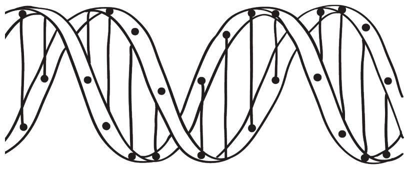

A1: *The MEAN Central Dogma*

Julian Mazzitelli, *iGEM UofT*

May 14, 2015

Due: May 22, 2015 or *anytime thereafter*

1 Introduction

This assignment will be an introduction to the modern full-stack web application while also promoting an improved understanding of the transfer of information within biological systems. You will build a single page application (SPA) with AngularJS which communicates with a RESTful Node.js (Express) *application programming interface* (API) and stores data on a MongoDB database. This is the MEAN stack:

MongoDB ↔ Node.js (Express) ↔ AngularJS

Moreover, you will write code in more than one language, interact with a number of frameworks/APIs, make a new best friend (documentation!), and consider the viability of your chosen data structures and algorithms for a given problem.

Your API will be capable of manipulating data forwards and backwards between each stage of biological information flow. You will not know what format the incoming data is, and will have to decide which conversion to use. That is, given DNA, RNA, or a polypeptide sequence, you must convert to each of the other two. Ultimately, you will describe the central dogma of molecular biology:

$$\begin{tikzcd}
DNA \arrow[r] & RNA \arrow[r] & protein \arrow[ll, bend left]
\end{tikzcd}$$

Note We will not consider DNA/DNA, RNA/RNA, or **protein/DNA** interactions in this assignment. Though you should realize the **crucial** implications of positive/negative regulation of DNA expression through protein/DNA interactions.

Prerequisites You **must** complete our Python DNA! assignment before starting this. You will call your python script from the back-end. You should also have completed javascripting, learnyounode, and expressworks. If you have trouble completing the DNA! assignment, consider trying out codecademy's Python tutorial.

2 Environment

It is necessary you have a proper development environment set up before beginning this assignment. You will need Python 3, Node.js and MongoDB. Your npm version should be up to date (Node.js comes with npm 2.7.4 but the latest is 2.9.something). Upgrading npm can be achieved with

```
$ npm install -g npm
```

If the previous does not upgrade your npm, it is most likely that you are on Windows and the path to Node.js occurs before the path to node_modules between your PATH and path environment variables (PATH loaded first). Once npm is up to date, you can install a few other global modules which you will be using:

```
$ npm install -g yo bower grunt-cli gulp
```

If you have issues starting a local MongoDB service, again, you are probably on Windows and what you need to do is

```
$ mkdir C:\data\db
```

and run (while you start `mongo` in another terminal)

```
$ mongodb -dbpath C:\data\db
```

If you have any issues getting set up, Google is your friend.

3 Requirements

Your app must:

- use a Python script to convert DNA → protein. You can a Python script from your request handler with ChildProcess.
- use JavaScript to convert protein → DNA
- define endpoints which can take in data from `req`, and send back the converted data. Perhaps one endpoint which catches all `get` requests and then internally sends it *post* endpoints which can accept params defining direction of conversion? It is all up to you. I will be looking for originality, clarity, and practicality.
- all converted sequences you have worked with should be stored as their DNA, RNA, polypeptide sequence (plus anything else worthwhile!) in a MongoDB database with Mongoose. Provide endpoints to retreive data from the database as well. (just some `get`s)
- use unit testing with Mocha

4 Boilerplates

You may find the following boilerplate generators useful. Try to reverse engineer the boilerplate and understand the flow of routes, models, etc. Look up all the `require` modules online to get an idea of what they do. However, in this tutorial we will be building our app *from the ground up*. That is, your solution should not be based off of any of these boilerplates, but checking them out and playing with their code is definitely recommended.

The generator made by the Express team:

```
$ npm install -g express-generator
```

A port of the above to yeoman:
`$ npm insall -g generator-express`
A popular MEAN boilerplate:
`$ npm install -g generator-angular-fullstack`
mean.io's generator
`$ npm install -g mean-cli`
A front-end only AngularJS generator
`$ npm install -g generator-gulp-angular`

5 The MEAN Stack

The MEAN stack is the acronym given to a full-stack web server using Node.js as it's serverside engine, with Express as a framework for Node, using MongoDB for a database, and using the frontend framework AngularJS which eases the difficulty of making single page applications. We will go through a quick intro to each of these members, but first an understanding of how a web application works must be acquired.

