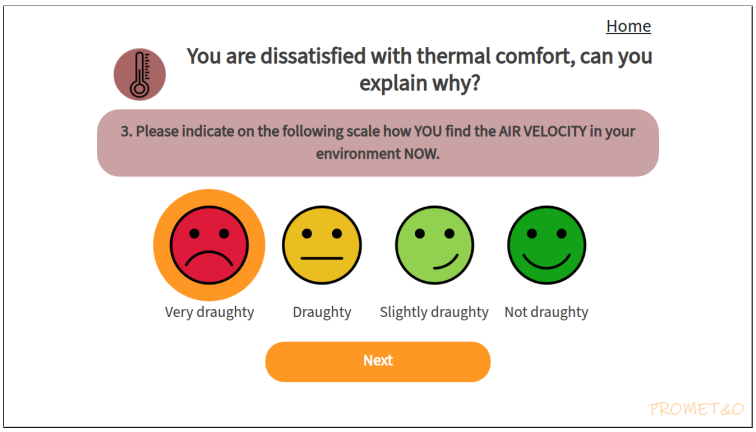



Figure 4.12: Thermal comfort picker (landscape)

## 4.4 Acoustic comfort questionnaire

The first page of the acoustic comfort questionnaire is shown below, the user will be asked to provide an overall rating about the acoustic comfort of the room.

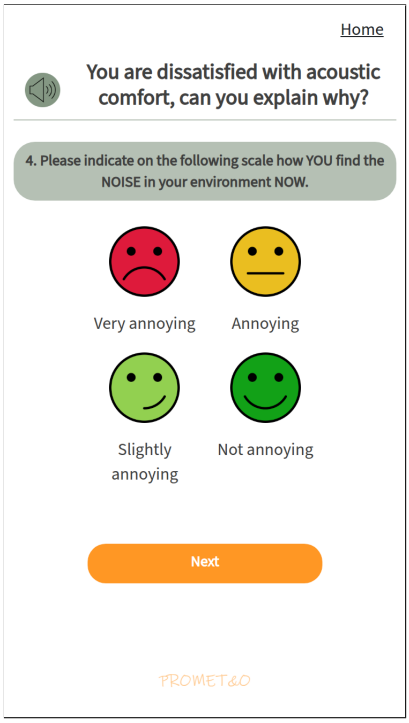



Figure 4.13: Acoustic comfort picker (1)

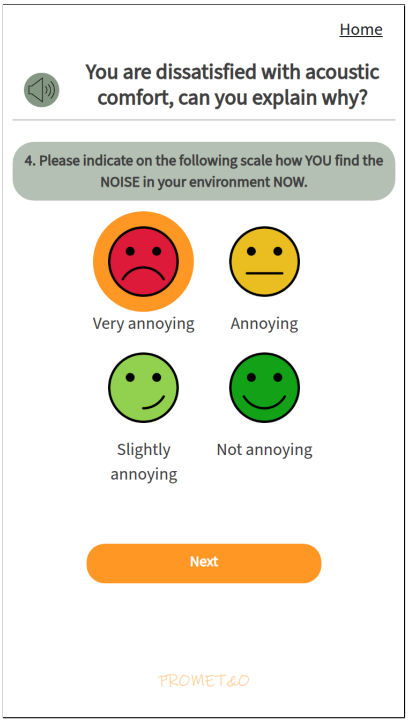



Figure 4.14: Acoustic comfort picker (2)

The style sheet applied on this form of questionnaire has already been covered by previous sections that contain image pickers, thus it will not be repeated here.

The following view, which consists of check boxes, is the second page of the acoustic comfort questionnaire. It collects the information on possible sources of environmental noises around the room. The user can select multiple options, and the view will display the selected options with a highlighted ring.

Note that the check boxes in the application often times has lengthy lines, which is the reason why the candidate has not added any bias horizontally under portrait mode, this helps better visualize the options and makes the view more appealing. The following SASS code is used to achieve this:

```
@media screen and (max-aspect-ratio: 1/1)
    // Checkboxes picker div
    #container-survey [id^="surveyModel"] .sv-question__content >
    ↪  .sv-selectbase:has(.sv-checkbox)
        padding-left:       50px    // Add padding to the left
        padding-right:      50px    // Add padding to the right
        display:            grid
        justify-content:    center
```

The candidate has introduced a "limit" rather than "bias" on both sides of the check boxes, its sole purpose is to further concentrate the user's focus on the options.

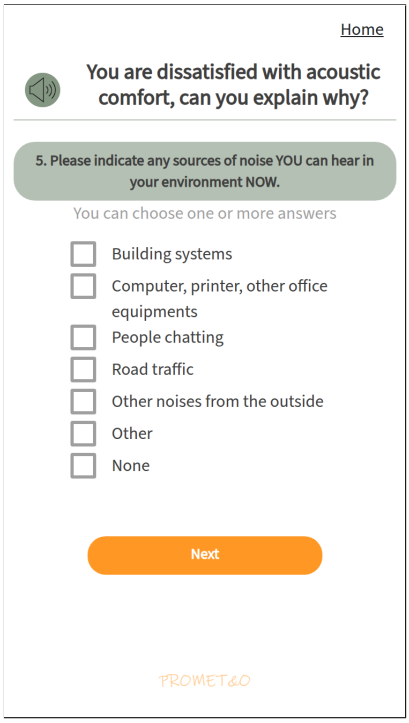



Figure 4.15: Acoustic comfort portrait (1)

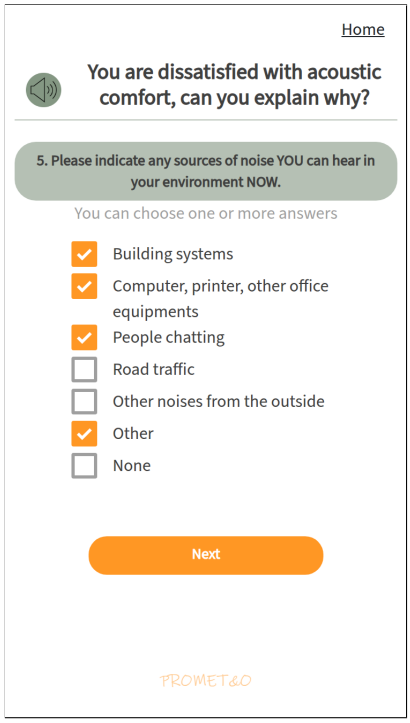



Figure 4.16: Acoustic comfort portrait (2)

A more interesting part about this view is the distribution of options on the radio checkers. In previous views it looks decent, but unfortunately the candidate cannot have the same conclusion on the checkers in this one. As shown below, there exists a more lengthy option among other options that are more or less at the same width level. To cope with this, the candidate chose to reserve the overflow in this page, considering that a specific fix for this issue would have an impact on both resources and maintainability.

```
@media screen and (min-aspect-ratio: 1/1)
    // Checkboxes picker div
    #container-survey [id^="surveyModel"] .sv-question__content >
    ↪  .sv-selectbase:has(.sv-checkbox)
        padding-left:           50px
        padding-right:          50px
        grid-template-columns:  repeat(2, 1fr) // Enforcing two columns
        display:                grid
        justify-content:        center
```

Notice there is a double-layer structure of helping messages, it may also contributed to the exhaustion of vertical space. But since they are vital components of both visual and operational purposes, the candidate will not make any alteration on them, thus the overflow stays. A simple overflow has become a more viable solution for more advantages it generates.

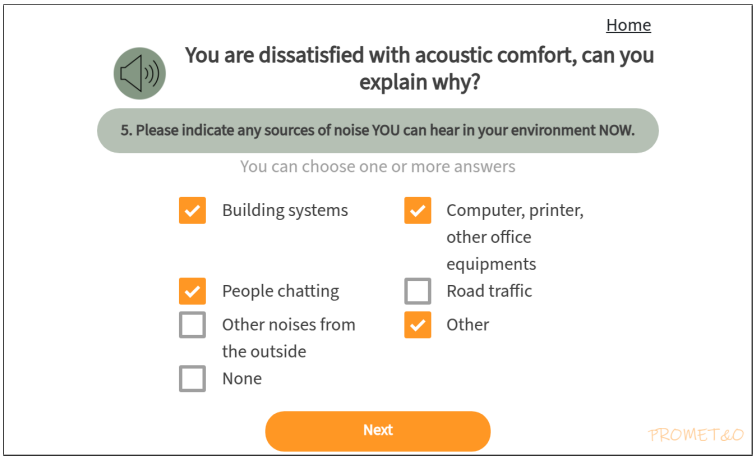



Figure 4.17: Acoustic checker landscape

## 4.5 Visual comfort questionnaire

If the user chose in section 4.1 the option for visual comfort to express their complaint about visual comfort, the system will display this view to gather relevant information.

The visual comfort questionnaire is a good example for summarizing the most repetitive procedures of the questionnaire, namely image picker, radio picker and check boxes. By comparing the layout between all three types of the form, it is very easy to see the consistency and visual hierarchy that the candidate has been emphasizing during this theses. At first, we will have a image picker in Fig. 4.18.

The main focus is surely the image picker, we reserve the most space for the options that are distributed over a two-by-two layout. The first priority is to provide to the user the room of operation, and secondly to reduce the probability of mishandling the choice. The color differences on each icon also helps the user to have an intuitive understanding of the selected options.

Next, the help message that is asking "the real question" right above the focus will provide further messages for users to assist them on better understanding their choice. The question text may appear in less significance than the two above but it still helps improving the user-friendliness of the view, the icon besides it also provides a visual cue.

The button below is separated enough from the main content to reduce the possibility of mistake and also build the visual hierarchy. And the logo is properly placed below it.

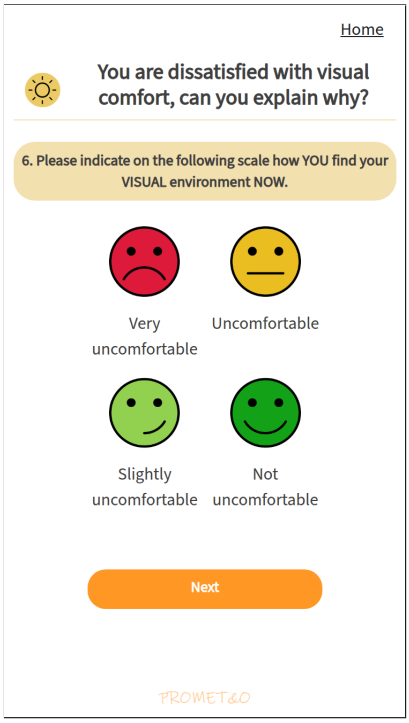



Figure 4.18: Visual comfort picker (1)

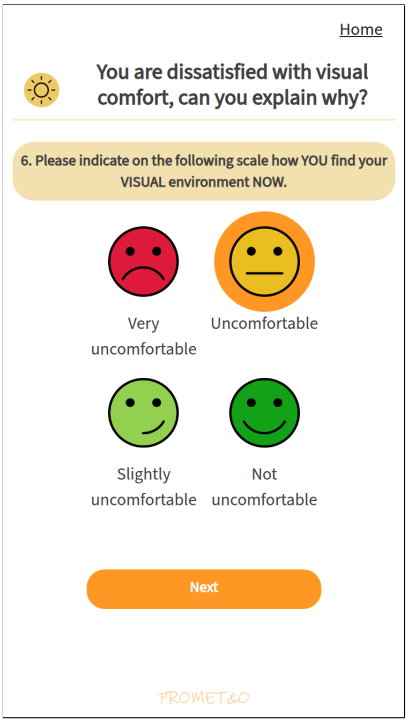



Figure 4.19: Visual comfort picker (2)

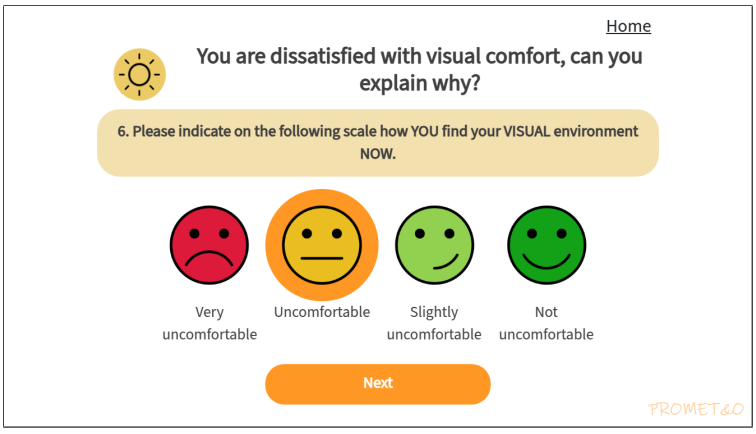



Figure 4.20: Visual comfort picker (landscape)

The design is consistent with the check box form in Fig. 4.21, where the user can select multiple ones from well distributed options. The help message is placed right above the focus point, and the question text locates above the help message. The button is placed below the main content, and the logo is placed below the button. And like previous views, the landscape mode will have the same layout but with a two-by-two layout of the options.

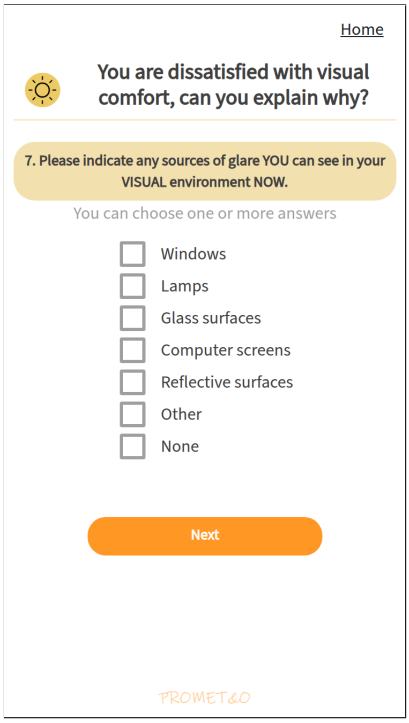



Figure 4.21: Visual comfort checkbox (1)

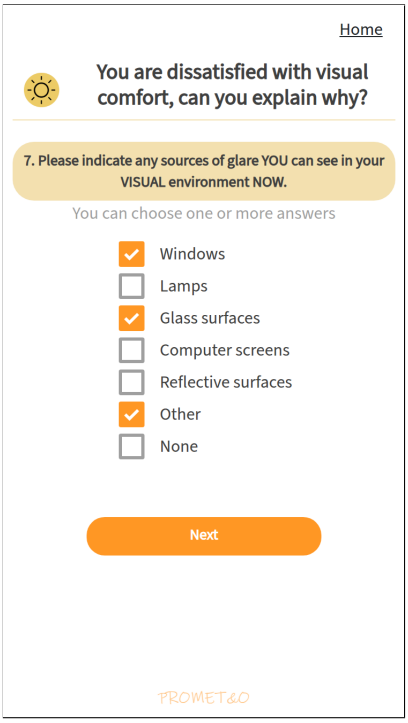



Figure 4.22: Visual comfort checkbox (2)

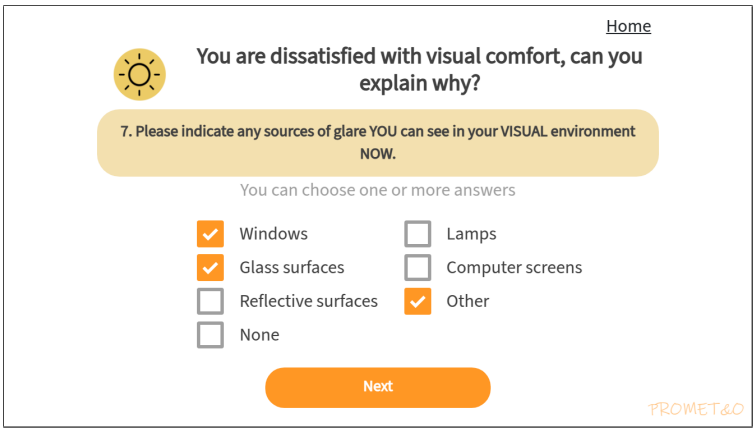



Figure 4.23: Visual comfort checker (landscape)

Also for the radio form demonstrated in Fig. 4.24, the user can only select one option. The help message is placed right above the main picker, and the question text is placed right above the help message. The button is placed below the main content, and the logo is placed below the button. The landscape mode will have the same layout but with a two-by-two layout of the options.

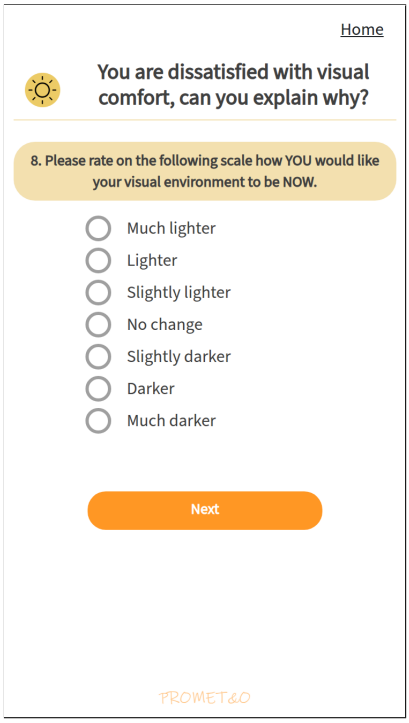



Figure 4.24: Visual comfort radio (1)

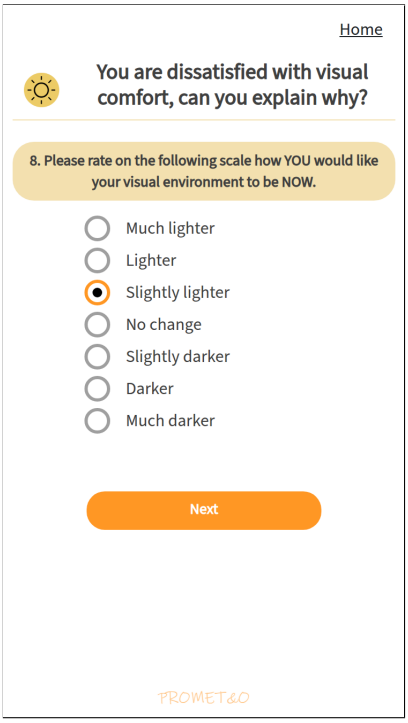



Figure 4.25: Visual comfort radio (2)

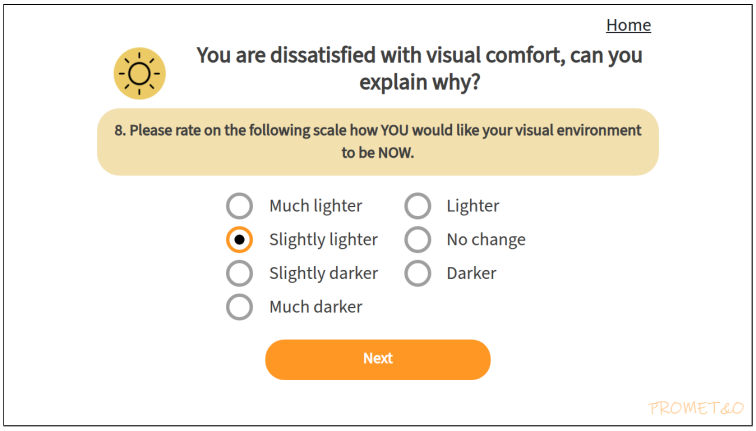



Figure 4.26: Visual comfort radio (landscape)

## 4.6 Air quality questionnaire

If the user has chosen the option for air quality to express their feeling about the complaint about air quality, the system will display this view.

At this point the user is likely to be adequately familiar with the procedure and visual of the questionnaire, and the candidate believes the pursuit of consistency during the design and development of the questionnaire is crucial for achieving this.

