# Grad Apps 2.0 Testing Documentation

**Edward Vaisman** **Sadman Sakib Hasan**

April 26, 2018

## Contents

## 1 System Overview

GradApps 2.0 is a business system application, which allows, our client to *automate* the selection of the best candidate into the EECS Graduate Program by *minimizing* the manual work to be done. Our client are the members of **The EECS Graduate Program**.

The application is broken down into three user levels: *Administrator*, *Committee Member* and *Professor*. Each of the roles play a crucial part in order to select the best candidate into the graduate program. GradApps 2.0 operates as a web application, hence, a reliable internet connection is required when interacting with the application.

# 2 System Summary

This section provides a general overview of the system. The summary outlines the uses of the system's hardware and software requirements, systems configuration, user access levels and systems behavior in case of any contingencies.

## 2.1 System Configuration

### 2.1.1 Browser Configuration

GradApps 2.0 operates as a web interface application. It supports all modern web browser, however, Chrome and Mozilla Firefox are the recommended browser for using the application. The application is recommended to be only used through desktop browsers.

**Recommended Browser(s):**

- **Chrome**: >= Chrome v60.0.3112
- **Mozilla Firefox**: >= Firefox 57 (v57.0a1)

### 2.1.2 Node.js Configuration

GradApps 2.0 is built using Node.js and it is vital to use the correct version of the node package manager (npm) and the node.

**Recommended Node.js version:**

- **Node**: >= v8.9.4
- **NPM**: >= 5.6.0

### 2.1.3 MySQL Configuration

GradApps 2.0 uses MySQL commercial database as the datasource manager.

**Recommended MySQL version:** >= 5.7.20

The following environmental variables are required to be set prior to starting the application.

- **MYSQL_HOST**: the host name of the database server

- **MYSQL_PORT**: the port number of the database server
- **MYSQL_USER**: the user name to access the database
- **MYSQL_PASSWORD**: the password to access the database
- **MYSQL_DATABASE**: the app database name (default: "gradapps")

The following environmental variables are required to be set prior to running the test cases on the application.

- **TEST_MYSQL_HOST**: the host name of the database server for testing
- **TEST_MYSQL_PORT**: the port number of the database server for testing
- **TEST_MYSQL_USER**: the user name to access the database for testing
- **TEST_MYSQL_PASSWORD**: the password to access the database for testing
- **TEST_MYSQL_DATABASE**: the test database name (default: "testdb")

### 2.1.4 Test Configuration

The testing framework used for automating GradApps 2.0 is **Mocha** for the underlying backend testing and **Protractor** for the frontend testing. The default browser used for Protractor is Chrome, however, other browsers can be added to the *config.js* file.

```
multiCapabilities: [{
  'browserName': 'firefox'
}, {
  'browserName': 'chrome'
}];
```

### 2.1.5 Other Configuration

The default port for the application web server is 3000. However, it can be set to one of your choice by enabling the PORT environmental variable.

## 3 Getting Started

This section explains how to get GradApps 2.0 on the machine, install it and start the application.

To install the application, it has to be pulled from the private GitHub repository as it is not a published application for other uses.

To clone the repository in the server machine, please make sure Git is installed in the machine. Along with Git, all the above configuration mentioned in Section 2.1 must be installed.

**Recommended Git version:** >= 2.3.2

1. Git clone the repository in the current working directory, run the following command:

```
$ git clone https://github.com/ssh24/EECS4090-Project.git
```

2. Change the working directory to the source of the project:

```
$ cd EECS4090-Project/src/
```

3. Install the required dependencies:

```
$ npm install
```

4. Set the required environmental variables:

```
$ SET TEST_MYSQL_HOST = <host>
$ SET TEST_MYSQL_PORT = <port>
$ SET TEST_MYSQL_USERNAME = <username>
$ SET TEST_MYSQL_PASSWORD = <password>
$ SET TEST_MYSQL_DATABASE = <database>
$ SET PORT = <app_port>
```

## 4 Test Suites

GradApps 2.0 has a list of exhaustive test suites to make sure the application works as intended. The two main test suites are:

- **Database Tests**: suite consists of running test cases against the test database to make sure the underlying backend system works as expected.
- **Application Tests**: suite consisting of a combination of acceptance tests with respect to the software requirement documentation and unit tests with respect to the software implementation. This suite is much more stronger than just an acceptance test suite as it covers more paths than an acceptance test suite would.
- **Overall Tests**: suite consisting both the database and application tests.

