Faculty of Technology – Coursework Brief 2019/20

Module name:	Advanced Programming		
Module code:	CTEC2902		
Title of the Assignment:	Assignment 2 – Group MVC Project		
This coursework item is:	Summative		
This summative coursework will be marked anonymously	No		
The learning outcomes that are assessed by this coursework are: 1. Present design documentation to correspond with a system. 2. Identify and program classes from a design of medium complexity. 3. Manage the development of a system with other team members. 4. Write syntactically correct programs using the constructs of the programming language. 5. Address issues of quality, maintainability, correctness, and robustness in the software development process. 6. Using modern frameworks and APIs to develop software.			
This coursework is:	A Group work	Assessed Individually	
You are required to build an MVC project in groups of 4 or 5. You will create a web-based system (using lab works as a guide) from the list under project briefs given below. All group members are required to interact via their GitHub Repositories, and they must push commits to their areas. This will be presented at the end.			
This coursework constitutes 30% **to the overall module mark.**			
Date Set:	Tuesday 21st January 2020 (Week 17 Lecture)		
Date & Time Due:	**Initial Group Presentation –** Week 21 Lab Session (w/c: 17/02/20) **Final Project Submission** – Thursday 26th March 2020 (Week 26), 14:00 (2pm) **Final Group Presentation (Viva)** – Week 27 Lab Session (w/c: 30/03/20)		
Your marked coursework and feedback will be available to you on: If for any reason this is not forthcoming by the due date your module leader will let you know why and when it can be expected. The Associate Professor Student Experience (studentexperience-tech@dmu.ac.uk) should be informed of any issues relating to the return of marked coursework and feedback. Note that you should normally receive feedback on your coursework by **no later than 20 University working days after the formal hand-in date,** provided that you have met the submission deadline.	TBC after all viva (Final presentations) has been completed.		
When completed you are required to submit your coursework to: 1. Group GitHub Repository			

Late submission of coursework policy: Late submissions will be processed in accordance with current University regulations which state:
*"the time period during which a student may submit a piece of work late without authorisation and have the work capped at 40% [50% at PG level] if passed is **14 calendar days**. Work submitted unauthorised more than 14 calendar days after the original submission date will receive a mark of 0%. These regulations apply to a student's first attempt at coursework. Work submitted late without authorisation which constitutes reassessment of a previously failed piece of coursework will always receive a mark of 0%."*

Academic Offences and Bad Academic Practices:
These include plagiarism, cheating, collusion, copying work and reuse of your own work, poor referencing or the passing off of somebody else's ideas as your own. If you are in any doubt about what constitutes an academic offence or bad academic practice you must check with your tutor. Further information and details of how DSU can support you, if needed, is available at:
http://www.dmu.ac.uk/dmu-students/the-student-gateway/academic-support-office/academic-offences.aspx and
http://www.dmu.ac.uk/dmu-students/the-student-gateway/academic-support-office/bad-academic-practice.aspx

Tasks to be undertaken: See the following section below

Deliverables to be submitted for assessment: See the following section below

How the work will be marked: See the following section below

Module leader/tutor name:	**Dr. Eseosa Oshodin**
Contact details:	**eseosa.oshodin@dmu.ac.uk**

The Task

For this assessment, you will build an MVC project in groups of **4 or 5**, with allocated **roles**. Using the lab work, you will create a web-based system using the **project briefs** listed below. All group members are required to interact with their **GitHub Repositories**.

On the following pages, there are three sets of **Project Briefs**. They're written in a vague way which customers use when they first approach developers. The first thing is to retrieve the requirements from these briefs and add them to your team's repository, which will be created once the groups are finalised.

For GitHub access, you are expected to have created an account on www.github.com.

The Roles

There are two roles you can take for the project. You will be marked individually on your submissions (i.e. the code/documentation you push into your group's GitHub repository).

- **Systems Analyst / Coordinator role**

  This role is a mixture of Information Systems Analyst and Project Manager. They will be responsible both for analysing, modelling, designing the system and assign new issues to a developer. They will be in charge of keeping the project information in GitHub neat and tidy. Documentation such as requirements, design (i.e. UI diagram, use case diagram, UML diagram etc.) and test plan will need to match the system. These are required by the developers who need to keep their work in line with each planned content presented in the documentation. While this role won't require much code to be written, the code by the developer is expected to be analysed and understood by those taking up the role (i.e. Systems analyst/coordinator). The systems analyst/coordinator will need to link development to the plan and other documentation.

