PROJECT SPONSOR NATION INSTITUTE OF HEALTH, NIDCD.

THIS WORK IS SUPPORTED BY THE NATIONAL INSTITUTE ON DEAFNESS AND OTHER COMMUNICATION DISORDERS, OF THE NATIONAL INSTITUTE OF HEALTH (NIDCD/NIH). R01DC015436: "A REAL-TIME, OPEN, PORTABLE, EXTENSIBLE SPEECH LAB" TO THE UNIVERSITY OF CALIFORNIA, SAN DIEGO.

HTTP://OPENSPEECHPLATFORM.UCSD.EDU

WEB-APPS USER GUIDE

OSP TEAM

PUBLISHED BY OSP TEAM

OPENSPEECHPLATFORM.UCSD.EDU

First printing, December 2018

Contents

List of Figures

Introduction

1.1 What is in this guide?

This version corresponds to Release 2018c. We present instructions on making sure the Embedded Web Server (EWS) is working properly. We provide steps for:

- Testing each web-app to make sure it is working as expected.(Additionally, we provide some documentation on the web-apps.)
- How to customize them

The guide is organized by web app, so that you can easily find the information relevant to your task.

Note: Please refer to the Getting Started Guide to install the EWS software before following the instructions in this guide: https://github.com/nihospr01/OpenSpeechPlatform-UCSD

1.2 What's the bigger picture?

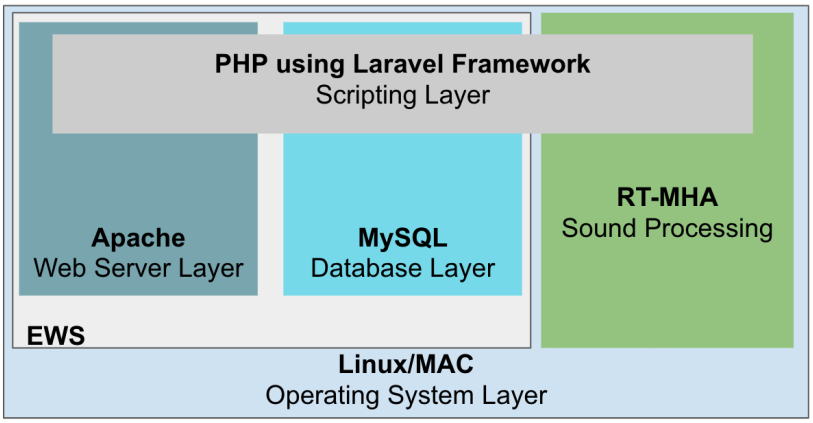

Figure 1.1: Diagram of OSP system. Everything in the white square of this diagram has to do with the embedded webserver (EWS) in this guide. The EWS uses PHP to communicate to the RT-MHA. The EWS literally hosts a webserver, on which users can access webpages or web-apps.

This guide is about the web-apps of the EWS. There is information about testing the web-apps, as well as documentation to help you

make new web-apps. The web-apps themselves are hosted by the EWS, and they use PHP to communicate with the other elements of the OSP. For example, the web-apps use PHP to send data and receive data from the mySQL database. In addition, the web-apps use PHP to communicate parameters to the RT-MHA, which results in real-time audio processing. The web-apps enable researchers and end users to affect the real-time engine, by simply clicking or tapping on elements of a webpage. The web-apps can be accessed from any browser-enabled device. If you are running OSP locally, like on a MacBook, you can use the web-apps by typing in the URL localhost:8000 However, if you are accessing the web-apps from a different device than the the one hosting the EWS, say the portable or wearable systems, you must type the IP address of the hosting device into the URL.

1.3 What's under the hood?

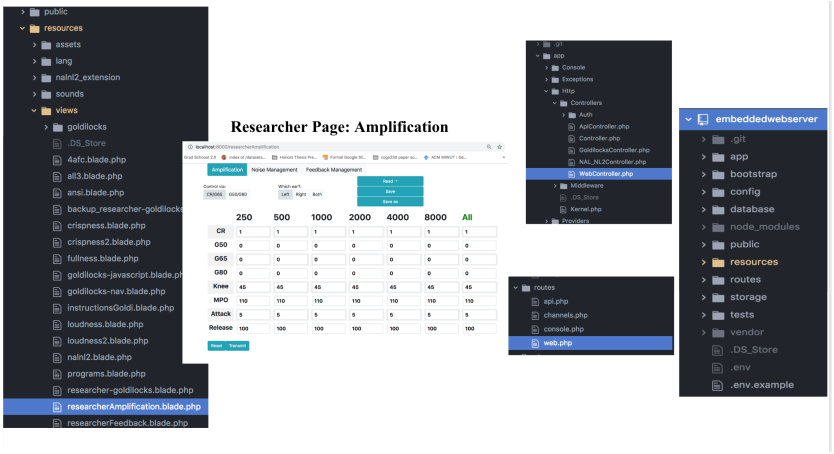

Figure 1.2: Diagram of the file-tree of the EWS, and the specific files involved in the rendering and functionality of the researcher page controlling amplification parameters.

This researcherAmplification.blade.php file located inside of the Resources>Views folder, contains the HTML skeleton of the web-app. It includes aspects of the page such as the tabs at the top, the table, and the labels of the table . This web.php file contains the routing instructions for the web-app. This file outlines which page is associated to which link, and points each page to associated controller. In this file we defined this page link as /researcherAmplification. (*The localhost in the URL refers to this app being served locally. If used remotely, this would be the IP address.) This WebController.php file contains all of the functions available for this page. This includes the function to view the page, which is called from web.php This

file includes the functions which send parameter information to and from the RT-MHA and mySQL databases.

Please be aware that line numbers in code screenshots may not exactly match line numbers in the code provided. Please refer to comments and specified section.

1.4 OSP web-app Landing Page

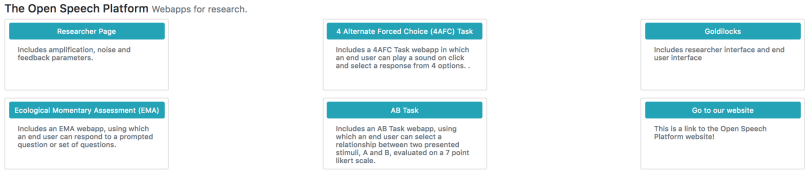

Figure 1.3: Diagram of the file-tree of the EWS, and the specific files involved in the rendering and functionality of the researcher page controlling amplification parameters.

The OSP web-app landing page is the place you end up if you simply type the URL localhost:8000 or IPAddress:. It contains links to all of the different web-apps that we have provided for OSP. It also has a link to our website.

1.4.1 Testing the OSP web-app Landing Page

In order to make sure the OSP web-app landing page is working properly, simply click on the blue "Researcher Page" button. You should be redirected to the researcher page, and see a table of amplification parameters. If the link works, everything is running as it should be and you can proceed.