### 4.1 Database Tests

The database test suite consists of test cases to run against the test database to validate the structure of the database. The framework used for database tests is **Mocha**. To change the current directory to the database tests directory, do the following from the source directory:

```
  $ cd test\specs\database
```

Following is an example of a database test case:

```
it('assign a valid review to a valid committee member', function(done) {
  review.assignReview(12, 20, 1, function(err, result) {
    if (err) done(err);
    assert(result, 'Result should exist');
    done();
  });
});
```

### 4.1.1 Add new test file

Test files are separated by feature types. For example: review test, application test and faculty member test. To create a test file, do the following from the database suite directory:

```
  $ touch <new_test_file>
```

Once a new test file has been creating, you can go ahead and start writing test cases. Following is an example test file with some test cases:

```
describe('Util Functionalities', function() {
  before(function(done) {
    connection = mysql.createConnection(creds);
    utils = new Utils(connection);
    connection.connect(done);
  });

  after(function(done) {
    connection.end(done);
  });

  describe('get member id function', function() {
    it('get member id of a valid username', function(done) {
      utils.getMemberId('arri', function(err, result) {
        if (err) done(err);
        assert(result, 'Result should exist');
        done();
      });
    });

    it('get member id of an invalid username', function(done) {
      utils.getMemberId('some_id', function(err, result) {
        assert(err, 'Error should exist');
        assert(!result, 'Result should not exist');
        done();
      });
    });
  });
});
```

### 4.1.2 Running test suite

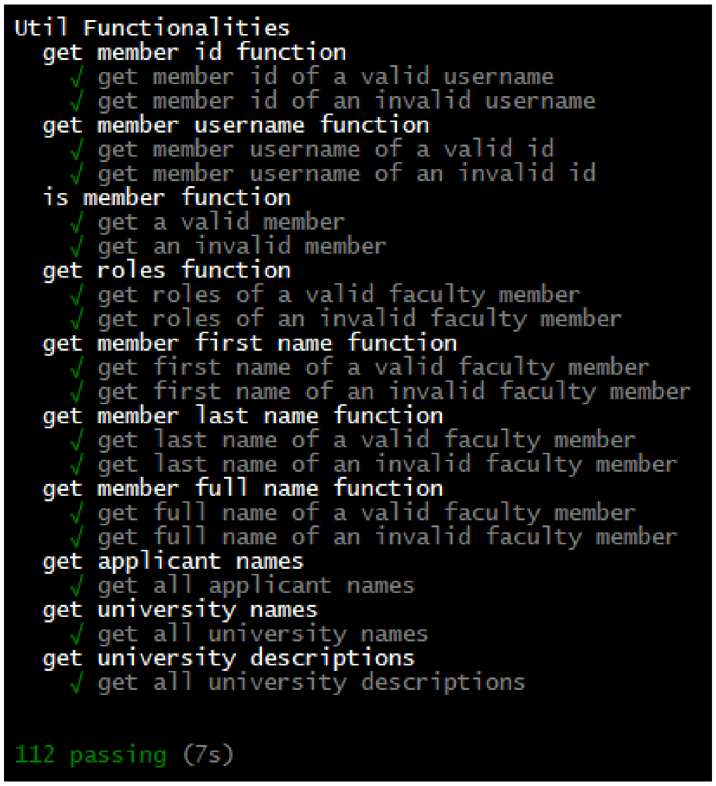
Once you have added a new test file with some test cases you can simply run the entire database test suite using the following command from the source directory:

```
$ npm run db:test
```

**Pro-tip**:

- To run a single test file or a single test case , add *.only* to the *describe* or *it* hook respectively.
- To skip a single test file or a single test case , add *.skip* to the *describe* or *it* hook respectively.

