Project 7: Web App Dashboard

Table of Contents

Sections of this Guide

- **How to approach this project** includes detailed guidance to help you think about how to organize your code, project and files.
- **How to succeed at this project** lists the grading requirements for the project, with hints, links to course videos to refresh your memory and helpful resources.

How to Approach This Project

There are a number of ways to approach and complete this project. Below, one approach is detailed in steps, but you do not have to follow this approach to the letter. You are free to

explore your own solution, as long as it meets the requirements laid out in the "How to succeed at this project" section below.

Helpful Hint:

This is the largest project you will have built to this point. It combines a lot of elements you have learned from previous projects, and introduces new concepts like CSS grid. It is important to break things into smaller, manageable sections, as well as testing your code often.

Step 1: Setup

Download and extract the project files. Inside the project folder there should be:

- A PNG mockup for the project
- An icon folder containing:
    - 8 SVGs to use for the navigation and social widget
- An images folder containing:
    - 4 profile pictures for use in the new members and recent activities sections.

Set up a new GitHub repository and push the project files to it.

- *Related video:* **Share Your Projects with GitHub**

NOTE: **The following detailed walkthrough is just one way to go about making this project... there are plenty of other methods out there that are just as good! If you have a different way you would like to use, by all means give it a try!**

Step 2: HTML

The first order of business is to create the HTML for the project. This project is a little bigger than those we have dealt with so far. It will be important to look at the overall layout and containers first before diving into the smaller components.

- Since your three main areas will be styled using CSS grid, structure these items first, using semantic tags where possible. Something like this will work nicely:

```
-- header
-- nav (for the sidebar navigation)
-- main
```

- Build out the HTML for each of these three major sections starting with the header. At its simplest level, the header has four elements in it, the title, the bell icon, a profile picture, and a profile name. One approach could be as follows:

```
-- header
    -- h1
    -- div (to contain the bell icon)
        -- svg
        -- div#notifications (container element for our notifications)
    -- img
    -- h3 (or similar element)
```

- After building the header, it's time to move to the sidebar navigation. Since the project does not require you to use the SVGs inline for this portion, you can use the `img` tag, or even set them as the background picture for your links. Using the first method, the HTML breakdown may look something like this:

```
-- nav
    -- a
        -- img
```

- The final portion of the main layout is the widget container. This section will be a little tricky, and there are many different ways of tackling it. It might be helpful to think of these areas in terms of rows and columns. In this type of grid, there would be some sections that have 1 column which spans the whole row, like the "Dashboard" heading, Traffic chart, and Social sections. The rest of the widgets fall into the category of each row having 2 columns.
    - **Note:** The rows shown below are for visualization only. We will be using CSS grid to control the columnar layout so we don't actually need separate HTML elements for the rows.

```
-- main
    -- row
        -- col 1
    -- row
         -- col 1
         -- col 2
```

- Now that we added the major parts of the widget container, we can break down the markup for each individual widget.

Note: Now that we have reached small components, the examples will be in actual HTML

- Dashboard header
    - This will include things like the word “Dashboard” displayed in a headline tag, like an `h2` and an `input` for the search bar.
    - Since the space between the word “Dashboard” and the search bar will change, we may want to wrap it in a separate `div` so that we can control the spacing easier with CSS.

```
<div class="main-header"> // row
    <h2 class="headline">Dashboard</h2>
    <div class="main-header-search">
        <input type="search" placeholder="Search..." />
    </div>
</div>
```

- Alert Bar
    - The alert bar will be a simple `div` container, because we will add the actual alert with JavaScript.
    - Make sure to give the `div` an `id` attribute so that we can easily reference it in the JS.

```
<!-- alert banner -->
<div class="alert" id="alert" >
    <!-- JS will insert alert here -->
</div>
```

- Traffic Section: Line graphs
    - The traffic section is a little challenging, it needs to have its own navigation component in its header along with the normal `h3` displaying “Traffic”.
        - Use a `ul` and `li`s to build out the navigation links for “Hourly”, “Daily”, “Weekly”, and “Monthly”
        - Don’t worry about `a` tags and `href`s, you will use JS for the actual navigation if you go for “exceeds expectations.”

