# Fischer Code Examples for MWM

The first bit of code comes from my SLR Maine project and it the routing model for the SMS emssaging system connection trough the Twilio API. Written in Go and featuring a few sepcific types of data-type maniuplation, I feel like it exemplifies my ability to quickly and efficiently learn a completely new language and its conventions.

SLR Maine Backend Repo

```
/**
* An Event happens from the front end there is a new message to be sent.
* It hits a route that triggers a database call to retrieve a list of
numbers.
* The message content makes it to the request call in the correct format
* A for loop calls the sms function for every number in the db
* general hilarity ensues 🌊
 */

package sms

import (
        "encoding/json"
        "fmt"
        "net/http"
        "net/url"
        "os"
        "strings"

        m "github.com/rafischer1/slr_capstone_go/models"
)

// if there is a POST to the /data route the handler could trigger first a
GetAll call to the subscribers table and pass that []string of numbers as
one of two parameters to the SendText(numbers, msg) => return twillio
http.statust

// SendText formats and passes the flooding event message to the Twillio
api
func SendText(Msg string) {
        accountSid, authToken, numberFrom := Init()

  // range through array to generate "To" number for all subscribers in db
        subscriptions := m.GetAllSubs()
        for i, sub := range subscriptions {
                fmt.Printf("SMS to Number**%v = %v\n", i, sub.Phone)
        }

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

```
        msgData := url.Values{}

        msgData.Set("To", "3024232120")
        msgData.Set("From", numberFrom)
        msgData.Set("Body", Msg)
        msgDataReader := *strings.NewReader(msgData.Encode())

        fmt.Println("msgDataReader:", msgDataReader)

        // 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"])
                }
        } else {
                fmt.Println("response println:", resp, resp.Status)
        }
}
```

This second entry is a portion of the App.js file from a React project which creates a clone site of an email inbox. This not only highlights React skills but also the often complicated managing of state, toggling of condition for rendering, and how I organize code.

React Inbox Frontend App.js

```
export default class App extends Component {
  constructor(props) {
    super(props)
    this.state = {
      messages: []
    }
    // console.log("state:", this.state.messages)

  }


  /******************************
  The Mark all as Read and Unread
```

```
    are good
    ********************************/
    allReadCallback = async () => {
      await this.updateMessages({
        "messageIds": this.state.messages.filter(message =>
  message.selected).map(message => message.id),
        "command": "read",
        "read": true
      })

      this.setState({
        messages: this.state.messages.map(message => (
          message.selected ? { ...message, read: true } : message
        ))
      })
    }

    allUnreadCallback = async () => {
      await this.updateMessages({
        "messageIds": this.state.messages.filter(message =>
  message.selected).map(message => message.id),
        "command": "read",
        "read": false
      })

      this.setState({
        messages: this.state.messages.map(message => (
          message.selected ? { ...message, read: false } : message
        ))
      })
    }

    /******************************
    Compose new message
    ********************************/
    openComposeCallback = () => this.setState({ compose: !this.state.compose
  })

    composeMessageCallback = async (post) => {
      let postBody = {
        subject: post.subject,
        body: post.body
      }
      let response = await fetch(
        `${REACT_APP_API_URL}/messages`,
        {
          method: "POST",
          body: JSON.stringify(postBody),
          headers: {
            "Content-Type": "application/json",
            Accept: "application/json"
          }
        }
      );
      this.openComposeCallback()
      this.getMessageState()
    }

  /****************************
    * Delete selected messages works
  *******************************/
    async deleteMessagesCallback(id) {
      let response = await fetch(
        `${REACT_APP_API_URL}/messages/${id}`,
        {
          method: "DELETE",
          headers: {
            "Content-Type": "application/json",
            Accept: "application/json"
          }
        }
      );
      if (response.status === 200 ) {
        this.getMessageState();
      }
  }

  /****************************
    * Star callback working well!
  ****************************/
    starCallback = (message) => {
      this.toggleFunc(message[0], 'starred')
    }

    ...

     /****************************
   Performance and update functions
    ********************************/
    updateMessages = async (body) => {
      let editBodyLabel = ''
      if (body.label === undefined) {
        editBodyLabel = ''
      } else {
        editBodyLabel = body.label[0]
      }

      body.messageIds.map(async (id) => {
        let editBody = {
          ID: id,
          Labels: editBodyLabel,
          Read: body.read
        }

        return await fetch(
          `${REACT_APP_API_URL}/messages/${id}`,
          {
            method: "PUT",
            headers: {
              "Content-Type": "application/json",
              Accept: "application/json"
            },
            body: JSON.stringify(editBody)
          }
        );
      })

    }

    toggleFunc = (message, property) => {
      const idx = this.state.messages.indexOf(message)

      this.setState({
        messages: [
          ...this.state.messages.slice(0, idx),
          { ...message, [property]: !message[property] },
          ...this.state.messages.slice(idx + 1),
        ]
      })
    }

    getMessageState = async () => {
      const response = await fetch(`${REACT_APP_API_URL}/messages`);
      if (response.status === 200) {
        let resJson = await response.json()
        // console.log('resJson', response)
        this.setState({
          ...this.state,
          messages: resJson
        })
      } else {
        // console.log('I broke on the GET fetch', response)
        throw new Error('Uh oh! I broke on the GET request')
      }
    }

    /******************************
    Component did mount is ☑️
    ********************************/
    async componentDidMount() {
      this.getMessageState()
    }

    /******************************
    Render below here ⌨️
    ********************************/
    render() {
      return (
        <div className="App">
          <Toolbar
          messages={this.state.messages}
          openComposeCallback={this.openComposeCallback.bind(this)}
          applyLabelCallback={this.applyLabelCallback}
          removeLabelCallback={this.removeLabelCallback}
          deleteMessagesCallback={this.deleteMessagesCallback.bind(this)}
          selectAllCallback={this.selectAllCallback}
          allReadCallback={this.allReadCallback}
          allUnreadCallback={this.allUnreadCallback}
          />
          {this.state.compose ? <Compose
          composeMessageCallback={this.composeMessageCallback} />
          : false
          }
        <Messages messages={this.state.messages}
        starCallback={this.starCallback}
        selectCallback={this.selectCallback.bind(this)}
        />
        </div>
      )
    }
  }
```