Assessment Brief

Module Leader: Dr. Carlos da Silva		Level: 6
Module Name: Applications: Architectures and Frameworks		Module Code: 55-604385
Assignment Title: Web Development Project		
Individual / Group Individual	Weighting: 50%	Magnitude: *wordcount/length of…* *40 hrs over 20 weeks*
Submission date/time: 10/03/2022, 15:00	Blackboard submission Y Turnitin submission N	Format: source code
Planned feedback date: 31/03/2022	Mode of feedback: Verbal during presentation and annotated rubric	In-module retrieval available: No

Module Learning Outcomes

- LO1: Critically assess the architectures of popular frameworks and the role which such frameworks have in modern system development.
- LO2: Implement software using a variety of frameworks.
- LO3: Examine comparatively systems architectures and demonstrate appropriate uses for them.
- LO4: Compare and contrast the frameworks used during the module.

Assessment Brief

Introduction

In this task you will build an application that makes use of modern application development frameworks and development techniques. You will demonstrate your understanding of the technical opportunities provided by such frameworks.

This application should be built to meet the needs of the clients in the case study presented below.

Case study

ReadBooks Online (RBO) went through a significant expansion in the past 10 years with ideas including adding the ability for users to request books and audiobooks. Although IT management is centralised in its head office, there are several other employees elsewhere.

After a customer satisfaction survey, it was found that substantial improvements could be made to their current catalogue system. A survey was carried out to find current customer trends and desires. The result of this is that any client of RBO can request books or audiobooks not currently in the RBO catalogue to be added for purchase.

This study resulted in a business process view of the system, which is presented in Figure 1. This system is security sensitive, only available to registered users and the RBAC[1] access control model is used to enforce security policies within the organisation. In this context, the catalogue editing process involves three roles: Client, Employee and Authoriser.

A Client is any user who can register an account and can subsequently request (audio)book(s) by opening a request ticket. An Employee is someone responsible for dealing with the request. An Administrator role, which comprises of an authoriser who can monitor/create/delete employee and client accounts as well as approving or denying requests.

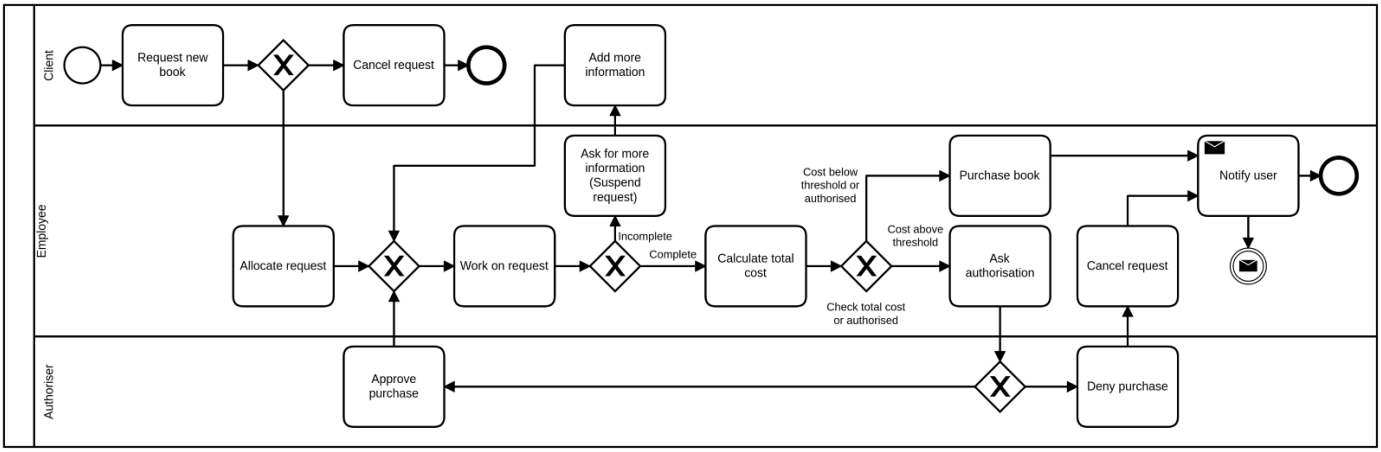

Figure 1. UML Activity diagram capturing the flow of a request in the proposed system.