Example of a running test suite:



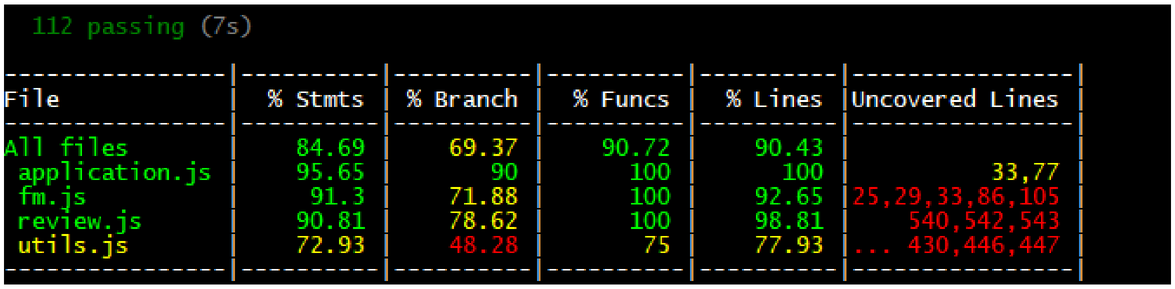
```
Util Functionalities
  get member id function
    √ get member id of a valid username
    √ get member id of an invalid username
  get member username function
    √ get member username of a valid id
    √ get member username of an invalid id
  is member function
    √ get a valid member
    √ get an invalid member
  get roles function
    √ get roles of a valid faculty member
    √ get roles of an invalid faculty member
  get member first name function
    √ get first name of a valid faculty member
    √ get first name of an invalid faculty member
  get member last name function
    √ get last name of a valid faculty member
    √ get last name of an invalid faculty member
  get member full name function
    √ get full name of a valid faculty member
    √ get full name of an invalid faculty member
  get applicant names
    √ get all applicant names
  get university names
    √ get all university names
  get university descriptions
    √ get all university descriptions


112 passing (7s)
```

Figure 1: Running Test

### 4.1.3 Test coverage

Once the test suite has completed running it will give a resulted output of the code coverage from the database test suite. The code coverage is also saved in the source directory under "coverage" folder.

Example of the code coverage output after running the database test suite:



```
  112 passing (7s)

----------------|----------|----------|----------|----------|----------------|
File            |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------|----------|----------|----------|----------|----------------|
All files       |    84.69 |    69.37 |    90.72 |    90.43 |                |
 application.js |    95.65 |       90 |      100 |      100 |          33,77 |
 fm.js          |     91.3 |    71.88 |      100 |    92.65 |25,29,33,86,105 |
 review.js      |    90.81 |    78.62 |      100 |    98.81 |    540,542,543 |
 utils.js       |    72.93 |    48.28 |       75 |    77.93 |... 430,446,447 |
----------------|----------|----------|----------|----------|----------------|
```

Figure 2: Database Suite Coverage

## 4.2 Application Tests

The application test suite consists of a combination of acceptance tests with respect to the software requirement documentation and unit tests with respect to the software implementation. This suite is much more stronger than just an acceptance test suite as it covers more paths than an acceptance test suite would. The framework used for application tests is **Protractor**. To change the current directory to the application tests directory, do the following from the source directory:

```
  $ cd test\specs\unit
```

Following is an example of an application test case:

```
it('- log in with invalid password', function() {
  login.enterUsername(config.credentials.app.admin.username)
    .then(login.enterPassword.call(login, config.credentials.app.admin
      .password + '1'))
    .then(login.clickLogIn.call(login))
    .then(expect(browser.getCurrentUrl()).to.eventually
      .contain('login'))
    .then(expect(login.getErrorMessage.call(login)).to.eventually
    .equal('Invalid password. Please try again.'));
});
```

### 4.2.1 Add new test file

Test files are separated by feature types. For example: review test, application test and faculty member test. To create a test file, do the following from the application suite directory:

```
  $ touch <new_test_file>
```