1.4.2 Helpful tools and tutorials

Most of the time, coding includes googling a lot of stuff, so we have compiled our most visited websites when creating these web-apps, and have included them below:

1. To get more familiar with Laravel, the framework we are using to enable simple communication between the EWS and the RT-MHA, follow along with these simple tutorials:

   Laravel Tutorials: `https://laracasts.com/series/laravel-from-scratch-2018`

2. To get all of the details on Bootstrap, the styling front end framework we are using to create sleek styling which supports multiple form factors, visit this website. Use the menu bar on the left to explore what bootstrap has to offer. As you go through this guide, you can visit this helpful website to learn more about any element used: Note: this particular link will take you to the section about the way in which Boostrap organizes the layout of pages, enabling you to customize content placement.It is good to be familiar with

   Bootstrap Documentation: `https://getbootstrap.com/docs/4.1/layout/overview/`

this grid system of rows and cols, so take a few minutes to read this page.

3. To learn more about HTML5, the front end skeleton all of the web-apps use, visit this helpful site: (This includes things like the <head> or <style> tags.)

HTML5 Documentationhttps://www.w3schools.com/html/html_basic.asp

1.4.3 *Under the hood of the OSP web-app Landing Page*

Figure 1.4: This is code from the web.php file, located in the Routes folder. You may notice that the code lines in the screenshots do not match the code lines in your download. Do not worry, this is simply the result of comments being added to the code, or things being moved around.

As we discussed in the under the hood section, this page can be broken down into three components:

- The view, which contains the HTML skeleton and some styling elements are located in the Resources>Views>welcome.blade.php file.
- The web.php file, located in the Routes folder, and contains information about the link to this page.(Reference Figure 1.4 and notice lines 14-16.) This code specifies that the interface in the welcome.blade.php file is accessible via the / link. This is the default home direction if no URL is specified. If you click on one of the other web-apps, you notice that the URL reads

  localhost:8000/nameof.blade.php page, or IPAddress:nameof.blade.php page.
- this page does not have any live updating functionality. It is actually just a list of links to different pages. The controller responsible for this page is called WebController.php, and is located in the app>Http>Controllers folder. The only function that it performs for this page is actually rendering the view. In figure 1.6, pay attention to lines 97-99, which shows the controller code that actually renders the view.

Reference figure 1.5 to look inside the welcome.blade.php file. This excerpt of code that makes the part of the landing page which has the Researcher Page tile. We use Bootstrap 4, an open-sourced front-end framework, which provides a library of stylized elements for designing web apps. For example, on line 55 in figure

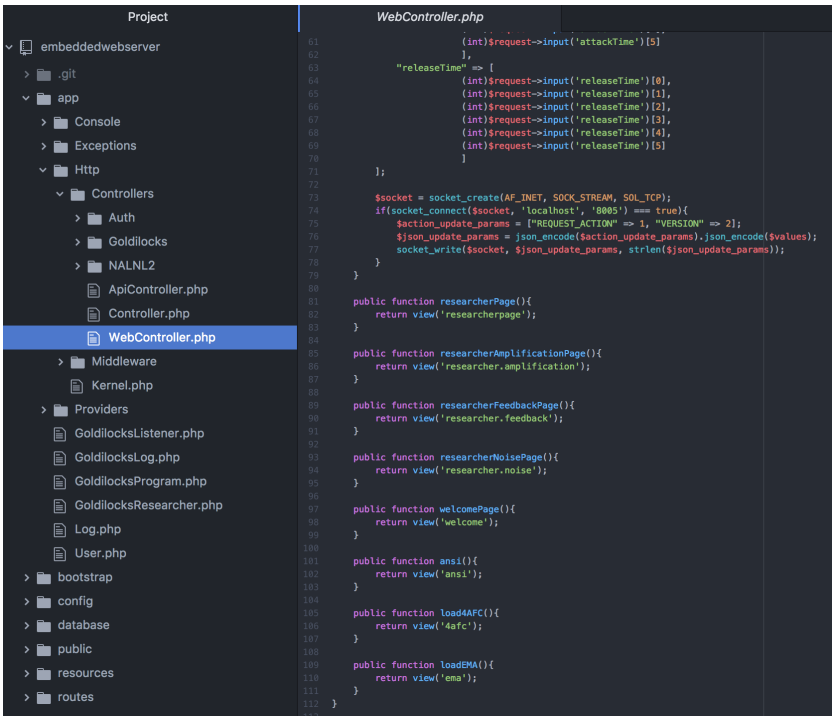

Figure 1.5: Screenshot of the controller for this web-app. This bottom portion that you see contains the functions which render many of the web-apps. The top portion of this screenshot features the code for pushing the last row of the Amplification page table. Notice that there are several web-apps which receive commands from this controller.

Figure 1.6: This is code from the welcome.blade.php file, located in the Resources>Views folder. It contains the HTML skeleton of the web-app, as well as styling elements. (Note: You can also follow along in your own development environment, browser or Finder. (note: these screenshots of code are taken in a development environment called Atom. If you download Atom for your computer, you can go to File>Open> and select the directory that you have this code saved in. In other environments, there may be no colors)

1.5, you can see that a class called "card" is called. This is a bootstrap component, which can be easily customized for diverse tasks. You can learn more about cards and other bootstrap elements here: https://getbootstrap.com/docs/4.0/components/card/

To change the name of this card to say something else, you would need to replace the text on line 56 that says Researcher Page. Once you make a change, save it and then re-start the server to make sure that change gets recognized. Line 60 is the part of this card, which contains the link to the researcher page. The syntax of line 60 corresponds to pointing to a URL in the Laravel framework. Laravel is like Bootstrap, but for the PHP side of the web-apps. Laravel is a framework, which comes with ready to use elements that every website needs. For example, Laravel enables us to designate a URL in this simple manner. On the second half of line 80, you notice the classes

"btn" and "btn-primary". These are also Bootstrap components, buttons. There is a whole section in the Bootstrap documentation website about buttons, and a number of different types to choose from. To change the button color, for example, you could change btn-primary to btn-secondary, or one of the other options described in the documentation.

The Researcher Page web-app

The researcher page is a tool researchers can use to control the hearing aid. All of the parameters of the real-time master hearing aid (RT-MHA) are exposed, and can be changed in real time. These parameters include controls for amplification, noise management and feedback management. The parameter states can be transmitted in real time to a connected device, creating real time sound processing.

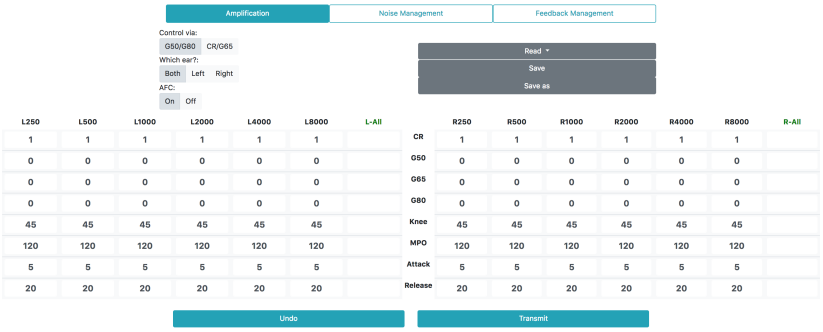

Figure 2.7: Researcher page web-app, in the amplification parameters section. The url of this page is /researcher/amplification Note that the URLs of the noise and feedback management pages are different.

