Assessment cover

OXFORD BROOKES UNIVERSITY

Module No:	COMP5047	Module title:	Applied Software Engineering

Assessment title:	Resit Coursework - Software Engineering of a Modern Computer Application

Due date and time**:**	9:00am, 14th April, 2025

Estimated total time to be spent on assignment:	84 hours per student

LEARNING OUTCOMES

On successful completion of this assignment, students will be able to achieve the module's following learning outcomes (LOs):
1. Demonstrate an understanding of the role of requirements analysis and specification in software engineering and to be able to use this knowledge to create use case models and functional models of computer applications.
2. Demonstrate an understanding of the relationship between requirements and design and to be able to apply the knowledge to create structural and behavioural models of computer applications.
3. Critically evaluate and utilise design paradigms of object-oriented analysis and design, component-based design, and service-oriented design.
4. Use software modelling language such as UML and modelling tools in the context of model-driven software engineering.
5. Work in a group to apply the knowledge and skills developed in this module

Engineering Council AHEP4 LOs assessed	
C3	Select and apply appropriate computational and analytical techniques to model complex problems, recognising the limitations of the techniques employed
C5	Design solutions for complex problems that meet a combination of societal, user, business and customer needs as appropriate. This will involve consideration of applicable health & safety, diversity, inclusion, cultural, societal, environmental and commercial matters, codes of practice and industry standards
C6	Apply an integrated or systems approach to the solution of complex problems
C14	Discuss the role of quality management systems and continuous improvement in the context of complex problems
C16	Function effectively as an individual, and as a member or leader of a team

Student Name: **Noah Aslan** **Student Id:** **19197248** **Subsystem:** **CloudTables-Manager**

Statement of Compliance ***(please tick to sign)***

☐ I declare that the work submitted is my own and that the work I submit is fully in accordance with the University regulations regarding assessments *(www.brookes.ac.uk/uniregulations/current)*

RUBRIC OR EQUIVALENT:

Marking grid and marking form are available on Moodle website of the module.

FORMATIVE FEEDBACK OPPORTUNITIES

(a) Discuss your work with your practical class tutor during practical classes; (b) Discuss your work with lecturer and/or practical class tutor in drop-in hours.

SUMMATIVE FEEDBACK DELIVERABLES

Deliverable content and standard description and criteria
Please see attached file of *COMP6030 Coursework Marking and Feedback* for feedbacks on your coursework, which include:
(a) Breakdown of marks on each assessment criterion
(b) Comments on each aspect of the assessment against assessment criteria
(c) Annotations on your submitted work

Task 1:

I create a GitHub repository with the name COMP5047_CW_G39-Noah-Resit, with the link https://github.com/Noah-Aslan/COMP5047_CW_G39-Noah-Resit

All the files will be visible within the repository.

The files and folders within my GitHub repository listed above:

ArcitecturalDesign with the document architectural design task 4.pdf

DetailedDesign is the next folder with the documents Behavioural model task 5 part A and Behavioural model task 5 part B.

The lasr Folder is called FunctionalModel with the files ActivityDiagram – Task3A and ActivityDiagram – Task3B.

The last document in my repository is the PDF document containing all my tasks.

Task 2:

1. Introduction

As the requirements analyst for the CloudTables project, I have been tasked with defining the quality requirements for the **CloudTables-Manager** subsystem. This subsystem is a critical component of the CloudTables SaaS platform, designed specifically for **restaurant managers** to configure and maintain their restaurant profiles through a web-based interface accessible via desktop or tablet.

The CloudTables Manager provides features such as setting up restaurant data, updating operational information, uploading images, managing staff profiles, and maintaining menus. To ensure the subsystem is effective, usable, and reliable at scale, I have outlined its quality requirements with a focus on the following four software quality attributes:

- **Security and Privacy Protection** – to ensure restaurant data, account credentials, and sensitive content are secure and private.
- **Performance** – to guarantee responsive, fast, and seamless interaction with the system under different workloads.
- **Reliability** – to support consistent operation, accurate data management, and fault recovery mechanisms.
- **Scalability** – accommodating multiple restaurants, data entries, and users while maintaining service quality.

