Assessment Brief

| Module Leader:  Soumya Basu | | Level:  6 |
| --- | --- | --- |
| Module Name:  Software Architecture and Design | | Module Code:  55-608213 |
| Assignment Title:  Portfolio of Software Development Artefacts | | |
| Individual / Group  Group | Weighting:  100% | Magnitude: *wordcount/length of…* |
| Submission date/time:  01/12/2022 | Blackboard submission Y  Turnitin submission N | Format: Software Design artefacts, source code, word/pdf document. |
| Planned feedback date:  22/12/2022 | Mode of feedback:  Verbal and written | In-module retrieval available: No |
|  | | |
| **Module Learning Outcomes**   * LO1: Select and apply contemporary design techniques, architectures and frameworks in the implementation of software applications. * LO2: Implement software using a variety of frameworks. * LO3: Examine comparatively systems architectures and demonstrate appropriate uses for them. * LO4: Use appropriate design strategies effectively to guide the production of solutions in a complex problem domain. | | |

# Introduction

In this assessment you will design and build a Web application that makes use of modern application development frameworks and development techniques. You will demonstrate your understanding of the design techniques, architectures and technical opportunities provided by such frameworks.

You will be working on teams of 3 to 5. Details about confirming your team composition will be circulated separately on the Blackboard site and during classes.

This application should be built to meet the needs of the clients in the case study presented in a separate document. The case study has been selected as it offers several challenges with regards to design, usability, security, and privacy. Your design should reflect these issues; we expect designs and implementations that deal with each of these key challenges. This assessment will require a coordinated approach and as such, we expect you to plan your time appropriately using an appropriate project management model.

Your design should consider the case study holistically. For implementation, you can select two or three major functionalities within your group. The selected functionalities should include at least two types of users.

The assessment is divided into three main groups:

Group #1 is the production of your solution's overall architecture and design upon which an implementation will be based.

Group #2 is the implementation of your solution using the technology and frameworks discussed throughout the module.

Group #3 is the production of a group project report and an individual reflection reports. Group project report is used for the assessment walkthrough.

You and your team will work together following the roadmap presented in this document. It includes a general guidance on the expected progress of your project as well as opportunities for formative feedback throughout the semester.

Your mark is composed by a group-based assessment, an individual reflection statement and a peer-assessment element. For the group assessment each member of your group will get the same marks. If you wish you may want to submit a peer review form in case, you feel some member’s contribution is not up to the mark. Based on tutor observation and peer review group assessment marks may be reduced for a member.

This document details each of the elements mentioned above.

# Design artefacts

Using the case study materials provided, you need to arrive at the requirements and create the overall solution and designs upon which an implementation will be based.

The requirements should include possible scenarios for different types of users. It should also include a formal requirement documentation (user story or use case or requirement specification) adhering to the SMART(T) criteria.

The solution should include system context (to show the system boundary and its interaction with external entities) and architecture overview (including the major subsystems of the system and how they communicate with each other).

The designs should include static and dynamic models.

Design artefacts should include

* User scenarios for different types of users and SMART(T) requirements linked with the scenarios. Requirements can be documented using user story or use cases or requirement specifications. Requirements should also consider both functional and non-functional requirements.
* A System Context diagram indicating the boundary of this solution and possible interaction with external entities.
* An architecture overview demonstrating the solution architecture.
* A static model (e.g., a class diagram, component relationship diagram, entity relationship diagram or other structural diagram) that shows the major units in your system and the relationships between them.
* A dynamic model (e.g., Interaction diagrams like sequence or collaboration, activity diagrams, state chart or other behavioural diagrams) that explains the interactions between units in your system to handle the core areas of functionality. This must include a dynamic model of a specific complex piece of functionality, e.g., how the system collects data from different sources at different rates.
* A textual description/annotation (as appropriate) about the primary responsibilities of different units in the system and the interaction between the units. Descriptions/annotations can relate to any of the above models.

The models may be submitted using a suitable modelling environment available on a standard university desktop (e.g., StarUML, Visual Paradigm or diagrams.net) or as good quality image files (i.e., ones that you can easily read all details) - Suggestion, use vectorial images when exporting from the chosen tool.

