Badger Bytes: Spike Journal

The Team: Austin Torres, Jiwon Song, John Eslinger, Rachel Hon, Yashishvin Pothuri

February 22nd (2-22-2021)

Team meeting is planned for Tuesday night, February 23rd. Team members do prep-work for meeting/Spike Exercise (e.g. Kash creates a basic diagram of the pages/UI).

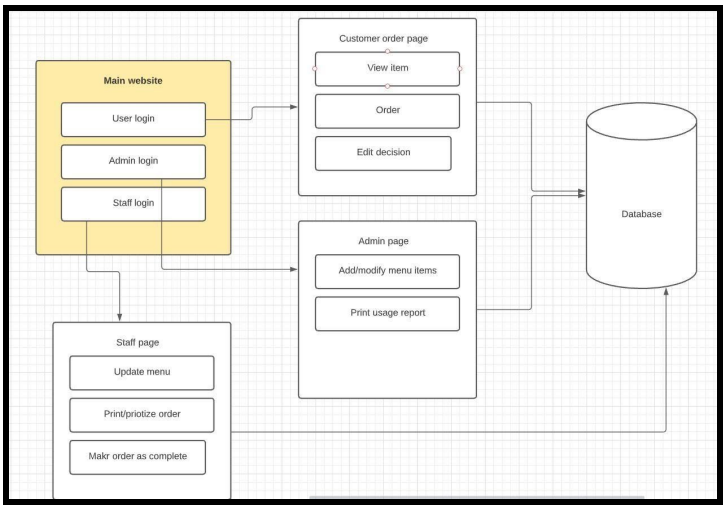
February 23rd (2-23-2021)

At the team meeting, the team begins pooling ideas. Kash suggests using React, a Javascript library, to create the front-end. With previous course experience and some old projects to serve as initial frameworks, Austin and Kash take up the front-end. Rachel and Jiwon, having experience with databases, begin looking into how to create a back-end database in React, namely at formats like SQLite and JSON files. Working off of UML diagrams, created by Jiwon, of Kash's initial designs, John begins working on organizing the data/actions into web pages (on top of reviewing Javascript/React tutorials with Rachel and Jiwon). The team agrees to meet up the following morning to discuss web page organization and a front-end/back-end interface.

React: a Javascript Library Kash and Austin have experience with
React – A JavaScript library for building user interfaces

Tutorials being viewed by Rachel, Jiwon, and John
Tutorial: Intro to React – React
Getting started with SQLite in React-Native | by Sambhav Jain
Building Basic React Authentication | by Denny Scott | Better Programming

UML diagram of Kash's initial designs (made by Jiwon)



February 24th (2-24-2021)

Meeting after a night's sleep, the team does a second round of brainstorming and refining. Starting with the meeting's original goals, the team expands on Kash's design (guided by John) to determine the web page organization of the site, basic user interaction with each page, and control flow between pages. Jiwon, after digging into database details, determines that SQLite and MySQL would provide better performance over JSON for the application as Austin finds an old React project that utilizes a MySQL database. With Kash and Rachel finding tutorials to help develop Rest API to tie React JS at the front-end to MySQL at the back-end, the team splits to begin developing and combining both ends. Meanwhile, a repository is created to store the code.

Later on, Kash and Rachel decide that React Native may be a better fit for the project over regular React. With Rachel looking into authentication for the login page, the team continues to develop code.

