Jonathan McCaffrey

CSCI 380

November 29, 2018

Writeup for Week 14 Assignment

**Current project hosted URL:** http://liontracks.teammccaffrey.com/

**Code hosted on GitHub:** <https://github.com/jmccaffrey42/csci380-project>

For my project I am building a basic task management system called “Lion Tracks”. The architecture for this system is as follows:

Back-end Application

PHP

Server

Browser

Front-end Application

JavaScript/HTML/CSS

Project Database

MySQL

There is a web frontend application written almost entirely in JavaScript using a few libraries such as ReactJs and jQuery. This application creates DOM elements as necessary to render an interactive user interface allowing someone login, navigate to the board, manipulate lists of cards, and update cards in detail. This front-end application then talks to the server over HTTP using REST verbs such as GET, POST, DELETE and PUT to transfer state contained in objects encoded in JSON.

The PHP service supports the following calls:

* **Card Lists**
  + GET /api/lists – Retrieves card lists
  + POST /api/lists – Create a new list
  + DELETE /api/lists/<id> – Delete a list
* **Cards**
  + GET /api/cards/<id> – Retrieve details about a card
  + POST /api/cards – Create a new card
  + PUT /api/cards – Update a card
  + DELETE /api/cards/<id> – Delete a card
* **Comments**
  + POST /api/comments – Create a comment
  + PUT /api/comments/<id> – Update a comment
  + DELETE /api/comments/<id> – Delete a comment
* **Auth**
  + POST /api/auth/login – Login
  + POST /api/auth/register – Create an account
* **Users**
  + GET /api/users – Retrieve all users
  + GET /api/users/<id> – Retrieve users by id
  + PUT /api/users/<id> – Update user

Calling the backend service with an HTTP client looks like this:

>> REQUEST

POST http://localhost:8000/api/auth/login

Content-Type: application/json

Accept: application/json

Cache-Control: no-cache

{

"email": "jmc@test.com"

}

<< RESPONSE

HTTP/1.1 200 OK

Host: localhost:8000

Date: Fri, 30 Nov 2018 07:59:31 +0000

Connection: close

X-Powered-By: PHP/7.2.12

Cache-Control: no-cache, private

Date: Fri, 30 Nov 2018 07:59:31 GMT

Content-Type: application/json

X-RateLimit-Limit: 60

X-RateLimit-Remaining: 59

{

"id": "486b857f-7c36-43fb-9af0-7384f108e22e",

"created\_at": "2018-11-26 06:38:49",

"updated\_at": "2018-11-26 06:38:49",

"email": "jmc@test.com",

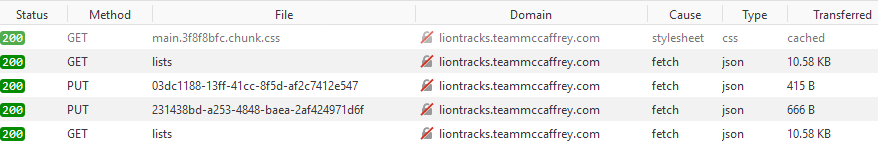
"name": "Jonathan McCaffrey",

"token": "486b857f-7c36-43fb-9af0-7384f108e22e"

}

When this request is received by the server, it issues a SELECT statement to MySQL which collects the correct row in the users table corresponding to the email given. If that row is found, it returns it to the PHP application which, in turn, encodes it in JSON and sends it back to the client as the HTTP response. The client interprets a “200” response code here as a correct login since this system doesn’t require a password.