As shown in the process diagram (figure 1), the process starts with the client raising a ticket (*Request new book*). New requests remain on a queue before being allocated to an employee to be worked on. A client can cancel a request that has not yet been allocated (*Cancel request*) - this caters for accidental requests as well as requests the client is no longer interested in. An employee can allocate a request to themselves (*Allocate request*). Once a request is allocated, an employee can process the request (*Work on request*) by ensuring all relevant information for the book has been submitted by the client. Requests with missing information is sent back to the client to be updated (*Ask for more information*) - the request will be suspended in this state until missing information is added (Add more information) and then automatically reallocated to the Employee that asked for more information. While working on a complete request Employees calculate the total cost of purchase (*Calculate total cost*) and needs authorisation if it goes beyond a configurable threshold. If the total value is above the configured threshold, then the Employee needs an authoriser (*Ask authorisation*) to approve (*Approve purchase*) or deny (*Deny purchase*) the request. If the purchase value is below the authorisation threshold, or it has been authorised, the Employee can proceed with the purchase (*Purchase book*). Clients are notified (notify user) if their request is authorised or declined. Clients, employees and authorisers should be able to see and search through a list of requests; the clients should only be able to view their own requests in this list.

Each one of these phases corresponds to a possible state a request might be on the system. This is important because the ReadBook Online management team is keen on being able to obtain reports on the current status of the requests and be able to measure some performance metrics.

Finally, it has been identified that a real time chat function would greatly improve the system, with chat history being saved and directly connected to a particular request.

The task

You must build a system that can be used to manage purchase requests. Your application must implement the functionalities described below.

A prototype system for RBO to deal with purchase requests. Your client wants a custom application that allows the tracking of book purchase requests from a centralised location.

For each purchase request the system needs to be able to record information about who raised the request (the client), details about the book and time/date information. Requests information also includes the Employee user allocated to it.

The system should also record the different change of states a request goes through with their respective time/date information and user that triggered the change. The states a request can go through should follow the process diagram description of the case study.

Employee users should be able to search for requests using any of the above-mentioned information. Client users should be able to list and search their own requests. Requests can only be modified by the appropriate roles according to the steps indicated in the process diagram. Users should be able to see the different changes a request went through, while maintaining the restriction that client users can only see their own requests.

Optionally, client and employee users should be able to communicate via a chatting system. Notice that the chat system is a synchronous real-time communication channel, different from the “Add more information” step in the activity diagram which is asynchronous. Chats are associated with a particular request.

You must build a system that can be used to manage purchase requests. Your application must implement the functionalities described below.

- Handle the creation and management of purchase requests and the information on them.
- Manage and track the change of status of purchase requests as well as controlling who can change a ticket based on its current status.
- Allow for client users to list, search and see details of their own requests.
- Allow for Employee and Authoriser users to list, search and see details of any request.
- Client and support users should be able to exchange messages through a chat system.
- Manage user authentication, sessions and access control.

You might think of other features that you may wish to add. For example,

In this task we require that you 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 MongoDB 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 middleware layer that accepts HTTP 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.

Demonstrating your work

Your work will be marked either at a walkthrough or via a screencast video. A form will be made available before the submission date on which you should register if you want to have a walkthrough. Alternatively, you may submit a video presentation in which you demonstrate the

functionality of your application and describe its implementation. The video and walkthrough will both follow the script that is given at the end of this assignment specification.

The video may be via desktop capture such as Open Broadcaster (https://obsproject.com/), Screencast-o-Matic (https://screencast-o-matic.com/) or you may use a camera/phone to capture both you and your screen. In either case it is important that the code and output are both clearly readable by a viewer. The video must be in a platform independent format such as MP4 and must be no longer than 15 minutes.

Submission

The deadline for this task is **3 pm on Thursday 10th March, 2022**.

- You must submit an archive of your code to Blackboard.
- Do not include the contents of the node_modules folder or the contents of your database.
- You must include a readme file with a link to your video and instructions for building and running your application. Your readme must be plain text or simply formatted using markdown[1].
- You must include instructions or a script for rebuilding and repopulating the database.
- Include any unit, integration or UI tests that you developed.
- You must prepare a running version of your solution in the Azure Labs VM allocated for you in this module.

Walkthrough script for presentation

This is a script to help you in your presentation. Create an example scenario that you will use throughout the presentation and make sure the database is pre-populated according to your scenario.

1. Explain WHAT you have done.
    a. Explain what it does but not how it does those things.
2. Demo of the application running (Be clear about the examples you have created, and you should not show any code at this point).
    a. Clearly explain the user interface.
    b. Demonstrate the main functionalities (open, list, edit, search requests).
    c. Demonstrate the Chat system.
    d. Demonstrate input verification.
3. Security and user management
    a. Demonstrate your user management interface.
    b. How are users authenticated throughout the application?
    c. How roles and permissions have been implemented?
4. Architecture
    a. How did you organise the architecture of your solution?
    b. How is this organisation reflected in code?
    c. What design or architectural patterns have you used?
5. Data Layer
    a. Show the data structure you use (database tables and model associations).
    b. Show the structure of your JSON document
    c. How is the database wrapped by the Data Access Layer?
    d. How models connect with DB methods? CRUD operations?
    e. How controllers connect to models and how models are validated?
    f. Can you return both single objects and sets of objects?
6. Business Layer - Talk us through the API you implemented.
    a. Are the URLs REST/RESTful?
    b. How the endpoints respond with JSON?
    c. How do you use callbacks, events and promises?
    d. What security features are incorporated into API?
7. Presentation Layer
    a. How UI model fetch data asynchronously?
    b. How inputs are validated?
    c. How errors are handled?
    d. How sessions are used?
8. Software testing and code quality
    a. What is your approach for testing?
    b. What are you testing? How is testing being performed?
    c. What do the results mean?
    d. Why are you testing those things?

