Lab Exercise: Responsive Web Layout

Part 1

Objective:

- To construct a responsive web layout utilizing CSS Flexbox properties and media queries, as well as JavaScript.

1. In the existing web app project, create a new Pug file. Name this file 'responsivepage.pug'. Structure the body section of your webpage according to the provided layout:

```
div#header Header Section
nav#menu
    label#menu-icon &#9776;
    ul#menulist
        li: a(href='/') Menu 1
        li: a(href='/') Menu 2
        li: a(href='/') Menu 3
div.container
    div#section1
        h3 Section 1
        p Some text..
    div#section2
        h3 Section 2
        p Some text..
    div#section3
        h3 Section 3
        p Some text..
```

2. Create a new CSS file and name it 'responsive.css'. The CSS below contain rules that adjust the layout based on the viewport's size. The web layout will differ for viewports that are less than 600px wide compared to those that are 600px or wider.

```
* {
    box-sizing: border-box;
}

.container {
    display: flex;
    flex-wrap: wrap;
    padding: 10px;
}
```
	* { box-sizing: border-box; }: This rule sets the box-sizing property for all elements (*) to border-box. This means the padding and border of an element are included in the element's total width and height.
	
	.container { display: flex; flex-wrap: wrap; padding: 10px; }: This rule applies to elements with the class container. It sets the display property to flex, which makes the element a flex container. The flex-wrap: wrap; allows the items to

```
#header,
#section1,
#section2,
#section3 {
    width: 100%;
    padding: 10px;
}
```
	wrap onto multiple lines if there is not enough space in the container. The padding: 10px; adds a 10px padding around the container.

```
@media only screen and (min-width: 600px) {
    /* For tablet and desktop: */
    #section1,
    #section2,
    #section3 {
        width: 33.33%;
    }
}

#header {
    background-color: #2d3142;
    height: 150px;
    color: #ffffff;
}

#section1 {
    background-color: #75b9be;
    height: 250px;
}

#section2 {
    background-color: #d0d6b5;
    height: 250px;
}

#section3 {
    background-color: #ee7674;
    height: 250px;
}

ul {
    list-style-type: none;
    padding: 0;
    margin: 0;
    background-color: #bfc0c0;
    overflow: hidden;
}
```
	@media only screen and (min-width: 600px) { ... }: This is a media query that applies the enclosed styles only when the viewport is 600px wide or wider. Inside this media query, the width of #section1, #section2, and #section3 is set to 33.33%, meaning they will each take up one-third of the container's width when the viewport is at least 600px wide.

```css
li {
    float: left;
}

li a {
    display: block;
    color: #000000;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
}

li a:hover {
    background-color: #fcfc62;
}

#menu-icon {
    display: none;
}

@media only screen and (max-width: 600px) {
    #menu-icon {
        display: block;
        cursor: pointer;
        font-size: 30px;
        padding: 10px;
    }

    #menulist {
        display: none;
    }

    #menulist li {
        float: none;
    }
}
```

The #menu-icon class is initially set to display: none;, meaning it is hidden.

@media only screen and (max-width: 600px) { ... }: This is another media query that applies the enclosed styles only when the viewport is 600px wide or narrower.

Inside this media query, the #menu-icon is set to display: block;, making it visible, and #menulist is set to display: none;, hiding the menu list.

This is likely part of a responsive design where a menu icon is shown on smaller screens and the full menu list is hidden.

3. Create a new JavaScript file for the client side and name it ‘responsive.js’. This script will enable a functionality where a user can click on a menu icon to toggle the visibility of a menu. If the menu is currently hidden, clicking the icon will display it. If the menu is currently visible, clicking the icon will hide it.

```
document.getElementById('menu-icon').addEventListener('click', function () {
  var menu = document.getElementById('menulist');
  if (menu.style.display === 'none') {
    menu.style.display = 'block';
  } else {
    menu.style.display = 'none';
  }
});
```

document.getElementById('menu-icon'): This line gets the HTML element with the ID ‘menu-icon’. This is likely the menu icon that users click on to show or hide the menu.

.addEventListener('click', function() {...}): This line adds a ‘click’ event listener to the menu icon. This means that the function provided will be executed every time the menu icon is clicked.

var menu = document.getElementById('menulist');: Inside the function, this line gets the HTML element with the ID ‘menulist’. This is likely the menu that we want to show or hide.

if (menu.style.display === 'none') {...} else {...}: This is an if-else statement that checks the current display style of the menu. If the display style is ‘none’, meaning the menu is currently hidden, it changes the display style to ‘block’, which shows the menu. If the display style is not ‘none’, it must be ‘block’, meaning the menu is currently shown, so it changes the display style to ‘none’, which hides the menu.

4. Incorporate the external CSS and JavaScript files you've created into your Pug file. For the JavaScript file, ensure that the link is placed after the last `div` element in your Pug file.

Part 2

Objective:
- To set up and test a local network connection between your mobile phone and laptop
- To access an Express app running on your laptop from your mobile phone.

1. On your mobile phone, go to the settings and turn on the mobile hotspot. This will create a Wi-Fi network that other devices can connect to.

2. On your laptop, go to the Wi-Fi settings and connect to the mobile hotspot you just created. Your laptop is now connected to the same network as your mobile phone.

3. Start your Express app.

4. You need to find the IP address that your laptop is using on the mobile hotspot network. Here's how to do it:
   - On **Windows**, open Command Prompt and type ***ipconfig***. Look for the "IPv4 Address" under the wireless LAN adapter that you're connected to.
   - On **Mac**, open Terminal and type ***ifconfig***. Look for the "inet" address under the en0 or en1 entry.
   - On **Linux**, open Terminal and type ***hostname -I***.

5. Open a web browser on your mobile phone and type in the IP address of your laptop (found in step 4), followed by the port number that your Express app is running on. For example, if your IP address is 192.168.43.217 and your Express app is running on port 3000, you would type 192.168.43.217:3000 into your mobile web browser.

6. On your mobile phone, open the responsivepage.pug page (created in Part 1) from your Express app using your laptop's IP address. Rotate your phone or resize the browser window to verify that the layout adapts correctly to different screen sizes.

Part 3

Objective:

- To practice deploying a local development server with Ngrok, generate secure public URLs, and test web applications on external devices or networks.

Ngrok is a tool that allows you to expose a local server to the internet securely. It's especially useful for testing webhooks, sharing work-in-progress websites, or accessing local apps remotely.

1. Install Ngrok
   - Go to https://ngrok.com and sign up for a free account.
   - Download the appropriate version for your OS.
   - Unzip and place the executable in a directory accessible from your terminal (e.g., add it to your PATH).

2. Connect Your Account
   After installation, run this command to connect your account:

   ```
   ngrok config add-authtoken <your_auth_token>
   ```

   You can find your auth token in your Ngrok dashboard.

3. Start a Tunnel
   To expose a local server (e.g., running on port 3000), use:

   ```
   ngrok http 3000
   ```

   This will generate a public URL (e.g., https://c6fd137d26f1.ngrok-free.app) that forwards requests to your local server. The public URL will be displayed in the terminal.

4. Ensure that your local Express server is running. The Ngrok public URL will not work unless your local server is active.

5. You can now use the generated URL to:
   - Share your local app with others
   - Access your web app remotely
   - Test your local web app on real mobile devices, even if they're on a different network.

6. On your mobile phone, open the responsivepage.pug page (created in Part 1) using the Ngrok public URL generated in Step 3. Rotate your phone or resize the browser window to verify that the layout and menu toggle work correctly across different screen sizes.