These quality attributes will be applied across the CloudTables-Manager subsystem and adapted to each of its core functionalities, as discussed in detail below.

2. General Quality Requirements

These overarching requirements guide how I expect the subsystem to behave across all functional modules.

2.1 Security and Privacy Protection

I require the system to protect all user data using **end-to-end encryption**, employing **TLS 1.3** for data in transit and **AES-256** for data at rest. Access to the manager portal must be secured via **multi-factor authentication (MFA)** and **role-based access control (RBAC)**.

In addition:

- All actions performed by a manager must be logged with a secure, tamper-proof **audit trail**.
- Only authenticated managers should be able to view and modify their restaurant's data.
- The system must be fully compliant with **GDPR** and similar data privacy laws, allowing data export and deletion upon request.

2.2 Performance

To deliver a quality user experience, I expect:

- **95% of user interactions** (clicks, submissions, tab changes) to receive a response in under **1.5 seconds**.
- Image uploads (up to 5MB) to complete in under **5 seconds**, and form submissions to process in **3 seconds or less**.
- The subsystem to support **1,000+ concurrent managers** without noticeable slowdowns.

2.3 Reliability

I require the system to operate with **99.9% uptime** monthly. To support this, the following must be implemented:

- All create/update actions must be **transactional and atomic**, with failure rollback mechanisms.
- Form inputs must autosave every 30 seconds to prevent loss of unsaved data.
- Session states should persist during brief connectivity losses, allowing reconnection without data loss.

2.4 Scalability

To ensure long-term viability, I expect:

- The subsystem to support at least **10,000 active restaurant accounts** with seamless performance.
- Backend services and storage to scale horizontally using modern cloud-native solutions like **Kubernetes** and **NoSQL/document-based databases** for unstructured content like images.
- Front-end performance to remain consistent regardless of geographic access location, with the use of CDNs for media.

3. Function-Specific Quality Requirements

Here, I break down the quality requirements for the two primary manager-facing functionalities defined in the system.

3.1 Setup Restaurant Tenant Data (FR-RM-1)

This functionality allows managers to initially set up their restaurant's profile, including name, address, operating hours, menu details, dining room layout, staff credentials, and food/environment images.

Security and Privacy

- I require access to the setup page to be restricted to verified users via token-based session control.
- Uploaded files must be virus-scanned before being stored in the cloud.
- Data isolation must be enforced per tenant so one manager cannot see or modify another's data.

Performance

- The setup interface must fully render in **under 2 seconds** on standard broadband.
- Searchable fields like cuisine types or map coordinates must use **autocomplete with real-time suggestions** to assist input.
- All form elements should be validated in real-time to reduce submission errors.

Reliability

- I expect unsaved data to persist in the browser using session caching, with a prompt to resume incomplete setup if the page reloads.
- The system must offer **offline editing** where feasible, queuing updates to sync upon reconnection.

Scalability

- I require the system to allow **simultaneous setup of 500+ restaurants** without slowdown.
- Uploaded images must be processed in the background using **asynchronous queuing** to avoid blocking the UI.

3.2 Update Restaurant Information (FR-RM-2)

This function enables daily or occasional updates to existing restaurant data such as special offers, new staff, or revised prices.

Security and Privacy

- Each update must be logged in an **immutable change log**, detailing who made the change and when.

- I require sensitive fields like prices and allergens to be protected using **double confirmation and inline encryption**.

Performance

- Updates must propagate across the system (including Customer and Staff subsystems) within **2 seconds**.

- The system should respond to inline updates (e.g., toggling availability of a dish) in **500 milliseconds** or less.

Reliability

- I require a **versioning system** to restore previous states within 10 minutes of submission.

- Form validations must occur both client-side and server-side to prevent data inconsistency or invalid entries.

Scalability

- The system must support **batch updates** for events like seasonal menu changes, and track the progress of multi-step changes.