- Create a container to hold the `canvas` element. This will be important later because of the way chart.js handles responsiveness
    - Give the `canvas` an `id` so we can use it in JS
    - *External Resource:* **Chart.js: Responsive Charts**

```
<!-- line graph charts -->
<section class="traffic">
    <div class="traffic-header">
        <h3>Traffic</h3>
        <ul class="traffic-nav">
            <li class="traffic-nav-link">Hourly</li>
            <li class="traffic-nav-link">Daily</li>
            <li class="traffic-nav-link active">Weekly</li>
            <li class="traffic-nav-link">Monthly</li>
        </ul>
    </div>
    <div class="widget-container-full">
        <!-- element line graph will attach to -->
        <canvas id="traffic-chart"></canvas>
    </div>

</section>
```

- Traffic Section: Bar and Doughnut Charts
    - We will need two separate `section` elements for the bar and doughnut charts.
        - Each section will need an `h3` for the title
        - Like the line graph section, we will need to wrap the `canvas` element for the graph in a container.
            - Make sure to give each canvas an id so we can reference it in the JS!
            - *External Resource:* **Chart.js: Responsive Charts**
        - Example is for the bar chart, the doughnut chart is similar

```
<!-- bar graph -->
<section class="daily">
    <h3>Daily Traffic</h3>

    <div class="widget-container-half">
```

```
        <!-- element bar graph will attach to -->
        <canvas id="daily-chart"></canvas>
    </div>
</section>
```

- Social Status
    - The social status section will be three containers
    - Each container will hold an inline SVG, and the text elements. For ease of styling you may want to wrap the 2 text pieces in their own container
        - Related video: [Embedding SVG XML]

```
        <div class="social-container">
            <div class="social-image"
            <!-- inline SVG not included for brevity -->
            </div>
            <div class="social-text">
                <p>Twitter</p>
                <p>10,345</p>
            </div>
        </div>
        <!-- add two more social containers and child
        elements for other social media items -->
    </div>
</div>
```

- New Members and Recent Activity
    - Both of these sections have almost the same HTML elements, just different text content.
    - Start with a `section` to contain each member.
    - You will need an `h3` that gives the section title.
    - Start with a `div` container for each person.
    - Inside of each container should be an image, another `div` to wrap the text, and finally a `p` tag to handle the date or the greater than arrow.
        - The text container for new members will have a `p` element for the name and an `a` element for the email.

        - The text container for recent activities will have 2 `p` elements, one for the comment, and one for the date.
    - The markup shown below is for a new member with sample data, recent activity is similar

```
        <!-- new members widget -->
        <section class="members">
            <h3>New Members</h3>

            <div class="members-container">
                <img src="images/Sept2018Profile.jpg"
class="profile-image">
                <div class="members-text">
                    <p>Victoria Chambers</p>
                    <a href="#">victoria.chambers80@example.com</a>
                </div>
                <p>10/15/15</p>
            </div>
```

- Messaging
    - HTML for the messaging section is very straightforward.
    - Use a `section` for the widget container like the other widgets
    - Add an `h3` with the section title: “Message User”
    - Add a `form` element, inside of it you will need:
        - A text type input for who the message is for
        - A textarea for the message
        - A button to “send” the message

```
        <!-- messaging widget -->
        <section class="message">
            <h3>Message User</h3>

            <form class="widget-container">
                <input type="text" placeholder="Search for user"
class="form-field" id="userField">
                <textarea placeholder="Message for user" class="form-area"
id= "messageField" ></textarea>
                <button class="button-primary" id="send" >Send</button>
            </form>
        </section>
```

- Settings
    - Use a section for the widget container like the other widgets
    - Add an h3 with the section title: “Settings”
    - Follow the directions for the custom CSS toggle switches found in the external links of the project instructions.
        - *External link:* **Custom CSS Toggle Switches**
        - The only file you need is the stylesheet inside the dist folder
    - Use a select with option elements for the timezones
    - Create a container to wrap the “save” and “cancel” buttons for easier styling.