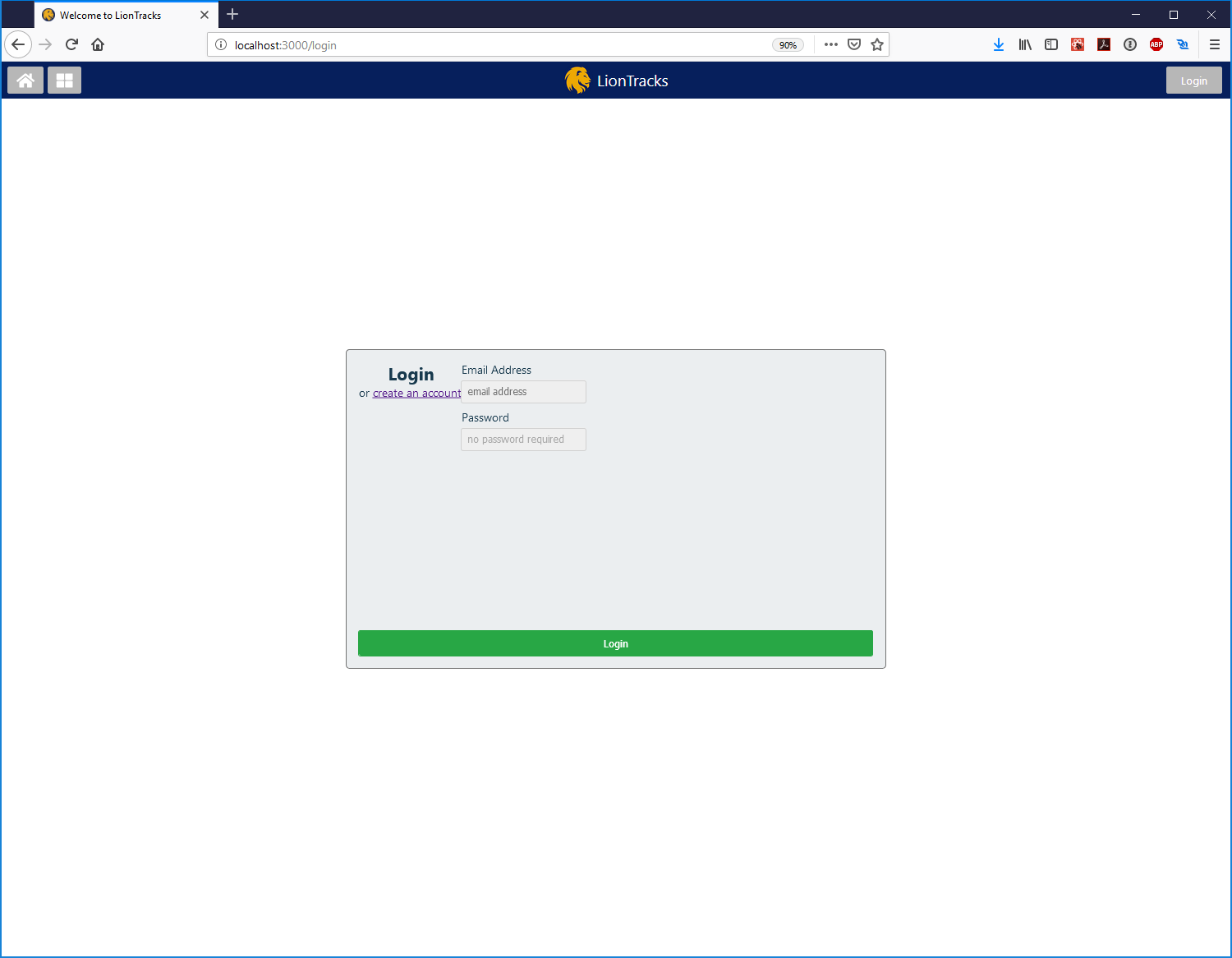
As you click around and edit things on the page, you can see in your browsers development console (under Network) that REST transactions are occurring to keep the server up-to-date with what you’re doing.