- Concurrency handling must ensure no race conditions or data corruption when multiple managers edit different sections simultaneously.

4. Testing and Validation Strategy

To confirm that all quality requirements are met, I will define the following testing strategies:

Security Testing

- Penetration testing will be conducted to check for vulnerabilities (e.g., SQL injection, XSS, CSRF).

- Role-based access and permission scopes will be verified through exploratory testing.

Performance Testing

- I will conduct **load testing** to simulate up to 1,000 concurrent users and identify performance bottlenecks.

- API endpoints will be profiled to ensure they consistently return responses within defined thresholds.

Reliability Testing

- Failover and crash recovery will be simulated using cloud service monitoring.

- I will verify that transaction rollbacks occur as expected on mid-process failures.

Scalability Testing

- Stress tests will be performed using datasets representing 10,000+ tenants and thousands of updates per hour.

- Auto-scaling triggers will be tested in cloud environments to ensure appropriate provisioning under increased load.

Functionality	**Security and privacy**	**performance**	**reliability**	**scalability**
Setup Restaurant Tenant Data (FR-RM-1)	- MFA and RBAC for access control - Encrypted data in transit (TLS 1.3) and at rest (AES-256) - Data isolation per tenant - Virus scan for uploads	- UI loads in under 2 sec - Image upload < 5 sec - Real-time field validation and autocomplete	- Autosave every 30 sec - Offline editing support - Persistent local cache of form data	- Handles 500+ simultaneous setups - Async processing for images - CDNs used for fast media loading
Update Restaurant Information (FR-RM-2)	- Immutable audit logs of changes - Protected fields with double confirmation - CSRF and injection prevention	- Inline updates in < 0.5 sec - Full updates propagate in < 2 sec - Batched change capability	- Version control with 10-min rollback window - Robust client/server-side validation	- Supports batch updates for menus - Handles concurrent edits without conflicts
General Subsystem Requirements	- GDPR compliance - Secure authentication - Role-based permissions	- 95% of interactions < 1.5 sec - Supports 1,000+ concurrent users	- 99.9% uptime - Atomic data transactions - Session recovery after brief disconnection	- Scales to 10,000+ restaurants - Cloud-native infrastructure (Kubernetes, microservices, CDN, etc.)

Task 3 Part A

Use Case Diagram Description – CloudTables-Manager Subsystem

As the requirements analyst for the CloudTables project, I have developed the following use case diagram to define the functional scope of the CloudTables-Manager subsystem. This diagram captures how the primary user of the subsystem—the Restaurant Manager—interacts with its key functionalities.

Actor Involved

- The Restaurant Manager is the sole actor in this subsystem. As they are responsible for configuring and maintaining their restaurant's profile, all use cases in this diagram are initiated by them.

Use Cases Identified

In my analysis, I identified the following key use cases based on the functional requirements of the system:

1. Setup Restaurant Profile
   I designed this use case to represent the manager's ability to input all essential data when registering a restaurant in the system. This includes general information like name, location, cuisine types, and hours of operation.

2. Define/Update Menus
   This use case allows the manager to create and modify their food menu, including pricing, nutritional information, allergy data, and associated images.

3. Manage Table Layout
   Here, the manager defines the physical layout of tables and seating arrangements, supported by images and layout diagrams.

4. Manage Staff Info
   I included this use case to capture the process of adding and updating staff information, such as chef experience, service roles, and qualifications.

5. Upload Restaurant Media
   Since media uploads are essential to both restaurant setup and marketing, I determined this should be treated as a reusable function. As such, I used the <<include>> relationship to show that Setup Restaurant Profile always includes media uploads.

6. View/Edit Restaurant Profile
   This use case represents the ongoing need for managers to review and revise restaurant data after initial setup. It covers changes to any aspect of the restaurant profile.