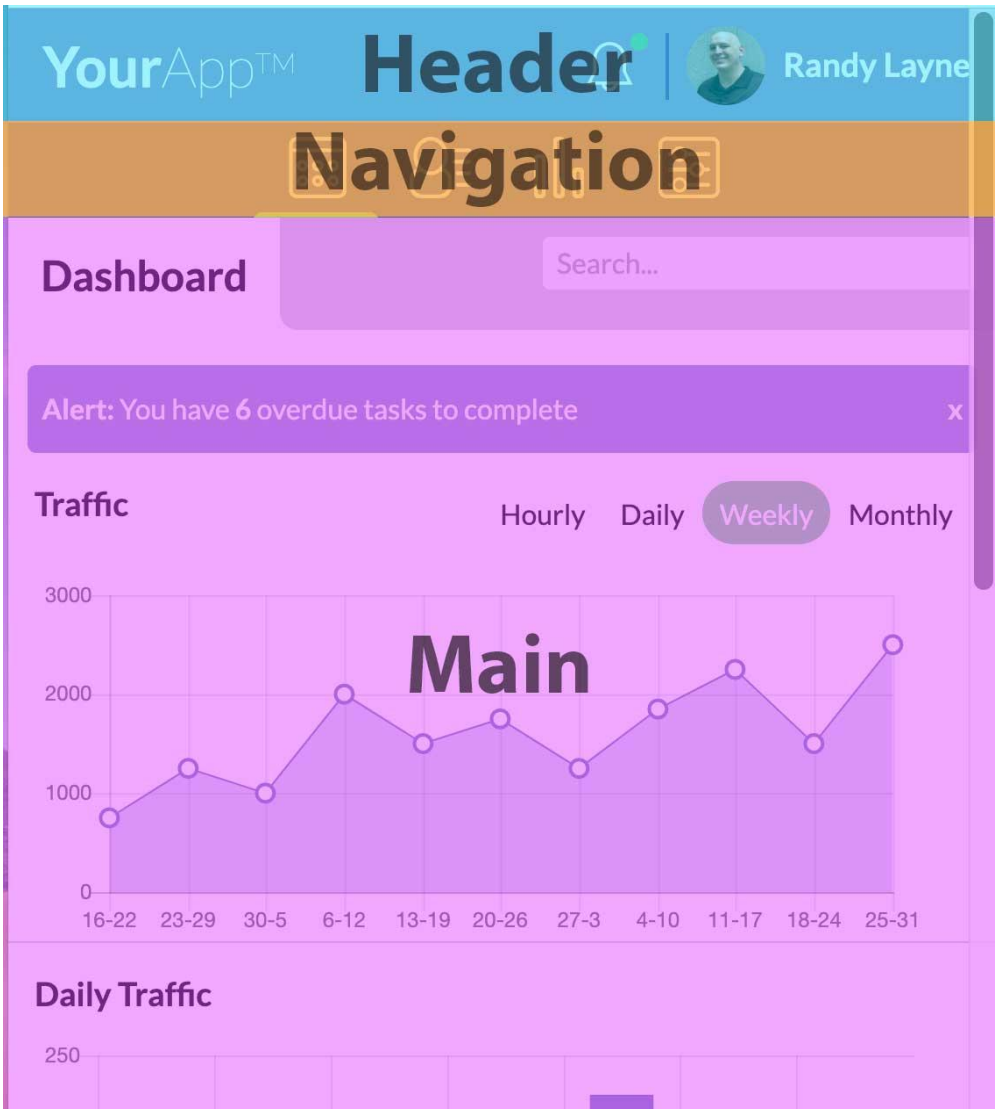
```
<!-- local settings widget -->
<section class="settings">
    <h3>Settings</h3>

    <!-- custom CSS toggle code-->

    <!-- custom CSS toggle code-->

    <select class="form-field" id="timezone">
        <option disabled selected>Select a Timezone</option>
        <option>Eastern</option>
        <!-- more options -->
    </select>
    <div class="settings-button" >
        <button class="button-primary" id="save">Save</button>
        <button class="button-disabled" id="cancel">Cancel</button>
    </div>
</section>
```

Note: You will add the actual charts later, so it may be helpful to use placeholder images of graphs as you are building out the HTML and CSS. This will help you define the space between the charts and surrounding elements.