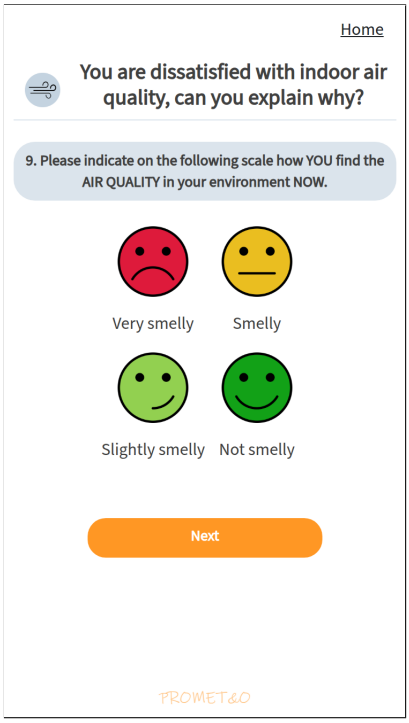



Figure 4.27: Air comfort portrait (1)

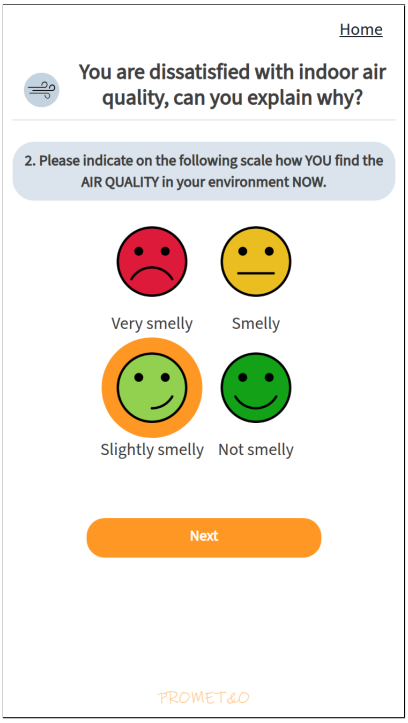



Figure 4.28: Air comfort portrait (2)

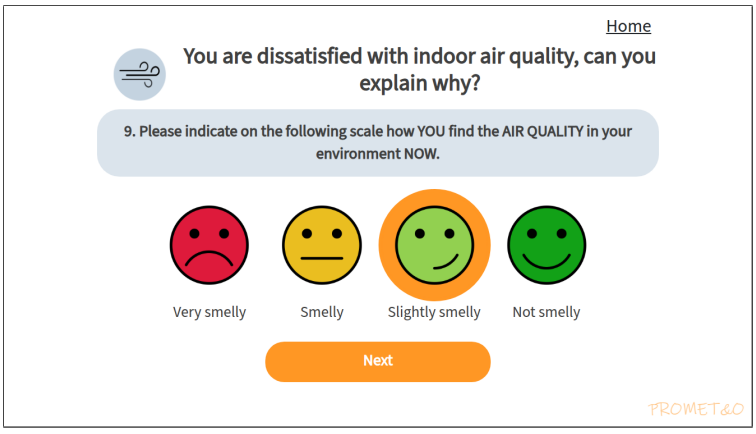



Figure 4.29: Air comfort landscape (1)

The next step is the system trying to acquire the reason why user is feeling discomfort in the room, it contains some common options of the reasons why the user is feeling discomfort in the room. This time, the options are not lengthy and very much numbered, this benefits the display under landscape mode. The same with all check boxes, the user can select multiple options.

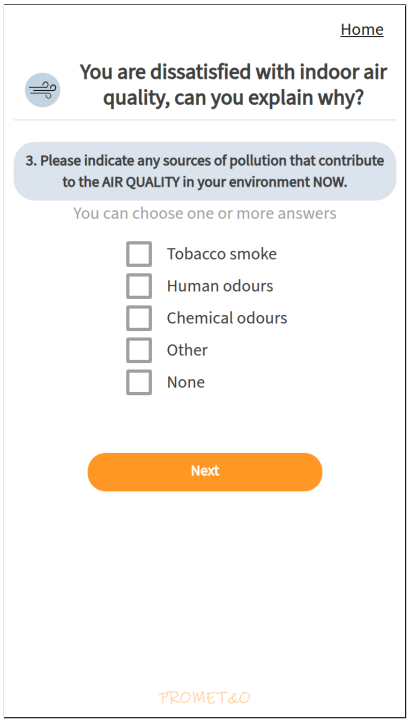



Figure 4.30: Air comfort portrait (1)

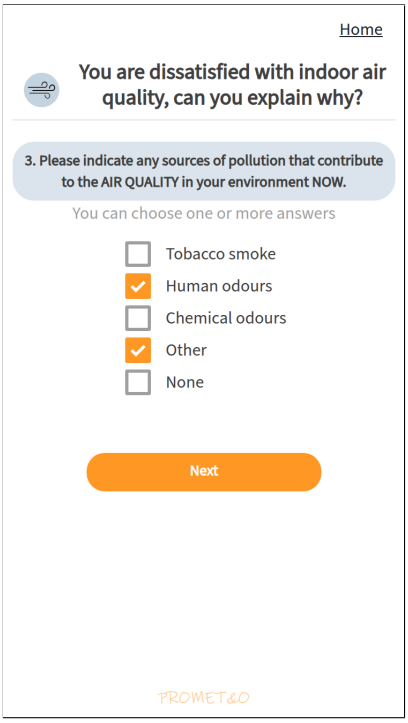



Figure 4.31: Air comfort portrait (2)

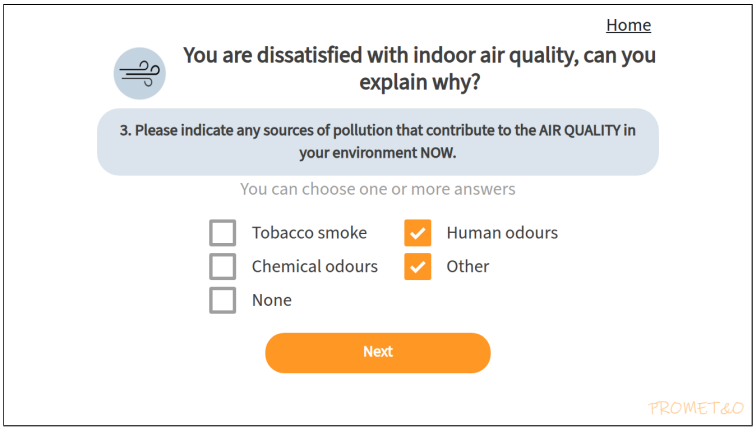



Figure 4.32: Air comfort landscape (1)

## 4.7 Final comments

After users finishes providing their feedback on their complaints, they reach the final step of the questionnaire, where the user is able to input a piece of message to express their feeling about the room.

This could be helpful to the system by collecting some informative data about the room besides implicit answers over a set of predefined answers, and also for the user to

have their answers justified or emphasized. However, the user is not supposed to enter too many words at a time.

Alternatively, the user is also able to simply skip this step if they have such intention by tapping the "Complete" button.

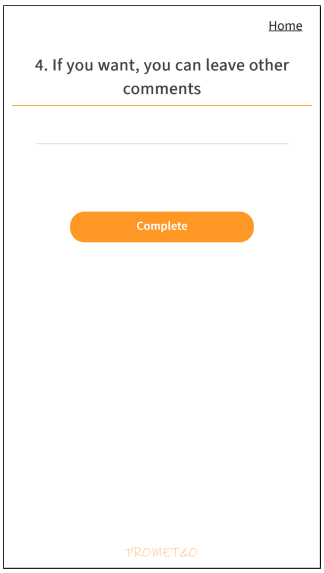



Figure 4.33: User comment portrait (1)

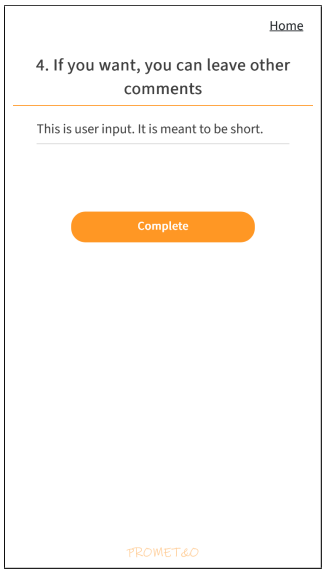



Figure 4.34: User comment portrait (2)

Discussions have been made about if the candidate should maintain a single line of text input, or instead replace it with a multi-line text input. The result is that by presenting a impression of short text input, the user interface can guide user to provide simple messages that tend to be straight to the point. This is beneficial for the space resources for communication and back-end.

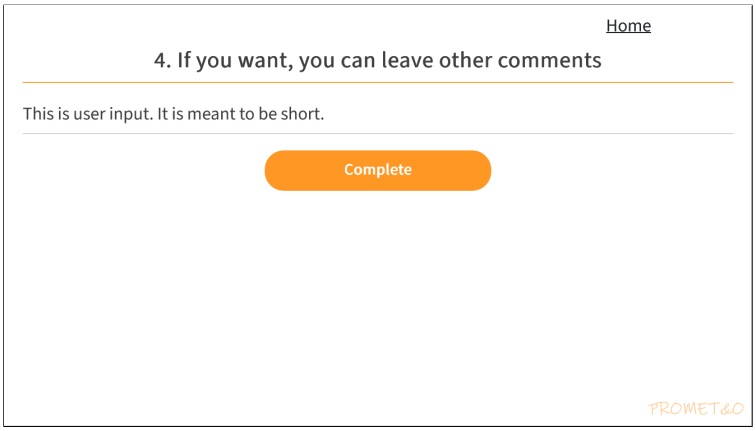



Figure 4.35: User comment landscape (1)

## 4.8 Notification on completion

After expressing their dissatisfaction about the quality of the IEQ indices, the user is asked to leave a short message about their experience. This message is then sent to the system administrator for further analysis, and the user is then redirected to this view to receive a piece of message of gratitude.

Other than the kind words, the user is also given a "receipt" for their completion of the questionnaire – it summarize the answers given by the user using a group of percentages reflecting the degree of the user's level of satisfaction based on their choices in the questionnaire. Not only does this indicate the completion of the questionnaire, but also serves as a reminder for the user about their experience and communicates the communication has been successful. The receipt is shown in Figure 4.36.

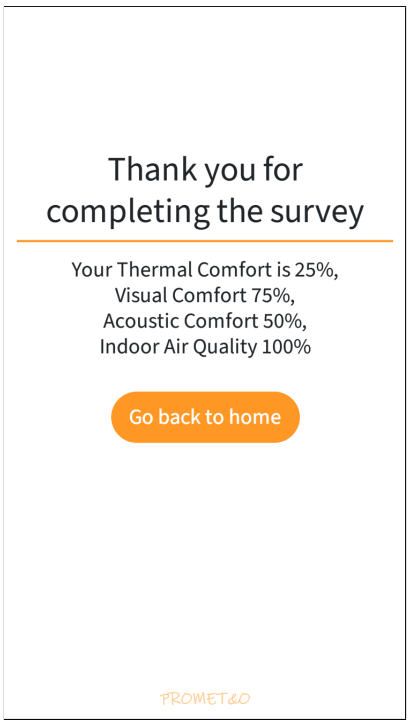



Figure 4.36: Receipt for questionnaire

# Chapter 5

# Profile page

## 5.1 Profile page

### 5.1.1 Showcase

Once users chose to tap on the button "Profile" on the welcome screen 3.5, they will be redirected to the profile page.

Despite the name being seemingly self-explanatory, the "profile" page is not as true to its name as it appears to be. Rather than a page that serves the purpose of displaying and providing the option to modify the personal information attached to the current user, it is actually more of a "hub page" that provides the user with a variety of options to navigate to different parts of the application based on the information it provides.

The view is separated, based on functionalities, three different sections.

On top, you have the welcome message that displays the username. Firstly the message greets the user, and secondly it confirms users that they have successfully logged in and they are on the right account at the moment. We do not provide any further information beyond first name basis for the sake of privacy.

Right under the divider stroke we have the main interactive area, which consists of two identical buttons labeled respectively "Dashboard" and "Personal". The "Dashboard" button serves as a pathway to the dashboard page, while the "Personal" button will lead the user to the personal questionnaire view.

Finally at the bottom area, the section that contains a page-based list view of past finished personal questionnaire records. Each record will show the date and time of completion, as well as the sensor number indicating where the user has completed the questionnaire. If the user has not completed any questionnaire yet, the list will be empty, as shown in Figure 5.1. If the user has completed at least one questionnaire, the list will be populated with the records, as shown in Figure 5.2.

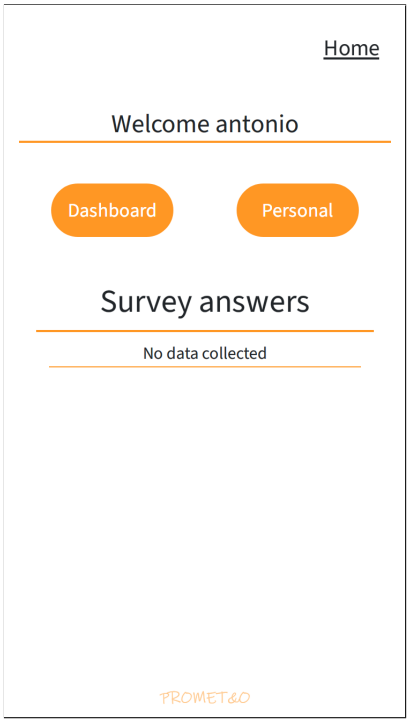



Figure 5.1: Profile with blank record

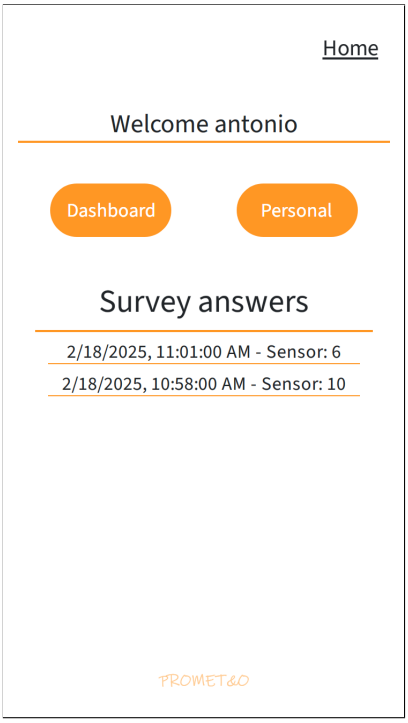



Figure 5.2: Profile with previous records

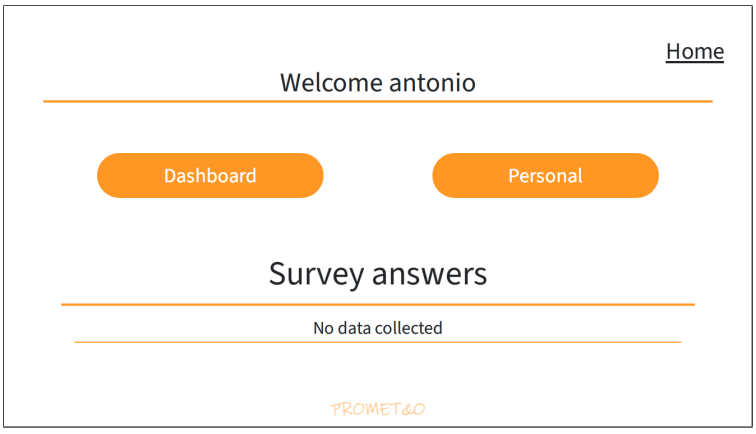



Figure 5.3: Profile with blank record (landscape)

### 5.1.2 Style sheets

Let us look at relevant SASS lines that are in charge of the view. Note that since the SASS file is dedicated to the mobile view (on desktop devices the application will import corresponding CSS file instead), for this reason we will not add the extra `@media` line to indicate whether the lines are applied to mobile view or not, as in this case, all lines are for the mobile devices by default.

Firstly, we need to set the proper and clear boundary for the profile page.

```
// Set boundary for container with id Profile
body.mobile-view #Profile
    padding-top:    getRelativeHeight(90)
    padding-left:   getRelativeWidth(30)
    padding-right:  getRelativeWidth(30)
    padding-bottom: getRelativeHeight(60)

// Set the font size for the welcome message
body.mobile-view #surveyTitle
    font-size:      getRelativeFontSize(30) + em
```

The usage of `getRelativeHeight()`, `getRelativeWidth()` is to ensure that the `padding` is set in a way that is relative to the screen size, so that the padding will be consistent across different devices. Same with some views in previous, due to the implicit nature of the high-fidelity prototype, we use functions to convert fixed pixel values to relative values.

Next we will set the style for the buttons. They are almost identical under both the portrait and landscape mode, with the only difference being the height of the buttons. Using the ancient art of "eye-balling", the candidate has determined that it needs a little more on height under landscape mode than portrait mode. The usage of `clamp()` function is to set a definitive boundary for button sizes.

```
@media screen and (max-aspect-ratio: 1/1)
    body.mobile-view #button-profile, #button-personal
        font-size:  getRelativeFontSize(18) + em
        height:     clamp(25px, getRelativeHeight(90), 50px)
        width:      clamp(25vw, getRelativeWidth(130), 40vw)
        margin:     getRelativeWidth(10)

@media screen and (min-aspect-ratio: 1/1)
    body.mobile-view #button-profile, #button-personal
        font-size:  getRelativeFontSize(18) + em
        height:     clamp(25px, getRelativeHeight(100), 50px)
        width:      clamp(25vw, getRelativeWidth(130), 40vw)
        margin:     getRelativeWidth(10)
```

As for the record list, the candidate did not make any modifications upon the original implementation, as the default style provided by the framework and previous work is sufficient for our needs after properly setting the boundary for the profile page.

# Chapter 6

# Personal questionnaire

## 6.1 General introduction

As the thesis approaching the later stage of development on the user interface, it may surprise readers that, among all the structures, layouts and components, it is the personal questionnaire page that the candidate finds to be one of the most complicated pages to implement, if not the most.

The personal questionnaire page is designed to be a view where the system collects information that is more personal from the user, including non-identifiable information such as age, job, education, and it will ask some general questions about the user's preferences and habits. The main difficulties come from two aspects in general;

- The information inside the view, especially one on a mobile screen, appears to be rather saturated, so it has exhibited a form of "magnifying effect" on defects, such as misalignment of contents. Any slight misconduct on the implementation may lead to a overly negative user experience. Such, the candidate has explored through multiple iterations of design and implementation to ensure maximum level of user experience.
- Just like the IEQ questionnaire, the personal questionnaire is also generated in advance by SurveyJS and has a certain degree of "immutability" we introduced in section 2.10. This means that the candidate has to work around the code to reach adequate level of code efficiency.

One may realize that reasons above will result in an excessive amount of trade-offs between aesthetics and functionality during the development. Moreover, a questionnaire is meant to be interacted with, which means not only should the user be able to see the questions, but also be able to answer them. Considering all obstacles, perfection can be a very challenging task to achieve, especially when the screen size is limited, in the next sections the candidate will make a lot of design choices along the way.

## 6.2 Questionnaire

The personal questionnaire consists of a two-page form asking about general user information, such as age, job, country of origin, education alongside with several experience-oriented close questions. In terms of the way the questionnaire is built, the questions can be classified into two categories:

- Drop-down selectors
  - Questions adopting drop-down selectors are in most cases open-questions that can divide its answers into multiple sub-divisions. An example of this could be acquiring the gender of the user, and the user can choose between several answers that are defined beforehand. Another example of this could be the requesting user's age, where all the possible answers are distributed evenly into several ranges of values.
- Radio selectors
  - Radio selectors are used for close questions that can be answered by a binary selection, oftentimes contradicting and completing another. One example being the question asking users if they are smoker or not, where users can only choose one between the positive and negative option.

One specific issue about the personal questionnaire is that it shares an amount of its class name with the IEQ questionnaire, which means that the candidate has to implement extra logic to make sure that the personal questionnaire is not affected by the IEQ questionnaire's style sheet, and vice versa. This is done by adding extra `id` to each main content container, which will only include content for personal questionnaire. In PROMET&O, for personal questionnaire, the `id` is `#personal`, you can find it being demonstrated in most of the SASS lines in this chapter.


