COSC 331
Lab 1: Building Our First Microservice

In this introductory lab, we'll be using Java to build a simple microservice from the ground up, including our own web server, API, and backend logic.

The Design of the Microservice

The microservice we plan to build is a dynamic CSS service, allowing custom CSS files to be generated and returned on demand. Such a service could be used, for example, to allow clients or users to customize the interface of a web application to meet their needs – for example, a color-blind user might opt to avoid certain colors, or a user who works primarily in low-light conditions might prefer to use dimmer colors. By designing this as a microservice, we can decouple the logic necessary for the styling change from whatever the underlying application is, while still only requiring one CSS template file for all users.

For this microservice, we will need to develop both the backend logic, which handles the actual reading of the template, and the substitution of our user-selected parameters:

- **main** – the main background color, in hex-code format (i.e. #D5D5D5)
- **accent** – a secondary color used for accents, in hex-code format
- **font** – a generic font family, one of ('serif', 'sans-serif', or 'monospace'), which is applied to the entire document

Each parameter has an appropriate placeholder in the CSS template file, denoted by the underscores around it's name:

- **_MAIN_COLOUR_**
- **_ACCENT_COLOUR_**
- **_FONT_FAMILY_**

When complete, your microservice should take these parameters as part of an HTTP GET request query string from the user/client, like this:

http://127.0.0.1/?main=d5d5d5&accent=1f1f1f&font=monospace

It will then parse out the values from the request, substitute them in the appropriate placeholder spot in the CSS, and then return the newly-generated CSS back to the user or client. The client can then use this generated CSS to style a page with their custom style.

Materials & Implementation

I have provided some files for you to get started with. This includes:

- **Template.css** – the basic CSS template that will be used as the basis for our custom generated CSS
- **Service.java** – This file includes a very basic implementation of a Java HTTP server using sockets, as well as skeleton signatures for the methods you will need to fill out to complete the lab
- **Demo.html** – This is a demonstration HTML file. It allows the user to select the main and accent colors, as well as a font type. When submitted, this page will use your microservice to generate a new CSS stylesheet, and then re-render to display the changed style. You can use this for testing your code.

The lecture two notes contain most of the code that you will need to complete this lab. I recommend that you re-read them carefully, as the code is split up across slides. Some of the implementation, such as the validation of the font input, is left up to you. In addition, the Service.java file includes TODO comments explaining what each function should be doing in your microservice.

You can run your microservice in the IDE, and then visit 127.0.0.1 or localhost to see what kind of response your code will generate (if any), as well as what data the client passes to the service. Make sure you shut down your service between runs, as otherwise your service will continue to listen for (and respond to) requests made to port 80 on your machine.

I recommend that you use either Chrome, Firefox, or a similar browser for this lab. Edge appears to work, but occasionally has hiccups when trying to communicate with our service.

Tasks

To complete this lab, you must first implement the following functions, signatures and descriptions for which are available in the comments of the service.java file, as well as examples in the lecture notes:

- **readCSS()** – reads the template CSS file into a String, and then returns it. This function should also minify the CSS.
- **substituteCSS()** – takes the template CSS and the three parameters and substitutes the parameters into the CSS, before returning the new CSS
- **verifyHex()** – takes a raw user input and verifies it is a hex number with between one and six digits, or returns the default value
- **verifyFont()** – takes a raw user input and verifies it is one of the three appropriate font families, or returns the default value

- **webServe()** – the webserver function. A basic web server has been provided, but you will need to modify it to appropriately handle the input and output into your other functions
- **parseHeader()** – takes raw input from the web server and parses out a specific parameter from the query string

I would recommend that you implement and test most of the functions first, before then integrating them into the webServe() function, as otherwise you'll likely end up with unexplained web errors that actually originated in your backend functions.

After you've finished your implementation, thoroughly test your code using the demo.html file, ensuring that the main and accent colors, as well as the font, change as expected. If one of them isn't working, go back and check your implementation.

BONUS: If you'd like a bonus mark, you can implement an additional feature. Either change an existing value in the template CSS file, or add a new one, with a new placeholder value. Include an appropriate parser call, validation function (if necessary), and substitution function step so that the user has an additional CSS parameter they can change. You can test this by manually entering the new parameter as part of a query string:

http://127.0.0.1/?myNewParameter=bold&main=5d5d5d

You do not have to modify the demo.html file to accept your new parameter, although you are welcome to do so.

Submission

For this lab, you should submit your completed **service.java** file on Moodle. If you also did the bonus work, you should also include your modified **template.css** file. Each implemented function is worth 1 mark each, with the exception of the webServe function, which is worth three marks. Marks will be deducted for implementations with bugs, or which don't respond or behave correctly. The bonus work is worth 1 additional mark.

Marks	Item
1 Mark Each	readCSS(), substituteCSS(), verifyHex(), verifyFont(), parseHeader()
3 Marks	webServe()
1 Mark (Bonus)	Bonus parameter implementation