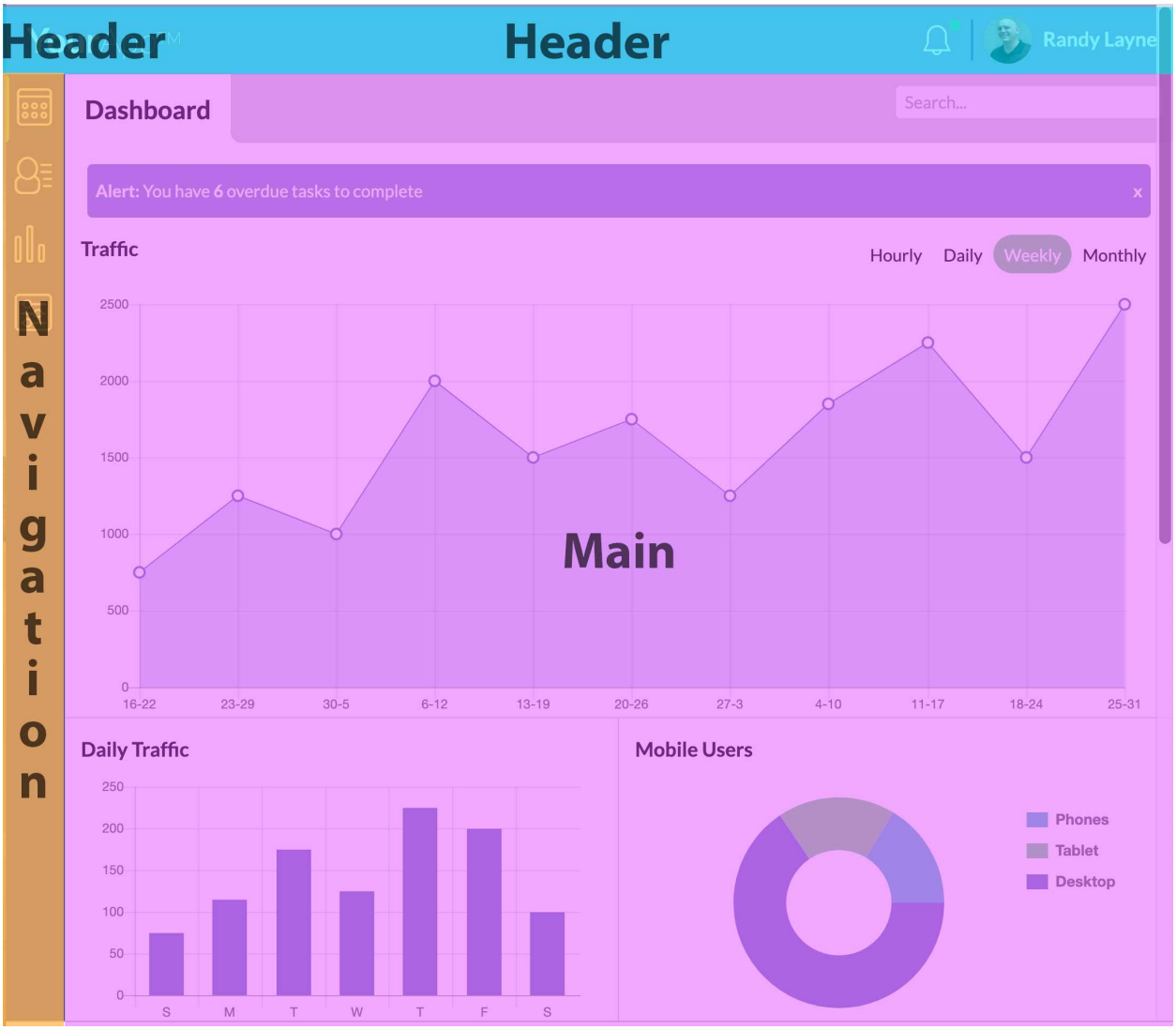
Step 3: CSS

Most of the styling used for this project is in line with what you have learned in previous projects. The main areas we will focus on for the styling of this project is the CSS grid portions, creating the 2 column layout on desktops using CSS grid, and making the graphs responsive.