2.5 Testing the Researcher Page

2.5.1 Testing Amplification Parameters

1. Run OSP-XCode with -t parameter enabled and -l parameter disabled
2. **Open the Researcher Page web-app in your browser** If you are running the web-app on the same device as you are running the OSP, type this URL into your browser:

   ```
   localhost:8000/researcher/amplification
   ```

   If you are running the web-app on another device, enter the following URL into the browser of your device:

```
IP_ADDRESS/researcher/amplification
```

3. **Add at least 5db to each cell in the G65 row.** You can use the All column to populate each cell in the row with the same value.
4. **Press transmit, and listen to the sound that plays.**
5. **Now, remove those 5 db from each cell in G65, and listen again.** If you hear a difference between the two parameter states, then you know the software is working as intended.

2.5.2 *Testing Noise Management Parameters*

1. **Open the Researcher Page and click on the Noise Management Tab in your browser** If you are running the web-app on the same device as you are running the OSP, type this URL into your browser:

```
localhost:8000/researcher/noise
```

If you are running the web-app on another device, enter the following URL into the browser of your device:

```
IP_ADDRESS/researcher/noise
```

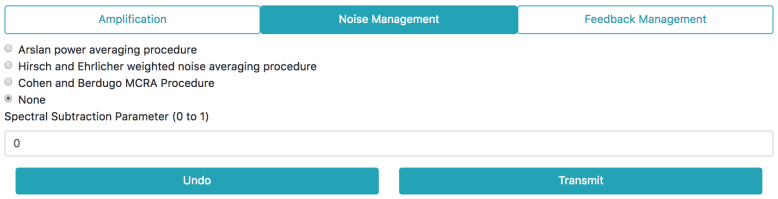

Figure 2.8: Researcher page web-app, in the Noise Management section. The url of this page is /researcher/noise.

2. **Click on the circle next to one of the available procedures to select it.**
3. **Notice whether or not you heard a change in the resulting sound. If you have, this page is connected**

2.5.3 *Testing Feedback Management Parameters*

This page is used to control the RT-MHA's adaptive feedback cancellation module. This page showcases the different options one has with regards to using feedback management. It also features a diagram of where this module fits into the bigger picture of the RT-MHA.

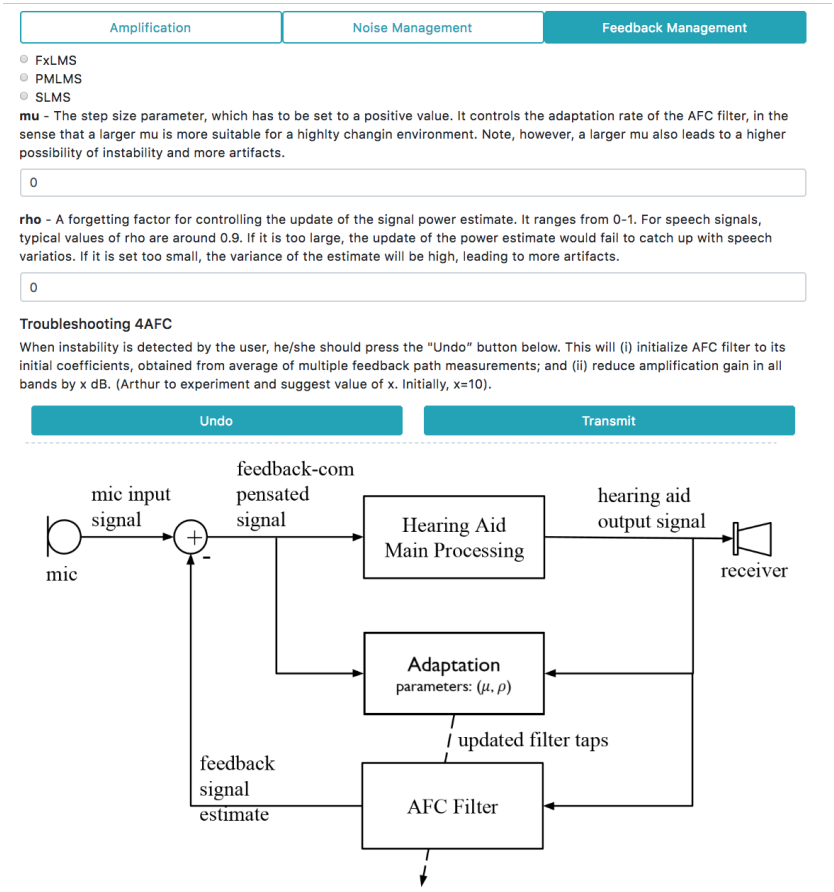

Figure 2.9: Researcher page web-app, in the feedback parameters section. The url of this page is /researcher/feedback

1. **Open the Researcher Page and click on the Noise Management Tab in your browser** If you are running the web-app on the same device as you are running the OSP, type this URL into your browser:

   ```
   localhost:8000/researcher/feedback
   ```

   If you are running the web-app on another device, enter the following URL into the browser of your device:

   ```
   IP_ADDRESS/researcher/feedback
   ```

2. **Click on the circle next to one of the top three lines to select it.**

3. **Notice whether or not you heard a change in the resulting sound. If you have, this page is connected**

2.6 Researcher Page Documentation

2.6.1 Amplification Parameters

This section includes screenshots of the code that makes the amplification parameters page work.

Recall the components of a web page we discussed in the under the hood section.

1. **The view**, which contains the HTML skeleton and some styling elements is located in the Resources>Views>researcher/amplification.blade.php file. The top of the amplification parameters page includes the navigation bar, which displays button links to the other researcher pages, as well as the the control via, save, save as and read buttons. The bottom of the amplification parameters page contains two tables: one for the left ear and one for the right ear, as well as the reset and transmit buttons. In the rest of this section, I will be referring to the views by either saying top or bottom.

Figure 2.10: This is a screenshot of the start of the table in the bottom of the amplification parameters page. You may notice that the code lines in the screenshots do not match the code lines in your download. Do not worry, this is simply the result of comments being added to the code, or things being moved around.

In the image above, you can see the HTML code to make the table in the bottom of the amplification parameters page. Look at line 242 in this screenshot to see how a table is initialized using Bootstrap. We create a div, and then call the class table, and table-sm. These both refer to the table class, but the sm makes the table adjust when presented on a mobile screen.

The <thead> denotes the start of the head of the table. Note that it is essential to close the head with a </thead> as you can see in the bottom line of the screenshot. Then the <tr> element denotes the start of a table row. Notice how there many lines labeled <th> until the closing of this row with the </tr>. Each <th> specifies what the head of that column will be called.

You can always visit: `https://getbootstrap.com/docs/4.0/content/tables/` to learn more about specifics about the table class.

Notice that once <tbody> starts, the columns are referred to using the <td> element instead. Here is an exact copy of first column in the first row, the Compression Ratio:

```
<td class="mha-td table-success"><input name="compression_ratio" data-side="left" data-index="0" id
    ="cr_0_1" class="form-control form-control-sm table-field font-weight-bold mha-input-field"
    type="number" value="1"></td>
```

Notice all of the many classes in use to make this table possible. Pay particular attention to the id, which is cr_0_1. This stands for

2. **The route** information for this web-app can be found in the web.php file, located in the Routes folder, and contains information about the link to this page. Reference Figure 2.10, and notice lines 19, which specifies the route for the amplification parameters page. It makes a call to the web controller, to access the researcher amplification page. Notice that the other two components of the researcher web-app: the noise and feedback management pages, have their own routes.