Figure 6.1: Personal questionnaire (1-1)

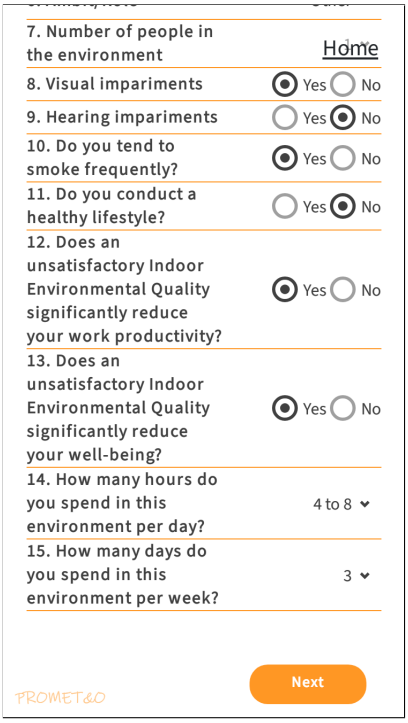



Figure 6.2: Personal questionnaire (1-2)


Figure 6.3: Personal questionnaire (1 landscape)

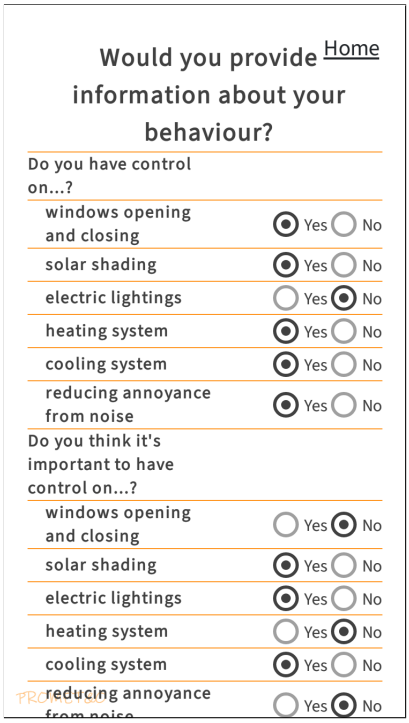



Figure 6.4: Personal questionnaire (2-1)

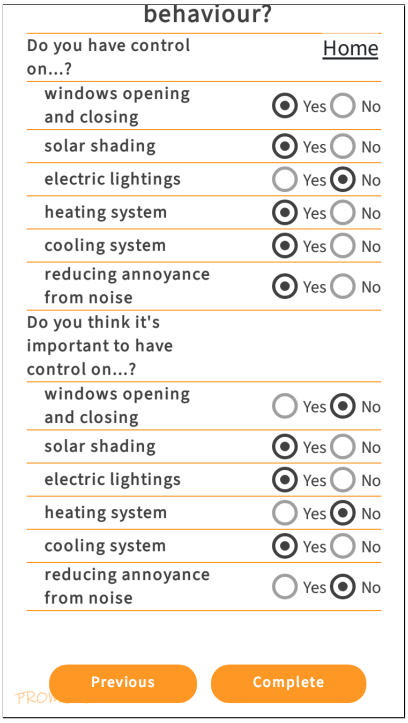



Figure 6.5: Personal questionnaire (2-2)

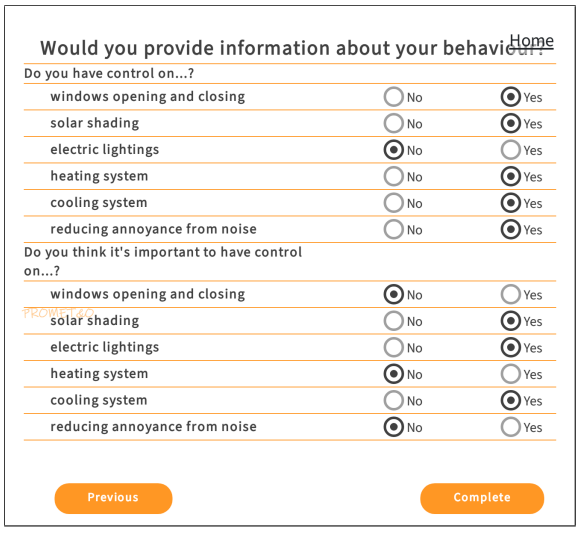



Figure 6.6: Personal questionnaire (2 landscape)

It is obvious that the horizontal space is more forgiving in the landscape mode, which leads to a more relaxed layout. Although the candidate agrees that cramped layout could impact user experience, the layout under portrait mode is considered a compromise between functionality and user experience. We do alter some of the aspects under portrait

mode, for example, making selectors aligned vertically in center to create an impression of space thus more relaxing atmosphere. Let us start the introduction from the top.

## 6.3 An inspection in detail into the style sheets

### 6.3.1 Navigation

Before starting with the main container content, some potential doubts from external need clarifying. Users can see the navigation button on the top-right corner of the view. This button is the only way to navigate back to the homepage or the profile page. The original button has itself styled as:

```
<p
  id="btn-home"
  role="button"
  style={{
    position:       "fixed",
    top:            25,
    right:          25,
    textDecoration: "underline",
    fontSize:       "130%",
    margin:         "0"
  }}
  onClick={() => navigate("/")}
  >
  Home
</p>
```

You can quickly understand the intended behavior of this button: it is fixed on the top-right corner of the view, and when clicked, it will navigate the user back to the homepage. The style of the button is set to be aligned with the high-fidelity prototype.

In this personal questionnaire view, as requested by the external requirements, the "Home" button needs to follow the users' view as they scroll down. But in way it will inevitably overshadow part of the content. The candidate has decided to produce a half-transparent background for the button, so that the content can maintain a certain degree of readability while have a certain degree of aesthetics.

```
div.container-fluid:has(div#personal)
  // Home button
  + #btn-home
    background-color:   #FFFFFF80
    // Floating logo
    + #prometeoLogo
      text-align:       start
      left:             3vw
```

The `background-color: #FFFFFF80` line is used to set the background color of the button to be half-transparent white, the candidate believes this is adequately friendly

to the visual. the `div.container-fluid:has(div#personal)` is used to ensure that the button is only applied to the personal questionnaire container, while `+ #btn-home` is used to select the button which is a sibling of the container. The subsequent `+ #prometeoLogo` is used to set the position of the PROMET&O logo at the bottom, which is also a subsequent sibling of the container. Also note that the indentations here is not to be messed with, as it regulates the cascaded relationship between said components, indeed the SASS compiler is sensitive to indentations.

### 6.3.2 Main container

Now we can get into the main content container. Firstly, the candidate would like to introduce the structure of the personal questionnaire; a list of `.sv-row` inside of a `.sv-page` container. Before we start modifying the style sheets, we must clear out general margins from all the containers to save space. If we need adding margins to some of the containers, we will do it specifically as those cases are definitely minor ones. In fact, the candidate has been making this type of modifications in most of previous style sheets, and it is now formally introduced.

```
body.mobile-view #personal .sv-page
  *
    margin: 0
```

The main title, being the largest in scale in terms of font size, is often omitted in the first sight as the attention from the user tends to be dragged onto the questionnaire content. However, the candidate is aware of that if we leave the main title alone then after the modification on other contents, the title will naturally stands out as a mess. So we need to properly style the main title first. The style of the main title is shown below:

```
// Main Title
body.mobile-view #personal .sv-page
  > .sv-title
    display:            flex
    justify-content:    start
    font-size:          getRelativeFontSize(24) + em
    margin:             zero()
    border-bottom:      1px solid $color-theme

    .sv-string-viewer
        width:          100%
        text-align:     center
```

Those are merely modifications on `font-size` and boundaries on its container. This style also assign the `font-size` into `24px` in terms of the relative size, making it the largest text in the whole view.

After that there are a piece of general configuration on each line of `.sv-row`. First we have the titles, the selector `.sv-row:not(:first-child)` here chooses every `.sv-row` except the first one, in which holds the main title.

```
body.mobile-view #personal .sv-page
  .sv-row
    padding-bottom:         zero()

    :not(:first-child)
        > .sv-title
          display:          flex
          justify-content:  start
          font-size:        getRelativeFontSize(24) + em
          margin:           zero()
```

Next we have some lines of more specific configurations on the selectors. At first we center the text, solidify the bottom border of each line.

The selector `&[style="..."]` is used to set the padding of the selector to be `5%`. This line is seemingly more complicated than others but it will turn otherwise if we look into it. Firstly it inherits the `.sv-question` selector which is the general selector for the question. Originally, this component will have an inline-style `--sv-...: 40px;"` generated by SurveyJS that adjusts the `padding` of the selector to specifically `40px`, however this value appears to be excessively large under the portrait mode view thus it is replaced by a horizontal bias of `5%` of container size.

In the end, we have the `> .sv-question__header` line that is to set the width of the header to be `50%`, it will keep the question text of every row to the left side.

```
body.mobile-view #personal .sv-page
  .sv-row
    // Question div
    .sv-question
        display:            flex
        justify-content:    space-between
        border-bottom:      1px solid $color-theme

        :has(div:empty)
            font-size:      getRelativeFontSize(16) + em

        &[style="--sv-element-add-padding-left: 40px;"]
            padding-left:   5% !important

            > .sv-question__header
                width:      50%
```

The following part is applied to the specific case where a container that has `.sv-radio` inside. The `display: flex` and `justify-content: space-between` lines are used to properly divide the left and right part of the each line. Inside of it, The `.sv-selectbase` stands for the selector part of the selector, which should be aligned in center vertically. The `label` stands for the text part of the selector, in its corresponding line, it will properly adjust the margin with respect to the selector.

```
// Where there is a radio selector
body.mobile-view #personal .sv-page
  .sv-row
    &:has(.sv-radio)
      display: flex
      justify-content: space-between

      .sv-selectbase
        display: flex
        align-items: center
        padding: 0

        label
          margin-left: 2em
```

### 6.3.3 Question rows

The following part is the most complicated part of the personal questionnaire view. The candidate has to make sure that the user can both see clearly the questions and answer them properly. Like the candidate has mentioned in previous chapters, there were a lot of back-and-forth on the design and implementation during the development of the personal questionnaire view.

The candidate roughly divides each row of the questionnaire into two parts; one that contains mostly text strings, and one that contains mostly interactive parts, more specifically, the two kinds of selectors we talked about in the previous section 6.2.

For the left-hand side, we have the following style sheets:

```
body.mobile-view #personal .sv-page
  .sv-row
    .sv-question__header
      width:              100%
      max-width:          50vw
      display:            flex
      align-content:      center
      margin:             0!important
      border-bottom:      none
      flex-wrap:          wrap

      .sv-title
          padding:        zero()
          font-size:      getRelativeFontSize(16) + em
          word-break:     normal
```

The style sheet above demonstrates a general structure; the one labeled `.sv-question__header` is the container that contains the question header, which is the the half on left side of the row, and `.sv-title` is the actual text component.

The combination of `width: 100%` and `max-width: 50vw` ensures the text container takes exactly half of the view width. The `display: flex` and `align-content: center` lines are used to center the text vertically within the container. `flex-wrap: wrap` wraps

the text when it exceeds the width of the container. Finally `word-break: normal` line sets the text to break normally when it exceeds the width of the container. In a word, we want the container to take half of the view width, and all its content positioned exactly at center. The subsequent `.sv-title` is used to set the font size of the text to `16px` in terms of the relative size. `word-break: normal` breaks the question text properly.

As the candidate finishes introducing the left-hand side, now he will discuss the right side, where the selectors are placed. The style sheets are shown below:

```
body.mobile-view #personal .sv-page
  .sv-row
    .sv-question__content
      display:              flex
      align-content:        center
      font-size:            getRelativeFontSize(16) + em
      flex-wrap:            wrap

      &--left > fieldset
        display:            inline
        justify-content:    end

      > div
        align-content:      center

      .sv-selectbase
        display:            flex
        justify-content:    end
```

Above is the style sheet for general cases. Like the left-hand side, the right-hand side is formed in a structured layout. The `.sv-question__content` is the container that forces an centered justifications and alignment of the content. The `flex-wrap: wrap` line is used to wrap the text when it exceeds the width of the container.

`&--left`, in combination of `.sv-question__content`, can be used to select `.sv-question__content--left`. This and following couple of lines are used to set the selector to be right-aligned, as contents on the right-hand side should be.

Now we can finally check the line specifically serves for the selectors. Firstly we have the drop-down selector.

```
body.mobile-view #personal .sv-page .sv-row
  .sv-dropdown
    width:          100%
    border-bottom:  none

    > .sv-dropdown__value
      transform:    translateX(-1vw)
```

Again, the selector appears to be container-content pair, where the `.sv-dropdown` being the container and `.sv-dropdown__value` being the content. The `width: 100%` line

is to set the width of the selector to be the same as the container, indicating it will take as much space as it needs, then it clears the bottom border. `transform: translateX(-1vw)` is used to move the selector to the left by `1vw`, adapting to the portrait mode view.

As for the radio selector, we have the following style sheet:

```
body.mobile-view #personal .sv-page .sv-row
  .sv-radio
    align-content:  center

    .sv-selectbase__column
        min-width:  fit-content

    .sv-selectbase__decorator.sv-item__decorator
        left:       -2em
```

Similar to the drop-down selector, those lines are mainly to adjust content to portrait mode view.

One significant difference than the personal questionnaire that never existed in the IEQ questionnaire is the addition of the "popups", where the user needs to interact with a popup to answer the question. Under portrait mode, we need to properly adapt to the view size, avoiding overflowing.

```
body.mobile-view #personal .sv-page .sv-row
  .sv-popup__shadow
    display:            flex
    justify-content:    center
    align-items:        center

  .sv-popup .sv-popup__container
    .sv-popup__body-content
        display:        flex
        max-width:      80vw
        max-height:     80vh
        overflow:       hidden
        margin:         zero()
        padding:        zero()

    .sv-action-bar
        padding-top:    zero()
        padding-bottom: zero()
```

Main focus here being `max-width: 80vw` and `max-height: 80vh` which limits the size of the dialog modal to at most 80% of the view width and height.

There is a potential conflict with an external component which also belongs to the class `.sv-action-bar`, but adding this specific selector under `.sv-popup` will make sure that the style is only applied to the action bar inside the popup.

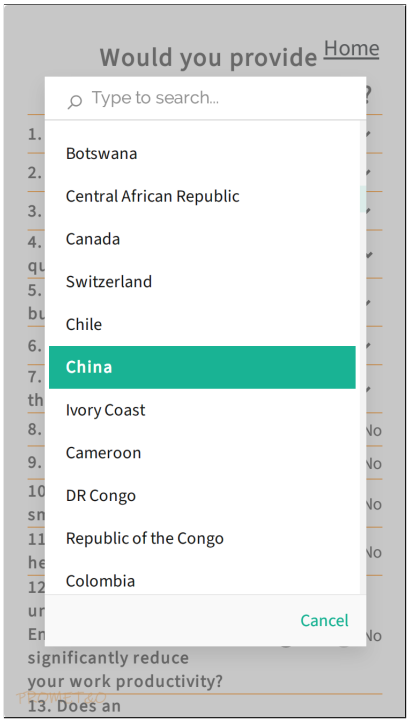



Figure 6.7: Personal popup (1)


Figure 6.8: Personal popup (2)

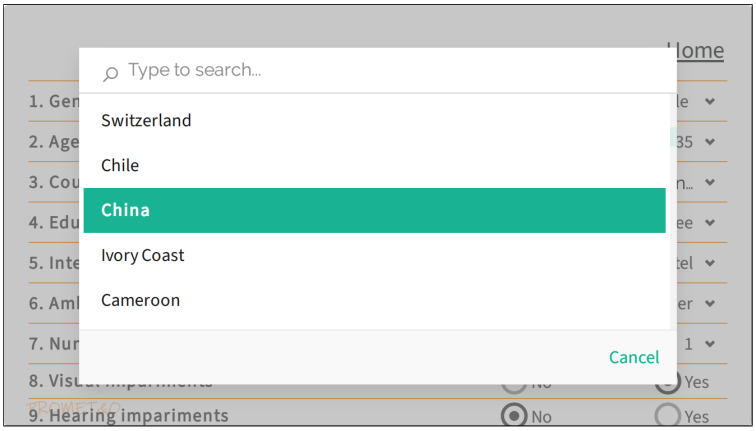



Figure 6.9: Personal popup (landscape)

### 6.3.4 Buttons

Finally, the last part for our introduction is the buttons for navigating within the questionnaire. The button group is placed at the bottom of the view.

```
body.mobile-view #personal .sv-page .sv-row
  .sv-action-bar
    display:                flex
```

```
  justify-content:        end
  margin:                 zero()
  padding-top:            2vh
  padding-bottom:         5vh

  .sv-action
    padding-bottom:       zero()

    input
      text-align:         center
      background-color:   $color-theme
```

The lines are mainly to adjust the buttons, along with its containers, to be fixed at bottom of the view. The unbalanced `padding-top` and `padding-bottom` is actually meant to avoid collapsing with the trademark logo.

# Chapter 7

# Dashboard

## 7.1 Overview

### 7.1.1 Dividing the devices

The original PROMET&O has its front-end user interface specifically designed for a certain device type, meaning it is heavily dependent on a specific resolution. But for the dashboard, simple modifications on style sheets are not adequate anymore for adapting to various screen sizes. Thus, after discussion, the candidate will use a separated JSX file that is specifically designed for the dashboard on mobile devices. This file will be responsible for rendering the gauges and other components that are unique to the dashboard. This way, the candidate can ensure that the dashboard will appear decent on various screen size, and it will be easier to maintain and update in the future.

The main method for differentiating device types and display modes are already discussed in the previous sections 2.6 2.8. Here we will show how to redirect to corresponding JSX component according to current aspect ratio, and in turn, a new set of style sheets.

```
<Route
    exact
    path="/dashboard"
    element={<ProtectedRoute logged={logged} anon={anon} />}
    >
    {!NO_DASH && (window.innerWidth < window.innerHeight || !isDesktop) ?
        <Route path="/dashboard" element={<PDashboard ita={ita} />} />
        : <Route path="/dashboard" element={<LDashboard ita={ita} />} />
    }
</Route>
```

Above is the `Route` component from React Router, it checks for the aspect ratio of the screen and the device type, and then renders the corresponding dashboard component. The `PDashboard` component is for portrait mode, and the `LDashboard` component is for landscape mode. Remember that the layout has to reserve the view for a certain device, so the `isDesktop` variable is used to determine whether the device is a desktop or not. The `isDesktop` variable contains this logic in its definition, which we demonstrated in the previous section 2.6.

### 7.1.2 The display of data

In the very beginning of the thesis, the candidate has explained the structure and purpose of the PROMET&O project. Its core feature evolves around the feedback loop for end users to collect their personal comfort about IEQ indices. Thus it makes sense that the "display of current data" has become one of the most, if not the most important functionality among all of them.

In general, IEQ indices, being much related to human comfort, has itself deeply intertwined with the concept of "range". For example, the temperature is considered comfortable when it is within a certain range, being it 20 $°C$ to 27 $°C$ or somewhat alike. The same goes for other indices like humidity, $CO_2$ concentration, etc. Therefore, it is appropriate to choose a type of display that can effectively show the current value of an index in relation to its range. In the end, the "gauge" type of display has been chosen for this purpose. An demonstration of the gauge display used in PROMET&O can be seen in Figure 7.1.

Although people have their own ranges defining personal comfort, there are guidelines on IEQ indices that are properly defined and widely accepted. One example being the ASHRAE 55-2017 standard [2] provides a range of temperature that is considered comfortable for the major popularity. It represents a balance between thermal comfort of the occupants and the energy efficiency of the building. What the candidate is trying to express is that, the choice for the range on the gauges are based on both objective and subjective perspectives.