- CSS Grid container for major components.
    - The main container of the project is the body element, which should be given a descriptive class name like “grid-container” in the HTML.
    - Start by breaking up the major elements on the page and decide how they should look on both mobile and desktop versions.
      Here are some suggested grid template areas for mobile and desktop views:



Mobile



Desktop

- Before we can start adding any CSS Grid styling, we need to change the ".grid-container" class's display property to grid.
- Next, for each of the elements that represent the header, main, and navigation container elements, we will need to add a grid-area property to them with the name of the area as the value, i.e. "header."

Reminder: We always use mobile-first techniques in the Techdegree!

- *Related video:* [Using a Mobile First Approach]

- Heading back to our .grid-container selector, it's time to add the grid layout properties for our mobile view.
  - Looking at the screenshot above,we all content should be in a single column layout for mobile view, and the rows are different heights based on the content. So, we will make our `grid-template-columns property` 100%, and our `grid-template-rows` set to `auto`.
  - Using the name of the areas we set in the `grid-area` properties, we'll set the grid-template-area property to have each area stacked above each other.
- Now that the mobile styles are set, it's time to work on the desktop view. Since we are using CSS Grid Layout, all the sizing and changes can be done straight from the `.grid-container` class.
  - First things first, you will need to add a min-width media query set to 768px.
  - Next, create a selector for the .grid-container class.
    - From the desktop screenshot, we can see that the first column should be fixed-width and the second column should adjust to fill the available space. So, we change our `grid-template-columns` property to reflect this, with something like: `55px 1fr`
    - The only thing left is to redistribute our grid-template-areas. Note that the header takes up both columns of the first row.

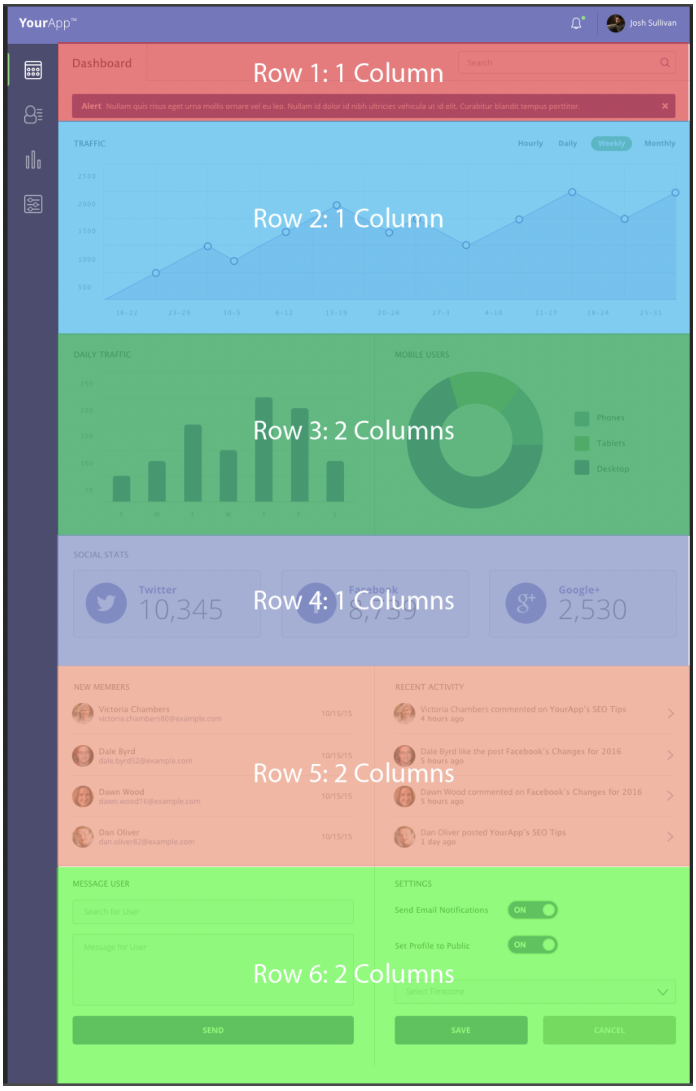
```
.grid-container {
  display: grid;
  grid-template-columns: 100%;
  grid-template-rows: auto;
  grid-template-areas:
      "header"
      "navigation"
      "main";
}

.header {
  grid-area: header;
}

.navigation {
  grid-area: navigation;
}

.main {
```

```
  grid-area: main;
}

@media (min-width: 768px) {
  .grid-container {
    grid-template-columns: 55px 1fr;
    grid-template-areas:
        "header header"
        "navigation main";
  }
}
```