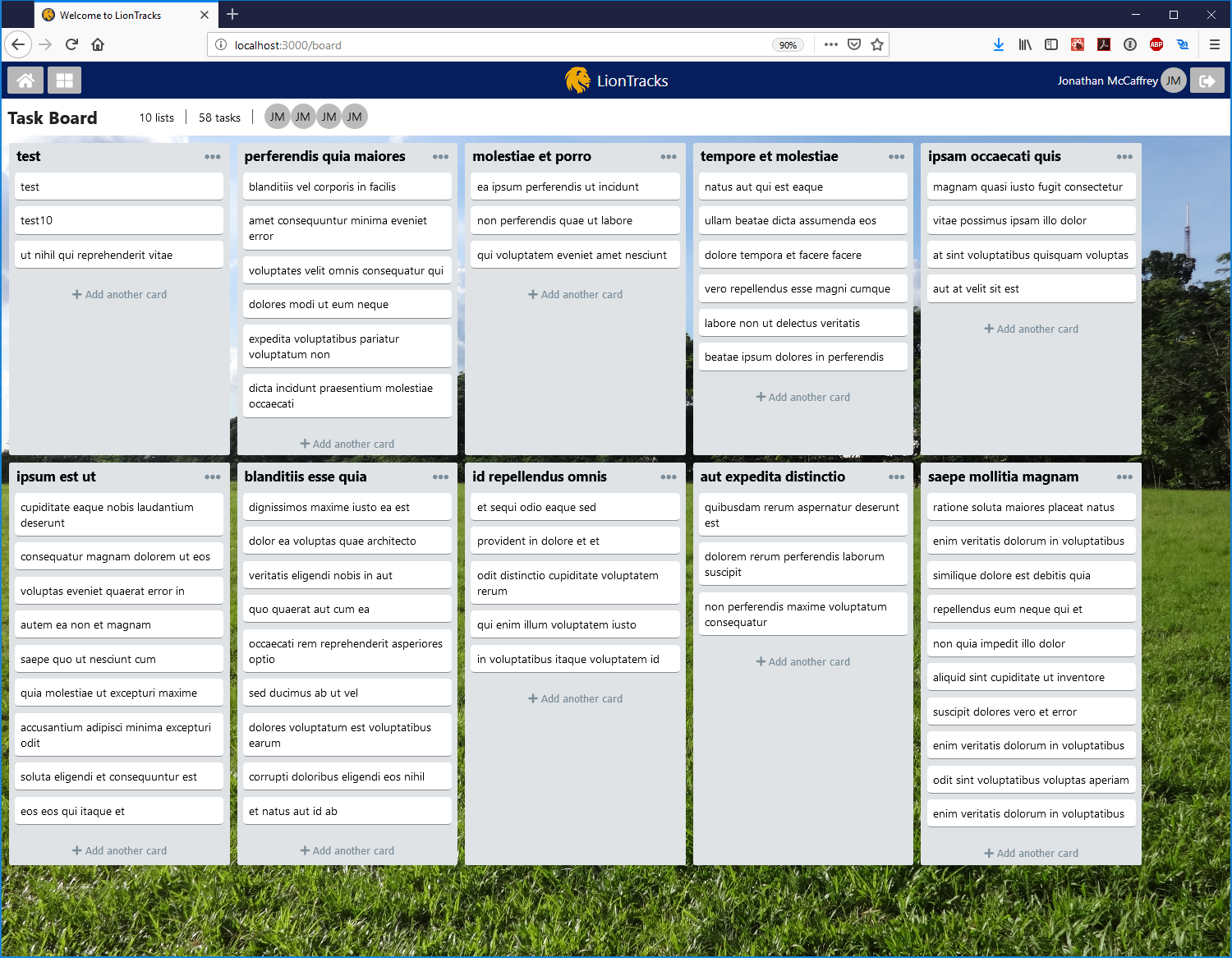
The JavaScript frontend contains the following routes:

* / - The home page, not yet styled but will contain an overview of the LionTracks system
* /login – The Login form with a link to the registration page
* /register – The registration form to create a new account
* /board – The interactive task board. This board has lists which contain cards. You can drag and drop the cards and lists to reorder them and move the cards between lists. Clicking on a title lets you edit it and clicking on a card takes you to a detail page where it can be edited or commented on.

