|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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. Identify and program classes from a design of medium complexity. 2. Create classes demonstrating appropriate OO principles. 3. Write syntactically correct programs using the constructs of the programming language. 4. Address issues of quality, maintainability, correctness, and robustness in the software development process. 5. Using modern frameworks and APIs to develop software. | | | | | | | | |
| **This coursework is:** | | | Group | | | | Assessed Individually | |
| You are required to build an MVC project in groups of 4 or 5. You will create, using the lab work, a web-based system based on the project briefs listed. All group members are required to interact with their GitHub Repositories and must push something to their areas. This will be presented at the end. | | | | | | | | |
| **This coursework constitutes** 30% **to the overall module mark.** | | | | | | | | |
| **Date Set:** | Tuesday 22th January 2019 (Week 17 Lecture) | | | | | | | |
| **Date & Time Due:** | **Initial Group Presentation –** Week 21 Lab Session (wc: 18/02/19)  **Final Project Submission** –  Thursday 28th March 2019 (Week 26), 14:00 (2pm)   **Final Group Presentation** –  Week 27 Lab Session (wc: 01/04/19) | | | | | | | |
| **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](mailto: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. | | | | |  | | | |
| **When completed you are required to submit your coursework to:**   1. Group GitHub Repository | | | | | | | | |

**Faculty of Technology – Coursework Brief 2018/19**

|  |  |
| --- | --- |
| **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 below** | |
| **Deliverables to be submitted for assessment: See below** | |
| **How the work will be marked: See below** | |
| **Module leader/tutor name:** | **Salim Hasshu** |
| **Contact details:** | **salim.hasshu@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 **projects briefs** listed. All group members are required to interact with their **GitHub** **Repositories**.

On the following pages are three sets of **Projects Briefs**. They’re written in a vague way which customers use when they first approach developers. The first thing is to gather 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](http://www.github.com).

**The Roles**

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

* **Developer role**

Developers should take ownership of one or more issues 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. Their code should interface elegantly with the parts of the system owned by other developers. Their classes should follow designs (of **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 well and be easy to deploy.

* **Systems Analyst / Coordinator role**

This role is a mixture of Information Systems Analyst and Project Manager. They will be responsible both for analysing, modelling and designing the system. They will be in charge of keeping the project information in GitHub neat and tidy. The documentation will need to match the system. This is dependent on the developers keeping their work in line with the documentation. While they won’t need to write much code, they will need to understand it, and link development of code to the plan and documentation.

# Key Dates

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

During this week we will finalise the groups on GitHub and create and group repositories. Access will be provided to your group to this repository by your tutor.

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

Each member of the team will also have to present their Wiki Page in their team’s repository on GitHub, which will explain what their role in the team is and which bit of the system they are responsible for. Each team member will also have to prove that they can push code / content to their repository (even if it’s just a small project). This is not a PowerPoint Presentation or similar. This is an opportunity to present your group’s GitHub, as well as your own.

1. **Final Submission (60%) - Thursday 28th March 2019, 14:00 (2pm) (Week 26)**

This is the **final** date and time any work you submit (by pushing it into GitHub) will be counted. GitHub is a public repository, so we can’t *stop* you submitting code after that point, but anything after that date / time won’t be marked.

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).

1. **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 weeks lab sessions. All member of the team must attend group presentations.

Project briefs

# Film review website

I want a website that lets users review films. I’m a big fan of films myself. I’d like to add a description of the film myself to get the ball rolling, and then let other users add their reviews about that film. So we would end up with each film having a page of some sort that listed all the reviews. I’d also like pages about actors and directors, which would link to all the films they had appeared in or made. I’d like users to be able to add comments about the actors and directors, 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 and delete user comments, and block users who make too many out of order comments, as well.

I’d like each review to have a “marks out of ten”, too, so that there’s a league table of films that you can use to see the most / least popular.

I’d also like people to be able to tag films by their genre (action, thriller, comedy etc), and then use those tags to be able to filter the best / worst tables (so you could list “the worst comedies”).

Plus I’d like people to be able to add film news stories, gossip etc.

**Example:**

<http://www.rottentomatoes.com/>

<http://www.imdb.com/>

<http://www.zergnet.com/>

# Link sharing / content aggregation site

I’m an investor with a pile of money who wants to have a go at the internet game. So I’d like a site that re-uses as much existing content (that’s already out there on the Internet) as possible. Obviously, I want my site to be popular, so I need it to have a popular theme that’s going to attract people: e.g. sport, music, gaming, baking, antique dealing / property restoration / interior design etc etc – I’d like you to suggest three or four topics that work well together (say “music, clubbing and fashion” or “sports, gaming and exercise”). I don’t want to get arrested or get in trouble, though, so they have to be “safe-for-work” topics.

I’d like my staff to be able to find some information that’s currently out there (say a news story on a music website) and then create a link to that content. Each link will have a little snippet of information explaining what it is, and some tags that relate it to the relevant topic. The links will be published to a “topic page” that gathers all the links for that topic into one place. Then I’d like to be able to track all the clicks that people make on that link. I’d also like to be able to place adverts (multiple adverts) on each topic page. I also want to know which ads get clicked on and which don’t.

I’d like users to be able to up and down vote the links, and write short, Tweet length comments about them.

I’d like to be able to search the whole site, and filter the results by topics and tags.

I’d also like to be able to embed Tweets about the topic from relevant famous people on each topic page, too.

**Example:**

<http://www.metafilter.com/>

<http://www.digg.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 in 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>

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **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 one issue in the repository but just that one* | *Has taken one issue, analysed it, derived further issues from it and resolved them all* | *Has broken one issue down, but 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 makes 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 on the build machine* | *Some of the code that they submitted breaks the build* | *Their code builds, but is 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 can be confusing / hard to follow* | *Variables are well named. Methods are refactored into more logical parts so 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 low* | *The main functions of the code have tests but there are plenty of functions that don’t have them* | *All the code that is not dependent upon databases etc has 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 Developer Role (60%)**

**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 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 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 and their states and interactions have been modelled.* | *All classes (including those in the GUI), and 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 present but no other attempt has been made.* | *Some contribution, with either a push to the repository or a basic wiki explaining the project* | *There was an obvious structure to the wikis explaining the project 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. 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* | *Gave a final presentation but it was ill-prepared and incoherent.* | *A basic presentation was given in which member roughly described their contribution.* | *Some contribution, but explained what they had done some-what understandably.* | *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 a outstanding presentation, including a system demonstration with no errors. All team members had a role in the demo and were enthusiastic. The presentation had a well - defined structure, with an introduction and conclusion.* |