Once a new test file has been creating, you can go ahead and start writing test cases. Following is an example test file with some test cases:

```
describe('Login Test', function() {
  this.timeout(20000);

  var login = new Login();
  var role = new Role();
  var utils = new Utils();
  var welcome = new Welcome();

  before(function () {
    utils.startApp();
    utils.openView('#');
    utils.maximizeBrowserWindow();
    welcome.clickSignInButton();
  });

  after(function (done) {
    require('../../pretest');
    browser.restart();
    utils.stopApp(done);
  });

  it('- log in with only username no password', function() {
    login.enterUsername(config.credentials.app.admin.username)
      .then(login.clickLogIn.call(login))
      .then(expect(browser.getCurrentUrl()).to.eventually
        .contain('login'));
  });

  it('- log in with only password no username', function() {
    login.enterPassword(config.credentials.app.admin.password)
      .then(login.clickLogIn.call(login))
      .then(expect(browser.getCurrentUrl()).to.eventually
        .contain('login'));
  });
});
```

### 4.2.2 Running test suite

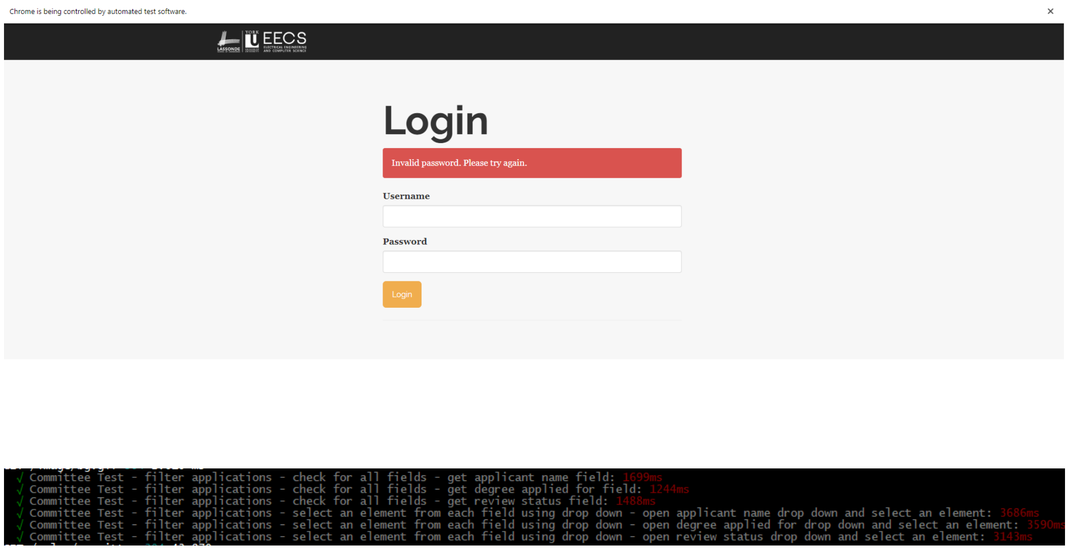
Once you have added a new test file with some test cases you can simply run the entire application test suite using the following command from the source directory:

```
$ npm run unit:test
```

**Pro-tip**:

- To run a single test file or a single test case , add *.only* to the *describe* or *it* hook respectively.
- To skip a single test file or a single test case , add *.skip* to the *describe* or *it* hook respectively.

Example of a running test suite:


Chrome is being controlled by automated test software.
EECS
Login
Invalid password. Please try again.
Username
Password
Login

