Final Exam Practice Questions

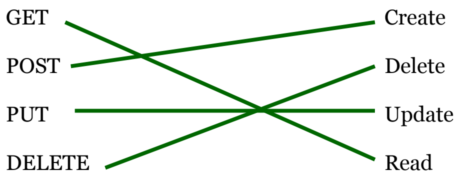
CSCI UA.0480-002 – Applied Internet Technology

1. What are some **options** (**name at least 3**) for **storing data** for your web application. **Discuss why you would or would not use** that option to store your data.

   Some possible answers:

   a) **in memory – however, data is lost whenever the application or server is restarted**
   b) **flat files on disk – persistent across application and/or server restarts, but may have a lot of overhead from manually reading, writing and querying data**
   c) **in a relational database – if data is highly structured, *entities* (or tables) are related to one another**
   d) **in a nosql database – great for quick prototyping or if you if you need a flexible data store where constraints and complex relationships are not required**
   e) **3rd party web services – for functionality that you don't want to implement yourself, client side heavy apps**

2. Draw lines to match the http request with the corresponding **CRUD** operation:



GET
POST
PUT
DELETE

Create
Delete
Update
Read

3. Answer the following questions about using MongoDB:

   a) How do you display all **databases** in the MongoDB shell?

```
show databases
```

   b) How do you display all **collections** in the MongoDB shell?

```
show collections
```

   c) What's a **collection**?

   A collection is a group of documents. (It's similar to a table in relational databases).

   d) Given the following movie related documents in a database called `moviedb` and collection called `movies`, what query would you use in the MongoDB shell to **retrieve all** movies?

```
[
    {'title':'Mauvais Sang', 'slug': 'mauvais-sang', 'year':1986},
    {'title':'Stroszek', 'slug': 'stroszek','year':1977},
    {'title':'Blue Velvet', 'slug': 'blue-velvet','year':1986},
    {'title':'Only Lovers Left Alive', 'slug':'only-lovers-left-alive', 'year':2013}
]
```

```
db.movies.find()
```

   e) Using the database/collection/documents from part d, what query would you use in the MongoDB shell to **retrieve all movies made in 1986**?

```
db.movies.find({'year':1986})
```

   f) Using the same database/collection/documents from the previous 2 questions, how would you **remove all of the movies**?

```
db.remove({}), db.movies.drop(), db.dropDatabase()
```

4. When generating an Express project, a project layout is automatically built for you. Describe what folders and files are created and what their significance is. Name and describe at least 4 files / directories.

```
project-name
            \_ bin
                  \_ www                entry point to your Express app, run with nodemon

            \_ public                   static files
                     \_ javascripts

                     \_ stylesheets

                     \_ images

            \_ routes                   where your application's routers are placed
                     \_ index.js

                     \_ users.js

            \_ views                    handlebars templates

            \_ app.js                   main app file, mostly for configuration

            \_ package.json             specifies your application's dependencies
```

5. What are the general steps for integrating Mongoose into your Express app?

   a) **require Mongoose**
   b) **define your Schema**
   c) **register your models**
   d) **connect to your database**
   e) **use models in your application (usually in your route files)**

6. Create a **Mongoose Schema** (or Schemas) to keep track of **contacts and their contact information**. It should support:

   a) storing a contact's first and last name, email, phone number and address
   b) each contact can have multiple addresses
   c) every address should contain a street address, city, state and zip code

   Write your Schema below. Although syntax doesn't have to be exactly correct, make an effort to write something that's reasonably close to what the actual code may look like.

```
var Address = new mongoose.Schema({
    street: String,
    city: String,
    state: String,
    zip: String
});

var Contact = new mongoose.Schema({
    first_name: String,
    last_name: String,
    email: String,
    addresses: [Address]
});
```

7. List at least two best practices for saving passwords in a database. Describe the rationale behind each best practice.

a) **passwords should not be in clear text – in case of unauthorized access to your database, you don't want passwords to be easily readable**
b) **passwords should be hashed – passwords should be hashed instead of encrypted so that they cannot be decrypted (you should use a one-way hash)**
c) **passwords should be salted – to prevent brute force attacks using known hashes, prefix your string with a *salt***

8. Using the sample movie database, collection and documents from question #3, write the code that would go into your router, `index.js`:

a) Assume that you have a corresponding Mongoose Schema that matches the fields of the documents from question 3 (**the slug will be handled automatically** by a url slug plugin, so you will not have to worry about that in your form or router)
b) Additionally, a **model** related to the schema has been registered under the **name `Movie`**
c) Imagine that you have the form markup listed at the end of this question
d) In your router, `index.js`...
e) (Remember to retrieving the model constructor, `Movie`)
f) Write **two route handlers**...
g) Create a route handler that accepts a **POST** from the form at the end of this question (determine the url to use from the form)
    - it should create and save a new movie, and redirect you to the movie detail page of the movie just created
    - the movie detail page will be at the url: `/movie/[movie-slug]` (for example `/movie/blue-velvet`)
h) Add another route that handles a GET to the url : `/movie/[movie-slug]`
    - this route handler should retrieve the movie identified by the slug in the url from the database
    - ...and it should send over the movie's information to a template called movie-detail

```
<form method=”POST”  action=”/movie”>
  <input type=”text” name=”title”>
  <input type=”text” name=”year”>
</form>
```

In `router.js,` you'll have some of the following setup code:

```
var express = require('express');
var router = express.Router();
var mongoose = require('mongoose');
```

Assume route handlers to get the form, etc ... are present in this file. Your code goes below (it does not have to be exactly syntactically correct, but it should be *reasonably* close):