Here is a screenshot of these pages in their current state:

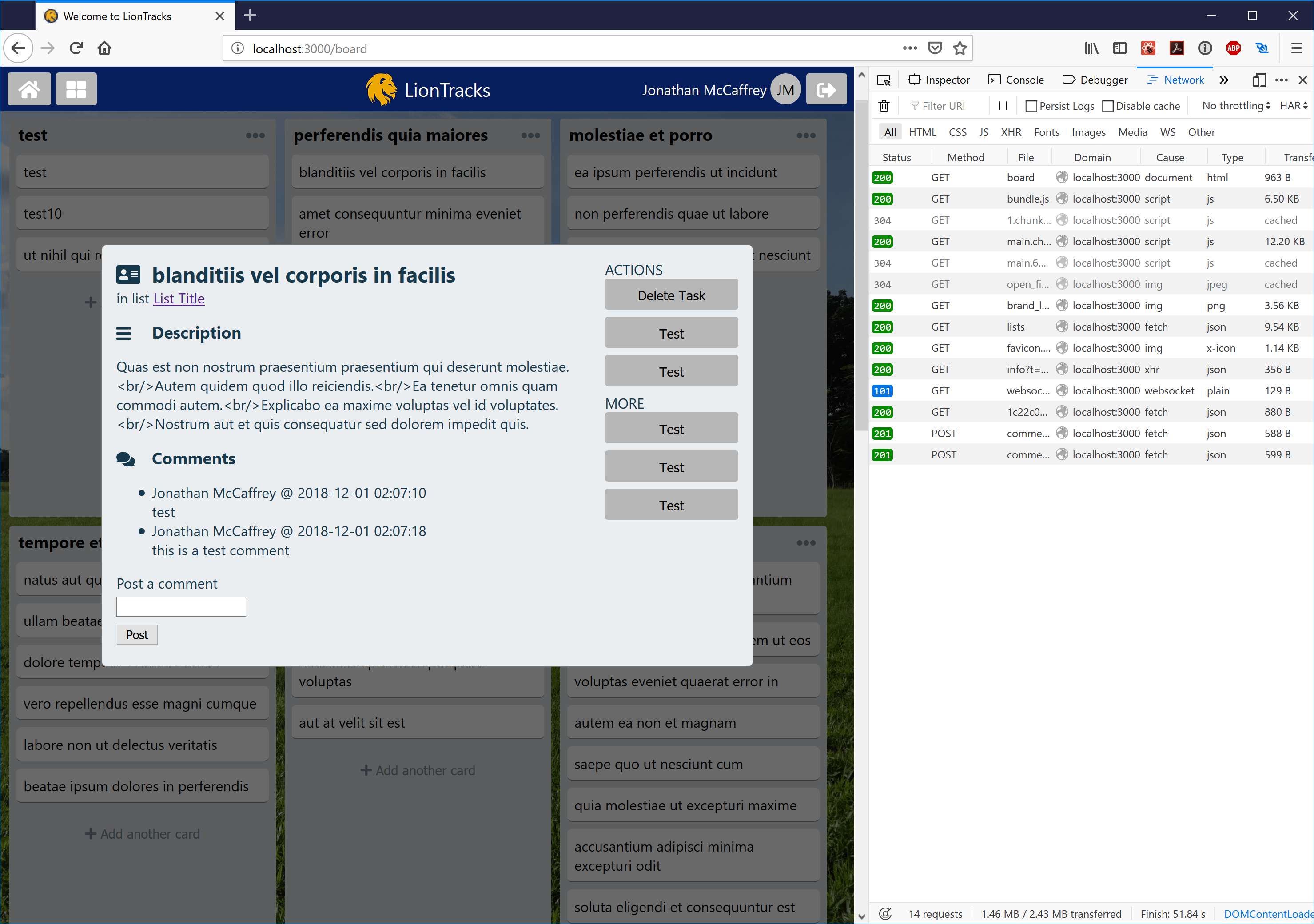
**Login Page**

**Board Page**



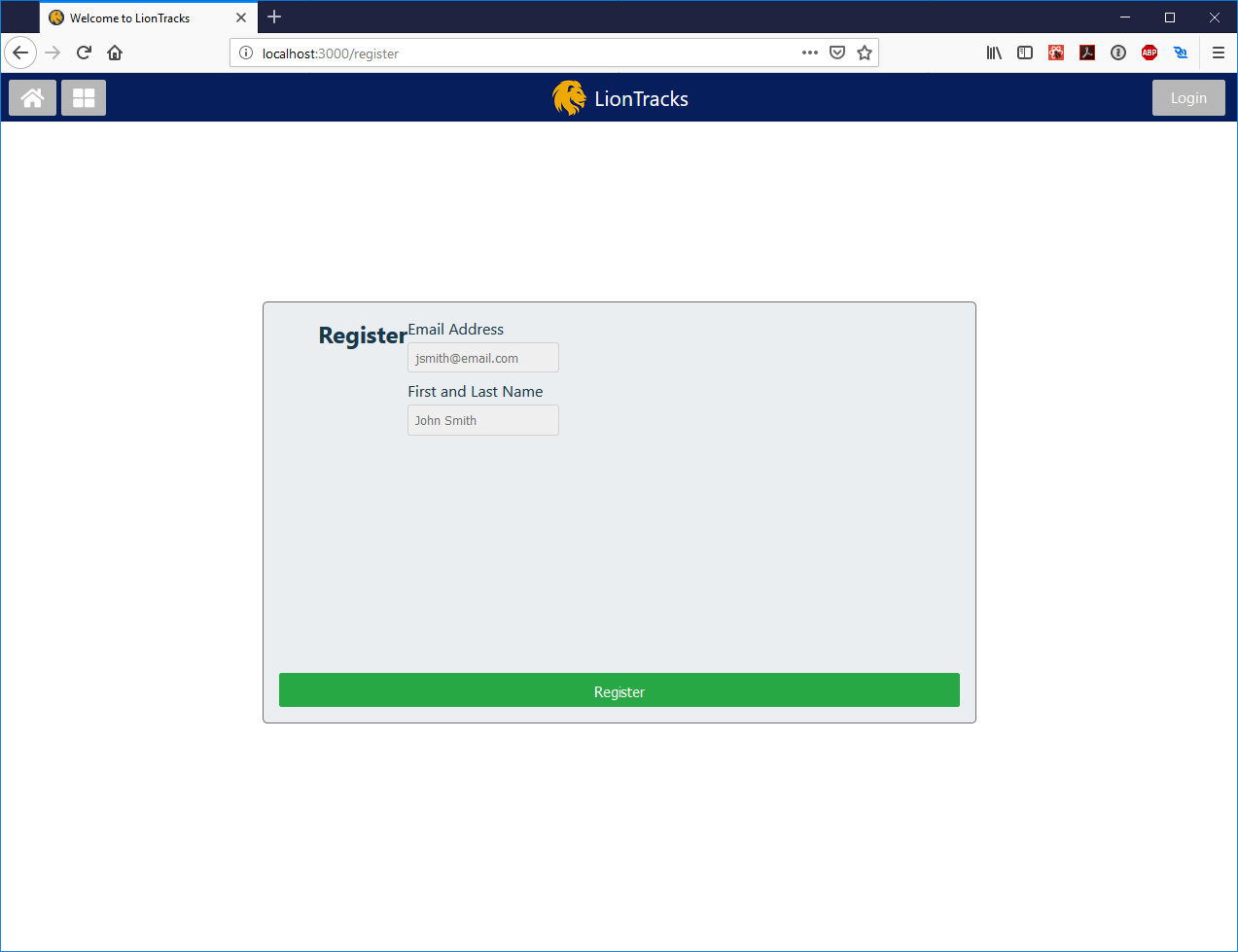
On this page users can create lists and cards, and then arrange them as they see fit. Right now in the screenshot I have lots of test data created so I can make sure and catch edge cases. When I deploy this for the final phase, I’ll clear out this data and populate it with more realistic cards and lists.