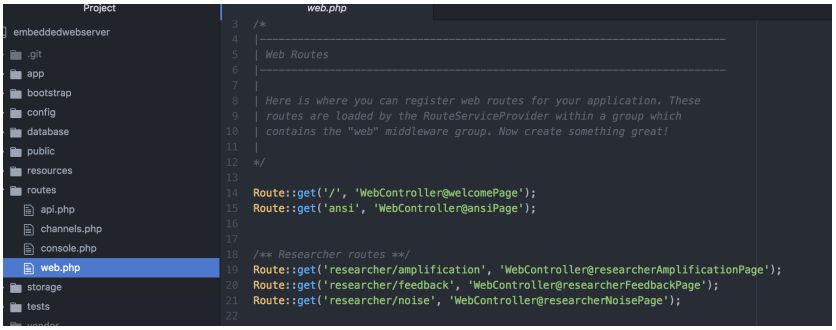

Figure 2.11: This is a screenshot of the code for the researcher page routes found in web.php Notice that the route for the amplification parameters page is the first line under the comment "Researcher Routes"

3. **The controller** on this page is responsible for the live updating functionality. The controller for the whole researcher page is called WebController.php, and is located in the app>Http>Controllers folder.

   You can see that the first main function of this file starts on line 17 in Figure 2.11, and is called "updateParams". This is a request, meaning it must be initiated. In the amplification parameters page, the request is initiated when a user presses "Transmit".

   Notice on line 23 in the Figure 2.11 that it asks for the "g50" values. In the lines of code before the "g65" label, you see requests to return integer values of the input in the g50 row. The numbers in orange, 0-5, correspond to the columns. As such, lines 23-29 in this screenshot can get the input from the g50 row on the amplification parameters page.

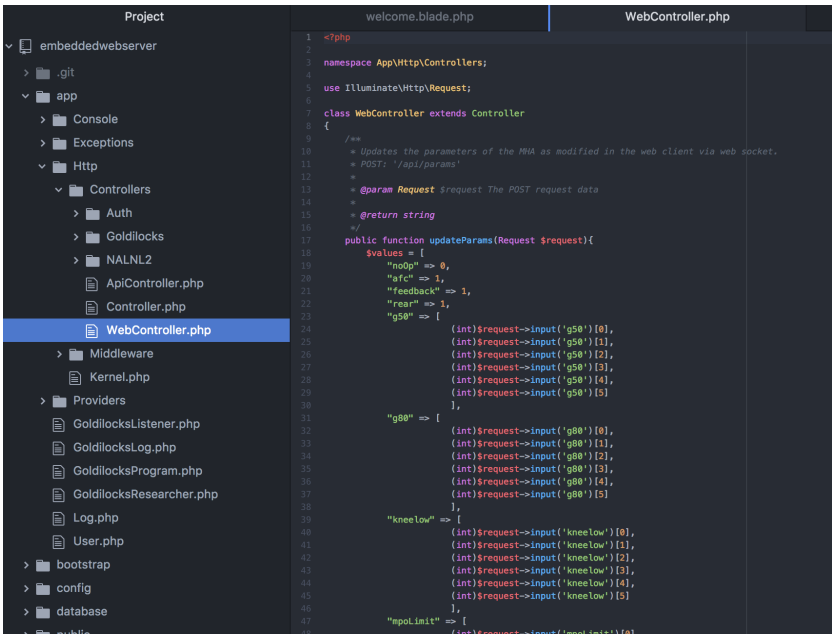

Figure 2.12: This is a screenshot of the controller code for this web-app. You can see

2.6.2 *Noise Management Parameters*

This section includes screenshots of the code that makes the noise management parameters page work. Recall the components of a web-page we discussed in the under the hood section.

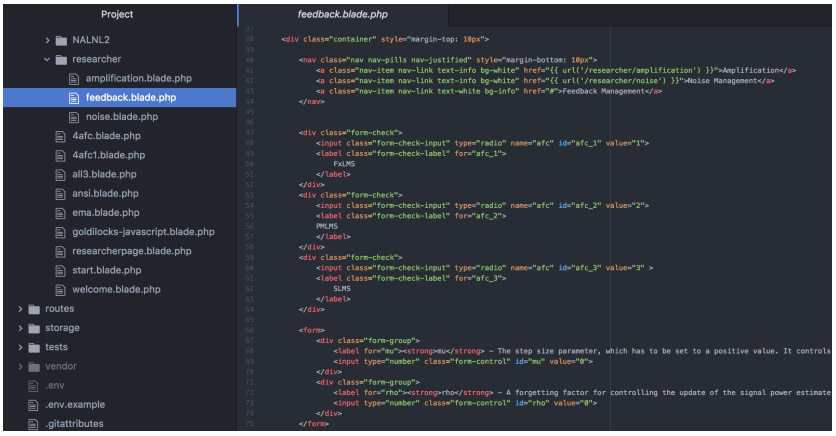

Figure 2.13: Screenshot of the code of the feedback.blade.php file, which contains the content that is used to visualize the feedback management web-app, also called the view.

1. **The View**, which contains the HTML skeleton and some styling elements is located in the Resources>Views>researcher/noise.blade.php file. This is where you can enter the text that you want to appear on a page, and describe any buttons and what they do. You can see in Figure 2.12, lines 41-45 create the navigation bar you use

in the interface to go between the three pages of the researcher web-app. Notice that line 42 and 44 include links to the respective pages, using the same syntax we saw in the welcome page links.

You can think of each component of the page being wrapped in its own div. For example, lines 48-53 wrap up the input for the Arslan power averaging procedure. Line 48 starts the div, and establishes that this will be a check form. Then, in line 49, the expected input is specified, it will be a radio type of input, indicated by a black dot in a circle, when selected. Lines 50-52 specify the "label" for this field in the form, and that is where you see the text "Arslan power averaging procedure", which appears on the page. You can see lines 54-59, 60-65 and 66-71 in this photo follow this pattern for the other options in the noise management parameters page.

2. **The route** information for this web-app can be found in the web.php file, is located in the Routes folder. It contains information about the link to this page. Reference Figure 2.10 and notice line 21, which specifies the route for the noise management parameters page. It makes a call to the web controller, to access the noise management page.

3. **The Controller** is located in the app>Http>WebController.php file. As described in the route section above, the route asks the controller to render this page in the routes. The actual function which renders the page lives in the controller, along with the other functions that encode the page's functionality.

2.6.3 *Feedback Management Parameters*

This section includes documentation and code snippets from the feedback management page. It will also be broken down into the components: view, route, controller.

1. **The View** file for this page, feedback.blade.php is located in resources>views>researcher, along with noise.blade.php and amplification.blade.php. Notice in figure 2.13, you see the same code to define the nav bar, only this time, there are links for the amplification and noise pages.

   Notice that the structure of this page is very similar to that of the noise management page. In particular, look at lines 47-52 in figure 2.13. You see the opening div with the form-check, followed by the line which specifies the input type as radio, followed by the label for the field, and ended by the closing field. This pattern repeats for the other options of AFC.