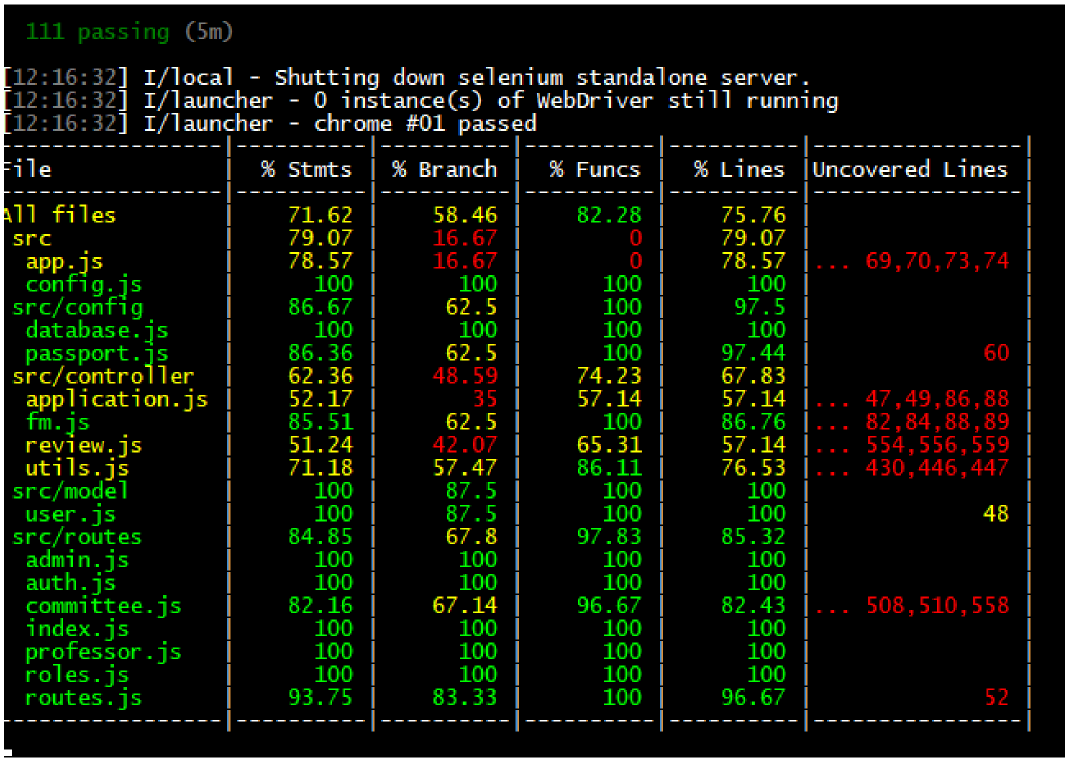
```
 √ Committee Test - filter applications - check for all fields - get applicant name field: 1699ms
 √ Committee Test - filter applications - check for all fields - get degree applied for field: 1244ms
 √ Committee Test - filter applications - check for all fields - get review status field: 1488ms
 √ Committee Test - filter applications - select an element from each field using drop down - open applicant name drop down and select an element: 3686ms
 √ Committee Test - filter applications - select an element from each field using drop down - open degree applied for drop down and select an element: 3590ms
 √ Committee Test - filter applications - select an element from each field using drop down - open review status drop down and select an element: 3143ms
```

Figure 3: Running Test

### 4.2.3 Test coverage

Once the test suite has completed running it will give a resulted output of the code coverage from the database test suite. The code coverage is also saved in the source directory under “coverage” folder.

Example of the code coverage output after running the application test suite:



```
  111 passing (5m)

[12:16:32] I/local - Shutting down selenium standalone server.
[12:16:32] I/launcher - 0 instance(s) of WebDriver still running
[12:16:32] I/launcher - chrome #01 passed
-----------------|----------|----------|----------|----------|----------------|
File             |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-----------------|----------|----------|----------|----------|----------------|
All files        |    71.62 |    58.46 |    82.28 |    75.76 |                |
 src             |    79.07 |    16.67 |        0 |    79.07 |                |
  app.js         |    78.57 |    16.67 |        0 |    78.57 |... 69,70,73,74 |
  config.js      |      100 |      100 |      100 |      100 |                |
 src/config      |    86.67 |     62.5 |      100 |     97.5 |                |
  database.js    |      100 |      100 |      100 |      100 |                |
  passport.js    |    86.36 |     62.5 |      100 |    97.44 |             60 |
 src/controller  |    62.36 |    48.59 |    74.23 |    67.83 |                |
  application.js |    52.17 |       35 |    57.14 |    57.14 |... 47,49,86,88 |
  fm.js          |    85.51 |     62.5 |      100 |    86.76 |... 82,84,88,89 |
  review.js      |    51.24 |    42.07 |    65.31 |    57.14 |... 554,556,559 |
  utils.js       |    71.18 |    57.47 |    86.11 |    76.53 |... 430,446,447 |
 src/model       |      100 |     87.5 |      100 |      100 |                |
  user.js        |      100 |     87.5 |      100 |      100 |             48 |
 src/routes      |    84.85 |     67.8 |    97.83 |    85.32 |                |
  admin.js       |      100 |      100 |      100 |      100 |                |
  auth.js        |      100 |      100 |      100 |      100 |                |
  committee.js   |    82.16 |    67.14 |    96.67 |    82.43 |... 508,510,558 |
  index.js       |      100 |      100 |      100 |      100 |                |
  professor.js   |      100 |      100 |      100 |      100 |                |
  roles.js       |      100 |      100 |      100 |      100 |                |
  routes.js      |    93.75 |    83.33 |      100 |    96.67 |             52 |
-----------------|----------|----------|----------|----------|----------------|
```

