Software Design and Architecture: Final Project Report

Inventory Management System ADD Process

Submitted: Dec 6, 2021

Submission By:
Ahmet Karapinar (100750048)
Joshua White (100747854)
Michael Metry (100747141)
Ridwan Hossain (100747897)

Table of Contents

1.1 - Elicit System Requirements

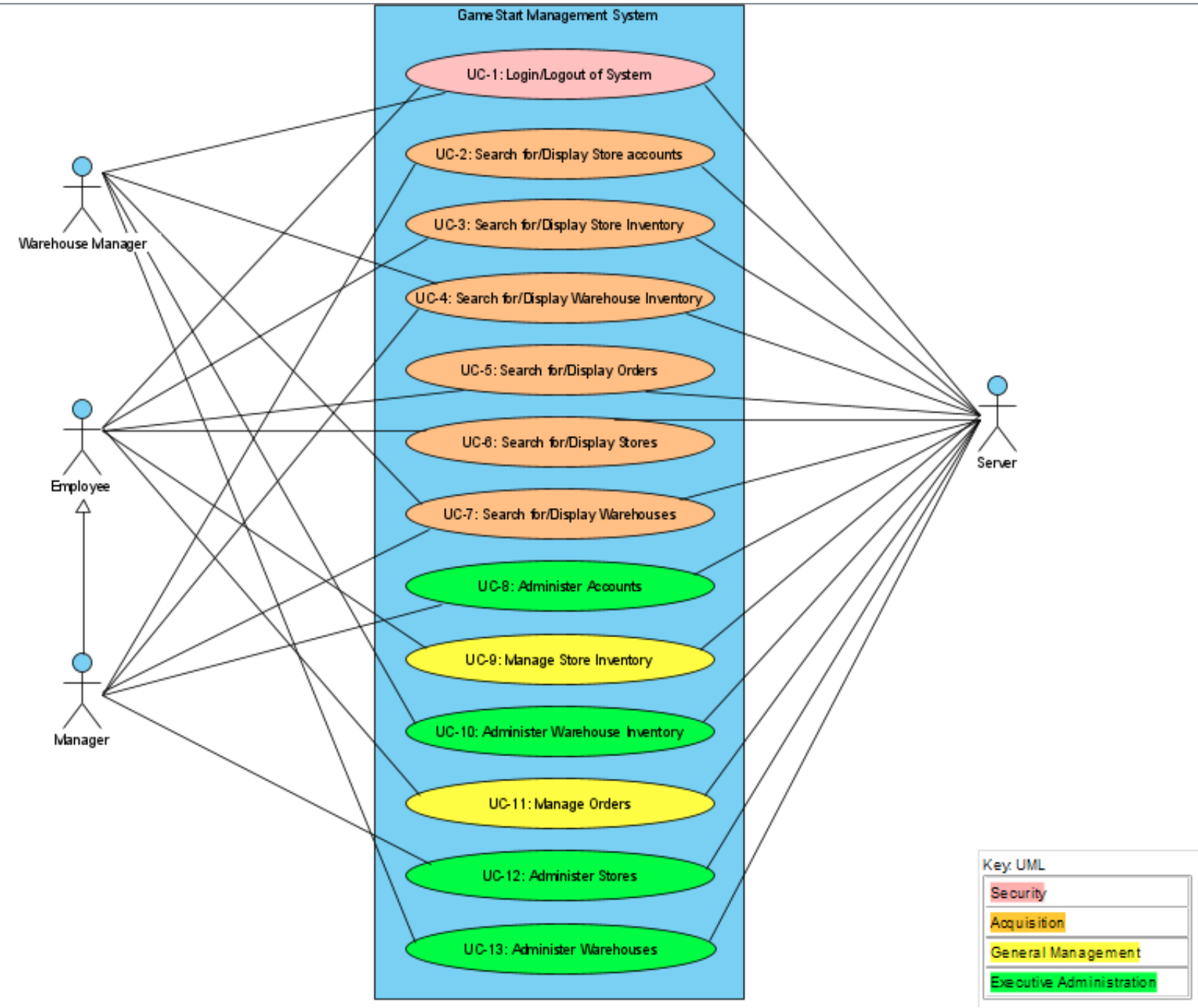

FIGURE 1.1 Use Case Model for the Inventory Management System

1.1.1 Use Cases

Use Case	Description
UC-1: Login/Logout of System	A user (Employee, Manager, Warehouse Manager) will enter account credentials (user ID & password) and upon successful login, the user will be granted access to their account (Where each type of user has varying levels of permissions). Upon completion of the desired tasks, the user will select the logout icon to deactivate the current session.
UC-2: Search for/Display Store accounts	A user (Manager) will enter account credentials (user ID & password), or the user will select an account from the dropdown box. Upon successfully entering the credentials, the user will receive appropriate data about accounts based on entered or selected information.
UC-3: Search for/Display Store Inventory	A user (Employee, Manager) will search the database system for the current store's inventory, where they will enter a store's id and receive a broad list of the current store's inventory of items.
UC-4: Search for/Display Warehouse Inventory	A user (Manager, Warehouse Manager) will search the database system for the current inventory of a warehouse, either by entering the warehouse ID or viewing a list of all inventory in the specified warehouse.
UC-5: Search for/Display Orders	A user (Employee, Manager) will search the database system for the customer orders associated with the current store through numerous methods. The user will either choose to display a specific store's orders, or search by entering a unique order number. After entering the information, data corresponding to that order will be displayed to the user.
UC-6: Search for/Display Stores	A user (Employee, Manager) will search the database system for other store locations by entering a store id, or by viewing a list of other store locations with their associated information.
UC-7: Search for/Display Warehouses	A user (Manager, Warehouse Manager) will enter a warehouse id into the database system. Given the warehouse id is correct and corresponds to an existing warehouse, information corresponding to that warehouse will be displayed.

UC-8: Administer Accounts	A user (Manager) will change account attribute(s) associated with a particular account id. The changes are then sent to the server for constraint validation. Upon successful validation, the changed attribute(s) are going to be updated in the database.
UC-9: Manage Store Inventory	A user (Employee, Manager) will either add items, remove items, or update particular details about items within the inventory of the current store.