You must also upload all designs to the module Blackboard site by the deadline. Design artefacts should be part of the group project report and should be submitted in a doc (word or pdf) format or combined in a single zip file.

.

# Development

Your team must build a prototype of a system that demonstrates your design. You are required to implement your designs for the case study and test your implementation. In your group you can select two or three major functionalities encompassing at least two types of users for the prototype development.

We recommend that your team adopts a feature-driven development approach, tackling one problem at a time and updating the initial design documents to reflect any changes identified during the implementation.

You must prioritise what functionalities will be explored by the team from the features described below.

We require that your group use JavaScript to demonstrate the following aspects of the application:

1. An application that runs on the node.js server.
2. A data access layer that wraps a suitable database, presenting records and tuples as JavaScript objects in the form of JSON documents. This must implement a full set of CRUD operations.
3. A back-end layer that accepts HTTP REST requests (at least GET and POST), validates those requests and routes them to the appropriate URL endpoints.
4. A front-end layer that implements views using a framework such as Vue, React or Angular.

Some specific aspects about your development approach are discussed in the sequence.

You are required to implement the system following the designs you created. You should only deviate from the designs if you can present a very good, critical reason for making the change. You will need to explain and justify any changes you make from the designs in your group project report.

While several designs for the case study may include mobile capabilities, you may choose to develop your applications for a desktop using a suitably restricted window size to simulate a mobile screen if you find this more convenient. You can mock some services if needed. You can also have prepopulated data, if required to implement any functionality.

We are looking for quality software engineering (good coding practice, testing, project management or any other applicable practices) over quantity. Please check with the module team to clarify the scope of your implementation.

While the primary focus of the implementation aspects is the end-product and the evaluation, we will be looking for quality within your engineering practice such as code commenting, documentation, careful testing, version control, static code review, containerised deployment etc.

In addition to implementing the design and producing a working system for the case study, you need to formally test it, and during the walkthrough you will be asked to demonstrate and discuss the testing that you have done. Your test cases should align with those described in your formal requirements specification.

# Documentation and Reports

As part of the assessment your group will submit a single report with the documentation of your project and each person will submit an individual reflection report.

## 4.1. Group Project Report

Your team will need to produce a report on your design and development approach. It should include the produced artefacts and important design/development decisions with justification, mainly when there are deviations from design in your implementation. This report is used for your assignment walkthrough.

## 4.2. Individual Reflection Report

As an individual you need to produce a reflection report. It should include the role you played, how you have worked within a group and your lessons learnt. You should include what you would have done differently as an individual given a chance to execute it again. The reflection should be up to 2 sides of A4 remember, quality over quantity is being assessed.

# Submission and Deliverables

The deadline for this task is **3 pm on Thursday 01st December 2022**. There will be a group submission point and an individual submission point to segregate group submission and individual submission. One of the group members can submit in the group submission point and the same will be reflected for all members.

ATTENTION TO THE FOLLOWING POINTS:

You **must** submit the following to the group submission point in Blackboard.

* + Do not include the contents of the node modules folder.
* You **must** include a readme file with
  + a link to your repository and video.
  + instructions for building and running your application.
  + Your readme must be plain text or simply formatted using markdown1.
* You **must** include instructions or a script for rebuilding and repopulating the database.
* Include any unit, integration or UI tests that you developed together with your test results.
* You must prepare a running version of your solution in a University PC or in the Azure Labs VM allocated for you in this module.
* Group project report including design artifacts
  + Scenarios and requirements
  + System Context and Architecture Overview
  + Static and Dynamic Model
  + Important design decisions with justification
  + Any design changes that you may have done during development in your group with justification.

You **must** submit the following to the individual submission point in Blackboard:

* Individual reflection

All the implementation and documentation created for the project must be uploaded as a single ZIP file to the module Blackboard site by the specified deadline. This deadline is shortly before the walkthroughs but must contain all materials you wish to present.