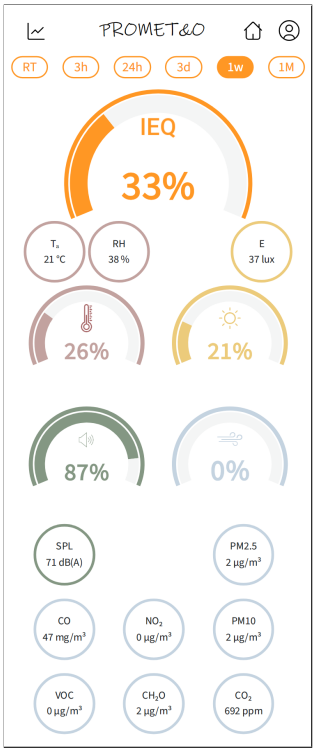



Figure 7.1: An overview of a gauge display

### 7.1.3 Navigation

How to efficiently navigate users to where they desire can sometimes be challenging; on a limited screen estate, there are choices that have to be made to perform constantly trade-offs from space to functionality.

In an application that emphasizes on efficiency, whether on the energy perspective or user experience, firstly the candidate does not want to add extra study curve for such application, since the users are supposed to merely use it for no longer than five minutes and forget about it ever since. Thus, a typical "header-bar, sidebar and main view" combination has been the choice for the general layout. The header-bar contains the title of the application and a couple of buttons that act as the navigation options. The sidebar contains the navigation links to different parts of the application alongside with other assistant features such as filters, and the main view is where gauges are displayed. A showcase of a sidebar can be seen in Figure 7.2.

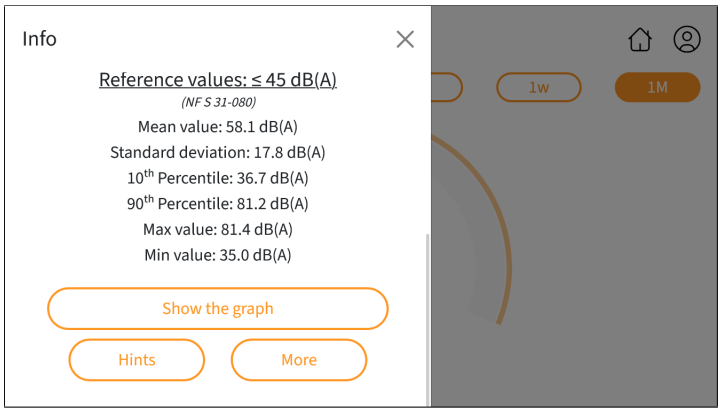



Figure 7.2: The sidebar

Also, the navigation on the same view is an important aspect that is worth for a discussion. One can quickly realize that, as in the application the view is constantly changing its content, a respective view size is adapting to it. The question is what is the optimal way to notify the user that the view has changed. The candidate, continuing the faith of "no extra study curve", has chosen to use a simple "cursor" animation. Such cursor will only appear when the view has changed, more detailed introduction can be found in the subsequent subsection 7.2.1.

## 7.2 Gauge view

The gauges view is the one that greets the users after they reaches the dashboard. It is the most important view of the dashboard, as it is the place where the users can see the current values of the IEQ indices. The gauges view is designed to be simple and straight-to-the-point, users can quickly see the current values of the IEQ indices and understand the current state of the environment.

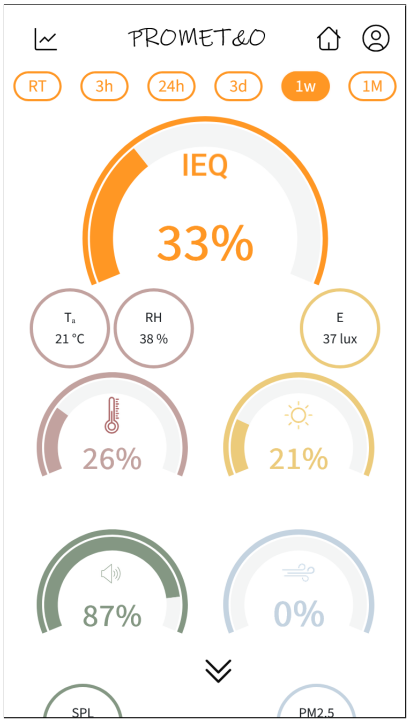



Figure 7.3: The gauges view (portrait) 1

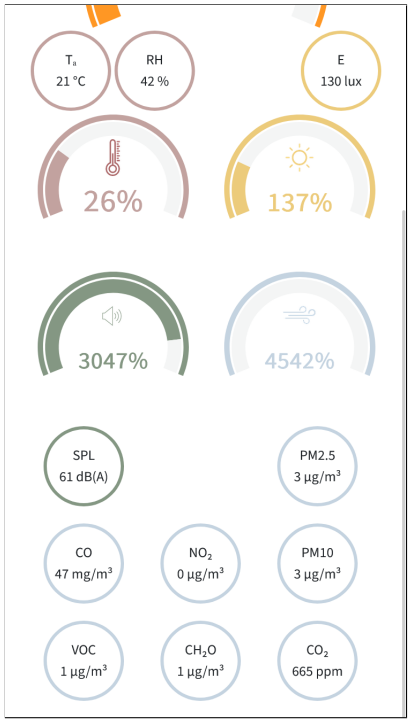



Figure 7.4: The gauges view (portrait) 2

As always we will start from top to bottom. The top of the view contains the header bar, which contains the title of the application and a couple of icons that acts as navigators. The right side of the bar, namely a "house" icon and a "person" icon, will redirect the user to the home page and the profile page respectively. The logo itself, acts as a "reset" button for the current view, which means upon each tap, the button will reset all the states in the view, including view states (e.g. redirecting back to gauge view from graph view), and the data states (e.g. resetting the values of the filters).

The candidate omitted the icon on left side deliberately, as it is a part of the offscreen sidebar component, which we will discuss in the sidebar section 7.3.

```
body.mobile-view
    // Main view
    #container-dashboard
        // Main container
        #row-nav
            max-width:  100vw
            margin:     0
```

By forcing `max-width: 100vw`, we ensure that the header bar will not overflow the screen width.

Right under the header bar, there is the button group for the time range filter. Its functionality is to let user choose the time range that will be used to filter out data. All the buttons in the button group share the class name `.btn-time`.

```
// Time range buttons (Container)
body.mobile-view
  #container-dashboard
    .col-btn-time
      display:          flex
      justify-content:  center
      padding:          0

    // Time range buttons (Button)
    .btn-time
      min-width:        12vw
      padding:          0
```

The candidate fixes a `min-width` to avoid the buttons becoming too narrow in width and thus difficult to interact with.

As we demonstrated in Fig. 7.3, the gauges in the view are divided into two classes. One is the single IEQ gauge, which indicates an overall rating of IEQ indices, and a group of four sub-gauges that represent corresponding values in detail. Firstly, we will look into the IEQ gauge.

```
body.mobile-view
  #container-dashboard
    // The main IEQ gauge(s)
    #gaugeIEQ
        // Svg container
      > .gauge-component-class
        max-width:  100vw !important
        // Actual svg
        // The values are to reflect a relation in ratio
        > svg
          width:    66vw
          height:   44vw
```

Due to the fact that IEQ gauge is the most important gauge in the view, it is logical that it takes most space from the view. The actual svg element is set to a fixed width and height, which has a ratio of 3:2. This ratio is chosen to ensure that the gauge will appear decent while takes as less space as possible.

In the next row, we can see three unequally distributed sub-gauges, which are the `Temperature`, `Humidity` and `Visual` gauges. The reason of this distribution is to form a visual hierarchy, as one can see the icons from left and right side have different `stroke-color`, meaning that some of them belong to one category while others from another. This line is invisible by default, it is only when the corresponding gauge or the IEQ gauge is tapped does the row appear.

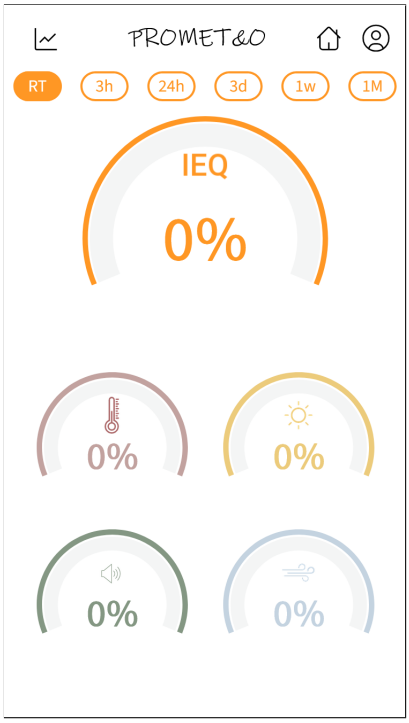



Figure 7.5: gauge-port-subgauges-invisible

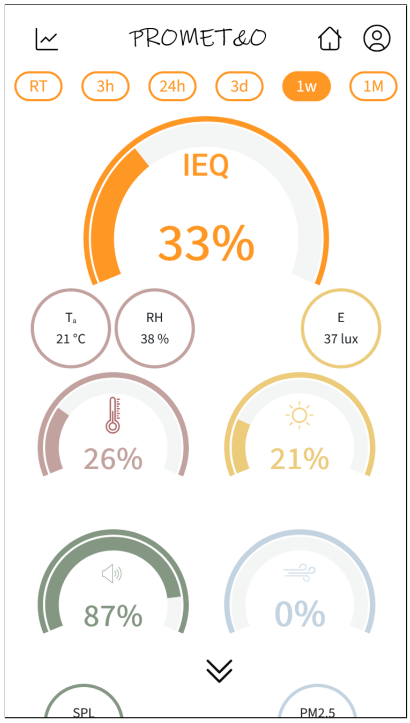



Figure 7.6: gauge-port-subgauges-visible

As one can see, in this specific row the view has three sub-gauges in total, which is quite ideal for a "gauge-gauge-divider-gauge" distribution, and this is what the candidate has chosen in the JSX file. Note that all JSX components that are demonstrated in this section are simplified by removing all properties for the sake of readability.

```
<Row className="row-sub-tmp-lgt">
  <Col className="d-flex justify-content-start">
    <Holder />
     
    <Holder />
  </Col>
  <Col className="d-flex justify-content-end">
    <Holder />
  </Col>
</Row>
```

By using the `Row` and `Col` we can construct a container-content layout. And naturally, `justify-content-start` and `justify-content-end` will align the content to the left and right side of the container respectively, dividing the columns naturally. Moreover, inside the left side column, the candidate added an extra ` ` to create a space between the two sub-gauges, we definitely have alternative methods for this purpose, but this is the most straightforward way to do it.

Major gauges except the IEQ one are designed to be smaller than the IEQ gauge, the rest are almost identical.

```
body.mobile-view
  #container-dashboard
    div.subGauge
      svg
        width: clamp(125px, 40vw, 250px)
```

The candidate simply adjusts the `width` of the `svg` element to be a value between `125px` and `250px`, depending on the screen width. This way, it will generate a moderate size of each sub gauge, the height will be automatically adjusted to maintain the aspect ratio of the gauge.

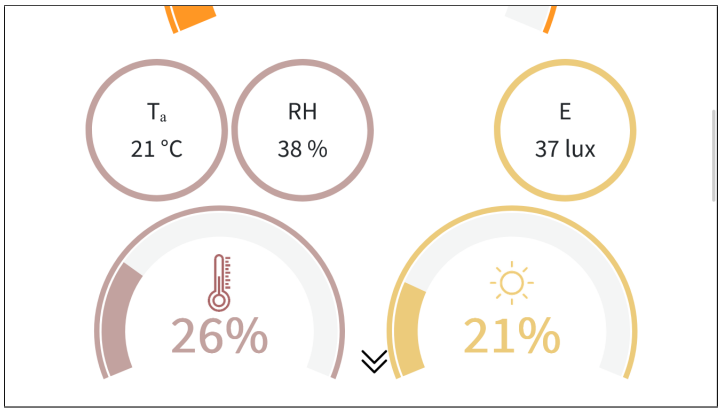



Figure 7.7: gauge-land-subgauges-visible

The style sheets used here is shown below, first the `Row` has a `flex-nowrap` property, which means that the columns will not wrap around (break) when the screen width is not enough. The `justify-content-start` and `justify-content-end` will make the gauges close to each other.

```
<Row className="row-tmp-lgt d-flex justify-content-start flex-nowrap">
  <Col className="d-flex justify-content-end">
    <HolderGauge />
  </Col>
  <Col className="d-flex justify-content-start">
    <HolderGauge />
  </Col>
</Row>
<Row className="row-snd-air d-flex justify-content-start flex-nowrap">
  <Col className="d-flex justify-content-end">
    <HolderGauge />
  </Col>
  <Col className="d-flex justify-content-start">
    <HolderGauge />
  </Col>
</Row>
```

The sub-gauges for $CO_2$ and `Noise` are placed in the next row, each of them will have

three equally distributed sub-gauges except the first row. The same principle applies here, the row is invisible by default, and only appears when the corresponding gauge is tapped.

```
<Row
  style={{
    padding:        "0",
    display:        expand ? "flex" : "none",
    justifyContent: "center"
}}>
  <Row className="row-sub-snd-air">
    <Col className="d-grid justify-content-start">
      <Holder />
    </Col>
    <Col />
    <Col className="d-grid justify-content-start">
      <Holder />
    </Col>
  </Row>
  //...
</Row>
```

The line `display: expand ? "flex" : "none"` will let the react state `expand` control the visibility of the whole container. Notice that, different from all the rest rows, the `Col` here has a `d-grid` property, which is a shorthand for `display: grid`, instead of `display: flex`. This will align the content on both side to their further end when used in combination with `justify-content-start`. Below is a general JSX component for the rest rows.

```
<Row className="row-sub-snd-air">
  <Col className="d-flex justify-content-start ">
    <Holder />
  </Col>
  <Col className="d-flex justify-content-start ">
    <Holder />
  </Col>
  <Col className="d-flex justify-content-start ">
    <Holder />
  </Col>
</Row>
```

All above illustrates the layout and relevant styles sheets applied on the gauge view. The candidate believes that it has successfully presented a clear visual hierarchy and is effective on displaying the data.

### 7.2.1 Notification on scrolling

One might have noticed that in the pictures there is an indicator for users to notify them when scrolling down is possible. This is an independent component that is defined exclusively for the dashboard.

```
const ScrollArrow = props => {
  const [visible, setVisible] = useState(false);

  useEffect(() => {
    const checkScrollNeeded = () => {
      const atBottom = window.innerHeight + window.scrollY >=
      ↪  document.documentElement.scrollHeight - 1;
      const scrollNeeded = document.documentElement.scrollHeight >
      ↪  window.innerHeight;
      setVisible(!atBottom && scrollNeeded);
      props.setShowArrow(!atBottom && scrollNeeded);
    };

    const handleScroll = () => checkScrollNeeded();

    const resizeObserver = new ResizeObserver(() => checkScrollNeeded());
    resizeObserver.observe(document.documentElement);

    window.addEventListener("scroll", handleScroll);

    return () => {
      resizeObserver.disconnect();
      window.removeEventListener("scroll", handleScroll);
    };
  }, []);
  // ...
}
```

The `useEffect` hook is used to add event listeners to the window object to check if the user has scrolled to the bottom of the page. If the user has not scrolled to the bottom of the page, the scroll arrow is displayed. This is achieved by comparing the height of the document to the height of the window. The boolean value `atBottom` is true if the `scrollY` value is greater than or equal to the height of the document minus the height of the window. `scrollNeeded` is true if the height of the document is greater than the height of the window. The scroll arrow is displayed if `atBottom` is false and `scrollNeeded` is true.

`ResizeObserver` is used to observe the document element and check if the user has scrolled to the bottom of the page when the window is resized. The `handleScroll` function is called when the user scrolls the page, it calls the `checkScrollNeeded` function to check if the user has scrolled to the bottom of the page. The `handleScroll` function is then added as an event listener to the window object.

Now we can look at the style sheet for the scroll arrow.

```
const styles = {
  arrow: {
    position:     "fixed",
    bottom:       "20px",
    left:         "50%",
    transform:    "translateX(-50%)",
```

```
    cursor:       "pointer",
    animation:    "waveShowHide 1s infinite",
    fontSize:     "2rem",
    color:        "#333",
    opacity:      1,
    transition:   "opacity 0.3s ease-in-out"
  }
};
```

where `waveShowHide` is a piece of animation that is defined as:

```
@keyframes waveShowHide
  0%
    opacity:    0
    transform:  translateY(-1em)
  25%
    opacity:    1
    transform:  translateY(0)
  75%
    opacity:    1
    transform:  translateY(0)
  100%
    opacity:    0
    transform:  translateY(1em)

.blink
  animation:    blink 1s infinite
```

In combination they will form a animation that can inform the user that the view can be scrolled down. It offers advantages for both aesthetics and functionality.

## 7.3 Off-canvas sidebar

In the previous section, the candidate briefly mentioned the design on the navigation and the entry point placed on the header bar. In this section, we will discuss the sidebar in detail.

The sidebar is a crucial part of the dashboard, as it contains the navigation alongside with assistant features such as filter groups to different parts of the application. The sidebar is designed to be an off-canvas sidebar, which means that it is hidden by default and can be toggled by tapping on a button. This design choice is made to save screen space and to make the dashboard look cleaner. The candidate also pays attention on the aesthetics and maintainability aspect.

If the user tap the left-side icon on header bar, the sidebar will slide in from the left side of the screen. From the gauge view, the sidebar will provide explanatory information on the current selected IEQ index. If the user did not select any specific index, the sidebar will display the information on IEQ in general.

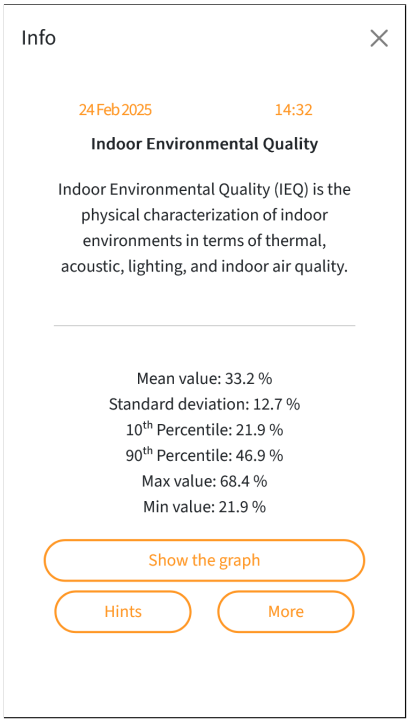



Figure 7.8: Sidebar (IEQ)

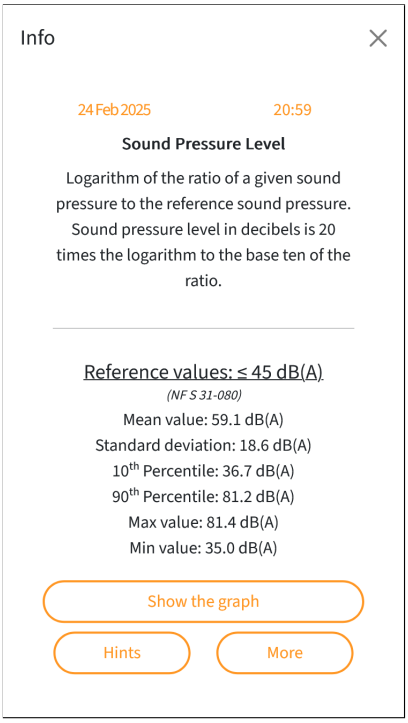



Figure 7.9: Sidebar (sound)

The first row reports the current local time, which could be useful to determine an "offset" with respect to normal values. For example, during the daytime, people in relevant rooms are more likely to experience higher room temperature, higher sound pressure, etc.

Inside the next container we can find the current index and explanatory information about the index. Below it is the reference values attached to each index. They are handpicked before the development of the dashboard, the user can find the source of those values in the display also.

The interactive part consists of a button that will lead the user to the graph view with the currently selected topic and time range, and a pair of buttons namely "Hints" and "More". The user can also find the button to close the sidebar.