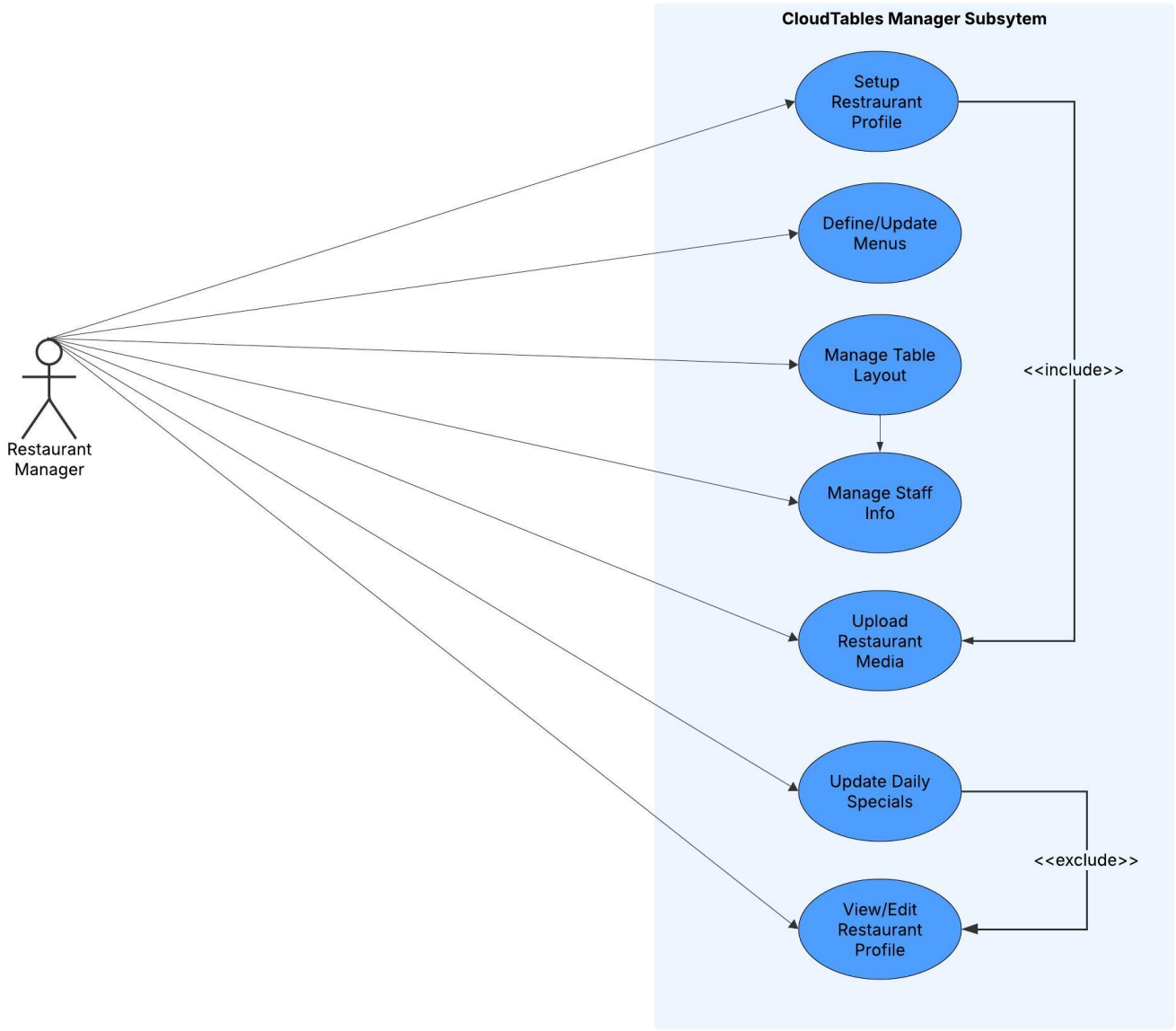
7. Update Daily Specials
   Because updating daily or seasonal menu items is an optional but frequent task, I modelled this as an <<extend>> relationship from View/Edit Restaurant Profile. This shows that while not always required, it extends the core editing functionality when needed.