# Roadmap/Milestones

The following table outlines a suggestion of weekly milestones that need to be adhered to in order to complete the assignment on time.

|  |  |
| --- | --- |
| **Date** | **Milestone** |
| Week 1 (30th September 2022) | Groups are formed. Clarification obtained about the case study. |
| Week 2(7th October 2022) | Requirements specification (User scenarios and SMART(T) requirements |
| Week 3(14th October 2022) | Overall solution architecture (System Context and Architecture Overview) |
| Week 4( 21st October 2022) | Static models |
| Week 5(128th  October 2022) | Dynamic models |
| Week 6(4th November 2022) | Implementation plan and implementation platform readiness |
| Week 7(11th November 2022) | Implementation checkpoint 1 with emphasise on security aspects |
| Week 8(18th November 2022) | Implementation checkpoint 2 with emphasise on testing |
| Week 9(25th November 2022) | Implementation checkpoint 3 with emphasise on software engineering practices |
| Week 10(1st December 2022) | Final Submission |

# Marking

Your group will present your solution (design and implementation) during a walkthrough. We will attempt to have a face-to-face walkthrough. If the situation does not permit, the walkthrough will be virtual either using Zoom or Blackboard Collaborate. You will be notified of the tool to use in due course. All team members must attend the walkthrough. Individuals who do not attend their group walkthrough without formal notification and proper justification will be given a mark of zero.

The walkthrough will last approximately 40 minutes. This will consist of 30 minutes for you to present your designs and 10 minutes for discussions and feedback. Time Slots for the walkthroughs will be made available via the module Blackboard site for your group to choose from.

It is up to you how you divide the work. If there are issues with the level of contribution within the team, these need to be raised in advance of the submission deadline with the module delivery team. By default, marks will be awarded equally for all students in the group; however, the lack of contribution by individual members (assessed via a peer review exercise and tutor observation) may result in some group members being awarded proportionally lower marks.

**Walkthrough script for presentation**

This is an indicative script to help you in your presentation. For demonstration you can select an example scenario. Make sure the database is prepopulated with required data to demonstrate your scenario.

1. WHAT you have done.
   1. Explain about different type of users you have considered
   2. Talk about different scenarios for all kind of users identified
   3. Show the formal requirement specifications created by your group.
2. Demonstrate your design
   1. Explain System Context
   2. Explain Architecture Overview
   3. Explain Static and Dynamic model
   4. What patterns have you used
3. Demo of the application running (Be clear about the examples you have created, and you should not show any code at this point).
   1. Clearly explain the user interface.
   2. Demonstrate the main functionalities.
   3. Demonstrate input verification.
4. Technical implementation
   1. Design implementation
      1. How does your code implement your design?
      2. How your code organisation relates to architecture overview?
      3. Is there any change in design from design phase to implementation phase?
   2. Data Layer
      1. Show the data structure you use (database tables and model associations).
      2. How is the database wrapped by the Data Access Layer?
      3. How models connect with DB methods? CRUD operations?
      4. How controllers connect to models and how models are validated?
      5. Can you return both single objects and sets of objects?
   3. Business Layer - Talk us through the API you implemented.
      1. Are the URLs REST/RESTful?
      2. How the endpoints respond with JSON?
      3. How do you use call-backs, events and promises?
      4. What security features are incorporated into API?
   4. Presentation Layer
      1. How UI model fetch data asynchronously?
      2. How are inputs validated?
      3. How are errors handled?
      4. How are sessions used?
5. Security and user management
   1. Demonstrate your user management interface.
   2. How are users authenticated throughout the application?
   3. How roles and permissions have been implemented?
6. Software testing
   1. What is your approach for testing?
   2. What are you testing? How is testing being performed?
   3. What do the results mean?
   4. Why are you testing those things?
7. Software engineering quality
   1. How did you manage versions?
   2. How did you manage the release?
   3. Do you want to highlight any good engineering practice that you have followed?

# Final Thoughts

The information in the case study document is typical of what you might expect to get from a client initially when asking for a solution (i.e., incomplete or asking too much, possibly ambiguous, and at times vague). This is a real requirement from a university.

