17-356/17-766: Software Engineering for Startups
Homework 3: Prototypes (Sprint 1)

100 points

Due: February 12th, Midnight (two weeks)

This homework is to be done with your group, but there's one individual component below

In this homework, you will work conduct your first sprint with your team towards your MVP. The core goal in this assignment is to quickly *prototype* the core elements of your product.

References:

- Design Methods http://faculty.washington.edu/ajko/books/design-methods/
- All about pull requests
- Thoughtbot guide on code reviews

The learning goals of this assignment are that, once completing it, you will be able to:

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

Your task for the next two weeks is to prepare a number of different kinds of prototypes:

Data Model

A data model is a way of specifying the format of data your application will interact with. There are many different ways of developing data models for your application; ranging from building objects (Javascript objects in this case) structured to contain data or by creating a database schema (like in SQL). Creating an data model is crucial any time different components of an application are to interact (for example, a database and server-side application, or an MVC-styled frontend and a server-side application).

In this assignment, we will be focused on creating a document-oriented data model. In Javascript, objects are 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",
    "mobius"
  ]
}
```

While this is a good example of a User, it is not entirely explanatory of the data types, optionals, and/or relationships expressed in the data. Likewise, we might write out a schema or spec (with additional comments) 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.
}
```

Developing such a schema or spec allows team members to coordinate development across multiple projects (say, frontend and backend APIs) and review needs. For example, to relate objects, you may want to add and generate a value for an *"id"* field. If you want to go further and more officially specify and validate your schemas, we highly recommend looking at JSON Schema.

For this component, your team should:

- As a group, decide on a list of relevant objects (Drones, Users, etc) that are part of your application and comprise your overall data model.
- For each of these objects, create a JSON example (like the first User example above). Note that certain objects may be composed of multiple variations if an object can be in more than one state depending on an action in your application.
- For each of these objects, create a corresponding data schema (like the second User example above).
- Put all of your JSON examples and schemas into your repo's Wiki.

Object 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 misuse your API, and hackers can attempt to abuse your data model for nefarious purposes.

To combat this, your application should programmatically validate objects and ensure correct data as per your schema. The way you should implement this depends on your choice of backend framework (from Homework 2):

LOOPBACK

Loopback includes built-in functionality for defining data models:

- Follow the loopback tutorial (check for updates if you're using version 4) for creating object models.
- Write some simple unit tests for your Loopback models either via jest (possibly helpful: loopback-jest), the loopback/testlab suite (keep an eye on loopback versions here), which uses sinon, or jest+supertest if you've integrated your data model within your application already, i.e. this example.

EXPRESS.JS

Unlike Loopback, Express.JS does not include built-in functionality for defining data models. Instead, you should install either the *Joi* object schema description language and validator, or the schm library for composable schema validation.

- Install and save your chosen package to your project repo.
- Create schemas for all of the objects that are part of your overall data model.
- Write some simple unit tests to ensure your models and schemas are working correctly either via jest or jest+supertest if you've integrated your data model within your application already, i.e. this example.

For either framework, if you end up using JSON Schema for specifications and validation (e.g. Loopback+Json Schema or Express-Json-Schema), we'll accept related examples, schemas, and tests against those schemas as well.

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 sketching 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 a wiki. You should turn in a link to the location in your your github wiki.

Frontend Prototype

As a startup with limited funding, you should be focused on building a presentable prototype as quickly as possible. However, as you don't want to do a lot of rework, you decide to develop your prototype using an existing framework, a tool which makes it very easy to create dynamic client-side applications. To keep things simple, all groups will be using **react.js** as our framework. You can familiarize yourself with **React** using the following resources:

- Create React App
- React Tutorial

Using **React,** 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 back end (data processing, validation, generation) at this point.
- **Verify this prototype correctly deploys to Azure**.

Code Collaboration Process/Workflow

- Never directly push code to master
- Every code change/update should happen through a Pull Request
- Each team member should leave at least one (and possibly more than one) **constructive comment** on another team member's Pull Request. The person making a change can also assign certain team members as reviewers.
- Once a reviewer is happy with the changes in a Pull Request, leave a +1 (or :+1: emoji) on the Pull Request thread

Team Deliverables

The deliverables for this assignment are:

- One link (per team) to the team's overall **data model** (examples and schemas) in their group repo Wiki
- A link to the paper prototype page of your team's repo. Each team member should upload their paper prototypes there; they can be organized on different pages, but they should all link from a single page.
- An implementation of the data model in either Loopback or Express.js, committed to your team's repo.
- An front-end prototype implemented in React, committed to your team's repo.

Individual Deliverables

Every team member should make pull requests (PRs) when adding/pushing new code. For this assignment, **every individual team member** should have a

- Pull Request merged with at least one +1 from a reviewer
- Add at least one constructive comment on another team member's Pull Request