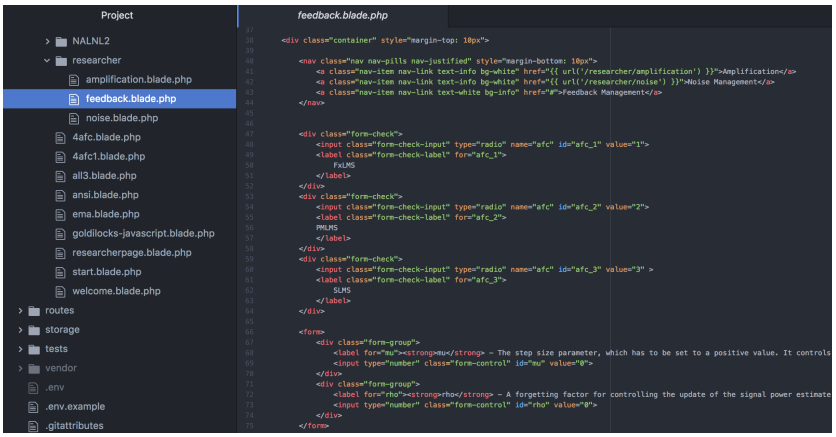

Figure 2.14: Researcher page web-app, in the amplification parameters section. The url of this page is /researcherAmplification Note that the URLs of the noise and feedback management pages are different.

Find the tag <form>, on line 66 in figure 2.13, and notice the similarities and differences between a form which has a check input, and a form which has a character input. This section encodes the mu and rho parameters, which users can enter by typing in numbers. As such, this is a basic and useful type of input which is good to be familiar with. To learn more about making forms with the bootstrap framework we are using, visit this website : `https://getbootstrap.com/docs/4.0/components/forms/`

2. **The Route** information for this page can be found in the web.php file, located in the Routes folder. Recall that this is the same file in which the other routes are, and reference figure 2.10. Line 20 in this image defines the route for the feedback management page, by calling the controller which contains the function which renders the page.

3. **The Controller** is located in the app>HTTP>WebController.php file. It is the same controller as the other pages of the researcher web-app. As described in the route section above, the route asks the controller to render this page in the routes. The actual function which renders the page lives in the controller, along with the other functions that encode the page's functionality.

4 Alternate Forced Choice (4AFC) web-app

This web-app is still in progress, and will soon be connected to the real time audio functionality. Please feel free to look at the user interface, and let us know if you have suggestions on improving the behavior flow!

The 4AFC web-app is a tool to measure outcomes, which allows researchers to run four choice question tasks using live sound processing, or canned stimuli.

3.1 Testing the 4AFC web-app

1. Run OSP-XCode with -t parameter enabled and -l parameter disabled

2. **Open the 4afc web-app in your browser** If you are running the web-app on the same device as you are running the OSP, type this URL into your browser:

   ```
   localhost:8000/4afc
   ```

   If you are running the web-app on another device, enter the following URL into the browser of your device:

   ```
   IP_ADDRESS/4afc
   ```

3. **Click on each of the four options in the first question** and confirm that the sound that plays is in fact the word that is written. If the words are playing correctly, the app is working properly. To learn how to edit the contents, see the documentation section below.

3.2 4AFC Documentation

In this section we will go over the three primary components we have been discussing: the view, route and controller, and introduce a special format for storing data to be read into a web page.

1. The View for this page, 4afc1.blade.php, can be found in the resources>views folder. What is important to understand about this web-app, is that each question is not an entirely new web page, but rather, the contents of the questions are dynamically read in, on click, from the word_sets.json file (Figure 3.15) in the public folder.

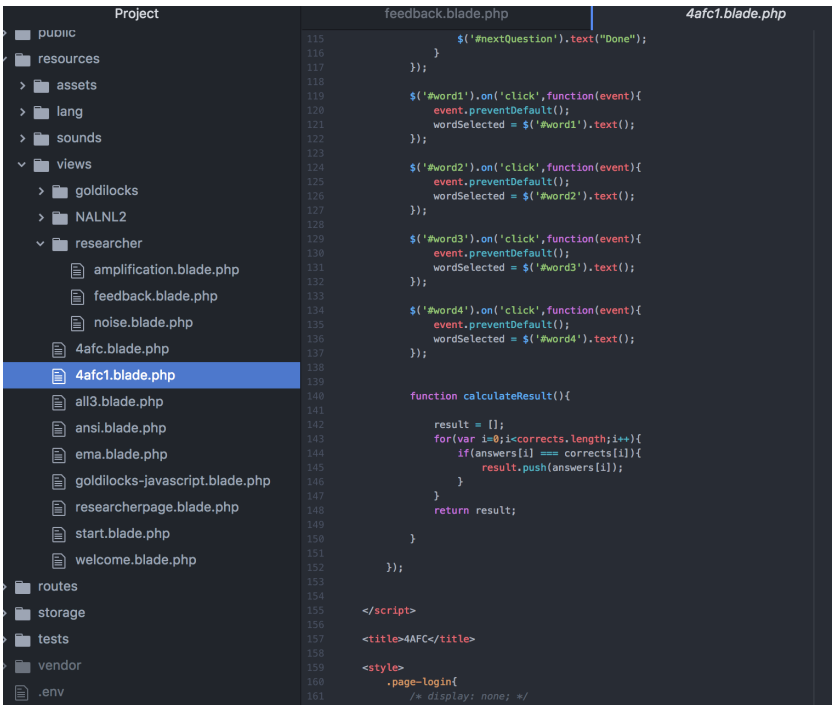

Figure 3.15: This is code from the 4afc1.blade.php file, which shows some of the JavaScript functionality, which contributes to the buttons being able to update with each question. .

Reference figure 3.14, and notice that all of this code is happening within the <head> html tag. You can see a <script> tag, which means that the code until the </script> tag will be in JavaScript, or another non-HTML language. The <style> tag, means that the code until the </style> tag is CSS styling code, which can be used to define the way content looks. It is also possible to add style directly into the HTML, which you can see in figure 2.9. Each row heading is styled to a certain font size.

Now, reference figure 3.15 to see the code to create the elements you see on the page. Notice the first section is a way to include a

login screen directly into the web-app. Following down the page, notice the code for the progress bar. This is a bootstrap class we use to create the look and feel of a dynamic web-app.

Towards the bottom of figure 3.15, you can see word1 and word2 being loaded into btn elements. This is how the words choices appear in the buttons on the 4afc web-app.

You can easily change the number of questions the app displays, or the contents of the questions using the steps provided next, without changing the styling aspects of the page. This will make it very simple for you to customize this web-app for your needs.