You are encouraged to talk to the module delivery team during face-to-face sessions or drop-in sessions to clarify any queries you might have as you design (and then implement) a working system.

Remember, your clients know the functionality required, they will rarely be computer savvy but thanks to search engines they may be aware of a lot of buzzwords even if not applicable in their context. Occasionally there will be developers in the client team, but remember they have a 'seen it all, done it all' feeling. It is your job as practicing software engineers/computer scientists, to interpret the English language musings into the solution, formal designs and ultimately a software package. If the client could do this, they wouldn’t need us.

# Assessment Criteria

|  | **FAIL**  (insufficient) | | | | **THIRD**  (sufficient) | | | **LOWER SECOND**  (good) | | | | | **UPPER SECOND**  (very good) | | | | | **FIRST**  (excellent) | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Zero | Low Fail | Mid  Fail | Marginal Fail | Low 3rd | Mid3rd | High 3rd | Low  2.2 | Mid 2.2 | | | High  2.2 | Low 2.1 | Mid  2.1 | | | High 2.1 | Low  1st | Mid  1st | High 1st | Exceptional 1st | | | Perfect 1st |
| Criteria and weighting | **<19** | | **20-39** | | **40-49** | | | **50-59** | | | | | **60–69** | | | | | **70-84** | | **85+** | | | | |
| **Design elements** |  | |  | |  | | |  | | | | |  | | | | |  | |  | | | | |
| **C1. Requirements specification**  Scenarios need to be identified for different kind of users. Formal requirements (functional and non-functional) following a SMART(T) approach  including, for example, a set of requirement specifications, user stories or use cases.  10% | No scenarios have been identified | | Scenarios have been identified but lack clarity and are not linked to the requirements. | | Scenarios are realistic and provide basic detail, and the requirements are specific at minimum. Requirements cover at least one type of user. | | | Scenarios have rich details and requirements have at least 2 SMART(T) criteria. Requirements cover at least two types of users. | | | | | Scenarios are linked well with the requirements and the requirements are mostly SMART(T). Requirements cover at least two types of users. | | | | | Scenarios consider both functional and non-functional requirements and the requirements are SMART(T). Requirements cover more than two types of users. | | Scenarios are supported by evidence from research  and analysis. Requirements are SMART(T) and cover all possible types of users. | | | | |
| ✔ |  | | | | | | | | | | | | | | | | | | | | | | | |
| **C2. Overall solution architecture**  An overall solution addressing the requirement in general, and the scenarios/requirements mentioned above in particular  5% | No system context or architecture overview provided. | | Very basic system context or architecture overview provided. | | System context shows the system boundary well.  Architecture overview shows layers/tiers of the system. | | | System context shows the system boundary well, including some of the functions provided by the system.  Architecture overview shows layers/tiers of the system and show some functionality residing on different layers. | | | | | System context shows the system boundary well, including most of the functions provided by the system.  Architecture overview shows layers/tiers of the system and which functionality resides on which layer. | | | | | System context shows the system boundary well, includes most of the functions provided by the system, and shows most external interfaces.  Architecture overview realises the scenarios/requirements reasonably well with few exceptions. | | System context shows the system boundary well, includes all major functions provided by the system, shows all external interfaces with communication protocols.  Architecture overview realises the scenarios/requirements nicely. | | | | |
| ✔ |  | | | | | | | | | | | | | | | | | | | | | | | |
| **C3. Static model**  A static model that shows the major units in your system and the relationships between them.  10% | Some basic units are identified but no details provided. | | Some basic units are identified without detail, and some relationships are shown. | | Basic units are identified with some detail, and some relationships are shown. | | | All units are identified with detail and most of the relationships are shown. | | | | | All units are identified with detail, and all relationships are shown with right cardinality. | | | | | Well-formed model with all major details included (with only minor omissions or errors) and offering an extensible design. | | Excellent, extensible design with no errors evident in syntax or design. Make use of several best practices. | | | | |
| ✔ |  | | | | | | | | | | | | | | | | | | | | | | | |