## 4.3 Overall Tests

The overall test suite consists of running both the database and application test suites.

### 4.3.1 Running test suite

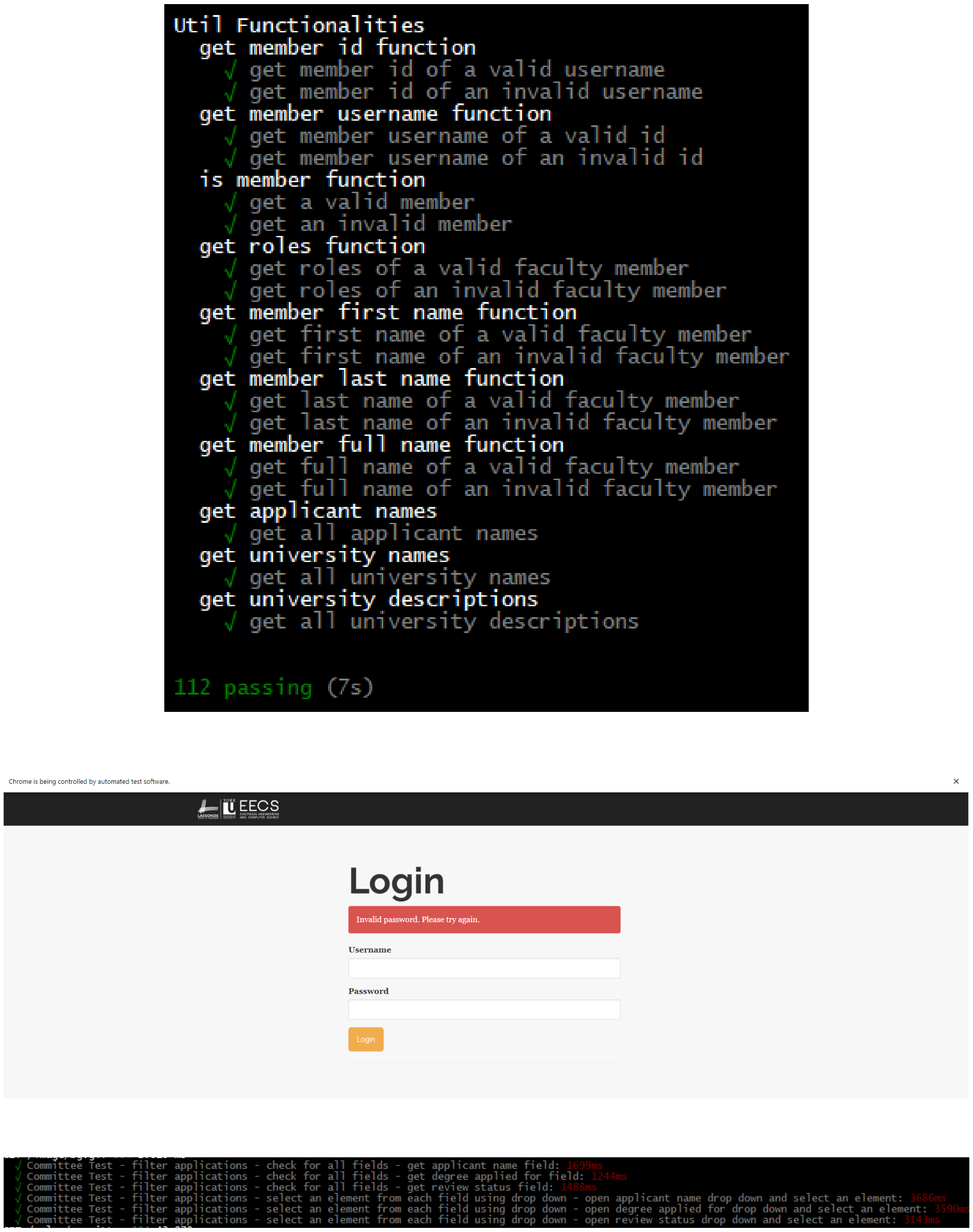
To run the overall test suite use the following command from the source directory:

```
$ npm test
```