UC-10: Administer Warehouse Inventory	A user (Warehouse Manager) will either add items, remove items, or update particular details about items within the inventory of the current warehouse.
UC-11: Manage Orders	A user (Employee, Manager) will view current “live” orders and will either add a new order, remove an order from the database system or update order details.
UC-12: Administer Stores	A user (Manager) will either add a new store, remove an existing one, or enter a store id in order to view information corresponding to that store. Given the id is correct, the user will update various details regarding that store.
UC-13: Administer Warehouses	A user (Warehouse Manager) will either add a new warehouse, remove an existing one, or enter a warehouse id in order to view information corresponding to that warehouse. Given the id is correct, the user will update various details regarding that warehouse.

1.1.2 Quality Attribute Scenarios

ID	Quality Attribute	Scenario	Associated Use Case
QA-1	Usability	Users of the database system should be able to easily perform the database’s numerous functions during normal operations. A user should take approximately 7 clicks to perform a desired task.	ALL
QA-2	Availability	A user sends a request via the server api to access the database system. The database system should be online to accommodate that request. The servers should run 100% of the time, it should be available 24/7.	ALL

QA-3	Security	A user attempts to access information or make changes which their account is not authorized to. Their request will be denied and their action will be restricted 100% of the time (no data in the system is compromised).	UC-1, UC-8, UC-10, UC-13, UC-13
QA-4	Availability	A user sends requests to the server api, most commonly read, write and update, for various entities such as items, orders, warehouses etc. At peak load; 99.99% of the requests are successfully processed by the system.	UC-2 to UC-13
QA-5	Performance	A user queries data from, adds data to, or modifies data within the database system. The system should be able to perform that request within 1000 ms.	UC-2 to UC-13
QA-6	Modifiability	When a new entity is introduced to the system (a new user type), the new entity should be able to be successfully added without needing to make major changes to the system's components or core functionalities. The modification should be able to be made with no defects introduced, within 3 hours, and minimal effort.	UC-1

1.1.3 System Constraints

ID	Constraint
CON-1	The Manager will be the only account level which has access to executive administration functions (See use case model) that pertain to a store.
CON-2	The Warehouse Manager will be the only account level which has access to executive administration functions (See use case model) that pertain to a warehouse.
CON-3	The maximum number of inventory items within a single store is 999 items.
CON-4	The maximum number of inventory items within a single store is 9999 items.
CON-5	The database system will only be available on local servers, one pertaining to each store.