- **Developer role**

  Developers should take ownership of resolving one or more issues (requirements or ideas which can be considered as tasks to be done) for the project, in order to make their contribution to the project clear. Developers should also help their fellow developers improve some of their code by pushing some modifications back into the repository; the developer can assign issues from incomplete tasks to fellow developers. Their code should interface elegantly with the parts of the system owned by other developers. Their classes should follow designs (created by **Systems Analyst / Coordinators)** and adhere to OO principles (e.g. encapsulation, inheritance and perhaps polymorphism). Their code should be clean and easy to maintain, robust, handle errors properly and be easy to deploy. Various tests should be carried out on all written methods or functions in the code.

Key Dates

1. **GitHub Group Repository Setup - Week 19 Lab Sessions**

   During this week, we will finalise the groups on GitHub. You all will be grouped (4 or 5) in a certain repository created. Access will be provided to each member of the repository group by your tutor.

2. **Initial Group Presentation (20%) - Week 21 Lab Sessions**

   Each member of the team will have to present their Wiki Page in their team's repository on GitHub. Individual member's page should explain what their role is in the team and which bit of the system they are responsible for. Also, each team member will have to prove that they can push code/content to their repository (even if it's just a small project). From the pushed content to the repository, the system analysts/coordinators should also describe the initial documentation for the proposed system and the developers should demonstrate the prototype system to reveal the rough view (Based on initial documentation) of the main page for the system. Please take note that this will not be a PowerPoint Presentation or similar. This will give you an opportunity to present your group's GitHub, as well as your own.

3. **Final Submission (60%) - Thursday 26th March 2020, 14:00 (2pm) (Week 26)**

   This is the **final** date and time for any work you submit (by pushing commits into GitHub) to be counted. GitHub is a public repository, so you can't be *stopped* from submitting commits after that point. But anything after that date/time won't be marked. The System analysts/coordinators should have the final version of all documentation uploaded to the repository. The developers should have the final version of the code committed and pushed to the repository.

   Also note – we'll take a look at everything pushed after that date, to see if you've been pushing last-minute fixes to your repositories which could affect your marks for management (see mark scheme).

4. **Final Group Presentation (20%) - Week 27 Lab Sessions**

   The final presentation will be half-an-hour long, which means each team member will get at least five minutes to describe their contribution to the project. Presentations will take place in the final remaining week lab sessions. All member of the team must attend group presentations.

Project briefs

Games review website

As your client, I want a website that lets users review games. I'm a big fan of games myself. I'd like to add a description of the game myself to get the ball rolling, and then let other users add their reviews about that game. So, we would end up with each game having a page of some sort that list all the reviews. I'd also like pages about developers, which would link to all the games they had made. I'd like users to be able to add comments about the developers, too.
I'd like users to be able to reply to each other's comments, too. But I'd also like to be able to moderate, delete user comments and block users who post too many out-of-order or offensive comments.
Also, I'd like each review to have a "**score** out of ten (10)" (e.g. Score: 7.5 out of 10), so that there's a league table of games that you can use to see either the most or least popular game.
I'd also like people to be able to tag games by their genre (action, adventure or miscellaneous etc), and then use those tags to be able to filter the best or worst tables with the games that are tagged with the specified tag (For example, user could list the worst "action" games in a table).
Also, I'd like users to be able to watch a trailer of the game.

Examples of some game review website:

https://www.metacritic.com
https://www.gameinformer.com/
https://www.destructoid.com/

Book Recommending Website

I am an investor who likes reading books, and I wish to invest in a website that provides recommended books to users. The users will need to have clicked on at least a link to an available book before they can receive the recommendation of similar books. Each available book's link should direct the users to external source where they could access or purchase the books. Books to be recommended should be filtered based on previously accessed books that have the same author or genre (For example, if a user has accessed a book with a particular author or genre, then books of the same author or genre should be recommended to the user). There should be a button available to the users to view recommended books that are filtered based on the similar author or genre to books already accessed or purchased.

As your client, I will want to be able to add new books and their links to where they can be accessed or purchased. I would like users to be able to vote or rate and write short comments or review books. Users will also be able to tag book by their genre (action, thriller, fiction, romance, horror fiction, adventure etc).

Examples:
https://www.goodreads.com/
https://www.whatshouldireadnext.com/
https://www.amazon.com/

Article / story website

I'm a traditional publisher who wants to go digital and start working with user-generated stories, so I'd like a website for people to write stories or articles. I'd like you to suggest the topic, though – it could be themed around a hobby or interest, politics, or be fan-fiction of some sort. Just as long as you think it's going to be popular, and it's safe for work (I don't want to get into trouble or damage my reputation).

