Problem 3: Sample Project - SLR Maine 🌊

Resources and Links

> Please provide the code to a sample project you've programmed as part of your previous work. If possible, please use a sample that you either worked on more independently or participated in the scoping/decision-making.

SLR Maine Backend - Golang/Postgres

SLR Maine React Frontend

SLR Maine Sprint Planning Board

Tell us about the project

SLR Maine is a prototype web application public engagement tool for awareness around coastal flooding and sea level rise built for the Gulf of Maine Research Institute and the City of Portland Maine. The site features an interactive parallax UI built in React with two part functionality: to provide a portal for interested citizens to sign up for SMS messages related to flooding events and to allow users to access important information related to current weather conditions and past flooding events in the Portland area (the chat functionality, originally a stretch goal, was added more recently).

The site features multiple React components interacting with a ***Golang backend API*** and ***PostgreSQL*** database provisioned and deployed on ***Heroku***. SMS messages are generated through the third party ***Twilio API*** and the chat functionality is an integration of the ***Stream Chat API*** (separate ***nodeJS API*** with a ***PSQL database*** deployed with AWS endpoints using ***Serverless AWS lambda and S3*** nodeJS Chat Github Repository Here).

Why was the project initiated? Why did you choose to build it the way you did?

Many years ago, in another lifetime, I worked for a marine mammal rescue and rehabilitation center in Westbrook, Maine. Around the same time the Gulf of Maine Research Institute was founded to be the policy, education, and research center for the Gulf of Maine. In Fall 2018, the City of Portland Maine passed a new budget measure to build upon the existing NOAA monitoring stations a better monitoring system for coastal flooding events related to sea level rise. Part of this funding was earmarked for public engagement. I initially contacted GMRI to ask about working with them on a potential capstone project for my Galvanize Web Development Immersive program. Through these talks I spoke with the planner with the City of Portland and together with the staff at GMRI we came up with the plan to create an 'alert' system for coastal flooding events. The main requirement from my clients was to create an interactive site with links to resources. It was left entirely up to me to determine how to send the SMS messages and what data to display to enable site interactivity.

Initial planning for the project consisted of creating wireframes and entity relationship diagrams in conjunction with my clients. This is helped solidify many of the user stories and influenced my decisions to create the backend routing with Go, PSQL, and to incorporate the Twilio API for the text messages. Working off of this agile backlog I began to break features down into digestible pieces (for example: the Moon component uses a basic lunar calculation to determine which moon phase image to load on the page).

I chose the parallax UI because the scrolling nature of parallax creates an inherent narrative as a user scrolls down the page. Different tidal predictions charts, a photo carousel, and fact boxes with links to resources move independently of each other while exploring the site. I had implemented a similar parallax scroll for a project in ***Vue*** and was interested in trying to make it work in React.

I chose the Twilio API for sending messages because I wanted to integrate a third party platform that could be easily routed from a Go backend and, honestly, to dive deep into the Twilio architecture and documentation (which turned out to be rather skimpy for Go!).

In short, I built this project with the primary goal of providing interested citizens accurate and compelling information related to coastal flooding events in Portland, Maine. This was achieved through the parallax UI, `react-vis/d3` tidal prediction charts, the `OpenWeather API` for weather conditions, `NOAA Ocean Monitoring Stations API` for current water level and tidal predictions, ***Materialize*** for UI component styling, and a SMS subscription service.

What challenges did you face when completing the project? How did you address those challenges?

- Data conversion from API and database data

Conversion Functions File

Working with large amounts of API data requires a lot of sorting and conversion. For example. the dates for the NOAA Api require a very specific format versus formatting the date for display in the tidal prediction chart. I very quickly came up with the idea to house all of my conversion functions in a separate file and export them so that I could call one from anywhere in the application. This created a modular and simplified project structure.

- Creating visual graphs of tidal data 📉

For the data visualization in this project I decided to use `react-vis` because it is an offshoot of `D3` created by Uber for integration in React. There were many challenges around formatting the data props in the correct way as they came down to the Child component. Since the same information is used to generate the tidal curve and the hint functionality Not to mention the difficult with the basic layout and CSS around position the charts on the page.