CON-6	The database system will only be accessible on devices that have the correct credentials to log into the system.
CON-7	The database system will be accessible through a web browser (Chrome, Firefox, Safari, etc.) on different platforms (Windows, Linux, OSX), given CON-6 is satisfied.
CON-8	The database system can support a maximum of 30 individual users per server (by extension, per store).

1.1.4 Architectural Concerns

ID	Concern
CRN-1	Administering a system structure that facilitates a reliable piece of software that adheres to all business requirements.
CRN-2	Delegate tasks to members of the development team
CRN-3	Leverage the team's knowledge about Python technologies, including Django, SQLAlchemy, Asyncio, MVC frameworks and the Python language.
CRN-4	To be able to sufficiently maintain a centralized, data strong application system.

1.2 - ADD Input Review

Category	Details
Design Purpose	This is a Greenfield System in a mature domain. It is a more traditional and well-known database type application and covers no new technological bases. The purpose of the design is to create an outline that will support the system and will be able to administer the final product's goals.
Primary Functional Requirements	From the use cases that was already defined, the primary ones are of the following: ● UC-3: Because it supports the core business ● UC-4: Because it supports the core business ● UC-5: Because it supports the core business ● UC-9: Because it provides the core functionality of the system ● UC-10: Because it provides the core functionality of the system ● UC-11: Because it provides the core functionality of the system
Quality attribute scenarios	From The previously defined list of scenarios described in 1.1.2, the below table has been weighted to assign priority amongst each of the quality attribute requirements.

	<table><tr><th>Scenario ID</th><th>Importance to the Customer</th><th>Difficulty of implementation according to architect</th></tr><tr><td>QA-1</td><td>Medium</td><td>Low</td></tr><tr><td>QA-2</td><td>High</td><td>Medium</td></tr><tr><td>QA-3</td><td>High</td><td>Medium</td></tr><tr><td>QA-4</td><td>Medium</td><td>Low</td></tr><tr><td>QA-5</td><td>Low</td><td>Medium</td></tr><tr><td>QA-6</td><td>High</td><td>High</td></tr></table> From the list; QA-2, QA-3, and QA-6 have been selected as QA drivers.
Constraints	From the list of described constraints in section 1.1.3 CON-1, CON-2, CON-5, CON-7 have been selected as drivers.
Concerns	CRN-1, CRN-3, and CRN-4 have been selected as drivers.

1.3: ADD Iteration 1 - Establishing an Overall System Structure

1.3.1 Establish Iteration Goal by Selecting Drivers

The goal of the first iteration encompasses the establishment of an overall system structure in regards to the design of a greenfield system. Addressing CRN-1, administering a system structure that facilitates a reliable piece of software that adheres to all business requirements, will help to achieve the first iteration's goal. All drivers that guide the system structure must be considered in order to successfully complete iteration 1. The drivers are:

- QA-2: Availability
- QA-3: Security
- QA-6: Modifiability
- CON-5: The database system will only be available on local servers, one pertaining to each store.
- CON-7: The database system will be accessible through a web browser (Chrome, Firefox, Safari, etc.) on different platforms (Windows, Linux, OSX).
- CRN-1: Administering a system structure that facilitates a reliable piece of software that adheres to all business requirements.
- CRN-3: Leverage the team's knowledge about Python technologies.
- CRN-4: To be able to sufficiently maintain a centralized, data strong application system.

1.3.2 Select which Elements of the System to Refine

As per the development process surrounding a greenfield system, the element to be refined in the first iteration is the entire Inventory Management System, which will be refined by decomposition.