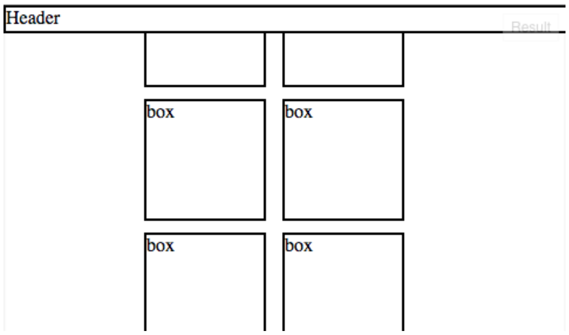
```
var Movie = mongoose.model('Movie');

router.post('/movie', function(req, res) {

    var movie = new Movie({
        title: req.body.title;
        year: req.body.year;
    })

    movie.save(function(err, movie, count) {
        res.redirect('/movie/' + movie.slug);
    });
});


router.get('/movie/:slug', function(req, res) {
    var slug = req.params.slug;
    Movie.findOne({slug:slug}, function(err, movie, count) {
        res.render('movie-detail.hbs', movie);
    });
});
```

9. In the previous question where your web application receives input from a user:

a) describe where in your application you can add validation
b) specify where ***must*** you have validation and include why

You can have client and server-side validation. You must have server-side validation because client side validation with JavaScript will not catch non-browser requests (for example, using an http request library, like node's request... or even just using commandline tools like curl).

10. Create the markup and CSS for the following layout below:



a) When the page is scrolled, the header does not move (it sticks to the top)
b) There is a grid of boxes on the page
c) The pixel dimensions do not have to be exact
d) The number of columns does not have to match exactly

Markup

```
<header>
    Header
</header>
<div id="container">
    <div class="box">box</div>
    <div class="box">box</div>
    <div class="box">box</div>
    <div class="box">box</div>
    <div class="box">box</div>
    <div class="box">box</div>
    <div class="box">box</div>
    <div class="box">box</div>
</div>




<!-- Looking specifically for position and display
properties that allow you to have a fixed header...
and a grid of elements with a height and width. Other
solutions, such as using a table or using floats
would work as well.-->
```

CSS

```
header {
  width:100%;
  border:2px solid #000;
  z-index:1;
  position: fixed;
  height:20px;
  top: 0;
  left: 0;
  background-color:white;

}

#container {
    margin-top:20px;
    margin-left: auto;
    margin-right: auto;
    width: 250px;
}

.box {
    display:inline-block;
    width: 100px;
    height: 100px;
    margin:5px;
    border: 2px solid #000;
}
```

11. **Write client side markup and code to**:

a) display a page with one button

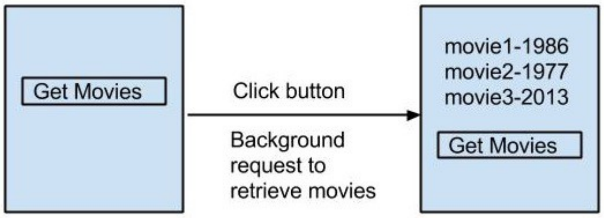
b) when the button is clicked, send a background request (that is, no page reload; your JavaScript should send a request *behind the scenes*)

c) the request will got an api; the api is located at localhost:3000/api/movies (assume that the domain is the same domain that your client side script originates from)

d) the response from the api will be a JSON document that is exactly the same as the list of documents shown in #3, part d

e) once it successfully retrieves the document, insert every movie into the page's body as paragraph tags

f) each paragraph tags will contain the title and year of a move separated by a dash



g) the code does not have to be exact syntactically, but *reasonably* close

```
<input id=”submitButton” type=”button” value=”get movies”>

document.getElementById(“submitButton”).addEventListener('click', function(evt) {
    var url = 'http://localhost:3000/api/movies';
    var req = new XMLHttpRequest();
    req.open('GET', url, true);

    req.addEventListener('load', function() {
        if (req.status >= 200 && req.status < 400) {
            var movies = JSON.parse(req.responseText);
            movies.forEach(function(movie) {
                var p = document.body.appendChild(document.createElement('p'));
                p.textContent = movie.title + '-' + movie.year;
            });
        }
    });
});
```

12. Write CSS selectors to pick out the specified elements from the markup below.

```
<h1>Header</h1>
<div>Hello!</div>
<div class="content">
    <a>Link 1</a>
    <p>
        <a>Link 2</a>
    </p>
</div>
<a>Link 3</a>
<a data-priority>Link 4</a>
```

a) All a tags _____a________________________

b) All the a tags nested under the div with class content _____.content a __________________

c) Only the a tags that are the direct descendent of the div with class content ____.content > a________________

d) Both h1 tags and div tags in one selector _____h1, div____________________

e) (Extra) Only the a tags with an attribute of data-priority _____a[data-priority]___________

13. List three methods/functions that allow you to pick out / obtain HTML Elements

document.getElementById

node.getElementsByClassName

node.getElementsByTagName

document.querySelector

document.querySelectorAll

14. When and where should client side JavaScript be loaded? Why?

a) **At the end of the body – in case code depends on DOM elements that need to be loaded**

b) **Preferably in an DOMContentLoaded event - guarantees that entire DOM is loaded**

15. Using the following CSS and markup, animate the div with id, animateMe, so that it travels across the page from left to right.

```
<div id="animateMe"></div>
```

```
#animateMe {
    position:absolute;
    width: 50px;
    height: 50px;
    background-color: #0f0;
    top:0px;
    left:0px;
}
```

```
setInterval(animate, 50);
var x = 0;
function animate() {
    var greenBox = document.getElementById('animateMe');
    greenBox.style.left = x + 'px';
    x += 1;
}
```

16. Describe 3 possible values for the CSS display property... and what effect they have on layout.

a) **block: line break, has width and height**

b) **inline: same line, does not have width and height**

c) **inline-block: same line, has width and height**