**Pro-tip**:

- To run a single test file or a single test case , add *.only* to the *describe* or *it* hook respectively.
- To skip a single test file or a single test case , add *.skip* to the *describe* or *it* hook respectively.

Example of running the overall test suite:



```
Util Functionalities
  get member id function
    √ get member id of a valid username
    √ get member id of an invalid username
  get member username function
    √ get member username of a valid id
    √ get member username of an invalid id
  is member function
    √ get a valid member
    √ get an invalid member
  get roles function
    √ get roles of a valid faculty member
    √ get roles of an invalid faculty member
  get member first name function
    √ get first name of a valid faculty member
    √ get first name of an invalid faculty member
  get member last name function
    √ get last name of a valid faculty member
    √ get last name of an invalid faculty member
  get member full name function
    √ get full name of a valid faculty member
    √ get full name of an invalid faculty member
  get applicant names
    √ get all applicant names
  get university names
    √ get all university names
  get university descriptions
    √ get all university descriptions


112 passing (7s)
```

Chrome is being controlled by automated test software.

Login

Invalid password. Please try again.

Username

Password

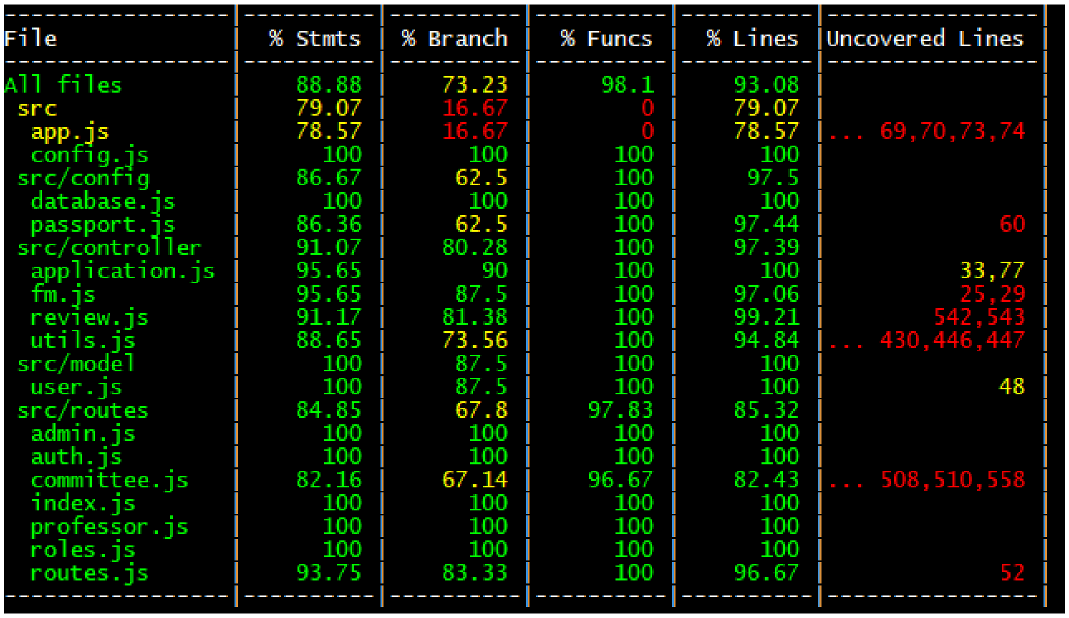
```
 √ Committee Test - filter applications - check for all fields - get applicant name field: 1699ms
 √ Committee Test - filter applications - check for all fields - get degree applied for field: 1244ms
 √ Committee Test - filter applications - check for all fields - get review status field: 1488ms
 √ Committee Test - filter applications - select an element from each field using drop down - open applicant name drop down and select an element: 3686ms
 √ Committee Test - filter applications - select an element from each field using drop down - open degree applied for drop down and select an element: 3590ms
 √ Committee Test - filter applications - select an element from each field using drop down - open review status drop down and select an element: 3143ms
```