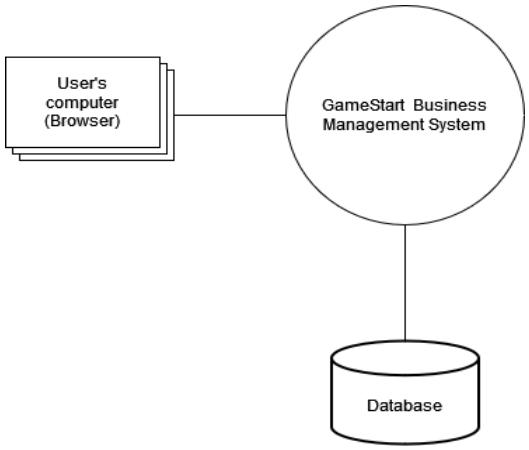

FIGURE 1.2 Context Diagram for the Inventory Business Management System

1.3.3 Select One or More Design Concepts that satisfy Selected Drivers

Design Decisions and Locations	**Rationale**
The client side of the system will be structured using the rich internet application architecture.	The rich internet application architecture focuses on a web browser based (CON-7) application whilst maintaining a rich user interface. This will allow the system to offload the majority of the work to the server side of the application. This would intake a GUI with minimal user activity which utilizes the server to query the database.

Discarded alternative	
Alternative	**Reason for discarding**
Rich client reference architecture	The rich client architecture is targeted to support limited to no network capabilities and imposes much more focus on the user's machine resources. Although it supports all other requirements this does not align best for the design goals of the project and reason for its disposal.

<table>
<tr>
<td>The server side of the system will be structured using the web application reference architecture.</td>
<td>This is a suitable architecture because it supports a separate application and server layer requiring requests to be made to the server from the local system(CON-5, CON-6). The server would access the database and would provide constant access to the user application (QA-2).
<table>
<tr><th colspan="2">Discarded alternative</th></tr>
<tr><th>Alternative</th><th>Reason for discarding</th></tr>
<tr><td>Mobile Application Reference Architecture</td><td>While this style of architecture aligns well with the design goals of the project, the system is not required to be accessed through mobile devices. Overall the web application reference architecture is most suitable for the pro</td></tr>
</table>
</td>
</tr>
<tr>
<td>The deployment will be structured using a two tier deployment (client-server) model.</td>
<td>Although it is a fairly basic deployment structure, two tier deployment(client-server) works extremely well for this project. Because the client is based on the access to the hardware and the server holds both the executable code and the database its preliminary structure already resembles that of the client-server model. This supports CON-7 as a web application will be used to access the second tier of the architecture.
<table>
<tr><th colspan="2">Discarded alternative</th></tr>
<tr><th>Alternative</th><th>Reason for discarding</th></tr>
<tr><td>Three tier deployment model</td><td>Although the three tier deployment model could have also worked for this project it was determined that separating the</td></tr>
</table>
</td>
</tr>
</table>

	application server and database was unnecessary as the extra security it provides being one of the main benefits is unnecessary for the design of the system.

1.3.4 Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Design Decisions and Locations	Rationale
Remove the plug-in execution container and isolated storage from the rich internet application	The rich user interface will be constructed using the Svelte front-end framework, which will exist on the server-side of the system and be provided to the client through browser protocols. Furthermore, no data is required to be persistent on the client-side of the system. As such, a plug-in execution container and isolated storage is not required in order to adhere to the design goals of the system.