I'd like each user to have a public profile that lists all their stories. I'd also like users to be able to comment on each other stories, too. If possible, I'd like them to put longer comments at the foot of each story, where they can discuss the story with other users. But I'd also like them to add short notes to the side of each story, too (like maybe if they think there's a typo or a mistake or something?)

Each user should have a profile page that gathers all their comments and stories together.

The homepage should list all the stories by the date they were last modified, with the most recent at the top. It would be nice if people can "up and down-vote" stories so we can list them by popularity, too.

I'd also like stories to be tagged with keywords, so that stories with the same keyword can be grouped together. And I'd like to be able to search and browse through all the stories.

I'd like people to be able to go back and edit their stories, and for the system to keep track of all the different versions. It would be nice for people to be able to preview a story before they publish it, too.

Example:

https://www.medium.com/
https://www.fanfiction.net/
https://www.fictionpad.com

Mark Scheme for Developer Role (60%)

	Clear Fail	Marginal Fail	Bare Pass	Clear Pass	Very Good	Excellent	Outstanding
	0% - 2%	3%	4%	5%	6%	7%	10%
Ownership of (an) issue(s) (10%)	*Barely pushed any code to the project repository.*	*Pushed small code changes to various parts of the repository, not linked to specific issues.*	*Pushed some code to one part, but barely resolved the related issue*	*Has resolved only one issue in the repository.*	*Has taken one issue, analysed it, derived further issues from it and resolved them all*	*Has broken one issue down and also designed new functions related to the initial issue.*	*Has developed an entirely new sub-system inspired by the original issue.*
Team working and code integration (10%)	*No evidence that they have worked on anyone else's code but their own.*	*Made one or two minor changes to another team member's code, but not improved it.*	*Made one significant improvement to another team member's code*	*Made several improvements to other code, and documented the changes in the issues list*	*Have worked closely with another team member to add tests to both sets of code and make the code robust.*	*Have worked closely with at least one other team member and developed classes with an understandable interface to the rest of the system. In other words, the developer's Model code makes logical sense, their Controller code is well integrated with the rest of the application, and their Views fit into the overall site design.*	
Understanding of OO principles (10%)	*The code is not broken down into meaningful, sensible classes*	*There are some classes, with names that make some sense, but it is not clear what those classes are supposed to do*	*The code is organised into classes with meaningful properties*	*The code is organised into classes with properly encapsulated properties, meaningful methods, and sensible method parameters*	*The code is organised into classes that make proper use of the basic OO structures such as collection classes, inheritance and polymorphism.*	*Code dependencies are managed using interfaces. Controllers are loosely coupled to their dependencies using Dependency Injection.* .	
Code cleanliness / maintainability (10%)	*Most of the code they submitted breaks the build. No comments provided.*	*Some of the code that they submitted breaks the build. Few comments provided but they are clear.*	*Their code builds, but looks messy. Variables are badly named. It is hard to work out what the code does.*	*It is possible to read the code, but methods are long and they can be confusing / hard to follow.*	*Variables are well named. Methods are refactored into more logical parts so that the flow of the code can be easily tracked.*	*It is clear what classes do, and there is a well-established, logical flow to the code, though comments are required to fully understand the code.*	*It is consistently easy to work out what their code does, without having to read comments.*
Code robustness (10%)	*Several parts of the code they wrote throws an error when the app runs*	*One or two pieces of their code cause errors*	*They've created code that doesn't cause any errors under most circumstances, but there's nothing to handle errors if they did occur.*	*The code runs properly, and the most common kinds of errors are anticipated and handled.*	*The code handles the most commonly-anticipated errors and logs them.* *Key variables are applied to the system configuration.*	*There is proper defensive coding in the interfaces between classes, as well as error handling and logging. The issue they own handles all exceptions gracefully.*	
Code testing (10%)	*There are no tests for the developer's code*	*There are a few tests but coverage is poor.*	*The main functions of the code have tests but there are still alot of functions that don't have their test.*	*All the code that is not dependent upon databases etc have coverage along its "happy paths".*	*There are tests for all the code, including alternate and exception flows.*	*There are tests for everything, and those parts of the system with data dependencies have had those dependencies broken with mock objects.*	

Mark Scheme for Systems Analyst / Coordinator role (60%)

