Foundation Degree in Digital Innovation

Module Name	**Testing and Automation**
Year of Study	**Year 1**
Assessment	**Module Assignment**
Name of Student	……………………..…………………
Name of Module Lecturer	Danny Smith
Assessment Weighting	100%
Release Date	2 November 2018
Submission Date	16 January 2019

PLAGIARISM: please refer to the student handbook for the college policy on plagiarism and procedures for correct referencing. Material should not be copied directly from the web or a publication without a correct acknowledgement ***at the place of insertion*** of the material. It is not sufficient to include a bibliography reference only. It is also an offence to copy work from another student.

Statement (to be signed by the student before hand-in):

"I confirm that the submitted coursework is my own work and that all material attributed to others (whether published or unpublished) has been clearly identified and fully acknowledged and referred to original sources. I agree that the College has the right to submit my work to the plagiarism detection service TurnitinUK for originality checks."

Signed…………………………………………………………………

Objectives:

- Demonstrate theoretical knowledge of various test levels and the principles of software testing.
- Write a test plan and test cases for a RESTful API.
- Use test-driven development (TDD) to implement a class-based system in Javascript, fully covered by Mocha unit tests.

Where coursework is submitted late and there are no accepted extenuating circumstances it will be penalised in line with the following tariff:

- Submission within 6 working days: a 10% reduction for each working day late down to the 40% pass mark and no further.
- Submission that is late by 7 or more working days: submission refused, mark of 0.

1st Marker Signature ………………………… Mark …………… Date ……………

Mark after late submission penalty (if applicable)………………

2nd Marker Signature………………………… Mark …………… Date ……………

Assignment Description

This assignment is designed to assess your understanding of testing theory and API testing, as well as your ability to use unit tests to drive your development.

The assignment has three tasks:

1. **Testing Theory**
2. **API Test Plan**
3. **Unit Testing and TDD**

Task One - Testing Theory (15%)

1. Briefly describe each of these terms: *Unit Testing*, *Integration Testing*, *System Testing*, *Acceptance Testing*. You should demonstrate that you understand each of the test levels, how they differ, and how they contribute to better quality software.

2. Explain the term **Regression Testing**. Comment on the disadvantages of regression testing (if any) and discuss potential methods of mitigating these disadvantages.

3. In a few paragraphs, using your own words, explain the **seven testing principles**.

4. Explain **Exploratory Testing**, making reference to charters and session sheets. How does exploratory testing differ from other types of manual testing?

Task Two - API Testing (35%)

You've been given some initial requirements for an API-based restaurant reviewing platform. Because early testing is usually a good idea, your project manager has asked you to write a test plan based on the requirements *before any development work takes place*. You only need to consider functional tests (so performance testing etc is out of scope). **You are not required to write any code** for this part of the assignment.

There are a few gaps in the specification, so you may have to make some decisions about the APIs expected behaviour on your own.

You'll be marked on:

- Your general approach to designing your test cases.
- The correctness of your tests, test data and expected outcomes.
- The completeness of your test coverage, including any edge-cases.

You should write your plan as a table/spreadsheet in the following format:

Test No	Name	Preconditions	Method	URL	Request Params	Expected Status Code	Expected Result	Expected Postconditions	Notes and Assumptions

Consider ease of use. Your cases should be relatively easy for a tester to run manually by making the HTTP requests described.

Here are the engineer's requirements for the API...

Review API Requirements

- No User interface - API only
- All endpoints should return `application/json` content-type.
- The API should be RESTful, and implement CRUD actions.
- Sensible status codes should be returned for all requests.
- Should be three collections: `Restaurants` and `Reviews`.
- No authentication for now. Anyone can call the API endpoints.
- Collections should be returned as JSON arrays (an empty collection should return and empty array).

Example Restaurant object:

```
{
 id: 1,
 name: "Joe's Diner",
 city: "London",
 url_slug: "joes_diner",
}
```

Example Review object:

```
{
 id: 1,
 author: "joe_bloggs",
 published: "2015-01-01",
 title: "Great food!",
 rating: 5,
 text: "The food was fantastic. The staff were lovely too."
}
```

These routes should be implemented:

```
GET /restaurants # Returns all restaurants
POST /restaurants # Creates a new restaurant
GET /restaurants/joe_diner # Gets all the data for a restaurant, not including their
reviews.
PATCH /restaurants/joe_diner # Updates a restaurant.
DELETE /restaurants/joe_diner # Deletes a restaurant.

GET /restaurants/joe_diner/reviews # Gets all reviews for a restaurant.
POST /restaurants/joe_diner/reviews # Creates a new review.
GET /restaurants/joe_diner/reviews/1 # Gets review with ID 1 for a restaurant.
PATCH /restaurants/joe_diner/reviews/1 # Updates review with ID 1 for a restaurant.
DELETE /restaurants/joe_diner/reviews/1 # Deletes review with ID 1 for a restaurant.
```