1.3.5 Sketch Views and Record Design Decisions

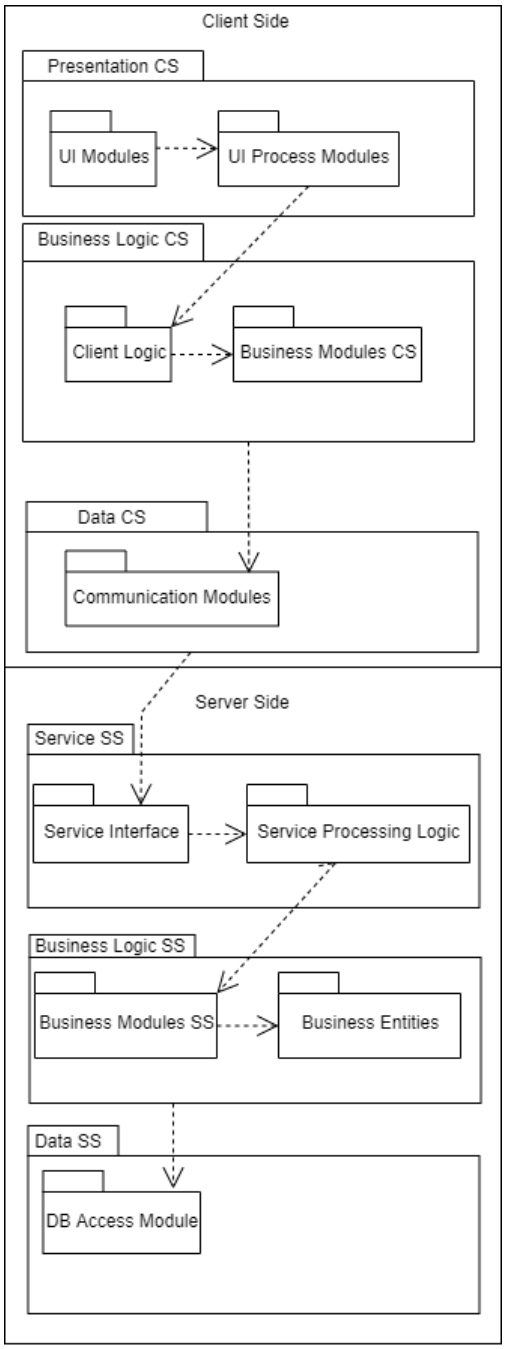

FIGURE 1.3 Modules Obtained from the Reference Architecture

Figure 1.3 was designed to demonstrate an early representation of the interactions between the modules of the client and server sides of the application.

The table below demonstrates a brief outline of each element's responsibility.

Element	Responsibility
UI Modules	This module displays the user interface and receives the user's input.
UI Process Modules	This module is responsible for the control flow of the system's use cases, such as navigation.
Client Logic	This module is responsible for controlling the processing of the user's input and client-side logic such as correctly responding to user commands.
Business Modules CS	This module exposes business functionality from the server-side of the system.
Communication Modules	This module is responsible for the communication with other layers of the system such as the server which hosts more in depth logic and the data base.
Service Interface	This module provides services to the client layer to be accessed and utilized.
Service Processing Logic	This module controls how the server responds to a request from the client's machine.
Business Modules SS	This module is responsible for implementing business logic and business operations in the server.
Business Entities	This module makes up the domain model.
DB Access Module	This module is responsible for the access and persistence of business entities (objects) in the relational database.

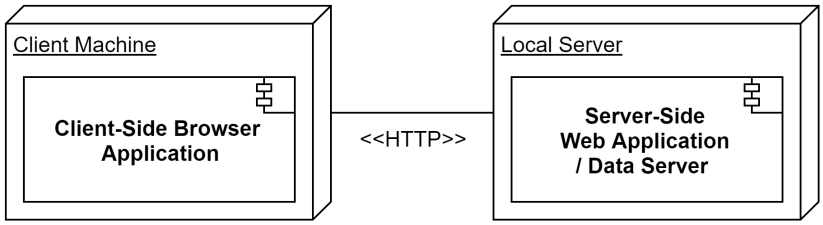

Figure 1.4 Initial Deployment Diagram for the Inventory Management System

Figure 1.4 illustrates the general composition of the system in it's deployment environment. It utilizes a two-tier deployment architecture as the layout for its distributed deployment. The client-side of the system will be contained on the user's browser, and the server-side will be contained in a specific location's local server.

The responsibilities of the elements found in Figure 1.4:

Element	Responsibility
Client Machine	The user's machine which will access the web application through a browser application residing on the client's PC, which handles all client-side logic.
Local Server	The server which is local to an establishment. It will host the web application and contains all the business and server-side logic along with all persistent data.

Description of relationships between elements in Figure 1.4:

Relationship	Description
Between client machine and local server	The browser application on the client machine will access services and communicate with the local server using HTTP protocols.

1.3.6 Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
		CON-5	A Reference architecture was selected that will support this type of functionality.
		CON-7	A Reference architecture was selected that will support this type of functionality.
QA-2			No relevant decisions made
QA-3			No relevant decisions made
QA-6			No relevant decisions made
		CRN-1	A prototype of the architecture has been developed
	CRN-3		Technologies for the base language have been chosen (Python). Front-end and back-end frameworks/technologies have been considered but not yet finalized.
		CRN-4	A prototype of the architecture has been developed
UC-3			No relevant decisions made

UC-4			No relevant decisions made
UC-5			No relevant decisions made
UC-9			No relevant decisions made
UC-10			No relevant decisions made
UC-11			No relevant decisions made

2.1: ADD Iteration 2 - Identifying Structure to Support Primary Functionalities

2.1.1 Establish Iteration Goals by Selecting Drivers

Throughout this iteration, the general architecture concerns will be addressed such as selecting the best structure in order to support the primary functionality of the project. This will be done by considering the system's primary use cases.