React Native

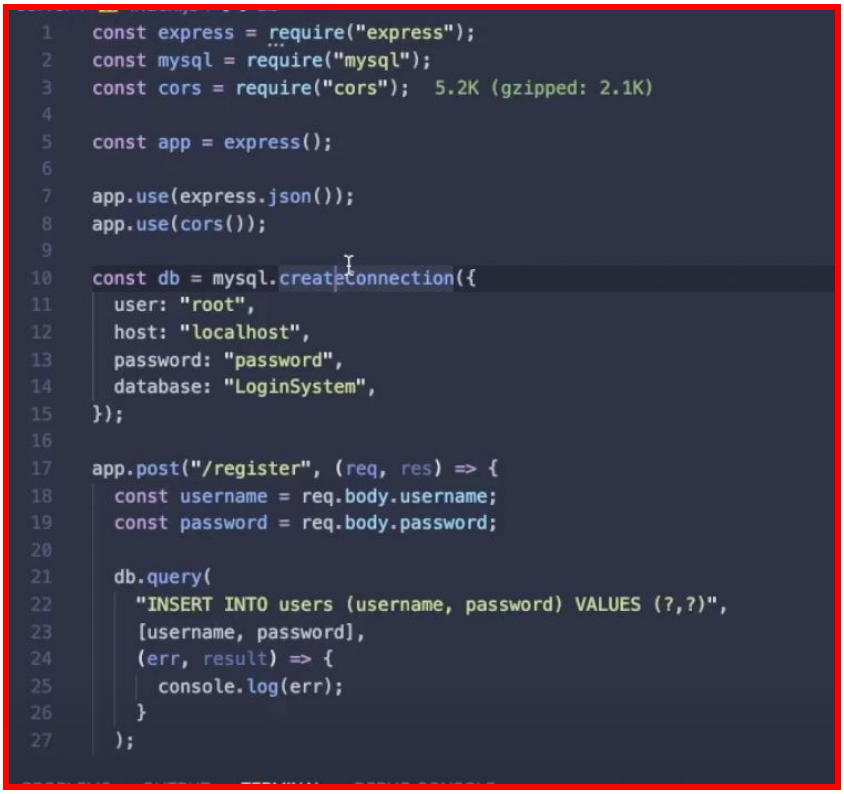
React Native · A framework for building native apps using React

Tutorials to help connect the front and back ends of the project

Understanding And Using REST APIs — Smashing Magazine

re: How can use React JS + Node JS + MySQL together?

Example of Login Page code using React JS, Node JS, and MySQL



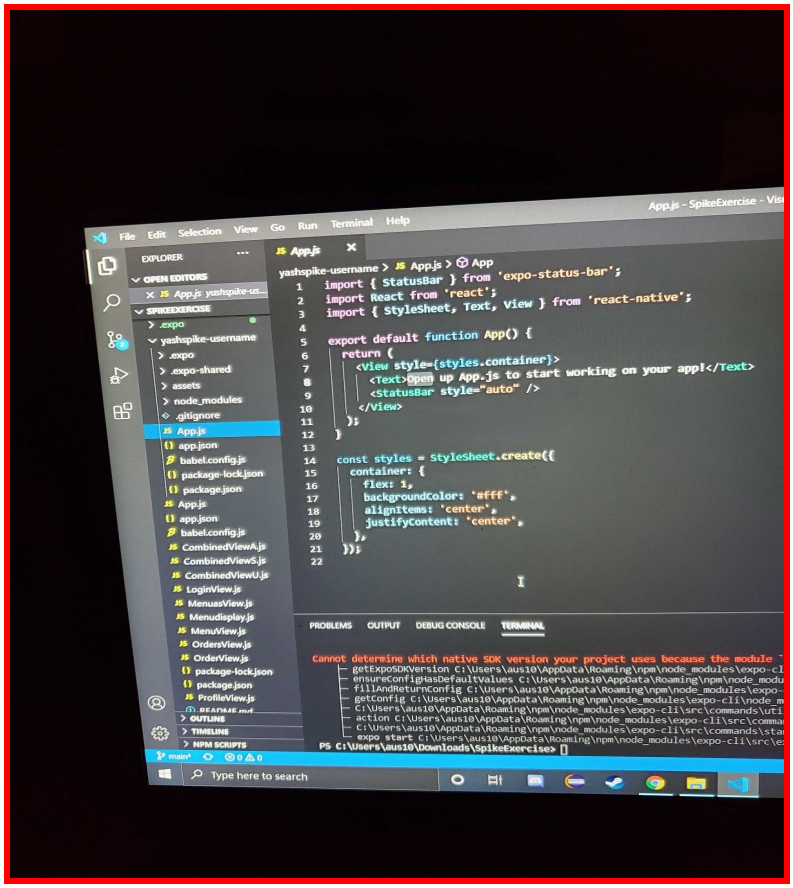
```
const express = require("express");
const mysql = require("mysql");
const cors = require("cors");  5.2K (gzipped: 2.1K)

const app = express();

app.use(express.json());
app.use(cors());

const db = mysql.createConnection({
  user: "root",
  host: "localhost",
  password: "password",
  database: "LoginSystem",
});

app.post("/register", (req, res) => {
  const username = req.body.username;
  const password = req.body.password;

  db.query(
    "INSERT INTO users (username, password) VALUES (?,?)",
    [username, password],
    (err, result) => {
      console.log(err);
    }
  );
```

February 25th (2-25-2021)

With a project to do, the team gets into the heavier development. Using previously written code as a base, Kash creates the page structure of the web app as Austin begins working on the visual and interactive design. Jiwon looks into Expo, a framework for React applications, as a means of creating a database to interface with the front-end. As Kash finishes the pages, he creates a list of requirements for API calls between the front-end and the database for the team to inspect.