Figure 4: Running Overall Test Suite

### 4.3.2 Test coverage

Once the test suite has completed running it will give a resulted output of the code coverage from the database test suite. The code coverage is also saved in the source directory under "coverage" folder.

Example of the code coverage output after running the overall test suite:



```
-----------------|----------|----------|----------|----------|-----------------|
File             |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines  |
-----------------|----------|----------|----------|----------|-----------------|
All files        |    88.88 |    73.23 |     98.1 |    93.08 |                 |
 src             |    79.07 |    16.67 |        0 |    79.07 |                 |
  app.js         |    78.57 |    16.67 |        0 |    78.57 |... 69,70,73,74  |
  config.js      |      100 |      100 |      100 |      100 |                 |
 src/config      |    86.67 |     62.5 |      100 |     97.5 |                 |
  database.js    |      100 |      100 |      100 |      100 |                 |
  passport.js    |    86.36 |     62.5 |      100 |    97.44 |              60 |
 src/controller  |    91.07 |    80.28 |      100 |    97.39 |                 |
  application.js |    95.65 |       90 |      100 |      100 |           33,77 |
  fm.js          |    95.65 |     87.5 |      100 |    97.06 |           25,29 |
  review.js      |    91.17 |    81.38 |      100 |    99.21 |         542,543 |
  utils.js       |    88.65 |    73.56 |      100 |    94.84 |... 430,446,447  |
 src/model       |      100 |     87.5 |      100 |      100 |                 |
  user.js        |      100 |     87.5 |      100 |      100 |              48 |
 src/routes      |    84.85 |     67.8 |    97.83 |    85.32 |                 |
  admin.js       |      100 |      100 |      100 |      100 |                 |
  auth.js        |      100 |      100 |      100 |      100 |                 |
  committee.js   |    82.16 |    67.14 |    96.67 |    82.43 |... 508,510,558  |
  index.js       |      100 |      100 |      100 |      100 |                 |
  professor.js   |      100 |      100 |      100 |      100 |                 |
  roles.js       |      100 |      100 |      100 |      100 |                 |
  routes.js      |    93.75 |    83.33 |      100 |    96.67 |              52 |
-----------------|----------|----------|----------|----------|-----------------|
```

## 5 SQL Injection

SQL injection is a code injection technique that may corrupt the underlying database of an application. It is one of the most common web hacking techniques in the world today. It works by simply placing malicious code in SQL statements, via web page inputs.

Our test suite, both database and application, covers protection against SQL injection. The application code for each of the controller in our MVC model asserts the input to be of the specified type we are hoping for. Any unmatched types will throw an exception making such asserts a crucial part of the application code.

Injecting malicious SQL statements through the controller will result in throwing such exceptions. This is because the SQL call is built by escaping the user inputs in a safe manner after the input is verified. Since, it can never reach the stage of a malicious type input within the controller, our application is safe from SQL injection.

## 6 Code Linter

The final execution after running a test suite is to lint the code to fix any syntactic or formatting issues in the code. This is done automatically as part of the post test, however, if one wants to lint the code at any point, they can simply do so by running the following command from the source directory.

```
 $ npm run lint
```

## 7 Coverage and Sufficiency

"Program testing can be used to show the presence of bugs, but never to show their absence!" - Dijkstra.

Testers live and breed trade-off's. Coverage measurements is a good tool to show *how far* you are from complete testing, but a lousy tool to be used for investigating how close you are to complete testing. The time needed for tests related tasks is infinitely larger than the time available. Hence, it is impossible to fully test a software system in a reasonable amount of time or money.

That being said for the purpose of this project we have chosen to meet a minimum requirement of 80% code coverage. With that we are confident enough to release a usable application.