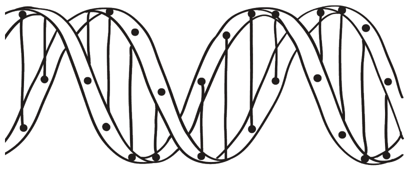

A1: *The MEAN Central Dogma*

Julian Mazzitelli, *iGEM UofT*

May 14, 2015

Due: May 20, 2015 or 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
- use JavaScript to convert protein → DNA
- accept data at endpoints of your choice
- provide endpoints to retreive data from database
- use unit testing with Mocha
- store converted data in the database

4 Boilerplates

You may find the following boilerplate generators useful.
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

5.5 MongoDB

5.6 AngularJS