**Card Detail**

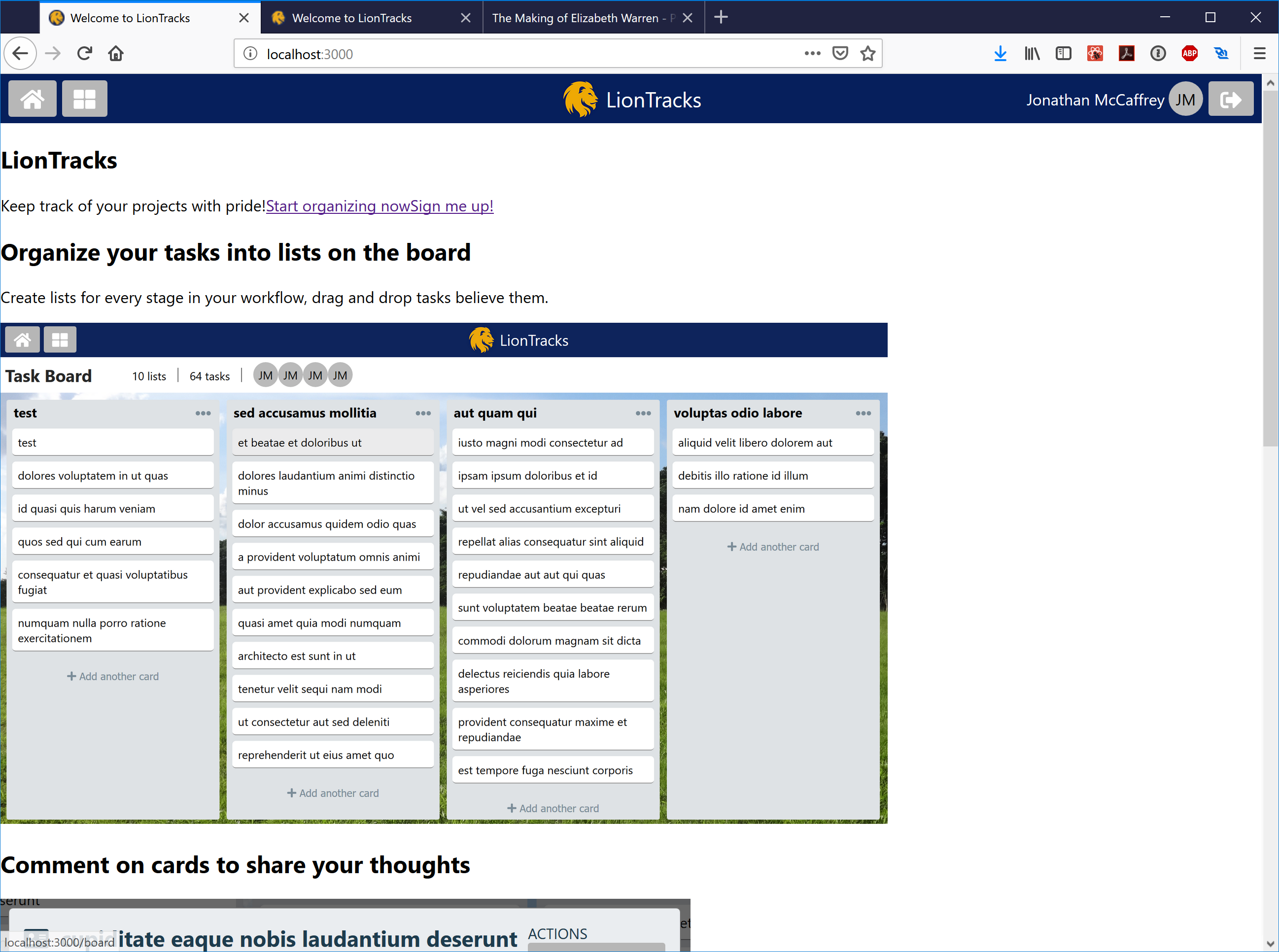


On this page the user can edit the card by clicking on either the title or the description and filling in what they want. When they press enter or click somewhere else, this change will be saved to the server by a PUT call. In the screenshot, the network debugger is open so we can see the REST transactions that the frontend is making with the backend as I post comments on the task.

**Register Page**



**Index Page**



This is an unstyled index page, I’m going to try and have it look more like a traditional marketing page with shadowed screenshots and a jumbotron image at the top.