You can find more information about Bootstrap classes including progress on: https://getbootstrap.com/docs/4.0/components/progress/

To learn more about the JavaScript used to actually make the words appear, check out: https://www.w3schools.com/js/js_json_html.asp

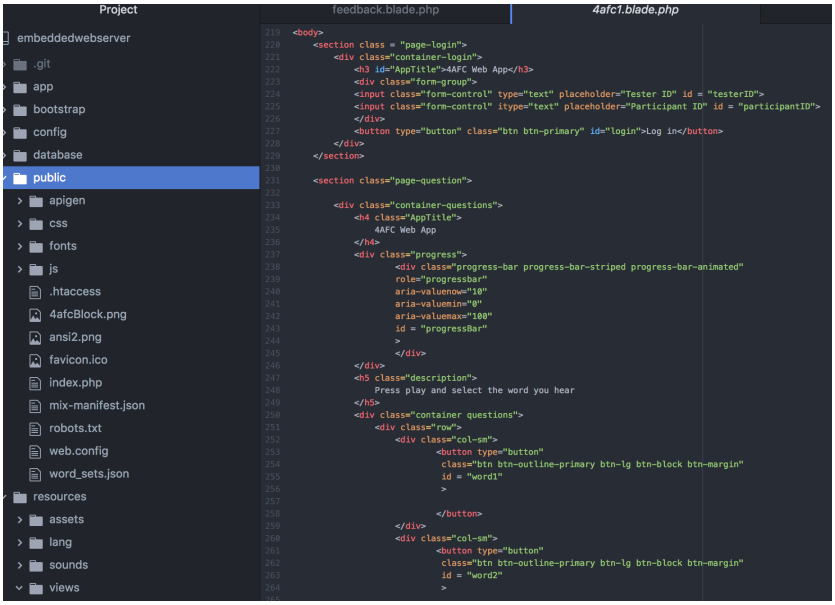

Figure 3.16: This is code from the 4afc1.blade.php file, which shows some the skeleton of what the user sees on the page, including the progress bar, and the button containing the first word.

(a) **Find the file in the Public folder called word_sets.json** This is a sample JSON data structure. JSON stands for Java Script Object Notation, and refers to a data structure format which can be easily used in a web page.

(b) **This JSON file starts opens and closes with square brackets [], and each segment in curly brackets , pertains to one question.** These segments are also in order, such that the first line of the first segment, named "words", would appear as the four answer choices for the first question. The line of the segment, named "correct_answer",corresponds to the correct answer for that question, which is important for scoring purposes. Finally, the last line of the segment, named "audio", contains the link to the audio file for that question.

Figure 3.17: This is code from the word_set.JSON file, located in the Public folder.

(c) **To edit the existing questions, replace the text in the word_sets.json file, that appears in green on the image below** For example, you could change the first question to have the options: add bad cad dad; pick bad as the correct answer and your resulting JSON segment would look as follows:

Figure 3.18: This would be the first segment of word_sets.json would look like if you performed step 3.

(d) **In order to change the audio files** first, organize your audio files such that there is a folder for each question, and the four options are inside of the folder. Second, copy paste these folders of sound files into the Resources>Sounds folder. Second, change the third line of each segment in the JSON file, named "audio", to reflect your sound files. For example, say you added a folder of sounds called "example", and inside it contained four sound files: 01, 02, 03, and 04. Let's say the correct answer was "03". This is what the "audio" line of code would look like:

```
"audio": { "Word_set": "/../../../resources/sounds/example/", "Actual_answer": "/../../../
    resources/sounds/example/03.wav" }
```

(e) **To add a new question, copy and paste the code for an existing segment, and replace its contents with desirable ones.** For example, we could add a fifth question, by copying and pasting the code for question one, after question four. Then, we need to change the "words" and "correct_answer" fields for the new question. Let's say the audio files you added to Resources>Sounds in step 4, are the correct sound files for this fifth question. Then, the whole JSON for all 5 questions would look like this:

Note: Notice that line 21 now has a comma after it, and there is no comma after line 26. This is because line 26 is the end of

Figure 3.19: This would be the first segment of word_sets.json would look like if you performed step 5.

the last segment in this file, so no comma is needed. However, between segments, it is essential to have a comma, as demonstrated in lines 6, 11, 16 and 21.

2. **The Route** information for this page is in the web.php file, in the Routes folder.

3. **The Controller** for this page is WebController.php, and is located in the app>Http>Controllers folder. The only function the controller has for this web-app, is the one that renders the view of the page.

AB Test web-app

This web-app is still in progress, and will soon be connected to the real time audio functionality. Please feel free to look at the user interface, and let us know if you have suggestions on improving the behavior flow!

The AB test web-app is a tool to measure outcomes, which researchers can use to run AB comparison tests between two particular MHA setting combinations. Users respond using a 7 point likert scale. Using this app, you can AB test different parameter "prescriptions", and reach the best prescription for the end listener. If you are using the portable version of our system, you can also deploy an AB test in real time, and the users can complete them in the wild.

4.1 Testing the AB Test web-app

1. Run OSP-XCode with -t parameter enabled and -l parameter disabled
2. **Open the AB test web-app in your browser** If you are running the web-app on the same device as you are running the OSP, type this URL into your browser:

   ```
   localhost:8000/ab
   ```

   If you are running the web-app on another device, enter the following URL into the browser of your device:

   ```
   IP_ADDRESS/ab
   ```

3. Click on the play button indicated by a triangle in a box in the upper right hand corner.
4. Click on A, and listen to the output
5. Then, click on B and listen to the output You may have to repeat steps 3-5 a couple of times to get the hang of it.

6. Make sure that you can hear a difference in the sound you hear after you press A or B If you hear a difference, the AB web-app is working properly. To learn how to customize the sounds and parameters of the AB web-app, read the documentation section below .

4.2 AB Test Documentation

1. **The View** file for this web-app is called ab.blade.php, and is located in the resources>views folder. You can see many similar elements from the other outcomes apps, such as the progress bar, and results table at the end.

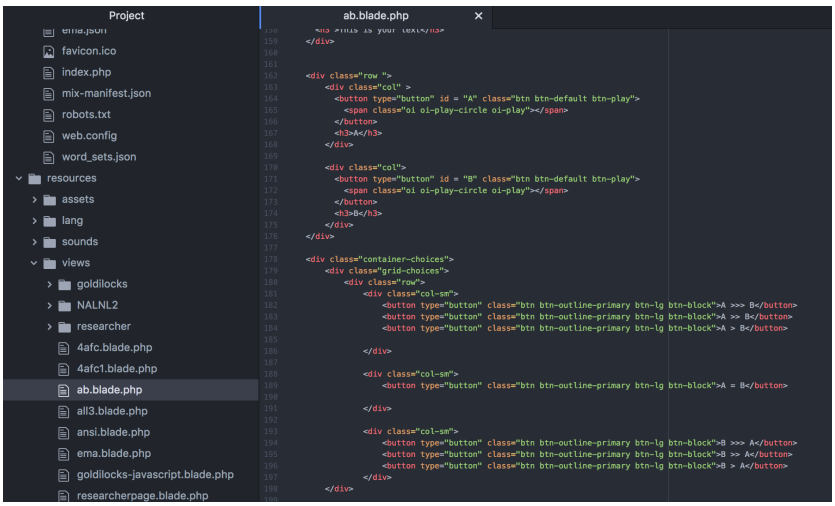

Figure 4.20: This is a screenshot of the part of the view code, which codes for the A and B buttons, as well as the option buttons you see in this web-app.

2. **The Route** information for this web-app is located in the web.php file. You can see that the controller, webController is being called to run the function which renders the webpage.

3. **The Controller** for this web-app is called has not been implemented yet, as the routes page contains the function to render the page.