Use Case Relationships

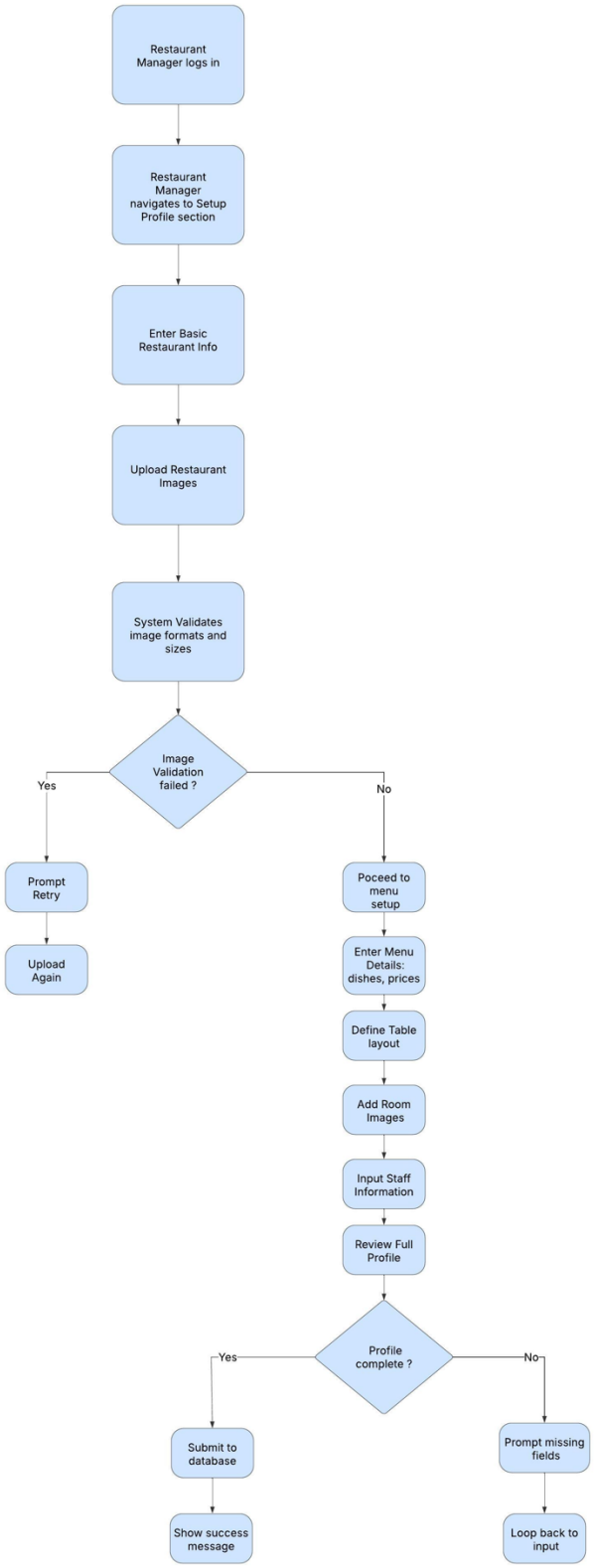
In the diagram, I have applied two key relationships to clarify behaviour:

- <<include>>
  I used this between Setup Restaurant Profile and Upload Restaurant Media to show that media uploads (e.g., photos of food, layout, ambience) are a mandatory part of the setup process.

<<extend>>
I used this from Update Daily Specials to View/Edit Restaurant Profile to indicate that this function may optionally extend the profile editing workflow when temporary menu updates are made.