The "Hints" and "More" button, upon tapping, will raise a modal dialogue box that contains more detailed information about the current index. The "Hints" button will provide the user with some tips on how to improve the current index, while the "More" button will provide the user with more detailed information about the index.

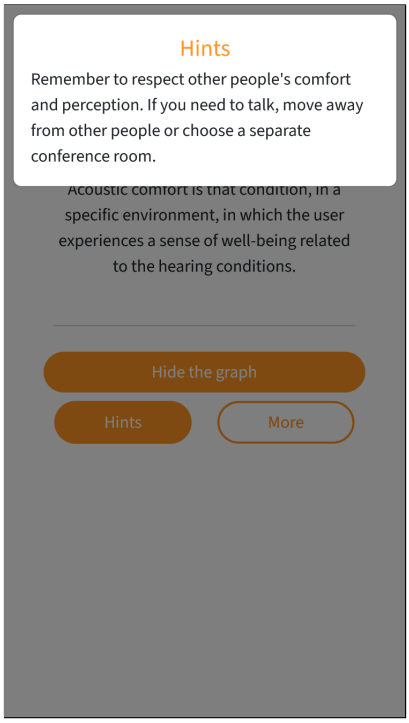



Figure 7.10: "Hints" button dialogue

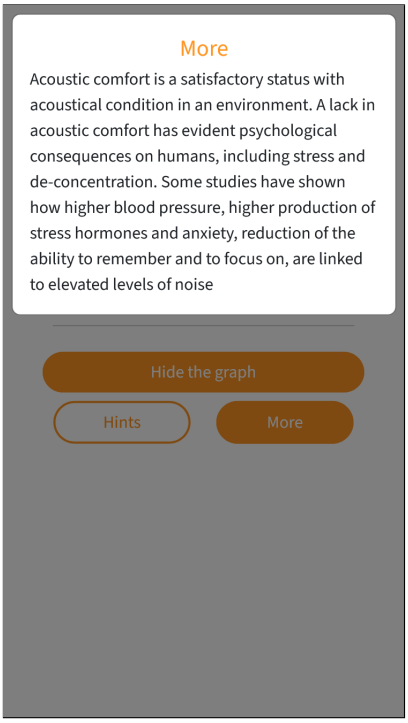



Figure 7.11: "More" button dialogue

The style sheets for modal dialogues in this view is also under the effects of `.modal-content` in 3.1.

```
body.mobile-view
    #div-offcanvas-body
        max-width:              90vw
        padding:                1em
        border-radius:          2em 2em 2em 2em

        // Buttons in general
        button:not(.accordion-button)
            max-width:          clamp(18em, 80vw, 20em)

        // Accordion buttons that are not collapsed
        .accordion-button:not(.collapsed)
            color:              #fff
            background-color:   $color-theme

        // Filter buttons by topics
        .col-btn-topic
            button
                font-size:      getRelativeFontSize(16) + em !important
                margin-bottom:  0.5em
                width:          100%
                padding-left:   0
```

```
            padding-right:  0

    // Container for filter buttons by time range
    .col-btn-time-offcanvas
        display:            flex
        justify-content:    center
        margin:             1%

        // buttons
        button
            font-size:      getRelativeFontSize(16) + em
            height:         clamp(2em, 2.25em, 10vh)
            width:          clamp(6em, 10vw, 70vw)
```

where specific values such as `font-size` are obtained through experiments.

## 7.4 Graph display

Another significant aspect about the sidebar is the navigation part of it. From the gauge view to graph view, from graph display to graph comparison, the sidebar is the main, if not the only entry point for the navigation.

We introduced the information section and the button group of "Hints" and "More" in section 7.3, in between of those two sections there is a button that will lead the user to the graph view with the currently selected topic and time range named "Show the graph".

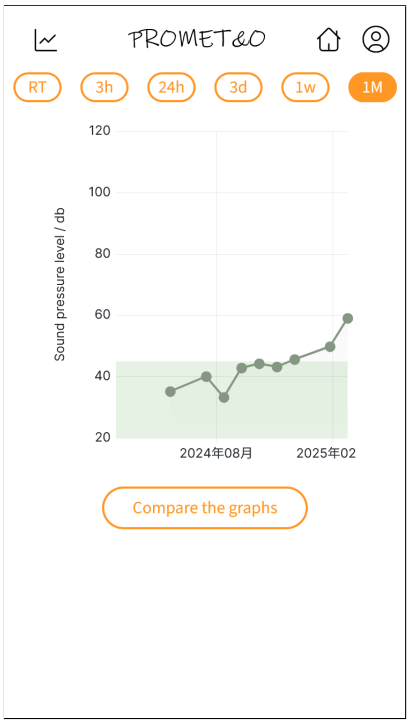



Figure 7.12: Graph display (sound)

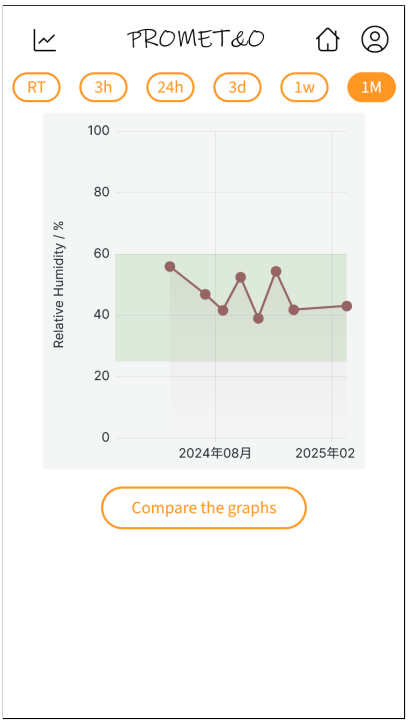



Figure 7.13: Graph display (humidity)

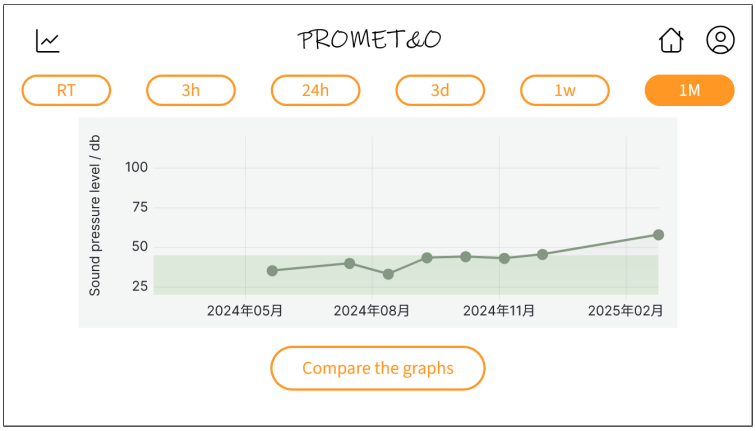



Figure 7.14: Graph display (sound pressure) - landscape mode

The candidate believes the layout is self-explanatory enough. So the candidate will directly jump into the SASS file.

```
body.mobile-view
  #container-dashboard
    #graphBox
      > iframe
        height:     50vh
        width:      80vw
        border:     zero()
        padding:    zero()
```

From the SASS lines, we can have a direct look on restrictions imposed on the scale of each graph. Note that this piece of code block is only applied to a single graph view (as the `>` selector only selects direct descendants), whereas the style sheets for graph comparison will eventually replace some of the properties, we will discuss them in the following sections.

The `height` and `width` are defined as 50% and 80% of the height and width respectively, where both values are subject to a personal preference of the candidate.

## 7.5 Graph comparison

### 7.5.1 The initial view

The graph comparison feature is one of the most stellar features of the dashboard. It allows the user to compare different graphs side by side, which is particularly useful when the user wants to compare the data of combinations from different topics and different time ranges.

When the user is in the graph view, under the frame for graph display, there will be a significant button that invites the user to compare the graph with another one. The button is named "Compare the graph".

After the user tapping the button, it will show the same graph the user previously saw in the display view, possibly with a different size, as the filter has not been changed yet. The user can then change the filter either through selecting the button group under the header bar, or tapping the icon on the same place where the original sidebar trigger button was. Now the icon will still trigger the sidebar, but content of the sidebar will be different than earlier, as there will be a filter group, with the button to exit the comparison mode.

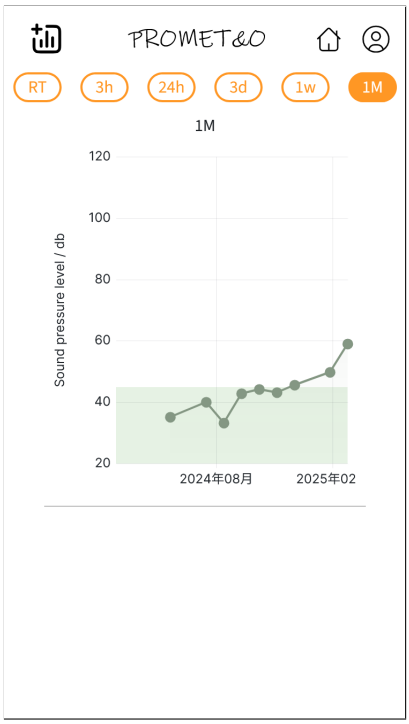



Figure 7.15: Graph comparison (sound)

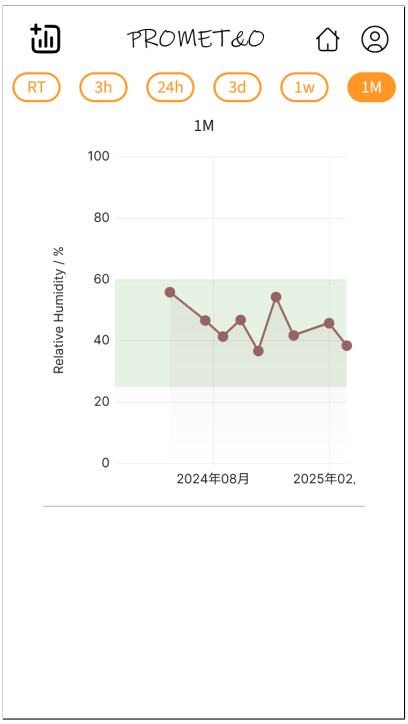



Figure 7.16: Graph comparison (humidity)

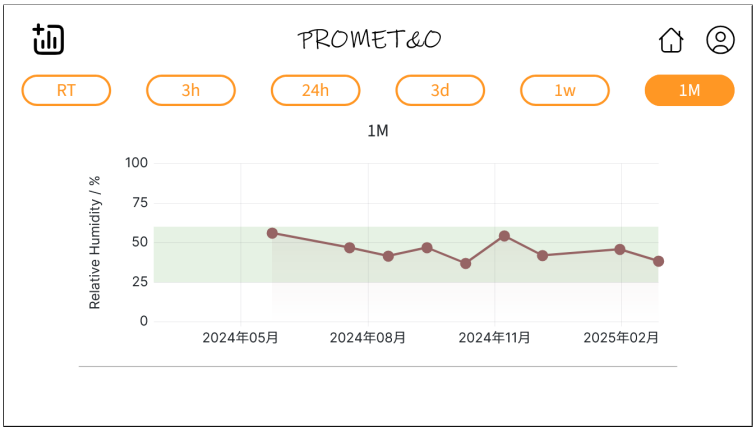



Figure 7.17: Graph comparison - landscape mode

At the first sight the user might find the comparison view is almost identical to the display view. For the sake of the comparison, the candidate will highlight here the main differences, namely:

- The filter tag
  - There will be a filter tag on the top of the graph, indicating the current filter settings. The tag is clickable, and will trigger the filter sidebar. In Fig. 7.15, it presents as the "1 M", referring to "1 Month", over the diagram.
- The divider stroke
  - The divider stroke is a vertical line that separates the two graphs. It is a thin line with a color that is slightly darker than the background color of the graph.

Those differences are illustrated by the JavaScript code below:

```
<Container style={{ padding: "0" }}>
  {graphs.map((graph, index) => {
    return (
      <Row key={index} className="d-grid justify-content-center">
        // The filter tag
        <p style={{ margin: 0, textAlign: "center" }}>
          {time.length >= index + 1 ? time[index] : null}
        </p>
        <Row className="row-graph m-0 p-0">{graph}</Row>
        // The divider stroke
        <Row className="mx-0" style={{ borderBottom: "1px solid #bbbbbb",
        ↪  marginTop: "10px" }} />
      </Row>
    );
  })}
</Container>
```

By using a `grid` display the view enforce an manageable layout for graphs. The `time` is of type `array` and contains the selected time ranges, it is used to display the time range of the graphs in the filter tag. The line there is used to avoid data leaks where the content in `time` and `graphs` do not match. The divider stroke, as one can see, is a single pixel thick line with a color of `#bbbbbb`.

### 7.5.2 The filter sidebar

The candidate has talked about the transform of the content in sidebar once the user enters the comparison view. From the perspective of the candidates this is actually never a good practice; it turns the initial knowledge about the sidebar all over and forces the user to learn about the mutation of it. However, the candidate believes that the filter sidebar is a necessary evil, as the user needs a component that includes sufficient view space for various filters.

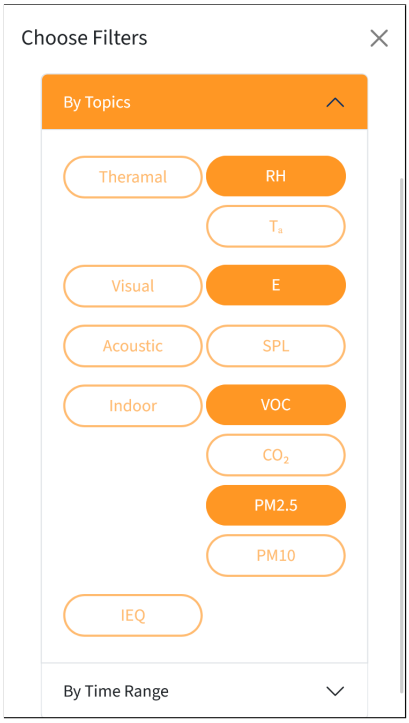



Figure 7.18: Filter sidebar (topic)

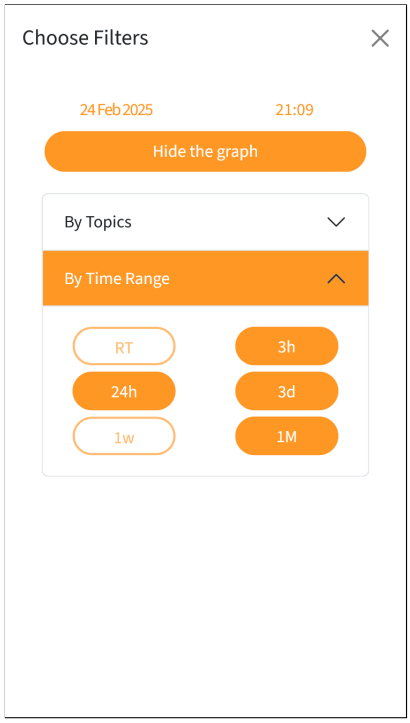



Figure 7.19: Filter sidebar (time range)

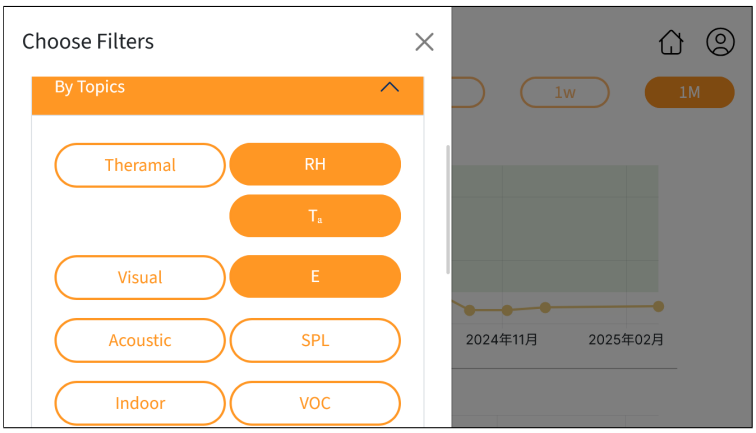



Figure 7.20: Filter sidebar (topic) - landscape

The filter is made possible through the JSX component `Accordion` from `React-Bootstrap`. The `Accordion` component is a container for a set of `Accordion.Item` components, each of which can be expanded or collapsed.

The `Accordion.Item` component is a container for a collapsible element, with a header and a body. The header is a clickable element that can be used to expand or collapse the body. The body is the content that will be shown or hidden when the header is clicked.

```
const AccordionFilter = props => (
```

```
  <div className="accord-filter justify-content-center" style={{ maxWidth:
  ↪  "100vw", paddingTop: "20px" }}>
    <Accordion defaultActiveKey="0">
      <Accordion.Item
        eventKey="0"
        onEntered={() => {
          props.setCompareTime([]);
        }}
        onExited={() => {
          props.setDisableTopics([]);
        }}>
        <Accordion.Header>By Topics</Accordion.Header>
        <Accordion.Body>
          <Container fluid style={{ padding: "0" }}>
            <CompareGraphsListPortrait
              measure={props.measure} />
          </Container>
        </Accordion.Body>
      </Accordion.Item>
      <Accordion.Item
        eventKey="1"
        onEntered={() => {
          props.setDisableTopics([]);
        }}
        onExited={() => {
          props.setCompareTime([]);
        }}>
        <Accordion.Header>By Time Range</Accordion.Header>
        <Accordion.Body className="px-0 d-flex justify-content-center" style={{
        ↪  flexWrap: "wrap" }}>
          {["RT", "3h", "24h", "3d", "1w", "1M"].map(tb => {
            return (
              <TimeButtonPortrait />
            );
          })}
        </Accordion.Body>
      </Accordion.Item>
    </Accordion>
  </div>
);
```

The `AccordionFilter` component is a container for two `Accordion.Item` components, one for the topic filter and one for the time range filter.

`Accordion.Item` components are used to create the collapsible elements for the topic and time range filters. Both of them has an `onEntered` and `onExited` event handler that will be called when the element is expanded or collapsed; the `onEntered` event handler will reset the time range filter when the topic filter is expanded, and `onExited` event handler will reset the topic filter when the time range filter is collapsed, and vice versa. Finally, the `CompareGraphsListPortrait` component is used to create the list of topics for the topic filter.

Note that, by external requirement, the maximum number of possible concurrent comparison graphs is four, being it a combination on different time range or different topics, so only two possibilities: four different topics over the same time range, or two different topics with two different time ranges.

### 7.5.3 Graph comparison view

After users choose the topics and time ranges for the graphs and collapsed the sidebar, they can get in the view for graphs they chose. The graphs will be displayed over a grid layout, with the same size and the same scale, like a waterfall.