| **C4. Dynamic model**  A dynamic model that explains the interactions between units in your system to handle the core areas of functionality. This must include a dynamic model of a specific complex piece of functionality: how the system collects data from different sources at different rates. You will be required to discuss the model and justify your design decisions.  10% | A basic model is provided with obvious omissions. | | A basic model is provided, but key details missing / flow of control incorrect. | | Reasonably formed diagrams provided for simple functionality. | | | Reasonable diagrams for complex functionality with major details & syntax correct. Sensible flow of control, but significant errors / difficulties. | | | | | Reasonable diagrams for complex functionality with major details & syntax correct. Sensible flow of control, but minor errors / difficulties. | | | | | Well-formed models that represent a feasible design and directly implementable. | | Excellent, well-formed models that show a reliable, robust design for complex functionality. | | | | |
| ✔ |  | | | | | | | | | | | | | | | | | | | | | | | |
| **C5. Consistency between design artefacts**  Consistency between requirement, solution architecture, static and dynamic model.  5% | Requirements, solution architecture, static and dynamic models are fully inconsistent. | | Very little consistency between requirements, solution architecture, static and dynamic models. | | Solution architecture satisfies the major requirements. Static model realises major part of the architecture. Dynamic model is consistent with static model with minor inconsistencies. | | | Solution architecture satisfies the major requirements. Static model realises major part of the architecture. Dynamic model is consistent with static model with no inconsistencies. | | | | | Solution architecture satisfies the major requirements. Static model realises the architecture. Dynamic model is consistent with static model with no inconsistencies | | | | | Solution architecture aligns with most of the requirements including all functional requirements. Static model realises the solution architecture. Dynamic model is consistent with static model. | | Solution architecture aligns with all requirements including functional and non-functional requirements. Static model realises the solution architecture. Dynamic model is consistent with static model. | | | | |
|  |  |  |  |  |  |  |  |  | |  |  | |  | |  |  | |  |  |  | |  |  | |
| **Implementation elements** |  |  |  |  |  |  |  |  | |  |  | |  | |  |  | |  |  |  | |  |  | |
| **C6. Functionality Implementation**  10% | Insufficient implementation or the implementation not related to the design. | | Core functionality is not present, and the user interaction does not follow the design. | | The components required to implement the base functionality have been implemented successfully following the designs.  The changes in system design (static and dynamic model) or user interaction is not highlighted in the group report. | | | The components required to implement the base functionality have been implemented successfully following the designs. Some of the changes in system design (static and dynamic model) or user interaction is highlighted in the group report. | | | | | The components required to implement the base functionality have been implemented successfully following the designs. The changes in system design (static and dynamic model) or user interaction is highlighted in the group report with no justification. | | | | | The components required to implement the base functionality have been implemented successfully following the designs. The changes in system design (static and dynamic model) or user interaction is highlighted and reasonably justified in the critical reflection. | | The components required to implement the base functionality have been implemented successfully following the designs. The changes in system design (static and dynamic model) or user interaction is highlighted and thoroughly justified in the critical reflection. | | | | |
|  |  | |  | |  | | |  | | | | |  | | | | |  | |  | | | | |
| **C7. Architecture implementation**  15% | No data layer, business layer or presentation layer implemented. | | Improper implementation of data, buiness and presentation layer. | | Basic document structures with no model associations.  Basic mapping of document to JavaScript objects. Basic seggregation between business and data layer is present, A clean simple usable user interface is provided. | | | Good document structure with basic use of model associations.  Documents and associations mapped onto JavaScript objects.  Connect models with MongoDB methods.  Basic separation of models, routes and controllers.  API endpoints respond with JSON.  HTTP verbs are used appropriately.  Basic input verification at html level.  Your stylesheets are reactive.  Sessions are used where necessary | | | | | Very good document structure and model associations.  Very good mapping onto JavaScript objects.  Proper validation added to all models.  Good separation of models, routes and controllers. Use of patterns for code organisation on models and routes.  Setting up validation middleware for incoming data.  Controllers for CRUD operations connected to models (Making calls to data layer classes).  Input verification at endpoint.  Input verification at framework level.  Proper use of design patterns on organising frontend components. | | | | | Excellent document structure and associations capturing all elements of the domain and beyond.  