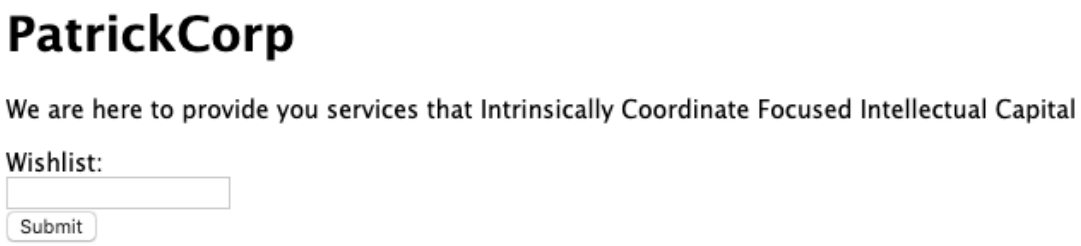
HTTP Backend

So far, we have been focusing on building "front-end" applications using JavaScript to execute within a user's browser and leveraged AJAX requests to an HTTP server ("back-end") for some functionality. In this assignment we will build our own backend web application using Express to handle the monotonous details of HTTP and make us productive quickly.

Requirements

The application will have a few disparate features so we can flex a few different skill sets. We are going to build our very own web API (kind of like the Weather API we used), create a dynamically generated HTML page, and use some basic session functionality to enable the app to hold a user's state across multiple page refreshes and navigations.

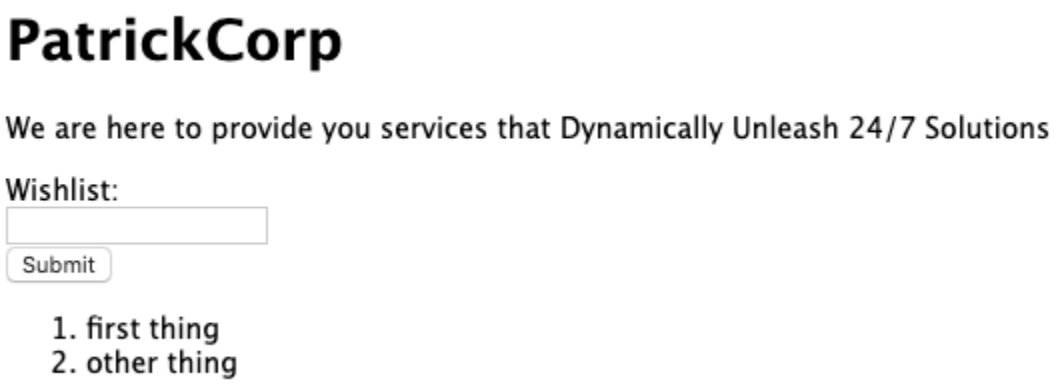
Dynamically Generated Homepage



The home page ("/") will be generated using the templated view system in Express. The title of the page should be some made-up business name, and there should be some text below it that incorporates the result of the Corporate BS Generator API.

Session Data

On the home page below the title and description, there should be a form that lets the user enter text input and click "Submit". Upon clicking Submit, a POST request should be sent to the server with the contents of the form input and added to the user's session state, and the response should redirect the user back to the root page ("/") where the items in the user's session will be displayed in a numbered list in the order they were added.



Basic Web API

There should be an API to return the square root of a number registered at the `/sqrt` route. The API has the following specifications:

- The number to find the square root for should be provided as a query argument named num
- If the number is not provided, a 400 error response should be returned to the user explaining the reason for the error.
- If the input number is invalid (such as a negative number), a 400 error response should be returned to the user explaining the error.
- A successful result should return a 200 response with a JSON object containing one key named result that maps to the number value of the result.

Sample error output:

```
$ curl --verbose localhost:3000/sqrt

* Trying ::1...
* TCP_NODELAY set
```

```
* Connected to localhost (::1) port 3000 (#0)
> GET /sqrt HTTP/1.1
> Host: localhost:3000
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 400 Bad Request
< X-Powered-By: Express
< Content-Type: text/html; charset=utf-8
< Content-Length: 15
< ETag: W/"f-TU573BK7UfJ7GDG0u2GT//FoCx4"
< Set-Cookie:
connect.sid=s%3AkiA1DzCN3TtaaWeR3U0P2fT53C55ULwU.9cFwdoSU7f0RuhZ%2B%2FcRucYk
cKF2S9PC%2B5Sn1pcBCOGk; Path=/; HttpOnly
< Date: Mon, 11 May 2020 03:26:28 GMT
< Connection: keep-alive
<
* Connection #0 to host localhost left intact
Number required* Closing connection 0
```

Implementation Notes

Start off with the home page. You can generate the express application with any view templating system preferred, but I recommend using Handlebars. Required reading: https://handlebarsjs.com/guide/#what-is-handlebars
Getting the corporate jargon is done similarly to how we have been doing it on the client side. We have been using Browser implementations of the fetch API so far, but there is an implementation developed for Node.js on the backend as well named node-fetch.

Before starting the session work, knock out the square root API. Register a route handler for `/sqrt` that behaves according to specification. See documentation for sending a response. Test it using your preferred tool of choice (e.g. your browser, Postman, curl, and more.)

Finally finish the session work by setting up a route that the user can POST their form data to. That router handler should add the user's input to the session state. The next time the home page is requested, the list of items in the session should be passed into the template rendering engine to generate the numbered list. Required Reading: express-session documentation & example at bottom.

Extra Credit

+5 points will be provided for the extra credit. The extra credit should be submitted as a ***separate*** submission (Homework 6 - HTTP Backend Extra Credit).

The purpose of the extra credit on this assignment is to use an external session store instead of the default in-memory one for the basic configuration. As discussed in the lecture, this will mean that multiple running instances of the application behind a load balancer will function correctly because the session will be stored in a centralized shared database of some kind. You can choose any external database you are comfortable with, **but keep in mind that you will need to include any set up instructions so that I can recreate the setup on my local machine**. So if you're going to be using some kind of crazy Oracle SQL database set up... I will most likely not be able to accept the extra credit. I recommend following the KISS principle here (keep it stupid simple). You can use Redis, as it's simple to install and use (demo provided at end of lecture.) Or something *even simpler* would be to just use a plain old file on the filesystem
- https://www.npmjs.com/package/session-file-store

To test this, you will need to set up a load balancer. So you can set up any HTTP load balancer you would like to distribute requests evenly between the two running applications. Nginx was provided as an example in class, but one could also use something like PM2 which will probably be a lot simpler and have less configuration.

Please use Piazza to ask any questions!