Expo: a framework for universal React and React Native applications
Expo.io

Snapshot of base front-end code worked on by Austin and Kash



Kash's list of API call requirements

```
For the Login Page
1) endpoint Login
   Get request:
     1)should return a token if returning a token is not feasible return a message saying that login was sucessful
     2)should return the permission the user has if he is a user
2) endpoint user
     Post request:
          1)Parameters required are username,password,permissions of the user
         2)If a post request is sucessful return a message saying sucessful
3)endpoint user/userid
           Put request:
             1)Parameters which are required are username,password,phonenumber,address,payment information
4)endpoint order
    required requests for this endpoint
    put,post,Get
      reuired parameters in the body for put and post
         Username,nameof the item,time,order status
5)endpoint Menu
     required requests for this endpoint
get,put,post
      reuired body parameters for the put and post request
         -Cost of the item
         -The url to the item's image
```

February 26th (2-26-2021)

As the final day starts, the team begins having some struggles with the backend. Jiwon determines Expo will not work with the system. Kash and Austin give Jiwon some helpful links, but despite trying multiple database frameworks (Parse and Back4App), little leadway is made. Rachel, trying Firebase, MongoDB, MySQL Workbench, and JSON formatted files, also struggles to find a way to create a compatible database. On a brighter note, Kash is able to finish the UI layout while Austin adds visual design to the UI. John puts together the Developer Log as of 9:57pm the night of and adds it to the repository.

Database frameworks reviewed

Expo.io

Parse Platform

Back4App: Low-code backend to build modern apps

Firebase

MongoDB

MySQL Workbench

JSON formatting (Intro provided by w3schools.com)

Links provided by Kash and Austin to Jiwon for support

The best backend platforms for your React Native app

Building Dynamic React Apps with Database Data

Error message Jiwon encountered trying to use Parse PI

Extra Resources involved in the Spike Exercise

Notable Links

[Github Repository]

[Initial Workspace Document]

API Information

The following API can be accessed at `https://mysqlcs639.cs.wisc.edu`

Route	Auth Required	Token Required	Get	Post	Put	Delete
/login/	✓		✓			
/users/				✓		
/users/`<username>`		✓	✓	✓	✓	✓
/activities/		✓	✓	✓		
/activities/`<activity_id>`		✓	✓		✓	✓
/meals/		✓	✓	✓		
/meals/`<meal_id>`		✓	✓		✓	✓
/meals/`<meal_id>`/foods/		✓	✓	✓		
/meals/`<meal_id>`/foods/`<food_id>`		✓	✓		✓	✓
/foods/			✓			
/foods/`food_id`			✓			

Auth and Tokens

For this API, users need to provide credentials in order to access information specific to themselves. They get these credentials by requesting tokens, which are short-lived codes which tell the server that you are who you are saying you are, without having to provide a username and password each time. The steps to get these tokens are outlined below.

Signup

This can be done with a `POST` request to the `/users` route. You will need to tell the API a bit about the user. You should provide this data in the message body (stringified) in the following form:

```
{username:<str>,                // Required
 password:<str>,                // Required
 firstName:<str>,               // Optional
 lastName:<str>,                // Optional
 goalDailyCalories:<float>,     // Optional
 goalDailyProtein:<float>,      // Optional
 goalDailyCarbohydrates:<float>, // Optional
 goalDailyFat:<float>,          // Optional
 goalDailyActivity:<float>      // Optional
}
```

Only the `username` and `password` fields are required. Don't worry about the other ones for creating a user, as they can be updated later with `PUT` requests.

If the user is successfully created, you will receive a positive message back from the server. You will then need to log in with that user.

Login

You can do this via the `/login` route with a `GET` request. You will need to send the username and password in the authorization header using the header `Authorization` with value 'Basic {base64enc(username:password)'. e.g. 'Basic bXl1c2VyOnBhc3MxMjM0' for a hypothetical user 'myuser' with password 'pass1234'.

You will receive back a token that you can use to access information from the API. The token you receive can then be added in the `x-access-token` header.