5.1 Static HTML

A simple, bare bones website is just an `html` file, or series of `html` files. HTML stands for *hypertext markup language* and is very simple to understand. Here is a basic `helloworld.html` webpage:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello world!</title>
    </head>
    <body>
        <h1>HELLO world?</h1>
    </body>
</html>
```

The first line must declare the doctype, and everything else must be contained within a `<html>` tag. A `<tag>` must always have its corresponding closing `</tag>`. Everything within `<head>` is where you declare the title which appears on your window, and other things such as stylesheets, JavaScript scripts, favicons, etc. All of the content that appears on your webpage will be placed inside the `<body>` tag. Here is a HTML5 cheatsheet, although the above and these are almost all the tags you will ever use:

```
<h1>...<h6>, <a href="www.igem.skule.ca">iGEM UofT</a>,
<a href="./localFile.html">tab 2</a>, <div>"division"</div>,
<ul> unordered list </ul>, <ol> ordered list </ol>, <li> list item </li>,
<img src="logo.png" />
```

Some tags are self closing, like `<img>`, but would still work as an open/close tag pair. Tags can have one id and multiple space seperated classes:

```
<tag id="myId" class="title active">
```

All that is left to finish this basic intro to a static web page is to mention stylesheets and scripts. I will leave this following example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>a naive randomized DNA sequence</title>
        <!-- <link rel="stylesheet" type="text/css" href="css/style.css"> -->
        <style type="text/css">
            * { margin:0; padding:0; }
            html, body { height: 100%; }
            #dna-bg { width:100%; height:100%; overflow:hidden; }
        </style>
    </head>
    <body>
        <div id="dna-bg"></div>

        <!-- <script src="js/main.js"></script> -->
        <script>
            var dna = '';
            for (var i=0; i < 100000; i++) {
                var r = Math.random();
                if (r > 0.75) dna += 'A';
                else if (r > 0.5) dna += 'T';
                else if (r > 0.25) dna += 'C';
                else dna += 'G';

                if (i % 1000 === 0 && i > 0) dna += '<br>';
            }
            var dnaBg = document.getElementById('dna-bg');
            dnaBg.innerHTML = dna;
        </script>
    </body>
</html>
```

Plain html pages work great for static websites like brochures and restaurant menus, but fail at delivering dynamic content.

5.2 Server-side processing

The creation of dynamic web pages such as forums required the development of server-side processing. There are many languages which can render html server side, among them Python(Django, Flask), .net, PHP and others. PHP has been the most popular choice for server side languages until Node came along. PHP stands for *Prepocessor Hyptertext: Php*, and it is exacly that. A PHP server will render each unique page of html and *then* send it to the client. So for example, on a forum, if you click on a thread, the client machine will tell the server which thread you want to look at, the server will then loop through each post in that thread in the database, compile a complete html page, and send it back to you. Visually, the flow is as follows:

$$\text{database} \leftrightarrow \text{server} \leftrightarrow \text{client}$$

Node can be used like this as well, to render each page. However in our webapp, Node will only act as an API, sending JSONs back and forth on get and post requests.

5.3 Node.js

Node.js is a runtime environment for server-side and networking applications. Node had it's first release in 2009. The core functionality of Node was written in JavaScript and C++ was used for connecting bindings and the operating system. Google's V8 engine, the JS execution engine built for Google Chrome, compiles JS into machine code, and is what Node uses to execute code. Previously, JS was interpreted in real time; V8 heavily optimized JS. Before Node, JavaScript could only be executed client-side. A big difference between Node and PHP is that PHP is a blocking language, while Node is non-blocking, allowing commands to execute in parallel and signal completion with a callback. A simple JSON API web server can be generated using Node with several lines:

```
var http = require('http');
var url = require('url');
var port = process.argv[2];

http.createServer(function(request, response){
    var pn = url.parse(request.url, true).pathname;
    var greeting = { from: 'me', to: 'you', message: '' };

    if (pn === '/hello') greeting.message = 'hello world!';
    else if (pn === '/bye') greeting.message = 'goodbye';

    response.end(JSON.stringify(greeting));
}).listen(port);
```

Test out this server yourself by running

```
$ node hw.js 9001
```

and visiting `http://localhost:9001/home` and `/bye`. For a more thorough introduction to Node.js, you are required to complete learnyounode. If you are unfamiliar with JavaScript, you may wish to do javascripting first.