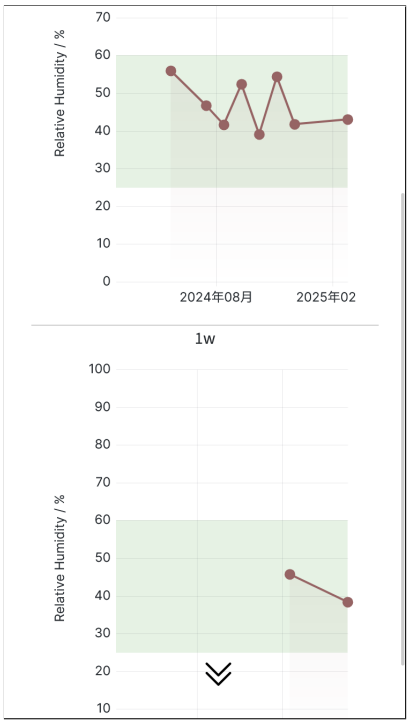



Figure 7.21: Graph comparison (topic)

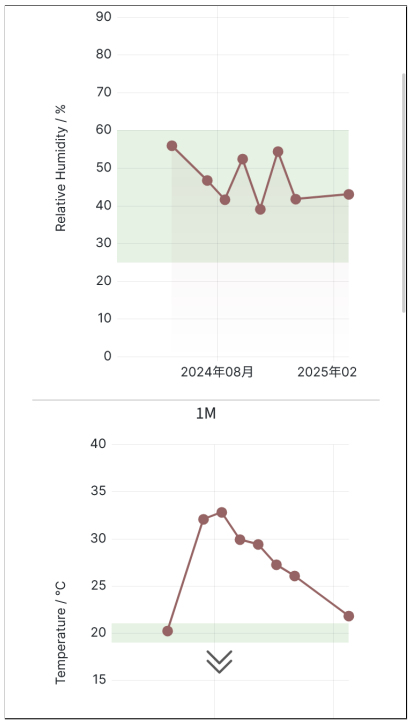



Figure 7.22: Graph comparison (time)

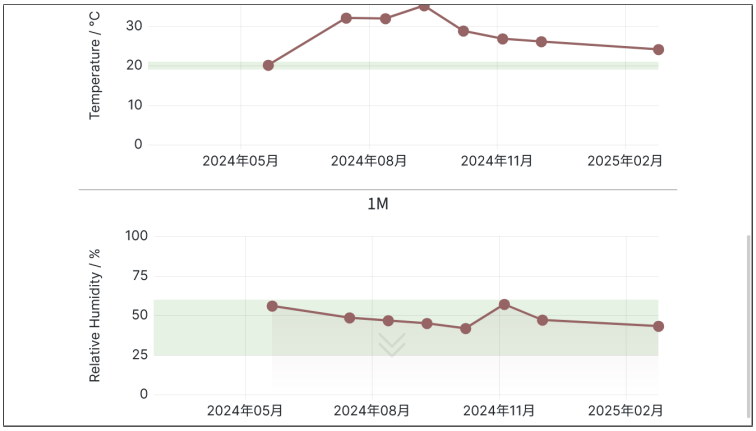



Figure 7.23: Graph comparison - landscape mode

The candidate has to admit this is not the best solution for such a feature, but in terms of maintainability this may come out as the optimal. The user can easily find the graphs they have chosen, and the user can manipulate the order of the graphs by adjusting their order of choice on the filter in the sidebar anytime they tap the icon on the header bar. For programmers, the layout is easy to understand, and can be easily decomposed if they have better ideas.

The candidate has already demonstrated the relevant JavaScript code in the previously on how the graphs are distributed in the waterfall view, and thanks to the responsive nature of CSS and React.JS, the candidate does not need extra style sheets for the landscape mode. However, for the sake of comparison, it needs a proper value set on the size of graphs, in order to allow the user to see at least two graphs in one view. After a few testing, the fitting value for comparison for a balanced configuration between portrait and landscape mode would be:

```
body.mobile-view
  #container-dashboard
    #graphBox
      .row-graph > iframe
        height:     40vh
        width:      90vw
        border:     zero()
        padding:    zero()
```

Reminding that the candidate has demonstrated before, for a single-graph view, the `height` and `width` value is 50% and 80% of the view port height and width respectively. Comparing to the single-graph view, the style sheet takes advantage of more horizontal space than under landscape mode, and the user get to see graphs in richer detail thanks to the more lengthy x-axis.

# Part II

# Reorganizing the back-end: integrating Apache APISIX®

# Chapter 8

# Background

## 8.1 A parallel in real life

Throughout the history of computer network, there existed many solutions on the model of communication between a more distributed computer with a rather centralized system. In most cases, the distributed computers can be named as "clients" while the centralized system called "server".

If one simplifies the communication to an extreme, one may get a loop of "clients send requests to the server, server respond with responses" until it ends. After generations of iterations the communication model has inevitably developed into protocols consist of series of complicated interactions, with both internal and external components participating in it.

Viewers can imagine the need of managing the request/response model of communication, if the candidate is required to provide a parallel, it is like two person talking to each other; this is an extreme case, and the candidate will try to add up other aspects to simulate the communication between server and clients.

The need of highest priority should be asking both person to speak the same language; the parallel of this in the context of network communication is requiring both ends to use the same protocol, preferably the same version, like one may experience obstacles when communicating with a time traveler who speaks the same language from three hundred years ago, the difficulty may come from the fact that the language itself has evolved, also the context for it has shifted rapidly. So it is obvious that communications need uniformed format and context.

Secondly, arguably is the security aspect of the communication. Depending on the context, it is very common that the user has polar opposite requirements on different types or context of the communication, but ultimately, the user needs containable, or even better, controllable security properties. A person may go from having a secret conversation about what he is hiding in his closet, to giving a speech on television to millions of people, as long as the person is aware of it. In computer networks, it should be among the top priorities for users to acknowledge the current state of security aspect of the communication.

If the communication is happening in a public office where people come from all over

the city to deal with official documents, then definitely a more complicated system is needed for management. Further requirements may include a ticket system that puts all requester in a order based on specific properties such as the date time when the appointment was made, security staff at the door that keep both the communications and people safe inside, officials at the door to deal with simpler daily tasks such as checking when documents will be ready, answering if the staff inside are on lunch break, or tell people that they need to go somewhere else, the list goes on.

The viewer may have guessed where the candidate is going to. If it is possible to make a parallel from public office to the server, then the system that is in charge of such management can also be found in the computer network world, for example, an API gateway. The candidate will introduce the concept in the following section.

## 8.2 API gateway

It is difficult, at least to the candidate, to suddenly prompt a solid definition on what an API gateway is. For its abstract nature, the candidate will continue to use the parallel example he used in section 8.1 to illustrate the concept.

A public service office is usually busy all the time, which makes the management on incoming guests a more crucial component among them all. More specifically, it might as well be a post office. It is in charge of sending and receiving a huge amount of paper mails and packets every single day, alongside banking services.

Every viewer might does not have experience in working in a post office but definitely has visited one in their life. The first thing they need in front of the post office is a ticket which confirms the guest's presence and his position in a line if there is one, and the management system is in charge of separating the traffic by types of services, and maintain a flexible schedule to distribute tasks to each desk. When a ticket gets printed, both the guest and the system will be aware of an approximate time slot for engaging the service. Actually, if the office has a line that is so long the guests cannot even see the end of it, then the guest is encouraged by staff at the door to look for another post office in the neighborhood.

When the guest is called to the desk, he will state his purpose to the attendant and optionally show relevant identity documents. The attendant will check for user information stored in the system, and get all necessary information needed for the service. If the guest is new to the postal office, the attendant can still manage the service in anonymous at a cost of efficiency.

The guests may want to send a mail, the attendant acknowledges this and will inspect the envelop. What the attendant will do is looking for specific information on specific position on the envelop. There might be situations where the guest did not provide sufficient information such as the postal code, or wrote it at a wrong position. The attendant will either return the mail to the guest asking for correction, or do the job for him by copying information to the right spot on the envelop. Anyway, the attendant will make sure the correct information are located in correct positions. Once all information are correctly filled, the attendant will put the mail in a queue or a box, waiting for further management, and wave the guest goodbye to conclude the interaction.

The real life example more or less describes the role of an API gateway during the communication between a host and a server. It reorders the incoming requests, possibly redirect them for load balancing, managing authentication and authorization process, and help forwarding requests by populating necessary blanks based on information the requests is carrying. Here the procedure is not meant to be authenticate, but to provide a simple glimpse on the concept.

### 8.2.1 Apache APISIX®

Apache APISIX® is a high-performance cloud native API gateway for API requests and microservices. [1]. Apache APISIX® provides state-of-the-art features for API gateway, thus introducing it has a huge potential in managing the RESTful API calls towards the PROMET&O server. It includes two main components, namely `Nginx` and `etcd`, they serves as reverse proxy and control center respectively.

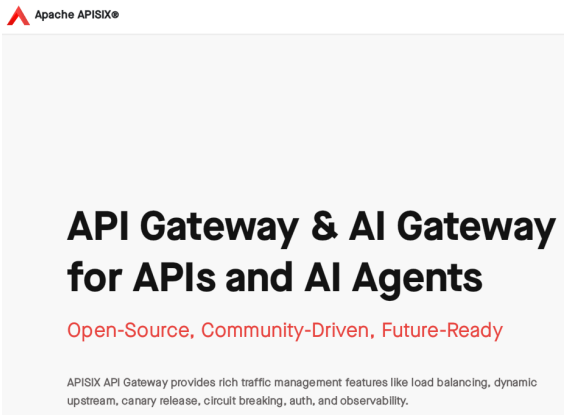



Figure 8.1: APISIX

One significant feature that is worth mentioning is the "hot configuration". Apache APISIX® supports configuration through both a local file, typically in `.yaml` format, or in a way that is called "hot configuration", which means administrators are able to configure the routes through API calls to administrative API endpoints without restarting the server. It credits to the `etcd` component that is part of Apache APISIX®, the topic of that is not much relevant to this thesis so the candidate will not dig deep into it. In conclusion, hot configuration provides great efficiency comparing to traditional maintenance procedure, where administrators typically need to modify local files and restart the server for the modifications to take effect.

Another stellar advantage of Apache APISIX® is its library of plugins, which everyone may would not like to call rich, but is adequate for majority of common operations over an API gateway. For example, its `redirect` plugin supports overwriting contents in the header from a HTTP request, the `openid-connect` plugin will hugely improve the communication efficiency for communicating with a regular authentication service using its prebuilt configurations based on openid-connect protocol. The list goes on.

Other advantages including high performance and low latency, wide support on current generation communication protocols like MQTT and WebSocket, and integrated observability, are not so obvious to regular users but indeed makes a difference.

The motivation for integrating Apache APISIX® into PROMET&O is also an insight for the future development. Previously, relevant colleagues have finished a magnificent job on current architecture. However, as the development goes on, the technical debt gets increased inevitably along the way, as `Nginx` may reveal to be less than sufficient for development purposes – features such as custom plugins and load balancer are all those who will gradually show their value for the long run. And, since APISIX includes `Nginx` as a part of it, it will not increase current technical debt.

In the next section 8.3 the candidate will introduce the structure of the architecture of PROMET&O and analyze potential risks.

## 8.3 Current state of the system

Right now, the architecture of PROMET&O is a regular client-server model. Currently there exists three different name spaces standing for three different system configured on different locations. Each name space can be divided into four parts in terms of functionality: the client, the back-end, the microservices, in addition to an external component Nginx as a reverse proxy.

The client, self-explanatory enough, is the single-page application created using React.JS and relevant tools that the candidate has participated in developing. It serves as the single user interface that users are supposed to interact with, whereas other services such as the authentication, provided by Keycloak, also exposes its ports in certain situations but merely temporarily. It is hosted on a simple server which the candidate calls "internal server".

The back-end is a simple server developed using `Express.JS` that is in charge of communicating user requests with all the microservices, and external data sources through the MTTQ protocol. In contrary to the "internal server", the candidate will refer to this server as "external server", whereas "external" indicates an "external layer" comparing to the internal one.

Next the candidate will list all microservices that currently resides in each name space:

- Authentication
  - The authentication service is provided by Keycloak
- Data source
  - Grafana is in charge of the accessing external data sources and transmitting data stream
- Database
  - The database for storing user feedback is written by MySQL

Those mere three microservices have composed adequate and well-functioned application for user needs.

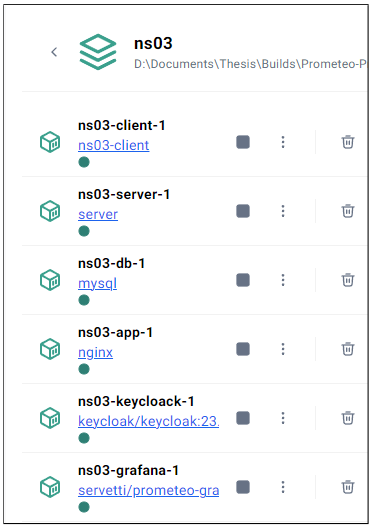



Figure 8.2: Original container using Nginx

The candidate will share some of his thoughts on possible improvement on current system structure.

Firstly, and in general, the candidate argues that the container needs proper decoupling. Taking the matter to an extreme, if someone who is not an expert in the web development field takes a look at this component list, he might have a hard time understanding what purpose each container serves.

Also, originally the microservices inside each `docker compose` are dependent on each other, which can be a subject to an argue; Shown in Fig. 8.3, it is not reasonable to prevent `client` from booting up simply because `db` is not starting properly. Doubts like this are among the top reasons of proposing an evaluation on current architecture.

Secondly, despite having nonidentical data sources in the back-end, the front-end user interface of each name spaces are majorly similar from each other. What's more, it actually does not make sense to have `client` coupled with other microservices both in terms of the perspective from the user and the developer. The candidate believes reason above are sufficient for a reorganization on the front-end component specifically.

Thirdly, it does make sense that each name space has its own authentication service dedicated to itself, but considering the maintainability on the service, and the realm feature that Keycloak expertise on, it would be more beneficial lifting the Keycloak service up to the same level as the API gateway, i.e. a centralized authentication service for all name spaces. The concept of realms will be further explained in section 10.1.

Above are the major points that can potentially benefit the system in terms of various aspects. For example, lifting both authentication service and the user interface will have

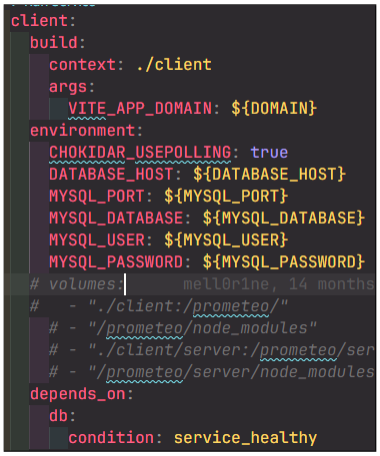



Figure 8.3: The dependency on db from client

great positive impact on maintainability and space resources at the cost of moderate development time.

# Chapter 9

# The new architecture

## 9.1 An overview on the new architecture

The candidate has shown a list of components within each individual name space in the previous section 8.3, as he discussed in that section, the first potential improvement brought upon by Apache APISIX® is a reconstruction of current architecture, which is the primary goal of this chapter. The candidate will follow a slowly progressing procedure and demonstrate the process of transformation.

The candidate briefly introduced the current architecture of PROMET&O. It can be summarized as Fig. 9.1.

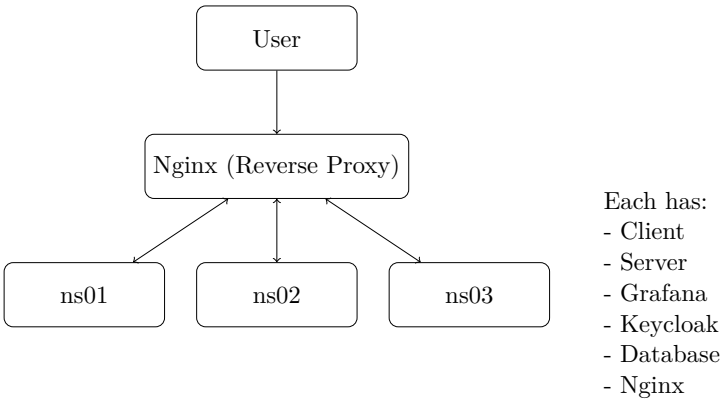



Figure 9.1: The old architecture

Like the candidate has mentioned before, some microservices can benefit the structure by getting lift up to the top level, namely client and Keycloak. This can be realized by simply separating the original `docker-compose.yml` file under each name space, and distributed them in their new locations. For example, the new client may have its `docker-compose.yml` file written like:

```
services:
  client:
```

```
    build:
      context:              ./
      args:
        VITE_APP_DOMAIN:    ${DOMAIN}
    environment:
      CHOKIDAR_USEPOLLING:  true
      DATABASE_HOST:        ${DATABASE_HOST}
      MYSQL_PORT:           ${MYSQL_PORT}
      MYSQL_DATABASE:       ${MYSQL_DATABASE}
      MYSQL_USER:           ${MYSQL_USER}
      MYSQL_PASSWORD:       ${MYSQL_PASSWORD}
    restart:                on-failure
    networks:
      apisix_apisix:
networks:
  apisix_apisix:
    external:               true
```

The highlight here is the newly added `network` property that connects this docker container with an external network that is created in another `docker compose` named `apisix`, this ensures connectivity between microservices in different `docker composes` to simulate domain resolution through `DNS` servers. Note that the container does not expose any port, as other `docker composes` can communicate with each other without exposing any port.

After the relocation, the general structure of each name space can be transformed into Fig. 9.2. Once everything is properly placed, then as the viewer can see, the work flow is already less confusing than the original architecture where client somehow resides alongside with back-end servers.

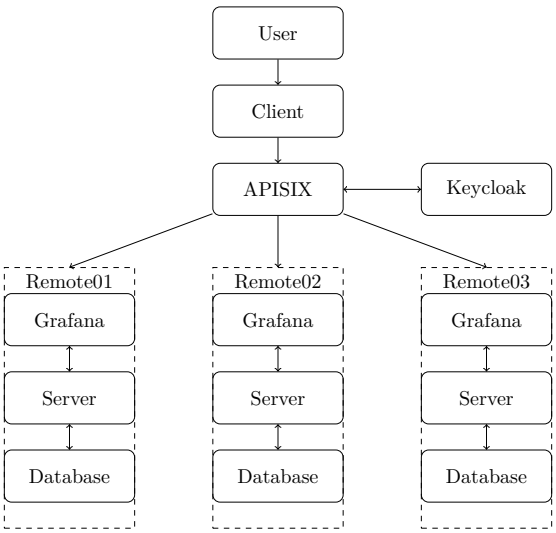



Figure 9.2: The new architecture

Taking an HTTPS request as an example, once the user before the client sends HTTPS requests to the remote server, and the request will pass through Apache APISIX® first, it performs the authentication and authorization by communicating with Keycloak using plain HTTP, then determines how to respond to the user.

## 9.2 Integrating the API gateway

Basically speaking, Apache APISIX® mainly performs according to the configurations on various routes. A **route** in the context of Apache APISIX® can be considered a rule set where it constitutes a specific way to handle requests based on its metadata like `host`, `uri`, etc. For example, a route can mandate requests from **https://localhost:443** requesting `GET /api` to the **http://backend:3001**.

The candidate deployed Apache APISIX® as an independent docker compose using resources obtained from its official website. The `docker-compose.yml` file of it is shown below:

```
services:
    apisix:
        image: apache/apisix:latest
        restart: on-failure
        volumes:
            - ./apisix_conf/config.yaml:
                /usr/local/apisix/conf/config.yaml:ro
            - ./apisix_conf/apisix.yaml:
                /usr/local/apisix/conf/apisix.yaml:ro
            - ./conf/cert/server_win.crt:
                /usr/local/apisix/conf/cert/server_win.crt:ro
            - ./conf/cert/server_win.key:
                /usr/local/apisix/conf/cert/server_win.key:ro
        depends_on:
            - etcd
        ports:
            - "9180:9180/tcp"   #admin port
            - "9080:9080/tcp"   #default HTTP port
            - "9091:9091/tcp"   #prometheus
            - "9443:9443/tcp"   #default HTTPS port
            - "9092:9092/tcp"   #prometheus
            - "443:443/tcp"     #HTTPS port in use
        networks:
            apisix:
```

Here the candidate mounted a pair of self-signed certificate and the corresponding secret key file, as the users are supposed to communicate with API gateway through the HTTPS protocol only. The file also properly exposes relevant ports including ones for requests and ones for administration. The candidate chooses to expose port 443 as it is the default port number used by HTTPS protocol thus simplify configurations in future development, but choosing other ports will not affect its functionalities.

When introducing Apache APISIX®, the candidate has highlighted its "hot configuration" feature, it proves to be a better method in general comparing to static configuration through modifying `.yaml` files, since during development the candidate will engage in a lot of back-and-forth when experimenting the configuration, in the final stage of the thesis, the candidate will export all dynamic settings into static file for consistent storage.