All CRUD operations are performed on all types.  Handling of deletion with associations.  Use of static and instance methods for models (to help with business functions).  Very good separation of models, routes and controllers.  Use of patterns for code organisation of models, routes and controllers.  RESTful URLs with proper verbs and paths.  Apply security privileges to API routes.  Specific controllers for business operations.  Proper error handling of backend calls.  Building UI modal to fetch data asynchronously. | | A relational, or NoSQL or a combination of relational and NoSQL database is used with justification.  Excellent code organisation and full separation of models, routes, controllers and business functions, with use of patterns for all elements.  URLs are fully adherent to RESTful principles (proper verbs, path and response code)  Proper use of JavaScript modules and classes to enforce separation of concerns.  Fully reactive PWA with input validation at all stages, error handling and asynchronous data fetching.  Use of design patterns for handling communication with backend. | | | | |
|  |  | |  | |  | | |  | | | | |  | | | | |  | |  | | | | |
| **C8. Security and User management**  5% | No implementation attempted. | | Poor attempt.  User login/logout not working properly. | | Basic user authentication with login/logout.  Hashed password.  No User management interface.  No access control. | | | User login/logout and user management interface.  Hashed password.  API endpoint protected by authentication. | | | | | User login/logout and user management interface.  Hashed password.  API endpoints protected by authentication.  Basic authorisation at backend. | | | | | User login/logout and user management interface (including user roles).  Hashed password.  Authorisation using role-based access control or other equivalent mechanism. Both front-end and back-end protected by authentication and authorisation. | | Full implementation of login/logout, user management (including roles and permissions) and authorisation, using role-based access control or other equivalent mechanism based on external authorisation infrastructure. Both front-end and back-end protected by authentication and authorisation. | | | | |
|  |  | |  | |  | | |  | | | | |  | | | | |  | |  | | | | |
| **C9. Engineering Practices**  10% | No documentation provided. No evidence of testing is provided. | | Basic evidence of testing is provided. | | Version control tool is used. Professional release management approach is followed. Unit testing has been carried out. | | | Basic documentation for the system is provided. Evidence of professional release management with version control, test, build, and management tools are provided. Extensive unit testing has been carried out. | | | | | Reasonable documentation for the system is provided. Evidence of professional release management with version control, test, build, and management tools are provided. Two forms of testing have been carried out including Unit testing. | | | | | Detailed documentation for the system is provided. Evidence of professional release management with integrated version control, test, build, and management tools are provided. Extensive testing covering test cases identified in requirements and including key aspects of the system. | | Detailed documentation for the system is provided. Evidence of professional release management with integrated version control, test, build, and management tools are provided. Extensive testing covering test cases identified in requirements and including key aspects of the system. Evidence of any additional good engineering practice (e.g., DevOps, Static Testing, Project Management, Deployment etc.) provided. | | | | |
|  |  |  |  |  |  |  |  |  | |  |  | |  | |  |  | |  |  |  | |  |  | |
| **C10. Individual reflection report**  20% | No reflection is provided. | | Very basic reflection is provided on the execution of the project. | | Reflection talks about the role played by the individual in designing and implementing the system. | | | Reflection talks about the role played by the individual in designing and implementing the system with a rationale on why the person has played the role. | | | | | Reflection talks about the role played by the individual in designing and implementing the system with a rationale on why the person has played the role. Additionally, it includes how the individual worked within the group. | | | | | Reflection talks about the role played by the individual in designing and implementing the system with a rationale on why the person has played the role. It includes how the individual worked within the group. Additionally, it talks about lessons learnt by the individual. | | In addition to the distinction criteria, the reflection contains what the individual do differently in his /her role and within the team given a chance to execute the project again. | | | | |
|  |  |  |  |  |  |  |  |  | |  |  | |  | |  |  | |  |  |  | |  |  | |
| **Overall Marks 100%** |  |  |  |  |  |  |  |  | |  |  | |  | |  |  | |  |  |  | |  |  | |