5.4 Express

Express is a "fast, unopiniated, minimalist web framework for Node.js" Essentially, Express provides an API for working with Node that makes *everything a lot easier*. Consider the Express implementation of the example above:

```
var express = require('express');
var app = express();

app.get('/hello', function(req, res){
    res.send('hello w0rlD');
}).get('/goodbye', function(req, res) {
    res.send('gooood bye');
}).listen(3000);
```

As you can see, there is less code than the pure Node example, and the semantics are more readable. But Express does more than just that, you can set up *middleware* very easily, which is essentially and function in the *"middleware stack"* which can intercept the process of a *route*, for example, like `'/home'`. Middleware can modify the request and response objects and if they do not end the response, must call `next()`. It is required reading for this assignment to read:

- Basic Routing Tutorial
- Routing
- Using middleware

All of which can easily be found from the Express site. In fact, I highly recommended cloning the GitHub repo for Express. That way you can open up some example files on your own computer, and test them out on localhost. Specifically, I recommend taking a look at the *routing* examples, shown on the Express FAQ under "How should I structure my application?" Which method do like best? Which do you think is more organized? Easier to understand?

5.5 MongoDB

From Wikipedia:

> MongoDB (from humongous) is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON).

Once you install MongoDB, run it from your terminal with `$ mongo`, and call `help()` and `db.help()` and `db.mycol.help()` for help on general methods, database methods, and collection methods, respectively.

There used to be a really awesome web pseudo-shell interface which ran you through a 15min tutorial of MongoDB. But I can't find it, maybe they got rid of it (perhaps used deprecated methods and they didn't bother making a new one). In the mean time you can check out the MongoDB manual and I'll try to write a neat mini-tutorial soon. But the basics to understand are that a *database* is made of a bunch of *collections* which are just an array of *documents* following a specific *schema*. Try to find out how make a new database, collection, and documents from the `mongo` command line interface.

5.6 AngularJS

Ahhh, AngularJS. Ahhhh, frontend JavaScript frameworks. Welcome to the land of no return. Where plenty abound. An no one knows which is best. Oh and by the way, the new framework just mastered, version 2.0 is out now and that means there is no guarentte 1.0 API calls will be supported (i.e. some got deprecated). Well you need to choose one. We will be going with AngularJS since it seems to be the most popular right now.

AngularJS eases the process of setting up frontend routes and controllers for templates. You can define a wrapper template, a header template, a navbar template, and a view template. That way when you switch pages by clicking on a tab in the navbar, only the view template needs to be changed. Furthermore you set a controller for each template. Basically a controller will define the functions used on that portion of the page, for example, given a login form, define which values store username and password, and define the function which is used to do the actual logging in. Controllers modify with the `$scope` of a page. Then "factories" are for getting data, usually that is where you put all your `get` and `put` requests. Or you can define a "service" which you might use for something you deem "service worthy"(??). Ideally you would keep out as much logic out of the controllers as possible and just modify variables, call functions from there. AngularJS is a lot to explain but you can learn the basics online, check out the examples on their homepage and watch a YouTube video or too. I will try to add a minimal example AngularJS app here when I have time.

6 Addendum

If you have any questions, or something I said is unclear, please let me know! Send me a direct message on Slack or post in the #drylabtutorials channel. This assignment/tutorial is a WIP, I will try to do what I can with the time I have. However, ideally, I would like you to go out into that realm of exploration known as the *internet* and find out how to teach yourself that which you do not know. I cannot stress the importance of practice, practice, practice. Nor documentation! Always have a documentation tab open for whatever technology you are using, and always check out the documentation for a new module. Usually a modules repo readme runs over the basics. Have fun! By the end of this assignment you will have developed a modern full stack web server tied to a novel database structure capable of performing the information flow of the central dogma.