Now the candidate will first configure the route to the homepage, a simple piece of configuration is performed:

```
{
    "id": "entry",
    "uri": "/*",
    "upstream": {
        "nodes": {
            "client:5000": 1
        }
    }
}
```

Taking it as an simple example the candidate can explain the basic structure of a route configuration request made to the admin control API of Apache APISIX®. The mandatory components of a configuration like this are:

- `id`
  - It is the sole identifier for every route, the administrator needs it for future management on the route defined.
- `uri`
  - It is used to match the incoming request. For example, for a request that is a `GET /api`, the settings defined for `/*` will take effect. It generally follows a "best-match" principle for multiple matches.
- `upstream`
  - It tells the gateway where this request should be forwarded to. For example, all requests relating to authentication should be forwarded to the upstream defined for Keycloak. It is possible to define multiple up-streams for a single route, thus achieving load balancing.

There are also optional fields such as `plugins` that will provide further help. In the following sections, plugins such as `serverless-pre-function` will prove to be useful for route configuration.

If the configuration is successful, Apache APISIX® will respond with a `200 OK` and the respond body that has been populated by the effective configuration on the domain, like what is shown below:

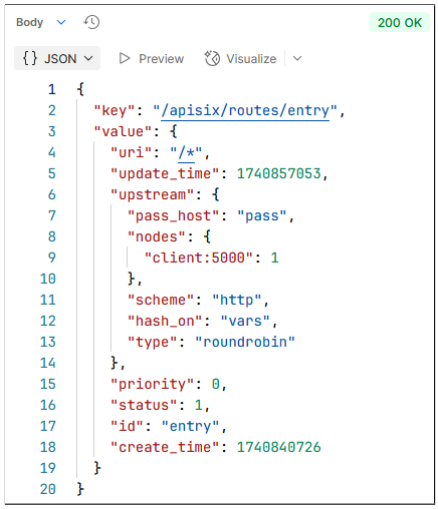



Figure 9.3: Successful configuration on a route

With knowledge on those components, it is obvious now what the candidate is attempting to do here; it simply forward every single request to the client through HTTPS. By default, everything still remains the original behavior; indeed, the web application originally had all its functionalities started from `client:5000`.

After this section, the candidate will start demonstrating how configuration on individual route are applied to each of them. Since the candidate has explained on the basics, he will omit some of the details in later sections and focus more on the purpose or meaning behind each step.

## 9.3 The authentication service

The Keycloak is the sole authentication service provider in our project, and possibly the single point of failure now in the new architecture, in exchange for higher maintainability. By taking advantage of the realm feature, we will have the ability to separate each name space into individual and independent realms, and maintain authentication and authorization for each of them on a centralized manner.

```
keycloak:
    ports:
        # - "8080:8080" # Can expose for debugging
        # - "8443:8443"
    image: keycloak/keycloak:23.0.7
    environment:
        KEYCLOAK_ADMIN: ${KEYCLOAK_ADMIN}
        KEYCLOAK_ADMIN_PASSWORD: ${KEYCLOAK_ADMIN_PASSWORD}
        KC_FEATURES: declarative-user-profile
        KC_PROXY: edge
    volumes:
```

```
        - ./keycloak/realm/.:/opt/keycloak/data/import
        - ./keycloak/plugins/.:/opt/keycloak/providers
        # - ./keycloak/data/:/opt/keycloak/data/h2/
        # - ./keycloak/certs/keycloak.crt:/etc/keycloak/certs/keycloak.crt
        # - ./keycloak/certs/keycloak.key:/etc/keycloak/certs/keycloak.key
    command:
        - start-dev
        - --import-realm
        - --hostname-url=https://${DOMAIN}/
        # - --https-port=8443
        # - --https-certificate-file=/etc/keycloak/certs/keycloak.crt
        # - --https-certificate-key-file=/etc/keycloak/certs/keycloak.key
    restart: unless-stopped
    healthcheck:
        test: "exit 0"
    networks:
        - apisix
```

There were some back-and-forth when determining whether to use HTTPS or HTTP to communicate with the Keycloak, as one can see from the file that the candidate has commented out all lines relating to the HTTPS option. This is mainly because the concept of `SSL` termination.

`SSL` termination means that the API gateway, in this case Apache APISIX®, upon receiving packets through `SSL` tunnel (HTTPS connection), terminates the `SSL` tunnel on the spot, and redirect the message to internal services (also known as "offloading") using plain HTTP connection. This is a common practice in network architecture as the decryption is a resource intensive job, so by dividing the computation load with the gateway it actually benefits performance on the back-end side, one can easily see this improves the immunity to malicious attacks with huge waves of incoming requests like the `DDoS` attack. The view can notice in the route configurations throughout this part that all internal communications will be through regular HTTP.

Once the Keycloak container has been successfully set up, it will automatically import data from local persistent storage thanks to the `--import-realm` option. This file is periodically maintained by the candidate.

The route for accessing Keycloak through Apache APISIX® is shown below.

```
{
    "id": "keycloak",
    "uri": "/keycloak/*",
    "plugins": {
        "proxy-rewrite": {
            "host": "keycloak:8080",
            "regex_uri": [
                "^/keycloak/(.*)",
                "/$1"
            ]
        }
    },
```

```
    "upstream_id": "keycloak-upstream"
}
```

- `host` modifies the host field of the request to `keycloak:8080`, this will for example, changing a request to `ns01-prometeo.polito.it/keycloak/` to `keycloak:8080/`. The main benefit of this is avoiding any potential cross-origin policy as it allows sending all front-end request from a uniform origin. The line below indicates the plain HTTP connection mentioned previously.
- `regex_uri` matches a sub-string of the `URI` and try to replace it with another string. In this case, it will replace the part starting from `/keycloak/` with all the things after it, effectively remove `/keycloak/`. This will correctly constructs the new request. For example, from `ns01-prometeo.polito.it/keycloak/realms/...` to `keycloak:8080/realms/...`. Combining with `host`, it redirects the request to the real internal address.
- `"upstream_id"` is referring to a predefined set of rules to be applied to a certain upstream. In this case, it is referring to one named `keycloak-upstream` . Its content is very self-explanatory. As shown below, it reads "redirecting all requests to `keycloak:8080`".

```
{
    "id": "keycloak-upstream",
    "nodes": {
        "keycloak:8080": 1
    },
    "scheme": "http",
    "type": "roundrobin"
}
```

In some occasions, the requests will inevitably call for `GET /realms` directly without adding the `/keycloak/` part. In those cases, a specific route is defined. Since all requests aimed for `/realms` naturally go to Keycloak, a simple switching on `host` is sufficient.

```
{
    "id": "realms",
    "uri": "/realms/*",
    "plugins": {
        "proxy-rewrite": {
            "host": "keycloak:8080"
        }
    },
    "upstream_id": "keycloak-upstream"
}
```

## 9.4 The Grafana routes

One vital feature for the dashboard is its communication with the data source set on Grafana, so naturally we want to improve the performance on those requests for better

efficiency. The candidate has configured the routes in Apache APISIX® for Grafana, specifically:

```
{
    "id": "chart",
    "uri": "/chart*",
    "plugins": {
        "proxy-rewrite": {
            "host": "grafana:3000",
            "scheme": "http",
            "regex_uri": [
                "^/chart/(.*)",
                "/$1"
            ],
            "headers": {
                "set": {
                    "X-WEBAUTH-USER": "${cookie_User}",
                    "X-WEBAUTH-ROLE": "${cookie_Role}"
                }
            }
        }
    },
    "upstream_id": "grafana-upstream"
}
```

This configuration is similar to the Keycloak one, where:

- `host` modifies the host field of the request to `grafana:3000` avoiding cross-origin issues.
- `regex_uri` removes `/chart/` for the URL.
- `headers` is self-explanatory, indeed it adds new headers to the request for upstream to use. It will be used by Grafana to differentiate user groups.
- `upstream_id` indicates all requests matched by the rule set are redirected to the upstream defined for Grafana.

```
{
    "id": "grafana-upstream",
    "nodes": {
        "grafana:3000": 1
    },
    "scheme": "http",
    "type": "roundrobin"
}
```

The configuration on this route also enables the usage of the built-in dashboard of Grafana for easier maintenance.

## 9.5 Review

In previous sections, the candidate has shown several examples on bridging the front-end container (Client) with the back-end container (Remote) or the authentication service (Keycloak). After correct configuration, the basic work flow of communications through the architecture the candidate has demonstrated can be described by:

- The client sends a request through HTTPS
- Apache APISIX® receives it and offloads
- Apache APISIX® forward the request to the back-end through HTTP

Now the new architecture works just like what the candidate has proposed in section 9.1. But it is easy to see that right now Apache APISIX® is merely a replacement for Nginx as a reverse proxy, and currently its sole mission is to perform the `SSL` termination. In current stage, this is adequate as it demonstrated the viability of the current architecture. In the next section, Apache APISIX® will appear as a powerful tool for processing requests on behalf of the backend and easing the pressure for it.

Despite not reaching its full potential, Apache APISIX® has already displayed several advantages and proves its value in the system, some of them could be:

- Flexibility
  - By comparing the new and old cluster, the viewer can easily observe that unlike the original Nginx reverse proxy, Apache APISIX® does not require an component inside the remote docker-compose, as Apache APISIX® presents itself as a replaceable plugin, which means upon unexpected error in the future development or further iteration over the system, it will be easy to detach it from the architecture.
- Efficiency
  - By utilizing `etcd`, Apache APISIX® provides huge improvement on efficiency by offering the possibility to modify its configuration over a snap of finger. It appears more obvious when doing experimenting on it due to repeat of identical operations.
- Performance
  - Apache APISIX® takes advantage of asynchronous and non-blocking operations, it appears to be more efficient than the event driven mechanism from Nginx in most cases.

On the other hand there are also several disadvantages against Apache APISIX® such as a steeper study curve. For example, one might need to get accustomed to the `lua` programming language to really take advantage on its plugin functionalities. Apache

APISIX® is also more focused on projects that is on larger scale, which PROMET&O has yet to fully become at this stage, while Nginx is more light-weighted but solid. Thus the candidate would never rule out the possibility of removing the gateway or replacing it with its alternatives in the future. When making such comparison and attempting to iterate, it oftentimes tends to be disagreement over philosophy rather than technique and design, but the opportunity on creating major iterations on technologies has also always been an excitement for developers.

# Chapter 10

# Centralizing the authentication service

## 10.1 Introducing realms in Keycloak

Throughout the development of PROMET&O, the candidate and colleagues before him have chosen Keycloak as the authentication service for the system. In Keycloak, the concept of "realm" is core to its management mechanism.

The candidate would like to consider a realm as an "isolated security domain". The keyword here is "isolated"; each authentication realm are supposedly independent to each other, with their own set of rules and configuration, and more importantly user information storage. To define it loosely, each realm can be seen as siblings with similar look from the outside, but can be different internally in terms of personality or mindset. They follow an identical data format, but the content of the data could differ. A pair of configuration of different realms is shown in Fig. 10.1.


Figure 10.1: Realm settings (1)

Realm is ideal for the main goal of this chapter. By setting up different realms for each name spaces, Keycloak could maintain independent configurations and data for each

remote back-ends, combining with upstream settings from Apache APISIX®, the project can have a true-to-the-name independent and centralized authentication service.

## 10.2 The indented workflow

As the candidate finished the introduction to the concept of realm, in this section he will explain the intended workflow of the authentication, and how realms are used to help building it.

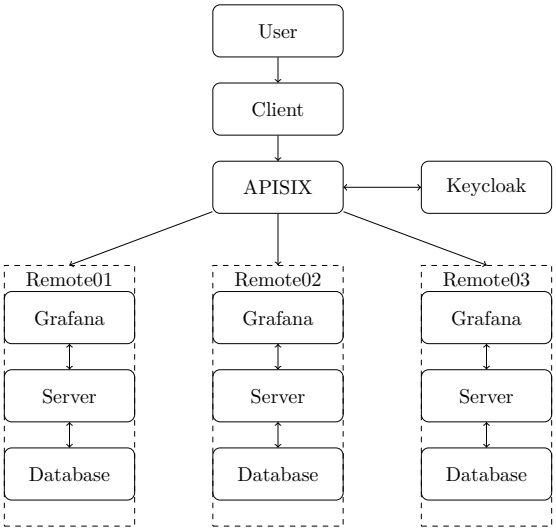



Figure 10.2: The new architecture

Let the candidate roll back to the content from previous chapter, as shown in this diagram, to centralize Keycloak and lift it to the top level of this architecture, it is necessary to have the following conditions.

- Keycloak does not belong to any other clusters, it will be on its own like Client, APISIX and Remote, and eventually does not shared the network with anyone except the gateway.
- Keycloak only contacts or gets contact by other components through the API gateway.

The first point is easy to implement, the candidate can simply move Keycloak to a new docker-compose of its own, with an independent `docker-compose.yaml` file. The second point, though it may not be as easy implementing, is fairly comprehensive; all requests and responds sent from Keycloak is going through the gateway without acknowledging other components, then Apache APISIX® is solely in charge of redirecting them.

To explain the workflow in its full extent, the candidate will make a more detailed example.

- A user access the client.

- The client will send a `GET /userInfo` request for user information through the gateway, Apache APISIX® will handle the request based on whether it has a valid cookie.
  - If the request does not possess a cookie, a `401 Unauthorized` indicates the user is currently anonymous and is able to perform a login.
  - If the user has the cookie, it decodes the cookie and sends the decoded user information back in a `200 OK`, and the user does not need to log in again until logged out.
- An anonymous user can choose to login through interacting with the front-end.
- Keycloak will respond with a self-generated form requesting credentials.
- The user fills the form and sends it back.
- Keycloak authenticates the credentials.
  - If the credentials are legal, Keycloak will send an `authentication code` back to the client.
  - Else, again a `401 Unauthorized` will be sent back.
- The client will use the code to request from Keycloak an `access token` which is a base-64 encoded JSON object. The client will add the tokens in cookies for continuous usage.
  - This is the cookie that the first request is looking for.
  - Upon decoding, the client will have all the user information it needs.
  - If the user access the homepage again he will not be anonymous anymore thus does need to log in again unless logged out.

The basic flow of the authentication is more or less the same as the Authentication Code Flow defined in RFC 6749 [3] with some inevitable tweaks.

## 10.3 The login flow

### 10.3.1 GET /auth/login

As the candidate discussed in section 10.2, when anonymous taps on the login button, the user will be able to claim a form generated by Keycloak. To request such form, it is needed to send a request to the `auth` endpoint on Keycloak, where extra credentials below needed:

- `response_type` should be `code` here to perform the `Authentication Code` flow defined by `OAuth` 2.0 standard.
- `client_id` is fixed as `Grafana` the same way as they are created in each realm.

- `scope` can be a various of values, in this case we will use the default value `openid`.
- `redirect_uri` specifies to Keycloak where it is supposed to redirect the user after successful login claim, in this case, as the flow has a two-step token exchange, it is needed to fill in `https://$host/auth/login/callback`, where `$host` is the built-in variable indicating the host of the request, this uniforms format of requests.

Finally, the configuration used for the flow would be like:

```
{
    "id": "auth-login",
    "uri": "/auth/login",
    "plugins": {
        "serverless-pre-function": {
            "phase": "rewrite",
            "functions": [
                "return function() local h=ngx.var.host or '';
                local ns=h:match('^(.-)-')or'AuthRealm';
                local r='https://'..h..'/auth/login/callback';
                ngx.redirect(
                'https://'..h..
                '/keycloak/realms/'..ns..
                '/protocol/openid-connect/auth?
                    response_type=code&
                    client_id=Grafana&
                    scope=openid&redirect_uri='..r,302);
                ngx.exit(ngx.HTTP_OK)
                end"
            ],
            "priority": 10
        }
    }
}
```

If the host contains a `-`, which holds true for all possible URLs for this project, it fetches the sub-string before it, so the possible outcomes are `ns01`, `ns02` and `ns03` in current stage. Else it takes the value `AuthRealm` that is used for development. In the end it save the string as a header. This is a common procedure shared by all routines for differentiating realms according to name spaces. Then this string is used to redirect to endpoints in different realms.

### 10.3.2 GET /auth/login/callback

At the end of last step, the user will eventually obtain an `authentication code` for further token exchange. It will be carried over to `/auth/login/callback` with a `code` parameter on the query. The token exchange will be performed by the API gateway on behalf of the back-end to reduce pressure.

```
{
    "id": "auth-login-callback",
```

```
    "uri": "/auth/login/callback",
    "plugins": {
        "serverless-pre-function": {
            "phase": "rewrite",
            "functions": [
                "..."
            ]
        }
    }
}
```

Where the inline functions can be expanded to the equivalent one demonstrated below.

```
local http = require("resty.http")
local cjson = require("cjson.safe")

local _M = {
    version = 1.0,
    priority = 1005,
    name = "keycloak-callback",
    schema = {}
}

function _M.rewrite(conf, ctx)
    local host = ngx.var.host or "AuthRealm"

    local realm_name = "AuthRealm"
    local client_secret = os.getenv("keycloakSecretId")
    local prefix = host:match("^([^%-]+)%-")

    if prefix and prefix:lower():sub(1,2) == "ns" then
        realm_name = prefix
        client_secret = os.getenv("keycloakClientSecret"..prefix:upper()) or ""
    end

    local code = ngx.req.get_uri_args().code
    if not code then
        ngx.log(ngx.ERR, "Missing authorization code")
        ngx.exit(400)
    end

    local params = {
        grant_type = "authorization_code",
        client_id = os.getenv("keycloakClientId"),
        client_secret = client_secret,
        code = code,
        redirect_uri = "https://"..host.."/auth/login/callback"
    }

    local httpc = http.new()
    httpc:set_timeout(5000)
```

```
    local res, err = httpc:request_uri(
        "http://keycloak:8080/realms/"
                ..realm_name..
                "/protocol/openid-connect/token", {
            method = "POST",
            body = ngx.encode_args(params),
            headers = {
                ["Content-Type"] = "application/x-www-form-urlencoded",
            },
            ssl_verify = false
        }
    )

    if not res then
        ngx.log(ngx.ERR, "Keycloak request failed: ", err)
        ngx.exit(502)
    end

    local data = cjson.decode(res.body)
    if not data or data.error then
        ngx.log(ngx.ERR, "Token response failed: ", data and
        ↪  data.error_description or "Decoding failed")
        ngx.exit(401)
    end

    local cookie_value = cjson.encode({
        token = data.access_token,
        refresh_token = data.refresh_token,
        id_token = data.id_token,
        authenticated = true
    })

    ngx.header["Set-Cookie"] = {
        "user_info="..ngx.escape_uri(cookie_value).."; Path=/; Secure;
        ↪  HttpOnly; SameSite=Lax",
        "access_token="..data.access_token.."; Path=/; Secure; HttpOnly;
        ↪  SameSite=Lax",
        "refresh_token="..data.refresh_token.."; Path=/; Secure; HttpOnly;
        ↪  SameSite=Lax"
    }
    ngx.redirect("https://"..host.."/")
end

return _M
```

The function does the following operations

1. Determining the realm by breaking down the `host`. The basic logic, being "splitting-then-comparing" is similar to the last routine

2. Retrieving `access code` from the query returned by the first step of the login flow

3. Forming a new request using both `realm` , `access code` and environmental variables
4. Set all necessary cookies

### 10.3.3 GET /userInfo

The inspection on the required cookie is performed by Apache APISIX® using a customized plugin. The route used for `GET /userInfo` are below:

```
{
    "id": "userInfo",
    "uri": "/userInfo",
    // "upstream_id": "server-upstream"
    "plugins": {
        "serverless-pre-function": {
            "phase": "rewrite",
            "functions": [
                "..."
            ]
        }
    }
}
```

To properly demonstrate the function used in `serverless-pre-function`, the candidate will demonstrate separately the code block because it was converted into a single line to adapt to the JSON criteria. A well formatted function is shown below:

```
return function(c)
    -- Get HTTP Cookie header
    local h = ngx.var.http_Cookie
    if not h then
        ngx.exit(401)
    end

    -- Extract user_info from cookie
    local u = h:match('user_info=([^;]+)')
    if not u then
        ngx.exit(401)
    end

    -- Clean and decode user info
    local d = ngx.unescape_uri(u)
        :gsub('^%s*[jJ]%%3A', '')
        :gsub('^%s*j:?', '')
        :gsub('[%c%z]', '')

    -- Decode JSON user info
    local j, e = pcall(require('cjson').decode, d)
    if not j then
        ngx.exit(400)
```

```
end

-- Extract token
local t = e.token
if not t then
    ngx.exit(401)
end

-- Extract and decode JWT payload
local p = t:match('[^.]+%.([^.]+)%.')
p = ngx.decode_base64(
    (p:gsub('-', '+'):gsub('_', '/') .. string.rep('=', (4 - #p % 4) % 4))
)

-- Decode JWT payload JSON
local x, y = pcall(require('cjson').decode, p)
if not x then
    ngx.exit(401)
end

-- Verify required fields
if not y.preferred_username or not y.realm_access.roles or
↪  #y.realm_access.roles < 1 then
    ngx.exit(403)
end

-- Set cookies
ngx.header['Set-Cookie'] = {
    'User=' .. y.preferred_username .. '; Path=/',
    'Role=' .. y.realm_access.roles[1] .. '; Path=/',
    'id_token=' .. (e.id_token or '') .. '; Path=/',
    'refresh_token=' .. (e.refresh_token or '') .. '; Path=/'
}

-- Set response headers
ngx.header['Content-Type'] = 'application/json'

-- Prepare response
local response = {
    active = true,
    username = y.preferred_username,
    role = y.realm_access.roles,
    email = y.email
}

-- Copy all additional fields from JWT payload
for k, v in pairs(y) do
    response[k] = v
end
```

```
    -- Send JSON response
    ngx.say(require('cjson').encode(response))
end
```

This lengthy `lua` function effectively is trying to do the following things:

1. It extract tokens obtained from login flow from the cookie.
2. It sanitizes the token, and decodes it.
3. It reorganize the necessary information into a new JSON object, and sends it back.
4. Upon any error, it sends back corresponding status code.

