THE UNIVERSITY OF SYDNEY

School of Information Technologies

COMP5347: Web Application Development
Week 9 Tutorial: Mongoose and the model layer

Learning Objectives

- Understand basic mongoose concepts and objects
- Integrate mongoose with Expressjs

Part 1: Add mongoose module

Task: Download the provided `week9-src.zip` from Canvas and extract the content in a directory. Start Eclipse, select "open projects from file system" under file menu, go to the directory you just extracted the content and click open. In this case, all the necessary node modules are specified in `package.json`. Right click it on the project explorer panel, select `Run as` then `npm install`.

Optional: Just keep in mind, in your own project, you need to run the following command in the terminal to install install mongoose manually.

```
npm install mongoose --save
```

Alternatively, you may open file "package.json" and add the following line at the end of the "dependencies"

```
"mongoose": "^4.3.0",
```

Save the file and right click it on the project explorer panel to `Run As` then `npm update`.

Part 2: Simple mongoose query

Task: Start the mongodb server using the following command, replace the "path" with a path to the local mongodb working directory where you create the `wikipedia` database with the `revisions` collections.

```
mongod --dbpath path
```

Run the code by right click the `/app/models/mongoose.revisions.js` on project explorer panel and run as `node.js application`.

Optional: line 3, the connection to mongodb is configured. If you give a different name to the Wikipedia data set when you import the data, the configuration might be different. In this case, you can replace the "dbname" in the following path with the name you created before. If you are not sure about the dbname, you may open the robomongo db browser and check it there.

'`mongodb:://localhost/dbname`'

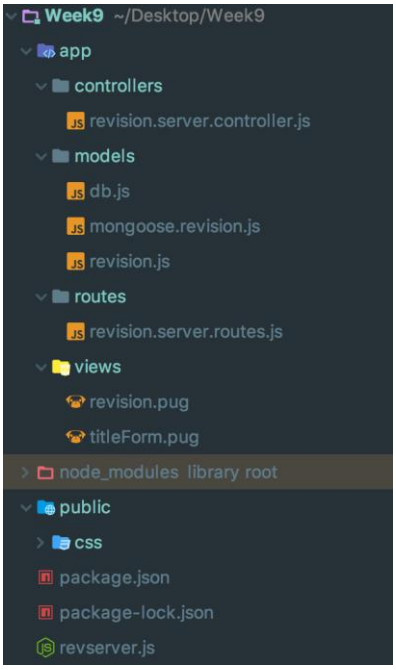
You need to uncomment the lines representing the update statement to try the update example. You may notice that writing queries, updates and aggregation statements in mongoose follow the same syntax as mongodb shell commands.

Try to write your own code to find the user who made the most non-minor revisions on article "CNN" and run `mongoose.revisions.js` again to test that you have obtained the correct results. You can also compare the results you get from mongoose to the results you get from respective shell command.

Part 3: Full MVC Express Application

This part will build a small application with mongodb as backend database server. It allows an end user to type in an article's title to find its latest revision data from the `revisions` collection.

Task: Understand the project structure



The view layer contains two views: one for displaying the form for entering title. The other for displaying revision data. Both views are provided as PUG files: `titleForm.pug` and `revision.pug`.

The model layer is supposed to handle data of the application. Database related code should be put in the model layer. Our simple application only needs to handle revision data, it has a module represent revision mode called `revision.js`. It also has a module called `db.js` which contains database initialization code.

Optional: Same as before**,** line 3 of db.js, the connection to mongodb is configured. If you give a different name to the Wikipedia data set when you import the data, the configuration might be different. In this case, you can replace the "dbname" in the following path with the name you created before. If you are not sure about the dbname, you may open the robomongo db browser and check it there.

'`mongodb://localhost/dbname`'

The `revision.js` script provides a static method `findTitleLatestRev`. This method will find an article's latest revision given a title. It does not define its own callback function.

The application also contains a controller `revision.server.controller.js`, which calls `findTitleLatestRev` and provides a callback function when the result is back.

The `revserver.js` contains code to create and start the application.

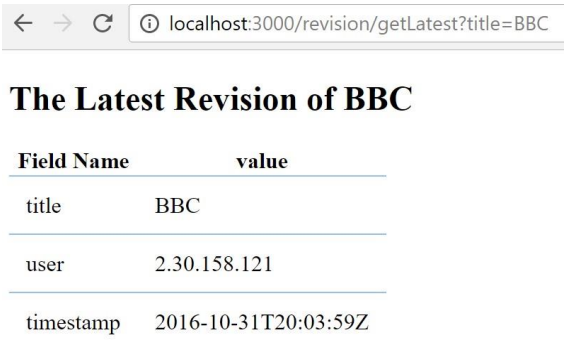
Task: Implement the router

The only part missing is a route file that will bind controller methods to urls. Inspect all relevant code to determine the content of the route file and implement it in `revision.server.routes.js` under `app/routes`

Task: Run the application

Run the start file as node.js application after you have implemented the route file. Once started, points your browser to `localhost:3000/revision`. It should display a form. When you type in a title "BBC", the request will be sent, and the server will query mongodb to find the latest revision on article BBC and send it back. See below screenshots for sample output.

(a) A simple form



(b) Result

Part 4: Introduction to Promise Library

Node.js naturally do not block IO as other mainstream programming languages do. In that case, if you want to write some code based on the output of previous function, you cannot just put the statement at the following line. Alternatively, you should use callback function to chain the following codes with the previous function. Most node core modules such as 'fs' module embrace this style in the nutshell.

When your logic gain complex, there will be a lot of layers of callback function. The term "callback hell" is coined to describe this situation. Promise is one of various remedial solutions, supported by various npm packages such as 'mongoose' and 'request.js'. Bluebird[1] promise library is taken as an example due to its popularity among node.js ecosystem.

Bluebird can be used to turn this:

```
fs.readFile("BBC.json", function (err, val) {
    if (err) {
        console.error("unable to read file");
    }
    else {
        try {
            val = JSON.parse(val);
            console.log(val);
        }
        catch (e) {
            console.error("invalid json in file");
        }
    }
});
```

[1] http://bluebirdjs.com/docs/getting-started.html

Into this:

```
fs.readFileAsync("BBC.json")
    .then(JSON.parse)
    .then(function (val) {
        console.log(val);
    })
    .catch(SyntaxError, function (e) {
        console.error("invalid json in file");
    })
    .catch(function (e) {
        console.error("unable to read file");
    });
```

Task: Download Promise.zip from Canvas and extract it to some local directory**.** Import the `promise.js` and `BBC.json` to the project root directory. Execute `promise.js` file by right clicking the file name on project explorer panel and `run as node.js application`. Uncomment the `fs.readFileAsync` function to compare the outcome of 'callback' style and 'promise' style code segments. Of course, the outcome should be the same.