- Use CSS Grid to make a 2 column layout on desktop screens
    - We will be using a slightly different method for our CSS grid on the widgets than for the overall layout. Instead of using grid template areas like we did for the overall layout, we will use a more flexible method that puts more control on the grid items than the grid container
    - main element
        - Start by making the `main` element a grid container by setting its `display` property to `grid`.
        - For desktop screen sizes we will need to change from one column wide to two equal sized columns. We do this by setting the `grid-template-columns` property to `1fr`

```
.main {
  display: grid;
}

@media (min-width: 768px) {
  .main {
    grid-template-columns: 50% ;
  }
}
```

    - Use the following screen shot to get an idea of which widgets need to be 1 column wide and which need to be 2:



- For the line graph and social media areas, the widget spans the width of 2 columns.
    - For desktop sizes, set the grid-column property to the first position (or grid line) and have it span two columns.

```
@media (min-width: 768px) {
      .traffic, .social {
            grid-column: 1 / span 2;
      }
}
```

- For all the bar chart, new members, and messaging widgets, they are in the first column position and only span one column width

```
@media (min-width: 768px) {
    .daily, .members, .message {
        grid-column: 1 / span 1;
    }
}
```

    - For the doughnut chart, recent activities, and settings widgets, they are in the second column position and span one column width.

```
@media (min-width: 768px) {
    .mobile, .activity, .settings  {
        grid-column: 2 / span 1;
    }
}
```

- Make the chart widgets responsive to screen resizing
    - Because of the way Chart.js handles resizing of the charts, it does not work well when the parent container of the canvas element uses percentage based widths.
    - In order to make the canvas container responsive, use the `vw` unit instead, which is a percentage of the browser window's width. For instance, a `width` of `70vw` would be 70% of the width of the browser
    - To account for the navigation element on the left, we can't use a full `100vw` on desktop screen sizes.
        - For mobile versions, all three charts can have the same width.
            - Start with `95vw` and adjust from there.
        - Desktop versions:
            - The line chart should have a `width` of about `90vw` to allow room for the navigation
            - The bar chart and doughnut charts should be about half the size of the line chart, or about `45vw`.
            - **Note:** Other styling like `padding` and `margin` may require you to adjust these widths, they are merely a starting point.

```
.widget-container-full, .widget-container-half {
    width: 95vw;
}

@media (min-width: 768px) {
    .widget-container-full {
        width: 90vw;
    }
    .widget-container-half {
```

```
        width: 45vw;
    }
}
```

- *External link:* [Chart.js: Responsive]

Step 4: JS - Alert Banner

- Alert Banner
  - The first thing we need to do is create a variable to store the div#alert element. The easiest and most efficient method is to use getElementById.
  - Since we only have the empty div in our index.html file, we will add the markup now using the innerHTML property of our alert element and a template literal to make the markup.

```
  const alertBanner = document.getElementById("alert");

  // create the html for the banner
  alertBanner.innerHTML =
    `
    <div class="alert-banner">
      <p><strong>Alert:</strong> You have <strong>6</strong> overdue tasks
to complete</p>
      <p class="alert-banner-close">x</p>
    </div>
    `
```

  - Add a “click” event listener to the alert banner.
    - Since you will be adding the markup using template literals, the close button doesn’t exist in the DOM yet so we will need to use event bubbling
    - Use a conditional (if statement) to determine if the user clicked the actual close button or just somewhere inside the alert banner.
    - Set alert’s display value to “none”

```
  alertBanner.addEventListener('click', e => {
    const element = e.target;
    if (element.classList.contains("alert-banner-close")) {
      alert.style.display = "none"
    }
```

```
});
```

- *Related video:* [Introducing Template Literals]
- *Related video:* [Practice Selecting Dom Elements]
- *Related video:* [Introducing Arrow Function Syntax]

Step 5: JS - Chart Widgets

Note: Working with deeply nested Object literals can get confusing quickly! We will be substituting in variables for different options to help make things more readable.