	Clear Fail	Marginal Fail	Bare Pass	Clear Pass	Very Good	Excellent	Outstanding
	0% - 3%	**4.5%**	**6%**	**7.5%**	**9%**	**10.5%**	**15%**
Management of issues (15%)	*The initial set of issues wasn't updated at all throughout the project.*	*Some updates to a few of the issues occurred, but not match with the code. Pushes are not traceable back to issues. Issues weren't assigned to the team properly.*	*The issues list was changed, but in chunks rather than constantly. Most of the updates to issues occurred in a rush at the end.*	*The issue list matches the state of the system at the end of the project. Issues can generally be tied back to the people that dealt with them.*	*The issue list was generally maintained throughout the life of the project and shows a reasonable amount of activity from most of the team members throughout.*	*The issue list was updated throughout. Issues that arose during the project (e.g. bugs, new ideas etc) are documented and can be mapped onto changes in the code.*	
Quality of documentation (15%)	*No documentation about the system was produced by the team.*	*Minimal documentation was produced by the team but it doesn't match what the system actually does.*	*The core parts of the code have been documented in a way that matches what the system does.*	*The code is documented. There is some high-level documentation in the Wiki.*	*All the code is well documented and how to use the core functions is described in the Wiki.*	*The code is thoroughly documented. The documentation is up to date. The Wiki explains what the system does. The issues can be tied to the documentation in key places.*	
	0% - 2%	**3%**	**4%**	**5%**	**6%**	**7%**	**10%**
System vision and context (10%)	*There is no documented vision for the system*	*An attempt has been made to document the system vision, but it is hard to understand and it doesn't really match the system.*	*A vision for the system has been created that matches some of the system, but does not really provide context.*	*The vision for the system has been defined and documented in a way that is understandable, and which provides basic context for it.*	*The vision for the system is short and clear. Some effort has been made to provide the core context of the system.*	*The vision for the system is clear, and there is some documentation of its context, inputs and outputs, and boundaries, though parts of this are missing.*	
System modelling (10%)	*There are no models at all for the system.*	*One or two core classes have been modelled, but the models do not make logical sense.*	*One or two obvious classes have been modelled in a way that has some logical sense to it.*	*All the basic classes for the system's business logic have been modelled in a way that makes sense.*	*Business logic classes, their states and interactions have been modelled.*	*All classes (including those in the GUI), their states at different points in time, and the interactions between them have been modelled.*	*There is a concise system vision that accurately sums up the purpose of the system. The system's context, boundaries and information inputs / outputs are documented.*
System user journey (10%)	*There is no evidence that the user's path through the system has been thought about.*	*The bare minimum path through the system for the most obvious function has been documented.*	*Most of the main functions have a planned path.*	*The basic "happy path" through the system for each function has been documented.*	*All happy and alternate flows have been planned and there are wireframe interface designs for some of the key functions*	*All flows (including the exception flows) have been documented and wireframed.*	

Mark Scheme for Initial Group Presentation (20%)

	Clear Fail	Marginal Fail	Bare Pass	Clear Pass	Very Good	Excellent	Outstanding
	0% - 4%	6%	8%	10%	12%	20%	
Initial Presentation (20%)	*Did turn up for the presentation*	*Present but ill-prepared with no GitHub account*	*GitHub account is presented but no other attempt has been made in pushing code files or uploading documents to the wiki.*	*Some contribution, with either a push to the repository or a basic wiki explaining the project. No demonstration of prototype.*	*There was an obvious structure to the wikis explaining the project's objective with explanation of individual roles, attempted to push to the repository.*	*It was obvious that the team had rehearsed it and helped each other work out what to show. Fully comprehensive Wiki page with project objects and updates, individual wiki pages highlighting roles and task. Multiple pushes individuals to populate group repository with initial documentation and project files. Prototype was well presented. Some issues (requirements) may also have been highlighted.*	

Mark Scheme for Final Group Presentation (20%)

	Clear Fail	Marginal Fail	Bare Pass	Clear Pass	Very Good	Excellent	Outstanding
	0% - 4%	6%	8%	10%	12%	14%	20%
Quality of presentation (20%)	*Did turn up for the presentation*	*A final presentation was given but it was ill-prepared and incoherent.*	*A basic presentation was given in which member roughly described their contribution.*	*Some contribution, but the explanation to what work they had done seemed understandable to an extent.*	*There was an obvious structure to the presentation. It was clear it had been thought through and prepared.*	*The presentation was well structured and coherent (I.e. it was easy to relate each piece to each other piece). It was obvious that the team had rehearsed it and helped each other work out how to deliver it and stick to time.*	*There was an outstanding presentation, including a system demonstration with no errors. All team members had a role in the demo and they were enthusiastic. The presentation had a well - defined structure, with an introduction and conclusion.*