## 10.4 The logout flow

The logout flow for each user is easy to operate, as the simplicity is one of the main goal of the design. In fact, any user can simply tap on the logout button that resides on top-right of the home view and that concludes the whole experience on the user's side. This is the original interaction logic that remains unchanged from previous architecture.

From the perspective of the developers, there are more complicated details behind it. The logout flow, defined by `OpenID-Connect` or `OAuth 2.0` standard, is similar to the log-in flow. It is basically a request to the logout endpoint with some prerequisites such as credentials (tokens) obtained through the login flow and should be stored in the cookies. More specifically,

The client fetches from the logout endpoint set by Keycloak, with a query including

- `client_id`
  - The client that the user logged into, in this project always "Grafana"
- `id_token_hint`
  - The token that obtained though login that indicates the id of the user.
- `post_logout_redirect_uri`
  - Similar to login, Keycloak needs to know where it should redirect the user to after the logout

The client also needs to clear all the relevant cookies that are used to store user information.

- Tokens, User, Role, etc...

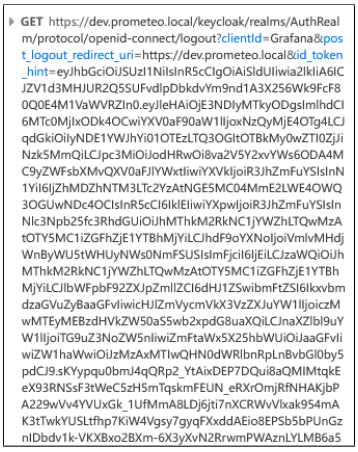



Figure 10.3: Successful logout (1)

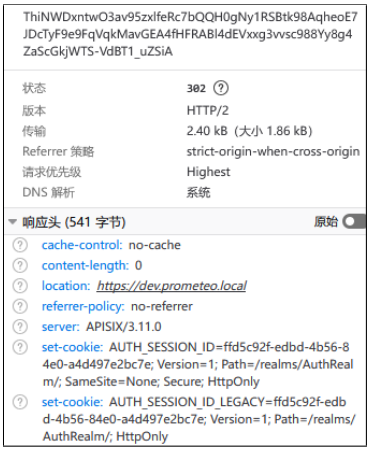



Figure 10.4: Successful logout (2)

The business logic of the logout flow has been constructed through an APISIX route on `POST /auth/logout`, for more details, check the code below.

```
{
    "id": "auth-logout",
    "uri": "/auth/logout",
    "plugins": {
        "serverless-pre-function": {
            "phase": "rewrite",
            "functions": [
                "..."
            ]
        }
    }
}
```

where a readable version of the functions is shown below.

```
function _M.rewrite(conf, ctx)
    local host = ngx.var.host or "AuthRealm"

    local realm_name = "AuthRealm"
    local prefix = host:match("^([^%-]+)%-")
    if prefix and prefix:lower():sub(1,2) == "ns" then
        realm_name = prefix
    end

    local cookies = ngx.var.http_Cookie or ""
    local id_token = cookies:match("id_token=([^;]+)")

    if not id_token then
        ngx.log(ngx.ERR, "Missing id_token in cookies")
        ngx.status = 400
        ngx.header["Content-Type"] = "application/json"
```

```
        ngx.say('{"error":"missing_id_token"}')
        ngx.exit(400)
    end

    local params = {
        clientId = "Grafana",
        post_logout_redirect_uri = "https://"..host.."/",
        id_token_hint = id_token
    }

    local logout_url = string.format(
        "https://"
            ..host..
            "/keycloak/realms/%s/protocol/openid-connect/logout?%s",
        realm_name,
        ngx.encode_args(params)
    )

    local cookie_list = {
        "Role=; Path=/; Expires=Thu, 01 Jan 1970 00:00:00 GMT",
        "User=; Path=/; Expires=Thu, 01 Jan 1970 00:00:00 GMT",
        "refresh_token=; Path=/; Expires=Thu, 01 Jan 1970 00:00:00 GMT",
        "connect.sid=; Path=/; Expires=Thu, 01 Jan 1970 00:00:00 GMT",
        "user_info=; Path=/; Expires=Thu, 01 Jan 1970 00:00:00 GMT",
        "id_token=; Path=/; Expires=Thu, 01 Jan 1970 00:00:00 GMT"
    }

    ngx.header["Set-Cookie"] = cookie_list
    ngx.redirect(logout_url)
end

return _M
```

The function will perform the following operations. Overall, it is very similar to the `GET /auth/login/callback` route introduced in last section.

1. Differentiate realms by breaking down the content of `host`
2. Retrieving `id_token` from the cookie
3. Forming a request to the logout endpoint exposed by Keycloak using available information
4. Clear out all relevant cookies

# Chapter 11

# Review on the new architecture

## 11.1 The objectives achieved

In the first chapter 8.3 of this part, the candidate has named several main objectives according to useful functionalities of Apache APISIX® and reorganizations in general. Those objectives are:

- Decoupling the microservices in the backend for a more clear and independent architecture. More specifically, by separating the microservices by its functionalities (API endpoint, authentication, and general microservices) the system will benefit in terms of efficiency and maintainability.
- Moving the client before the API gateway as the original placement does not make sense, the client is supposed to be an edge component to the system.
- Centralizing the authentication service by exploiting realms in Keycloak to increase efficiency by avoiding duplicate or repetitive configurations.

Now after a series of configuration and optimization, the candidate can declare that at current stage, all of the three objectives have been achieved successfully. Looking back on the original proposal of the new architecture, the system has also obtained in extra

- A "serverless" solution to the authentication service, as now there are no more API endpoints relating to authentication are still remaining in the back-end. It further proves that after proper immigration, eventually it is possible to convert all kinds of functionalities in the back-end over to the gateway, if ever needed. The candidate has not performed the same configuration on other microservices such as the database and observations. The reasoning of it can be, for example, the communication with the database can involve sensitive data and requires proper care.
- Relatively independent microservices. For example, previously the client strictly needs Keycloak since all functions in the back-end are built according to it. After integrating the API gateway, client does not care anymore about which authentication service is in the system. Being it Keycloak or not, it is eventually Apache

APISIX® that handles the requests based on route configurations are ensure that client get what it needs from authentication. This significantly improves the maintainability of the system

What's more interesting is the performance increase brought by Apache APISIX®. The candidate has mentioned the potential improvement on that category but never expected it to be worth highlighting. Indeed, in theory, the advantage from Apache APISIX® comes from its capability of load balancing, and it will not become obvious until multiple back-end nodes are deployed, which is not yet the case for PROMET&O. However, a simple and not rigorous pressure test may appear to be a disagreement, as shown in the pictures below:


Figure 11.1: Parallel test: APISIX vs Nginx

It is shown in the picture that in the `100 * 10` pressure test, the processed requests achieved by Apache APISIX® is 22% higher than `Nginx`, which is indeed surprising to see, but not such a shocking truth considering the capability of the API gateway.

What's more, the candidate put a vertical test for Apache APISIX® under both `100 * 10` and `1000 * 10` pressure test, and the results are shown in Fig. 11.2.

Surprisingly, it appears that Apache APISIX® is able to perform better under relatively higher concurrency pressure. This shows that daily pressure on PROMET&O has not exceeded the limit of Apache APISIX®, which is inspiring to hear, and leave great space for further development.

The experiments above are not very well designed as they are never meant to be rigorous study on the performance. The meaning of them is to show the fact that APISIX is very capable of its daily mission, and can provide extra benefits beyond its original purposes.


Figure 11.2: Vertical test for APISIX

## 11.2 Limits and reasoning

The viewer may have already come up with some doubts or judges on whether the flow is in its correct state or is performing with its full potential. Some of the questions could be:

- The client is now an edge component, but at what cost?
  - During the development, the candidate has kept the assumption that clients for different remote servers are identical, thus the client can be uniformed into one single node. But the candidate has come to understand that this will results in lower capability on customization, as devices relating to different name spaces can be distributed in different environment, which will eventually ends in different configurations.
  - In defend for the candidate himself, the client has reserved the capability of duplicating itself and apply different set of configurations thanks to the flexibility of Apache APISIX®. The topology of the architecture may change under this condition, as shown in Fig. 11.3.
- Is communicating using cookies safe?
  - The external communication channel is secured using HTTPS as one can see from the configurations the candidate has showed, and cookies themselves are also secured by setting their `secure`, `httpOnly` and `SameSite` to appropriate values. To be meticulous, the cookie is never the perfect solution for security, but mostly a compromise between security and efficiency.
  - On the other hand, cookies are vulnerable to various of malicious attacks. It can be overwritten pretty easily when being outside of HTTPS connection. It also adds overload to regular communications that may not need cookies.

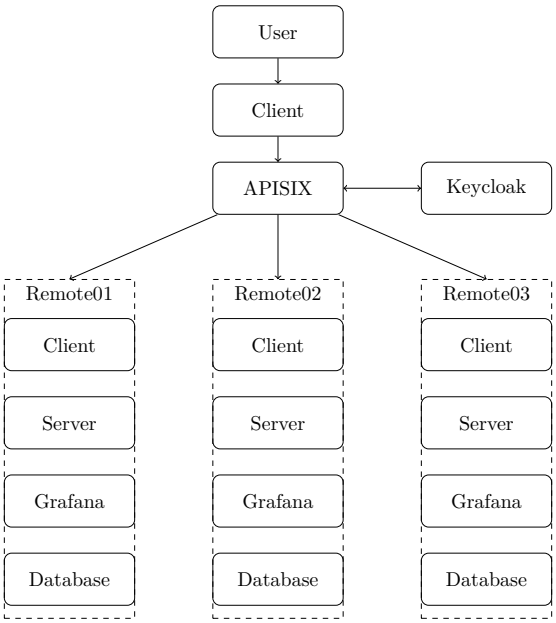



Figure 11.3: New architecture with distributed clients

- Is it possible to achieve "authentic serverless"?
  - The candidate has shown multiple routes in the gateway that originally handles API calls towards back-end now do not require the involvement of the back-end, so the possibility exists. Since the candidate has limited experience in `lua` programming language and knowledge about `nginx` reverse proxy, he has to admit the potential has shrink in his hand. Has a more experienced developer involved in the process, "serverless" will not be theoretical anymore.
  - But the candidate is thrilled that "serverless" being a amazing and inspiring concept on the first sight, it may not be the optimal option for every scenario. Taking PROMET&O as an example, the communication between microservices such as MySQL database and client is both important and sensitive, bold moves such as replacing back-end may introduce unforeseen risks that, considering on the future of the project, could prove to be major losses.

In conclusion, the candidate is not hesitant to say that the current architecture is far from a "prefect" one. To lay a basis for future development, the candidate most times chooses the most comprehensive and optimal option for maintainability over maximizing security and marginal efficiency.

## 11.3 A perception to further development

Throughout the chapter, the candidate has mentioned the concept of "serverless" multiple times. In order to clarify potential misinterpretation on what the candidate actually means, here the candidate will discuss this topic in a more detailed way.

The "serverless" architecture means that, by dynamically allocating resources over the cloud in responding to real-time needs from actual use cases, the back-end server can be seen as "intangible" or "shapeless" as it is not "solid" anymore.

Consider the traditional back-end server that is developed using `Express.js`, the back-end will define each API endpoint for the client to call and communicate with, and in turn inside it, layers such as `services`, `Data Access Object` will handle detailed business logic and data manipulation. Each of the components should be defined explicitly and unit tested before deployment.

Now, a "serverless" architecture will appear differently; it does not require each component or function to be explicitly defined by the developer, but rather focus on higher layer such as business logic. Taking the example of using SQLite and Google Firebase, traditionally the developer needs to define `DAOs` and write specific functions to interact with the database according to its policy, such as schema, locking mechanisms. But Google Firebase simply provides the developer a rather predefined data fetching function like a regular function call, making the implementation details invisible and stay in the cloud. A not-so-perfect example like this displays a fraction of differences in traits between those two options.

The optimization performed through this thesis is the first step into a more general concept of "serverless", as the functionalities of a tangible back-end server is gradually replaced by routes and custom plugins from the API gateway, the back-end server becomes more and more light-weighted. It is difficult for the candidate to predict if the back-end server will eventually disappear or not, but he is aware that certain degree of such load reducing for the server is generally beneficial to the system in terms of both performance and resource efficiency.

# Chapter 12

# Conclusion

As the candidate reaches the end of his development journey for the PROMET&O project, while looking back, the candidate is confident to say that he has completed all main goals that was determined at the beginning, and has gained much more than what was expected by himself in the beginning.

For the first part of the development, the candidate has finished the design and implement of the brand new user interface for the project on mobile views. Over the procedure, the candidate has looked into various of ways to organize the components on screens with limited size to emphasize visual hierarchy, and attempt to balance between the visual and operational aspects of the user interface. The lengthy procedure of evaluating different choices based on different requirements, each focusing on various aspects, has always been memorable for the candidate, and he considers this experience to be even more valuable than the final product. Although the design may appear to be not the most skillful as the candidate does not expertise on such subject, he still manages to come up with different methods and ideas on how to solve most problems he encounters.

For the second part, the candidate has finished the optimization on the network architecture of the project, his main goals includes decoupling microservices, centralizing the authentication service, along with other relatively minor improvements. The meaning of the modification consists both current and future advantages in terms of efficiency and maintainability. By experimenting towards the concept of "serverless", the candidate maintains the effort on exploring different possibilities to solve most problem, and also tries to establish a comprehensive and easy-to-development basis for future effort. Due to the immature design, some choices may be overturned by colleagues coming afterwards, but the candidate still believes that is exactly the point of his work in this part, as no design is perfect, and the attempt on reaching perfect never seizes.

Imagining the future for PROMET&O, the candidate is able to foresee a lot of possibilities; for example, the front-end user interface can benefit from more matured design and more efficient implementation. What's more, switching to other survey content providers other than SurveyJS can also be optimal. As for the architecture, the candidate never achieves the full potential of the gateway and its rich library of plugins. The candidate

can easily predict a more skilled colleague that will be able to take the advantage and make the project much better in terms of efficiency. A further decoupling of microservices can also be a good choice.

In conclusion, PROMET&O is a promising project; it has a divine objective of emphasizing on the environment and energy efficiency, and it has a solid basis for future effort, which will potentially make real differences in IEQ management around workplaces. The candidate is truly honored for having the opportunity to participate in the development for it.

# Chapter 13

# Appendices

## 13.1 List of the SASS items

The candidate has discussed his motivation and methodology relating to SASS files in section 2.4, one specific reasoning of using such a CSS super-set is the capability of lightweight programming, which means simple functionalities such as defining variables, numerical calculating, meanwhile it does not sacrifice most of the readability for the benefit and ideally, a flat learning curve.

In this section, the candidate will compose a list and discuss all the variables and functions he defined and used, and he will attempt to comment on the reason of their existence, how much value they provided, and whether potential improvement exists.

```
// Default screen sizes (iPhone 14 Pro Max)
$default-width-portrait:        430
$default-height-portrait:       932
$default-size-font:             16

$threshold-width:               900px
$trigger-width:                 ($threshold-width + 1)

// Gaps
$default-gap-vert-outer-border: 16px
$default-gap-hor-outer-border:  16px

$color-theme:                   #ff9724
$color-theme-accent:            #ffb347
```

The list above consists of all variables that are used in the development.

The first part defines the standard screen size defined for testings, which is an outdated setting since the device mentioned there is not an extreme case, thus is not the most appropriate choice. The choice for `font-size`, however, is one of the most popular choice across the community.

Next block, it sets a threshold for determining whether the user is under landscape mode or not. The choice is not supported by any authority, but is a legacy value that is

consistent throughout the development life of PROMET&O, so the candidate decided to keep the value unchanged. The candidate also would like to mention that, in most cases, the user interface differentiates the display mode through the computed value of aspect ratio, as the thesis explained in section 2.8.

The next two lines determines the default value for gaps between components, those values are not very frequently used since in most situations the user interface has very limited space to handle, so it does not make much sense that the system having a uniformed value for gaps.

Next we defined the theme color and an accent color, those are values that are obviously frequent during the implementation.

Then the candidate will show the functions.

```
// Returns the relative font size based on the default font size
@function getRelativeFontSize($num)
    @return calc($num / $default-size-font)

// Returns the relative font size for landscape orientation
@function getRelativeLandFontSize($num)
    @return clamp(#{$num}px, #{getRelativeFontSize($num)}em, #{$num * 2}px)

// Returns the relative height based on the default portrait height
@function getRelativeHeight($num)
    @return calc($num / $default-height-portrait)*100 + vh

// Returns the relative height for landscape orientation
@function getRelativeLandHeight($num)
    @return clamp(#{$num}px, getRelativeHeight($num), #{$num * 1.25}px)

// Returns the relative width based on the default portrait width
@function getRelativeWidth($num)
    @return calc($num / $default-width-portrait)*100 + vw

// Returns the relative width for landscape orientation
@function getRelativeLandWidth($num)
    @return clamp(#{$num}px, getRelativeWidth($num), #{$num * 2}px)

// Returns zero with !important flag
@function zero()
    @return 0 !important
```

Most of the functions serve the purpose of transforming values between units. We mentioned earlier that the usage of those functions are due to value settings in high-fidelity prototype in early design stage which fixes values in pixel unit, which is obviously not suitable for responsive design.

Throughout the user interface part, the candidate has demonstrated various functions he has used during the implementation. As a matter of fact, although they do appear

quite frequently, but the functionalities are either plain naive, or are results of temporary needs and requires a revision considering its global value.

# Bibliography

[1] Apache. Get apisix. *Apache APISIX Official Website*, 2025.
[2] ASHRAE. Ashrae standard 55. *ASHRAE*, 2017.
[3] Dick Hardt. The oauth 2.0 authorization framework. *RFC 6749*, October 2012.
[4] Wikipedia. Wikipedia - cascading style sheets. *Wikipedia*, 2016.
[5] Wikipedia. Wikipedia - visual hierarchy. *Wikipedia*, 2016.