- Line Graph
  - Store the canvas element with the id of "traffic-chart" in a variable named trafficCanvas.
  - Create an Object literal representing the data for the traffic chart
    - For the labels key, enter an array of values using the mockup as a guide.
    - Create a datasets key
      - Add a data key. For its value enter an array of numbers using the mockup as a guide.

      Note: there must be the same number of values as there is in the labels array
    - Create keys to change different properties of the data set. For example:
      - backgroundColor
      - borderWidth

```
 let trafficData = {
    labels: ["16-22", "23-29", "30-5", "6-12", "13-19", "20-26", "27-3",
"4-10", "11-17", "18-24", "25-31"],
    datasets: [{
      data: [750, 1250, 1000, 2000, 1500, 1750, 1250, 1850, 2250, 1500,
2500],
      backgroundColor: 'rgba(116, 119, 191, .3)',
      borderWidth: 1,
    }]
  };
```

  - Create an Object literal to set the options you want to change for the chart
    - Set the scales option to start the chart at zero
    - Hide the legend

- **Recommended:** set the animation duration to zero to make chart more responsive
- Other options to personalize it

```
let trafficOptions = {
  aspectRatio: 2.5,
  animation: {
    duration: 0
  },
  scales: {
    y: {
      beginAtZero: true
    }
  },
  plugins: {
      legend: {
      display: false
    }
  }
};
```

- Create the chart itself.
  - Create a variable named trafficChart and set it equal to a new instance of Chart
  - In the constructor call, pass the trafficCanvas variable so that the graph knows which canvas element to use to display the graph
  - As the second argument for the constructor, create an object literal.
    - Create a type key and set the value to “line”
    - Create a data key and set it equal to trafficData
    - Create an options key and set it equal to trafficOptions

```
let trafficChart = new Chart(trafficCanvas, {
  type: 'line',
  data: trafficData,
  options: trafficOptions
});
```

- *External Link:* **Chart.js: Usage**
- *External Link:* **Chart.js: Line**
- *External Link:* **Chart.js: Elements Configurations**

- Bar Graph
    - Creation of the bar graph is very similar to the line graph.
    - Store the canvas element with the id of “daily-chart” in a variable named dailyCanvas.

```
const dailyCanvas = document.getElementById("daily-chart");
```

    - Create Object literals for the data and options like you did for the line chart
        - Only the label and data arrays need to be different, following the mockup as a guide.

```
// data for daily traffic bar chart
const dailyData =  {
  labels: ["S", "M", "T", "W", "T", "F", "S"],
  datasets: [{
      label: '# of Hits',
      data: [75, 115, 175, 125, 225, 200, 100],
      backgroundColor: '#7477BF',
      borderWidth: 1
  }]
};

const dailyOptions = {
  scales: {
    y: {
        beginAtZero: true
    }
  },
  plugins: {
      legend: {
      display: false
    }
  }
};
```

    - Create the chart itself
        - Follow the example for the line chart substituting in the bar(daily) chart variables.
            - Give the type key a value of “bar”

```
let dailyChart = new Chart(dailyCanvas, {
  type: 'bar',
  data: dailyData,
  options: dailyOptions

});
```

    - *External Link:* **Chart.js: Usage**
    - *External Link:* **Chart.js: Bar**
    - *External Link:* **Chart.js: Elements Configurations**
- Doughnut Chart
    - Store the canvas element with the id of “daily-chart” in a variable named mobileCanvas.

```
const mobileCanvas = document.getElementById("mobile-chart");
```

    - Create an Object literal for the data
        - The data Object is similar to the Line and Bar charts, just with different array data for the label and data keys

```
const mobileData = {
  labels: ["Desktop", "Tablet", "Phones"],
  datasets: [{
    label: '# of Users',
    data: [2000, 550, 500],
    borderWidth: 0,
    backgroundColor: [
      '#7477BF',
      '#78CF82',
      '#51B6C8'
    ]
  }]
};
```

    - Create an Object literal for the options
        - Unlike the bar and line charts, you will need to set options for the legend
            - For the legend key, create an Object literal to set the position to right.