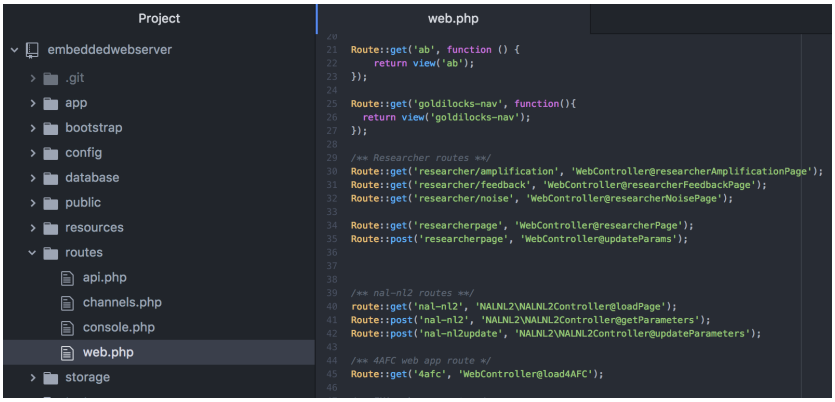

Figure 4.21: This is a screenshot of the part of the Route file, web.php. Notice at the top, you can see the function which returns the view of the AB web-app. Note that this is not done by calling a controller.

Environmental Momentary Assessment (EMA) web-app

This web-app is a tool for sampling the listenerâĂŹs mental state when they had a problem with the system so researchers have additional information to improve the hearing aid process. The questions can be easily changed to meet the needs of any study. One could even apply machine learning to this tool to auto-adjust the hearing aid. If you are using the portable version of our system, you can also deploy these assessments in real time, and the users can complete them in the wild.

5.1 Testing the EMA web-app

1. Run OSP-XCode with -t parameter enabled and -l parameter disabled

2. **Open the EMA web-app in your browser** If you are running the web-app on the same device as you are running the OSP, type this URL into your browser:

   ```
   localhost:8000/ema
   ```

   If you are running the web-app on another device, enter the following URL into the browser of your device:

   ```
   IP_ADDRESS/ema
   ```

3. **Go through the questions, select an answer, and then go onto the next question** At the end, you will come to a summary table of your answers. If you get to this page, your web-app is working! To learn how to edit the questions, see section 5.2 EMA Documentation

5.2 EMA Documentation

The EMA web-app is very similar to the 4AFC web-app in structure, and also has a JSON data structure, which you can edit to change the questions presented. We will first go over the three components, and then how to change questions in the EMA app.

1. **The View** file for this web-app, ema.blade.php can be found in the resources>views folder. Notice how Figure 5.22 is mostly written in JavaScript. You can see the part of the code (at the top), which actually saves the tester and listener ids. Next, notice on line 43, there is a call to fetch ema.json. This is the part of the code that loads in the JSON file.

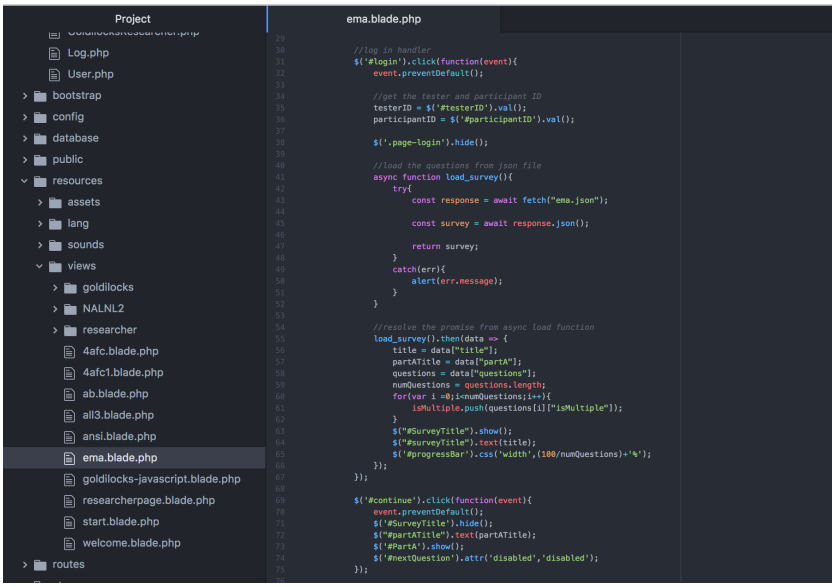

Figure 5.22: This is a screenshot of the ema.blade.php, or the view file for the EMA web-app. Notice the differences between this top part of the view, and the following picture, from later in the same file.

This next image, Figure 5.23, is also a screenshot of the ema.blade.php page, but a bit further into the code. Here you can clearly see the HTML skeleton that makes up the app. Notice that lines 384-489 encode the form which takes in the tester and listener ID, as well as the login buttons.

The next section, titled "Section for Task Description", is the code for generating the first page you see in the EMA app, the one that contains the note from the researcher about what the task will be, and is a great place to put a question about consenting to the task. The next section, "Section for Part A Description", is the second page that you see, which offers more space for waiver or task description information.

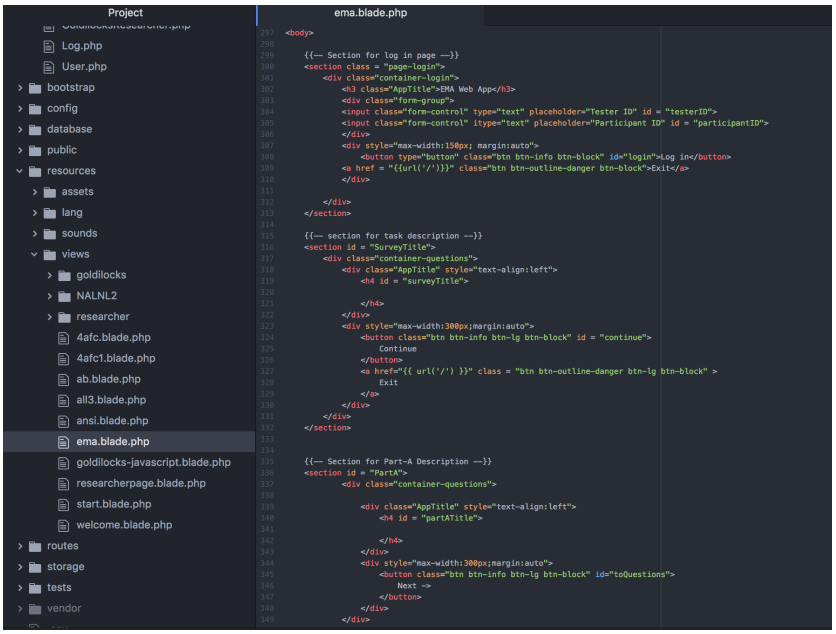

Figure 5.23: This is a screenshot of the further down the ema.blade.php, which shows the HTML elements you see on the screen.

2. **The Route** for this web-app is in the file web.php, located in the Routes folder. Reference figure 5.19. The EMA route starts on line 49 in the screenshot. Notice that this route is calling a function that returns the view of the EMA page. This is how the browser knows to display a specific page, given a particular URL. In this case, the URL is /ema, as you can see in the URL bar of your browser when you are on the page.

