HW3 Sprint 1

Homework 3: Sprint 1

Due on Gradescope Thursday, February 15, 2023 11:59pm

This homework is to be done and submitted as a team.

Learning Goals

- Develop data models for representing data and relationships.
- Use an iterative design process to develop rapid user interface prototypes.

References

- Design Methods E-Book
- SQL vs. NoSQL: The Differences Explained + When to Use Each

Homework Tasks

In this homework, you will work to conduct your first sprint with your team towards your MVP. The core goal in this assignment is to prototype the core elements of your product. Your task is to prepare a number of different forms of prototypes.

Please continue to use your current Dronuts repository that you generated as part of Sprint 0.

Designing your Data Model

An Object Data Model is a way of specifying the format of data your application will interact with. There are many different ways of developing such a model; ranging from building object-oriented class structures to to generating database schemas (like in SQL), or even as a formal UML diagram. Creating a a data model is crucial any time different components of an application are to interact. Examples include a database and server-side application or an MVC (Model, View, Controller) frontend and a server-side application.

In this assignment, you will focus on defining a data model, represented in code with either an Object Document Mapper, ODM (for document-oriented NoSQL databases like MongoDB), or an Object Relational Mapper, ORM (for SQL databases like PostgresSQL). In pure Javascript, objects can be represented in a format called JSON.

For instance, a `User` might be represented as follows:

```
{
   "username" : "ricksanchez",
   "password" : "735ed12abec12bdcb42b",
   "full_name" : "Rick Sanchez",
   "followers" : [
      "msmith",
      "bperson",
      "kmichael",
      "mpbh"
   ]
}
```

While this is a good example of a user, it is not entirely explanatory of the datatypes and relationships expressed in the data. Likewise, we might write out a document that looks something like this to help guide development:

```
{
   "username" : String,     // UNIQUE, URL-SAFE
   "password" : String,     // Salted, Hashed Password.
   "full_name" : String,
   "twitter_id" : ?String, // Optional, hence the question mark
   "followers" : [String]  // Reference to other USER objects
}
```

Much like developing in an object-oriented fashion within a single language, developing an object data model like this allows team members to coordinate development even across multiple projects (say, frontend and backend). As such, your team should:

- As a group, decide on a list of relevant objects (Drones, Users, etc).
- For each of these objects, create an example object (like the first `User` example above). Note that you may need to have multiple example objects if an object can be in more than one state.
- For each of these objects, create a generalized, corresponding data mapping (as shown above), mapping object fields to types, while acknowledging if fields are optional, nullable, required, etc. Comment as necessary to explain these constraints to us.
- Put all of the above examples and objects into your repository Wiki.

Database Choice

Express.JS does not expose built-in functionality for defining data models. **First, you'll need to choose a database**. We recommend using either MongoDB as a document-oriented database or PostgresSQL as a relational one.

Then, **you'll need to use a corresponding library** for mapping your Object Data Model to/from a relational database or a document-oriented one. We recommend either Mongoose (for MongoDB) or Sequelize (for PostgresSQL).

Sequelize If you go this route, you will install Sequelize, a package for validating models and generating SQL queries for the PostgresSQL database (or any SQL variant).

- Install the Sequelize package dependency via npm/yarn.
- Define models, with datatypes, for all of the objects in your Object Data Model. Follow this guide.
- Write some simple unit tests for your model validation. Follow this guide.
- *Note*: Follow along with this tutorial on setting up Sequelize to work with Typescript.

Mongoose If you go this route, you will install Mongoose.js, a package for validating JSON objects for the MongoDB database:

- Install the Mongoose package dependency via npm/yarn.
- Create schema objects for all of the objects in your Object Data Model. Follow this guide.
- Write some simple unit tests for your schema validation. Follow this guide.

Validation

In an ideal world, all actors (developers on your project, users of your API, etc.) would respect your data model. However, this isn't always going to be the case; developers can make mistakes, API users can mis-use your API, and hackers can attempt to abuse your object model for nefarious purposes.

To combat this, your application should programmatically validate objects, as defined by your data model. The way your team should implement this depends on your choice of database (MongDB vs PostgresSQL) and corresponding library: Mongoose (for MongoDB) or Sequelize (for PostgresSQL).

For example, in Sequelize, a `User` model with constraints may be defined like this:

```
const { Sequelize, Op, Model, DataTypes } = require("sequelize");
const sequelize = new Sequelize('postgres://user:pass@example.com:5432/dbname')

const User = sequelize.define("user", {
  username: {
    type: DataTypes.TEXT,
    allowNull: false,
    unique: true
  },
  password: {
    type: DataTypes.STRING(64),
    validate: {
      is: /^[0-9a-f]{64}$/i
    }
  }
});

(async () => {
  await sequelize.sync({ force: true });
  // Code here
})();
```

where validations are automatically run on create, update and save. You can also call a `validate()` method to manually validate an instance.