As an example, the following is the relatively simple implementation of the visual bar chart on the 'admin' page around flooding event categories:

```
let category = this.props.floodData;
let l = this.getCategoryData(category);

  const categoryData = [
    { x: 'Splash Over', y: l['Splash Over'] },
    { x: 'Minor', y: l.Minor },
    { x: 'Moderate', y: l.Moderate },
    { x: 'Major', y: l.Major },
    { x: 'Extreme', y: l.Extreme }
  ];
```

```
    return (
      <MediaQuery minDeviceWidth={950}>
        {matches => {
          if (matches) {
            return (
              <XYPlot
                className='barChart'
                xType='ordinal'
                width={550}
                height={200}
              >
                <XAxis style={{ fontSize: '12px', textShadow: 'none' }} />
                <VerticalBarSeries
                  data={categoryData}
                   style={{ stroke: '#576FC9', fill: '#576FC9' }}
                 />
              </XYPlot>
              ...
```

- The Twilio API routing for Go

One of the most gratifying moments of the last 12 months was the first text message I sent myself through the command line. The Twilio API integration was certainly difficult to get up and running. Over the course of two days I dove headfirst into the documentation and worked on finding a solution to initializing the API in Go on the backend and creating the proper route from the `admin` page of the frontend. This was probably my most favorite feature to work on in this application because of the challenge of taking the third party API documentation and code base and transforming it to meet my needs. The following code is the primary function that handles the Twilio API connection request and response:

```
// SendText formats and passes the flooding event message to the Twilio
api
func SendText(Msg string) {

    // Init() initializes the env variables
    accountSid, authToken, numberFrom := Init()

    urlStr := "https://api.twilio.com/2010-04-01/Accounts/" + accountSid +
"/Messages.json"

    // load all the data to send
    msgData := url.Values{}

   // SQL get all and range through for alert send through the
Models.getALLSubs() SQL call
    subs := m.GetAllSubs()
    for i, sub := range subs {
        fmt.Printf("SMS to Number**%v = %v\n", i, sub.Phone)
        msgData.Set("To", sub.Phone)
    }
    msgData.Set("From", numberFrom)
    msgData.Set("Body", Msg)
```

```
        msgDataReader := *strings.NewReader(msgData.Encode())

        // request body is formed and sent
        client := &http.Client{}
        req, _ := http.NewRequest("POST", urlStr, &msgDataReader)
        req.SetBasicAuth(accountSid, authToken)
        req.Header.Add("Accept", "application/json")
        req.Header.Add("Content-Type", "application/x-www-form-urlencoded")

        // response is read and evaluated
        resp, _ := client.Do(req)
        if resp.StatusCode >= 200 && resp.StatusCode < 300 {
            var data map[string]interface{}
            decoder := json.NewDecoder(resp.Body)
            err := decoder.Decode(&data)
            if err != nil {
                fmt.Println(data["sid"], err)
            }
        } else {
            // log of the successful response
            fmt.Println("send text response println:", resp, resp.Status)
        }
    }
```

Anything else you'd like to share that gives us insight into your working style?

I am committed to working within an Agile environment. In terms of workflow, I find it to be the best system for easily breaking down large problems into digestible chunks. This is at the heart of building a great project.

For a working style, I like to have a clear idea of my responsibilities. This helps to not have a key feature overlooked and it also helps me stay on track and not 'over-engineer' or bleed my features. One of my main goals in pursuing this career is to work in an environment where I not only challenge myself, but I am also reviewed and challenged by my teammates to improve my code quality and communication. I want to continue to learn and to eventually grow into a leadership position.

The SLR Maine application is a great representation of the complex work I am capable of creating. It was wonderful to have the opportunity to give something back to a non-profit whose work I greatly admire on a subject that I am very concerned and motivated around. Because education and specifically education equity is such a huge and essential issue I would love the opportunity to bring the same passion, enthusiasm, and creativity to projects with LEE.

> If you would like to send me a text message through the Twilio API: please login in to the `/admin` page with the password: `slrmaine`. This page contains information related to users in the system as well as a log of all messages recently sent (a lot of tests for the Chat Api integration). Once you send a message it will also automatically appear in the Chat on the site as an SLR Admin post. (If the login does not work and no error message appears, the heroku database may have recently timed out because of developer account settings. Please wait a few seconds or navigate away and back and try again. 🌊)