Task Three - Unit Testing and Test-Driven Development (50%)

Your task is to implement a holiday booking system using JavaScript classes, which is fully-covered by unit tests. A specification for the interface is given below. Aim to implement the spec exactly (and if you need to deviate from it for some reason, be sure to leave a comment in the test explaining your decision and rationale).

The vast majority of marks will be awarded for:

- The design and structure of your **tests**.
- Your test **coverage** and correct use of Mocha/Chai.
- The neatness, readability and quality of your code.

A few marks are also available for:

- The design and structure of your classes and functions.
- The quality of your "boilerplate" code (README.md, package.json, folder structure etc)

Project Setup

Set up your project environment in a similar way to the **Hotel Project**. This might include:

- Creating a project README, and initializing a `package.json`.
- Installing Mocha and Chai with NPM.
- Adding an npm script to run your tests.
- Adding test and models folders.

Once you're set up, implement the following classes **using TDD**. Remember to refactor your code once you have a **green** test.

The Booking Class (25%)

An instance of this class represents a holiday booking made by an employee. Bookings can be authorized by a manager. The class should expose the following interface:

```
booking = new Booking(new Date("2018-09-01"), new Date("2018-09-05"))
booking.startDate //=> Date object for "2018-09-01"
booking.endDate //=> Date object for "2018-09-05"
booking.numberOfDays() //=> 5

booking.isAuthorized() //=> false
booking.authorizedBy() //=> null
booking.authorizedOn() //=> null

booking.authorize("Mr Boss Man")
booking.isAuthorized() //=> true
booking.authorizedBy() //=> "Mr Boss Man"
booking.authorizedOn() //=> Date object for today's Date

booking.authorize("Mr Boss Man Again", new Date("2018-07-01")) // This
function takes an optional argument representing the date it was
authorized on. [Advanced Criteria]

booking.isAuthorized() //=> true
booking.authorizedBy() //=> "Mr Boss Man Again"
booking.authorizedOn() //=> Date object for "2018-07-01" (the date
provided)
```

The Employee Class (25%)

This represents an employee of the company and should expose the following interface:

```
employee = new Employee("E123", "joe bloggs", "joe@bloggs.com", 25)
employee.payrollNo //=> "E123"
employee.name //=> "Joe Bloggs" (note the capitalization)
employee.email //=> "joe@bloggs.com"
employee.bookings //=> []
employee.holidayAllowance //=> 25
employee.bookings = 'nonsense' //=> Throw Error to prevent overwriting
(no need to write a test for this) [Advanced Criteria]

employee.daysRemaining() //=> 25
employee.daysBooked() //=> 0
employee.daysBookedAndAuthorized() //=> 0

employee.makeBooking("2020-09-01", "2020-09-05") //=> a Booking object
with these start/end dates.

employee.makeBooking("2018-01-01", "2018-01-05") //=> a Booking object
with these start/end dates.

employee.daysRemaining() //=> 15
employee.daysBooked() //=> 10
employee.daysBookedAndAuthorized() //=> 0

employee.futureBookings() //=> Array of all future bookings
employee.pastBookings() //=> Array of all past bookings

employee.futureBookings()[0].authorize("Mr Boss Man") // Authorise a
booking as before.
employee.daysBooked() //=> 10
employee.daysBookedAndAuthorized() //=> 5
employee.futureBookings(true) //=> Only include authorized bookings
employee.pastBookings(true) //=> Only include authorized bookings
```

END OF ASSIGNMENT

Submission

All submission is via GitHub Classroom.

- Part *One* must be submitted as a **PDF document** or **Markdown file**, placed in the part1 directory.
- Part Two must be submitted as a **PDF document**, placed in the part2 directory.
- All code for *Part Three* should be placed in the part3 directory.

Notes

All your deliverables should be high-quality. This means that your submissions for part one should be well-formatted and easy-to-read (remember – this is a business document) and your code should be as clean and well-organised as you can manage.

- Before starting a new part, read the whole specification and think about the problem as a **whole**. Spend some time planning your approach before diving into the solution.
- Once you've completed all three parts, go back and check your tests for Parts *Two* and *Three* - are there any more tests you should add?