In Mongoose, upon defining your Schema object, you may get validation errors on save or by calling `validateSync`:

```
const schema = new Schema({
  username: {
    type: String,
    required: true
  }
  phone: {
    type: String,
    validate: {
      validator: function(v) {
        return /\d{3}-\d{3}-\d{4}/.test(v);
      },
      message: props => `${props.value} is not a valid phone number!`
    },
    required: [true, 'User phone number required']
  }
});

const User = db.model('user', schema);
user.phone = '201-555-0123';

// This person has no username :(
const user = new User();

let error;
try {
  await user.save();
} catch (err) {
  error = err;
}

assert.equal(error.errors['username'].message,
  'Path `username` is required.');

error = user.validateSync();
assert.equal(error.errors['username'].message,
  'Path `name` is required.');
```

Paper Prototype

To help you synthesize all your team's ideas into a single design, you will start by developing a series of paper prototypes. **Each individual team member is responsible for generating a total of 6 different paper prototype sketches**. **Each person should create 3 sketches of the UI for the customer page, and 3 sketches of the UI for the employees**. The UIs do not need to be "good", but they should be all different from each other. After generating these sketches individually, your team should meet to synthesize your design into a single design that incorporates the best of all the designs.

To turn these in, each team member should scan or take a picture of their sketches, and then upload them to your GitHub Wiki. Your team should turn in a link to the Wiki page where your pictures live.

Frontend Prototype

As a startup with limited funding, your company should be focused on building a presentable prototype as quickly as possible. However, as you don't want to do a lot of work from the ground up, you've decided to develop your prototype using a frontend user-interface "framework", a library which makes it very easy to create dynamic client-side applications. To keep things simple, we recommend that groups use React to build your frontend applications. If your team has another choice you are all more comfortable with (e.g. Vue, Angular, or Svelte), **please let the TAs know ahead of time**. We'll be focusing on React in recitations however.

Your team can familiarize themselves with React using the following resources:

- Intro to React Tutorial
- React Tutorial
- React Fundamentals

Using a frotend framework like React (which is *MV*'ish*, but not explicitly MVC), your team should develop a prototype:

- Develop a **view-only** prototype for the 1-3 largest user stories in your project. What you do not accomplish in this sprint, you will need to accomplish later, so budget your time appropriately. This should be a prototype, so while it should demonstrate how a user will interact with the system, it does not need to be "pixel perfect." Because it is view only, you should be able to navigate across pages, but you do not need to implement any part of the backend (data processing, validation, generation) at this point.

- Your team will be deploying your application hosted on the cloud. Again, we recommend Render for deployment. Please ensure that your submission deploys correctly upon submission of your assignment.

Use of Typescript In this class, we **highly** recommend the use of Typescript instead of Javascript for your Dronuts prototype. Typescript is a much stricter language that enforces types within your Javascript code. For this assignment, your frontend prototype and backend **should** be implemented using Typescript. You will be introduced to this in recitation, but here is a tutorial that discusses Typescript further.

We expect your team to set up CI for your web apps. We recommend the use of Github Actions as your CI service, however, other options such as Circle CI are acceptable. Here are some links to get you started:

- Using Starter Workflows
- How to build a CI/CD pipeline with GitHub Actions in four simple steps
- Creating a CI/CD pipeline using Github Actions

Releasing your work

Just before turning in your assignment, one person on your team should create a GitHub Release.

Deliverables

The deliverables for this assignment include:

1. Providing a link to your team's Object Data Model, which should be documented in the group's repository Wiki.
2. Submitting a link to the paper prototype section of your team's repository. Each team member must upload their paper prototypes to this section. Although the prototypes can be on different pages, ensure all of them are accessible from a single, unified page for easy navigation.
3. In your Wiki, add an entry explaining why the group chose MongoDB (NoSQL) or PostgresSQL (SQL) as your database of choice.
4. Sharing a link to the GitHub webpage where your team's deliverables are released (one per team). This release should contain:
    - **Your team's Data Model implemented** with either an Object Document Mapper or a Relational Mapper in Express.js. Utilize Mongoose or Sequelize libraries for this purpose, and do some initial model validation testing. Ensure this code is committed to your

    team's repository and merged into a `main` branch. *Note*: all work should be PR'ed first!
  - **A Typescript-based frontend prototype** developed in React (or a framework of your choosing), committed to your team's repository. *Note*: Once again, if you choose a different frontend framework, please discuss this with the TAs first!
5. All links should be submitted as a single document via gradescope (one per team).

Additional Criteria

We will be strictly enforcing and grading your team’s process; i.e., we want to see all PRs, close issues, and issues tied to cards on your team's Github Project board with individuals assigned to each. Failure to adhere to these practices will result in point deductions.