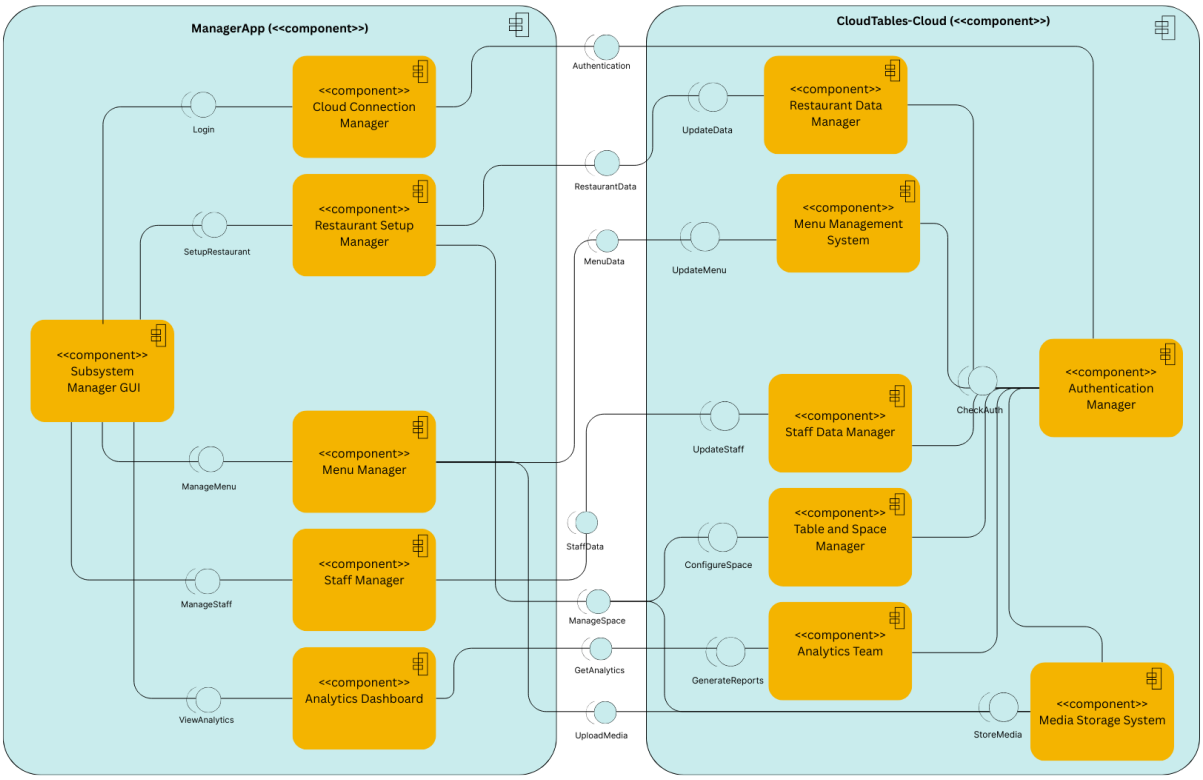


This task was done using Lucid Chart.
Task 3 Part B:



This task was done with Lucid Chart.

Task 4:



This task was done using Canva.

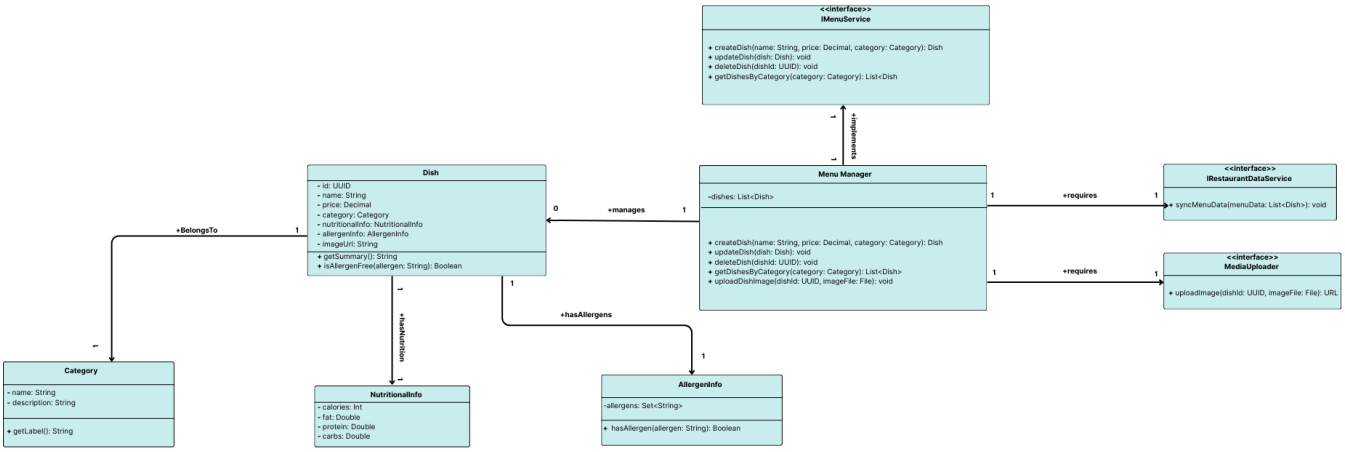
Task 4 continuation:

Component	Connector	Method	Parameters	Functionality
Manager GUI	SetupRestaurant	SetupRestaurantTenantData	basicData, menuData, SpaceData, staffData	Initial setup of restaurant tenant profile with all necessary details.
Manager GUI	UpdateMenu	updateManuInfo	dailySpecials, uodatedPrices, images, allergensInfo	Update existing menu items or add temporary specials.
Manager GUI	UpdateData	updateRestaurantInfo	openingHours, cuisines, address, directions	Modify existing restaurant metadata.
Manager GUI	UpdateStaff	updateStaffInfo	newHires, roleChanges, staffQualifications	Manage staff data, qualifications and changes.
Manager GUI	ViewAnalytics	generateReport	timePeriod, metricType	Access analytical reports for decision making.
Manager GUI	Login	authenticateManager	Username, password	Login to the system.
Menu Manager	ManageMenu	createOrUpdateMenu	menuData	Create or manage menu structure.
Staff Manager	ManageStaff	manageStaffData	staffData	Handle staff-related data.
Analytics Dashboard	GetAnalytics	fetchAnalyticsData	metricType, filters	Provide visualised performance data.
Authentication Manager	CheckAuth	validateLogin	Username, password	Validate credentials and initiate session.
Media Storage System	StoreMedia/UploadMedia	uploadMediaFile	File, filetype	Store images of dishes or restaurant layout.
Table and space manager	CnfigureSpace/ManageSpace	setupOrUpdateSpace	tableCount, roomLayout, parkingInfo	Define or adjust the physical layout of

				the restaurant.

Here is a table summarising the components and connectors including their methods and parameters with their functionalities and meanings which I have used to create my architectural design of the subsytem.