- There are a lot of options for the labels to fine tune the size and style of the legend labels.

```
const mobileOptions = {
  plugins: {
    legend: {
      position: 'right',
      labels: {
        boxWidth: 20,
        fontStyle: 'bold'
      }
    }
  }
};
```

- Create the chart itself
  - Follow the example for the line chart substituting in the bar(daily) chart variables.
    - Give the type key a value of “doughnut”

```
let mobileChart = new Chart(mobileCanvas, {
  type: 'doughnut',
  data: mobileData,
  options: mobileOptions
});
```

- *External Link:* **Chart.js: Usage**
- *External Link:* **Chart.js: Doughnut**
- *External Link:* **Chart.js: Elements Configurations**
- *External Link:* **Chart.js: Legend Configuration**

Step 6: JS - Messaging Section

- Create variables to store the form fields.
  - User field
  - Message field
  - Send button

```
const user = document.getElementById("userField");
const message = document.getElementById("messageField");
const send = document.getElementById("send");
```

- Create a click event listener on `send`
    - Create a conditional to check if both the `user` and `message` fields are empty
        - If so, create an `alert` dialogue to inform the user that both fields must be filled out
        - Otherwise, if only the user field is empty, alert the user that the user field must be filled out
        - Otherwise, if only the message field is empty, alert the user that the message field must be filled out
        - Otherwise alert the user that the message was sent.

```
send.addEventListener('click', () => {

  // ensure user and message fields are filled out
  if (user.value === "" && message.value === "") {
    alert("Please fill out user and message fields before sending");
  } else if (user.value === "" ) {
    alert("Please fill out user field before sending");
  } else if (message.value === "" ) {
    alert("Please fill out message field before sending");
  } else {
    alert(`Message successfully sent to: ${user.value}`);
  }
});
```

How to Succeed at This Project

Here are the things you need to do pass this project. Make sure you complete them before you turn in your project.

Mobile First

- ❑ The HTML file includes the viewport meta tag in the head of each document.
- ❑ A mobile-first approach is utilized using `min-width` properties for media queries.
- ❑ Appropriate media queries are in place and the content responds to mobile (320px), tablet (768px) and desktop (1024px) screen sizes.
- ❑ Use CSS grid to lay out the main elements (header, sidebar navigation, main content) on the page.

Alert Notifications

- ❏ Includes alert banner that user can close.
- ❏ Includes an alert icon in the header with a marker to notify the user of new alerts and messages.
- ❏ Add a CSS transition to the bell icon when the user hovers over it.

Chart Widgets

- ❏ Successfully implements chartjs.org for the charts:
  - ❏ Web Traffic (line chart)
  - ❏ Daily Traffic Bar Chart (bar chart)
  - ❏ Mobile User Pie Chart (donut chart)
- ❏ General spacing and arrangement of the elements matches the layout of the widgets in the mockup.

Social Network Information

- ❏ Includes a widget (or three separate widgets) that display social network stats for three social networks.
- ❏ SVG icons are added as inline SVG's.
- ❏ SVG fill colors have been changed to match the mockups.
- ❏ General spacing and arrangement of the elements match the layout of the widget in the mockup.

New Members and Recent Activity Listing

- ❏ Includes a widget that lists new members and displays an avatar, member name, email and join date for each member.
- ❏ Includes a Recent Activity widget that displays an avatar, type of activity, and time since activity for each member.
- ❏ General spacing and arrangement of the elements matches the layout of the widgets in the mockup.

Message User Widget

- ❏ Implement a messaging widget that includes the following:
  - ❏ A field for searching for a user. Real search functionality is not required.
  - ❏ A message textarea field that lets a user add a message.
  - ❏ A “Send” button that uses JS to allow a user to submit the form and display a confirmation the message was sent

- ❏ Uses JS to display error messages if both or either the *user* or *message* field is empty.
- ❏ General spacing and arrangement of the elements matches the layout of the widget in the mockup.

Settings Widget

- ❏ The settings widget has been created and displays the following settings options:
  - ❏ An on/off widget for whether to send email notifications.
  - ❏ An on/off widget for whether to set profile to public or private.
  - ❏ A dropdown to select timezone options.
  - ❏ Save and Cancel buttons (these do not have to do anything functional for “meets expectations”).