Redogörelse individuell uppgift 2, Tom Ederstål.

Inledning

Mitt mål med denna uppgift var att skapa en porföljsida som ser elegant och simpel ut. Jag ville också utveckla den på ett sätt som gör det enkelt för mig att lägga utöka och bygga på när jag har mer att visa.

Styling

Jag började med att kolla på andra utvecklares portföljer för att få en idé om hur jag ville att sidan skulle se ut. Jag funderade på vad jag skulle använda för CSS-förprocessorer och kollade upp SASS som jag tyckte verkade väldigt bra. Det som är intressant med SASS är att det hjälper till att organisera designkoden då man kan deklarera en variabel och ge den styling och sedan kalla den variabeln på de element man vill styla. SASS ger oss även valmöjligheter i hur man vill skriva sin syntax, något som utvecklare saknar i LESS.

Passade även på att läsa mer om olika CSS-ramverk. Började med det jag uppfattar som mest populärt, nämligen Bootstrap. Detta ramverk introducerade responsiv design på en större skala. Det var det första ramverket som främjade mobile first-tänket. Ramverket introducerade dessutom grid som delar upp skärmen i kolumner och rader vilket gjorde det lättare att designa responsivt.

Materialize CSS är Googles egna design-ramverk som de lanserade 2014. Det kommer med färdiga klasser och komponenter. Detta ramverk använder sig av Googles Material Design som är ett språk där man precis som i Bootstrap använder sig av gridbaserad layout. Det kommer även med responsiva animationer samt djup- och ljuseffekter.

Jag valde till slut att jobba med styled components då jag tyckte att det var enkelt att komma in i samt att koden ser snyggare ut i helhet än vanlig css. Jag implementerade även grid för att enklare placera mina komponenter där jag ville.

JavaScript

ECMAScript även känt som JavaScript uppfanns 1995 och har sedan 1997 varit ECMA-standard. Versionerna som släpptes fram till 2015 kallades ES1 till och med ES6. Därefter är de inte numrerade utan helt enkelt döpta ES 2016 - ES 2018. De största uppdateringarna som de flesta pratar om är ES5 och ES6. ES5 introducerade stöd för JSON samt olika väldigt användbara array- och sträng-metoder.

ES6 släpptes med en ny standard för att deklarera variabler. Istället för att skriva “var” så kunde man nu skriva antingen “const” eller “let”. Skillnaden är att en const-variabel skall vara konstant och aldrig ändras medans “let” låter oss ändra värdet på variabeln.

Webbläsaren kan köra JavaScript genom att en har en inbyggd JS-tolk som hittar och läser JavaScripten på sidan och sedan kör den.

Ett JavaScript-bibliotek är som en fil som innehåller redan skrivna funktioner. Detta kan göra arbetet mycket enklare och mer tidseffektivt då man slipper skriva en stor del av koden själv.

Till skillnad från bibliotek där man implementerar redan skrivna funktioner i sin kod får man i ramverk vissa riktlinjer och verktyg. Dessa verktyg gör det enklare för en utvecklare att bygga en hemsida från grunden. Jag har för detta projekt använt mig av ramverket React som är komponentbaserat för att göra det lättare att organisera projektet och dess filer och mappar.

Nedan visar jag min JavaScript kod.

app.js:

```
//api call api.openweathermap.org/data/2.5/weather?q=London,uk
const API_KEY = 'c6eb35300aec01814f790a3987bcac43';


//App
class App extends React.Component {
  constructor(){
    super();
    this.state = {
      city : undefined,
      country : undefined,
    };
    this.getWeather();
  }

  calCelsius(temp){
    let cell = Math.floor(temp - 273.15);
    return cell;
  }

  getWeather = async () =>{
    const api_call = await fetch(`http://api.openweathermap.org/data/2.5/weather?q=Stockholm,se&appid=${API_KEY}`);

    const response = await api_call.json();

    console.log(response);

    this.setState({
      city : response.name,
      country : response.sys.country,
      celsius : this.calCelsius(response.main.temp),
      error : false
    })
  }
```

I koden ovan har jag börjat med att deklarera en variabel i app.js som heter API_KEY för att spara nyckeln som jag fick från OpenWeatherMap. Därefter skapas app-komponenten.

I början av komponenten deklarerar jag mitt initiala state som består av city och country. Eftersom detta API ger mig temperaturer i fahrenheit måste jag skapa en funktion som jag döpte till calCelsius som tar in temp, deklarerar en variabel (cell) och subtraherar temp med 273.15. Därav får jag fram vad temperaturen är i celsius.

getWeather-funktionen är en asynkron funktion som inväntar(await) hämtningen(fetch) från OpenWeatherMap. Asynkron JS eller AJAX(Asynchronous JavaScript And XML) gör det möjligt läsa data från en server efter att hemsidan har laddats, uppdatera en sida utan att ladda om den och skicka data till servern i bakgrunden.

I api-länken kan jag manuellt skriva in vilken stad jag vill visa temperaturen i samt att jag med hjälp av template strings kan använda mig av JavaScript för att inkludera API_KEY i länken.

Därefter deklarerar jag en variabel som jag kallar för “response” som inväntar svaret från api_call i form av Json.

Det sista jag gör i denna kodsnutt är att jag sätter mitt state till att city och country ska vara det jag får i svar från mitt API (response.name, response.sys.country och this.calCelsius(response.main.temp)).

weather.component.jsx:

```
const Weather = (props) => {
return (
    <WeatherContainer className="container">
        <div className="cards">
            <h2>{props.city}, {props.country}</h2>
            <h3>{props.temp_celsius}&deg;C</h3>

        </div>
    </WeatherContainer>
);
};

export default Weather;
```

I denna kodsnutt skapar jag en egen komponent för att rendera mitt API i HTML format samt lägga till ett grader-tecken efter temperaturen. Denna komponent importerar jag sedan till app.js som “weather” och lägger den som en tagg i min render. Detta syns i nästa skärmdump.

```
render() {
  return (
    <BrowserRouter>
    <div className='App'>
      <Banner />
      <Header />
      <About />
      <Projects />
      <Contact />
      <Weather
        city={this.state.city}
        country={this.state.country}
        temp_celsius={this.state.celsius}
        description={this.state.description} />
      </div>
      </BrowserRouter>


  );
}
```

DOM eller Document Object Model är ett plattformsneutralt gränssnitt som dynamiskt hämtar och uppdaterar innehållet, strukturen och designen på en hemsida.

HTTP 1.1 introducerades 1991 och lade grunden till GET, HEAD, PUT och POST requests. HTTP 2.0 introducerades 2015 och kom med nya kompressions algoritmer för att hantera tyngre hemsidor men behålla en snabb hastighet.

Slutligen hostade jag min hemsida via surge.

Länk:
http://tomederstal.surge.sh/