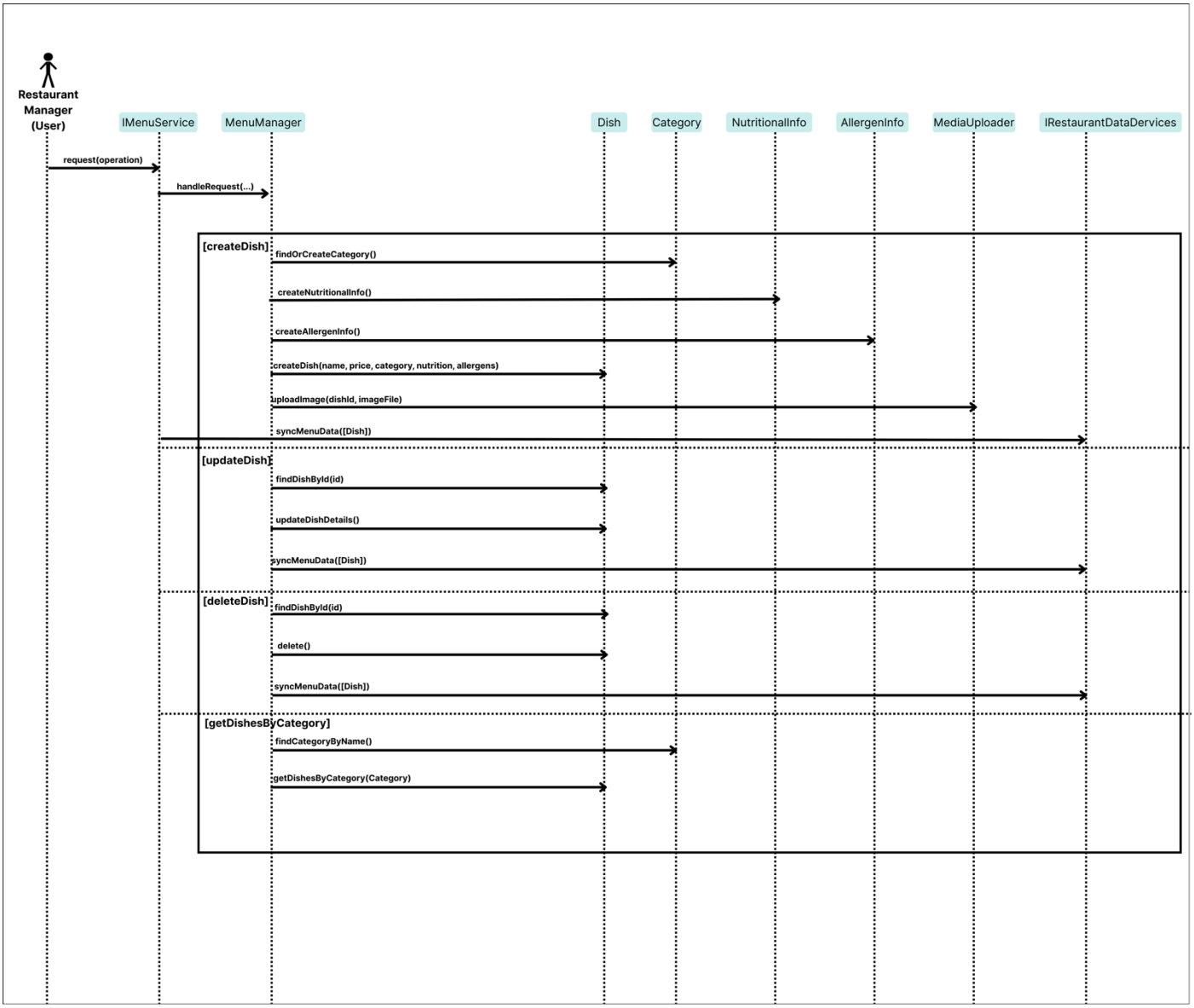
Task 5 Part a:



This task was done using Canva.

I created a class called Dish. My dish class includes the attributes – id, name, price, category, nutritionalInfo, allergenInfo and imageUrl.
Then I made a MenuManager class, the MenuManager class's attributes help to maintain a list of dishes and provides the methods to create, update and delete/remove dishes, get dishes by specific category and upload images for dishes.
Then, I created and defined the category, NutritionalInfo and allergenInfo classes. The category class has the name and description attributes. NutritionalInfo class includes the attributes calories, fat, protein and carbs. AllergenInfo class includes a list of allergies, allergens and a method hasAllergen.
Next, I implemented the interfaces IMenuService with syncMenuData, IRestaurantDataService with uploadImage, MediaUploader for image handling.
Lastly I created and established the relationships between all the classes I made – each dish has a category, nutritional info and allergy information. These relationships are all one to one relations.

Task 5 Part B:



This task was done using Canva.

I identified the user, the manager as the main user.
I created and structured the user interactions with the system I created. The user will perform tasks such as: createDish(), updateDish(), deleteDish() and getDishesByCategory().
I structured and designed the system's internal workflow for each specific individual operation: For creating a dish: I used methods such as findOrCreateCategory(), createNutritoinalInfo(), and syncMenuData() to update and renew the system.
For updating and/or deleting a dish: I added and included a step to findDishById(id) before updating and/or removing that dish. After the changes, syncMenuData() is called to reflect the updates.
I included an image upload facility – I showed how uploadImage(dishId, imageFile) are used throughout the IRestaurentDataService interface.
Finally, I visualised the sequence and logic of all operations – all the methods and data flow have are illustrated in the diagram above for representation.