Assessment Criteria

	FAIL (insufficient)				THIRD (sufficient)			LOWER SECOND (good)			UPPER SECOND (very good)			FIRST (excellent)				
	Zero	Low Fail	Mid Fail	Marginal Fail	Low 3rd	Mid 3rd	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+**		
C1. Management of purchase requests 20%	No implementation attempted.		Poor attempt. Application not working properly.		Basic implementation handling creation and management of purchase requests.			Good implementation handling creation/management of requests. Managing and tracking change of status of requests.			Very good implementation handling creation/management of requests. Full tracking of change of status. Report/List requests supporting ordering based on different fields.			Full implementation of the case study process. Full tracking of status change. Full support to report/list and search based on different fields.		All the previous plus extra functionalities. E.g., support for items beyond (audio)books; Extra steps in the process; More complex rules for demanding authorisation; Automation of some steps; More advanced reporting capabilities; etc.		
✓																		
C2. Security and User management 10%	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 JS middleware/framework. Both front-end and back-end protected by authentication and authorisation.		
✓																		
C3. Chat system 10%	No implementation attempted.		Poor attempt at implementing chat. Not working properly.		Basic chat functionality without authentication (login).			Basic chat functionality protected by authentication (only logged in users can use).			Chat protected by authentication (only logged users can use) with chat room bound to request.			Chat protected by authentication (only logged users can use) with room bound to request and history.		All of the others plus extra functionalities. e.g., pop-up chat window; Search of chat messages. Etc.		
✓																		
C4. Architecture 20%	No separation of code.		Basic separation between frontend and backend.		Separation on Frontend/backend. Basic separation of models from other elements of backend.			Separation on Frontend/backend. Separation of models and use of design pattern for code organisation. Basic separation of routes and controllers.			Separation on Frontend/backend. Good separation of models, routes and controllers. Use of patterns for code organisation on models and routes.			Separation on Frontend/backend. Very good separation of models, routes and controllers. Use of patterns for code organisation of models, routes and controllers.		Separation on Frontend/backend. Excellent separation and code organisation of models, routes, controllers and business functions, with use of patterns for all elements.		
✓																		
C5. Data layer 10%	No data layer implementation.		Improper implementation of data layer.		Basic document structures with no model associations.			Good document structure with basic use of model associations.			Very good document structure and model associations.			Excellent document structure and associations capturing all		All of the previous plus combination of relational and NoSQL databases.		

	FAIL (insufficient)				THIRD (sufficient)			LOWER SECOND (good)			UPPER SECOND (very good)			FIRST (excellent)				
	Zero	Low Fail	Mid Fail	Marginal Fail	Low 3rd	Mid 3rd	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+**		
					Basic mapping of document to JavaScript objects.			Documents and associations mapped onto JavaScript objects. Connect models with MongoDB methods.			Very good mapping onto JavaScript objects. Database queries implemented using promises. Proper validation added to all models.			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).				
C6. Business layer 10%	Not implemented.		Improper implementation of business layer. No business functions.		The logic of the application is clearly separate from model and view. Routes are coherent and can be deduced once the basic structure is understood.			API endpoints respond with JSON. HTTP verbs are used appropriately.			Setting up validation middleware for incoming data. Controllers for CRUD operations connected to models (Making calls to data layer classes). Input verification at endpoint.			RESTful URLs with prover verbs and paths. Apply security privileges to API routes. Controllers for business operations.		URLs are fully adherent to RESTful principles (proper verbs, path and response code) Proper use JavaScript modules and classes to enforce separation of concerns.		
C7. Presentation layer 10%	Not implemented.		Improper implementation of frontend (not working).		You create a clean, simple and usable UI.			Basic input verification at html level. Your stylesheets are reactive. Sessions are used where necessary.			All the previous plus input verification at framework level. Proper use of design patterns on organising frontend components.			Proper error handling of backend calls. Building UI modal to fetch data asynchronously.		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. Software Testing and Code Quality 10%	No tests done		Some comments and standard code writing etiquette.		Basic unit testing of backend. Basic code commenting.			Automated testing of backend. Use of positive/negative tests. Appropriate code indentation.			Automated unit/integration testing of backend. Use of Web GUI testing. Proper code documentation with comments.			Fully automated unit/integration testing of backend including business functions and API. Appropriate and consistent naming conventions.		Fully automated unit/integration testing of both frontend/backend including Web GUI. Code fully documented with comments. API documentation (e.g., openAPI specification).		
Overall mark:																		