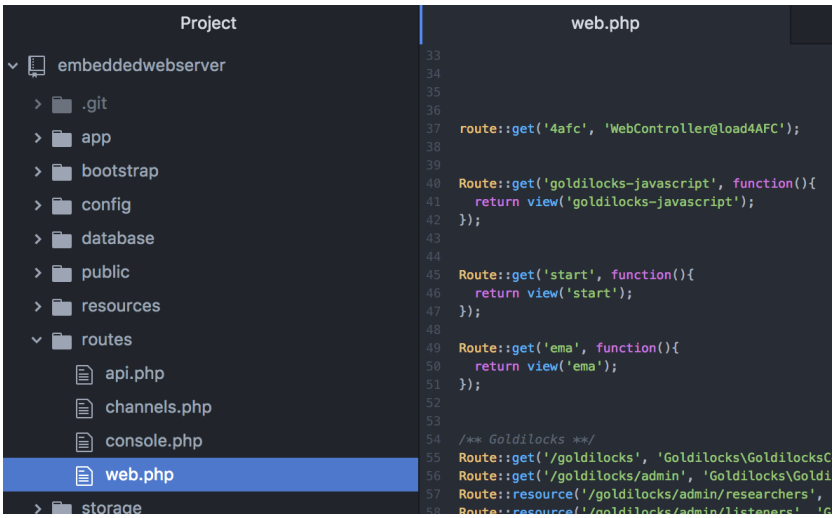

Figure 5.24: This is a screenshot of the web.php file, which shows the route for the EMA web-app

3. **There is no controller for this web-app**

4. **The JSON** file associated with this webapp, ema.json, is in the public folder. As you look through the file, notice that it has all of the text content of the EMA web-app. This includes the task descriptions, as well as the contents of the questions. This is, in fact, the file that is being fetched in Figure 5.22.

Figure 5.25: This is a screenshot of the ema.json file, which holds all of the content that will be read into the web-app. This is the file you should edit to change the contents of the app.

In order to change the contents of the first page, which has the "title" text, simply change the text in this file to what you would like it to say. The next line "Part A" holds the instructions for the first part of the task. Then, all of the questions and question choices follow. Notice the line in the each block of question choices, "isMultiple". This is a binary variable, either 1 if the question could have multiple answers, or 0 if there is only one answer choice.

There are many example questions provided that you could change the text of in order to make your own custom questions. Simply change the text that appears green from an existing question, restart and reload. Say we changed the last question to ask about how old the participant is. Because the participant should only have one answer for this question, I will set isMultiple=0.

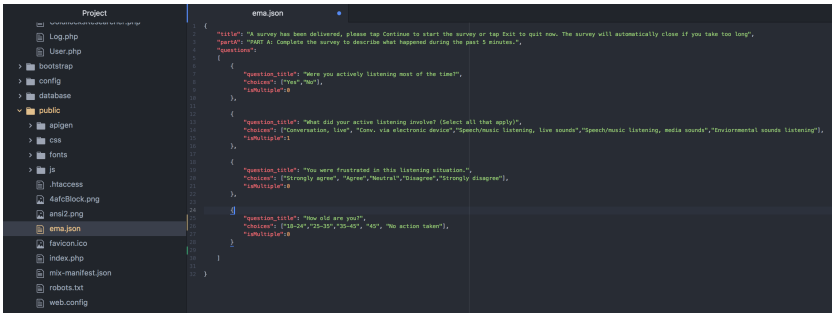

Figure 5.26: This is a screenshot of the ema.json file,with the last question changed to ask about the particpant's age

See figure 5.26 to look at the JSON file including this question. See figure 5.27 to see how this actually looks in the web-app! Editing

the content of this web-app is as simple as changing the text in this file.

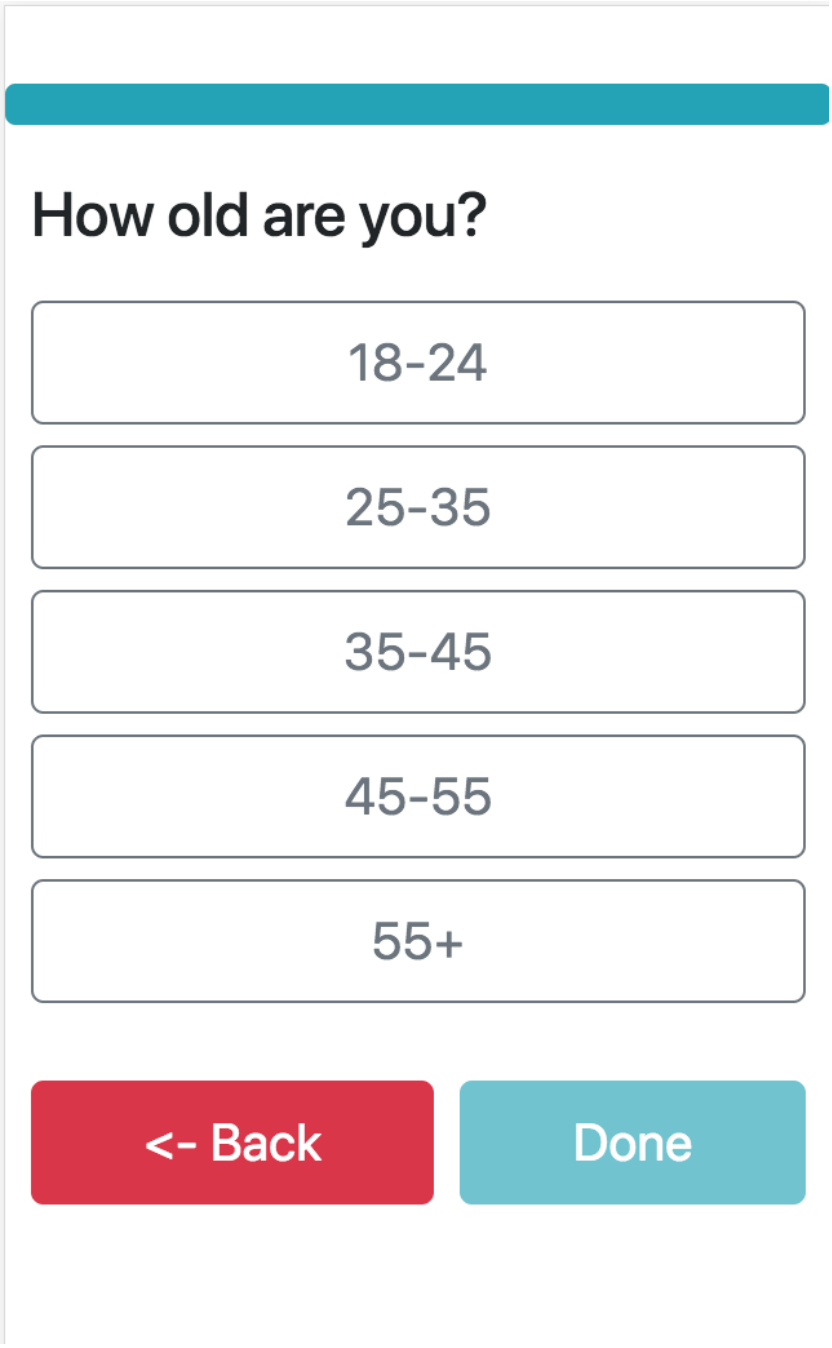

Figure 5.27: This is a screenshot of the EMA web-app, which shows the updated question