- UC-3
- UC-4
- UC-5
- UC-9
- UC-10
- UC-11

2.1.2 Select One or More Elements of the System to Refine

In this iteration, the elements that will be refined are the modules that most pertain to the primary use cases as outlined above. The modules of focus will be components that are involved with communication between client-side and server-side, api endpoints, and the security of data querying.

2.1.3 Select One or More Design Concepts that satisfy Selected Drivers

Design Decisions and Location	Rationale and Assumptions
Define explicit interfaces of communication between the client-side and server-side architectures.	Utilizing explicit interfaces ensures that there is separation and encapsulation between crucial communication components of the system.
Use Svelte Framework for client-side	Svelte is a Javascript framework for building intuitive user interfaces. Svelte allows for the construction of an entire front-end application that is composed of child components. Svelte applications communicate among its components through secure event-based communication

	methods and forms of prop passing. Furthermore, the Svelte framework allows for very easy raw data communication with a server-side api by exposing certain endpoints used for data transfer.
Use Django REST framework for server-side	Django REST is a widely used framework that allows for the easy development of powerful back-end web apis. It contains a robust serialization engine which is compatible with both ORM and non-ORM data sources. Django REST has easy to use emitters, parsers, validators, and authenticators. Django REST provides generic classes for CRUD operations, and has advanced HTTP response handling. All these capabilities make the Django REST framework very fitting to be implemented as the server-side api for the system.

2.1.4 Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Design Decisions and Location	**Rationale and Assumptions**
Only contain UI process modules and Client logic on the front-end	By having all UI process modules contained strictly within the client-side api, all the client logic and control flow will be managed by the front-end framework, allowing the system to separate it's client-side architecture while having certain endpoints be exposed for communication to the server-side.
Contain all business entity information and critical business logic on the back-end	Having the server-side provide services through a back-end api, endpoints can be exposed as means of communication whilst business logic may reside on the server-side of the system. This ensures only raw data is communicated and any logic is handled separately on the server-side. There is a secure transfer of information between endpoints while all constraints, business operations, and critical business logic is handled by the back-end api.

2.1.5 Sketch Views and Record Design Decisions

In order to support the chosen architectural design concepts, it is vital to build an initial domain model for the system to demonstrate the important entities in the domain, as well as their relationships. This will be done to provide a general overview of the structure of the system's domain elements, and ensure they contribute to addressing the chosen design concepts to satisfy our selected drivers.

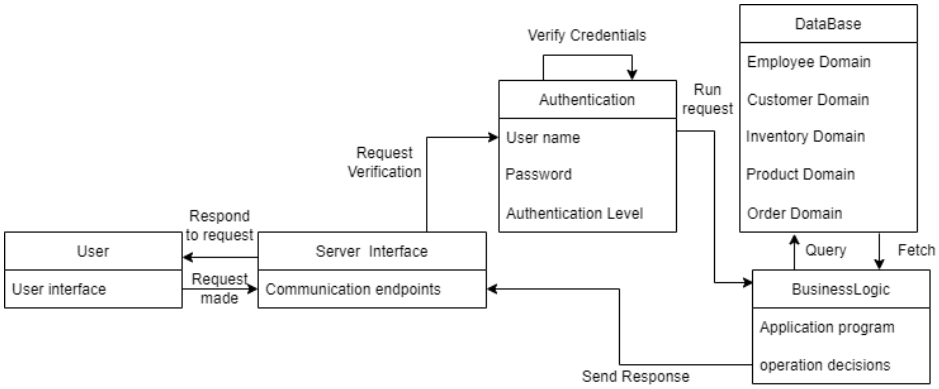

Figure 2.1 Initial Domain Model

Figure 2.1 shows an initial domain model of the system in order to support the design decisions made in iteration 2.

The responsibilities of the elements found in Figure 2.1:

Element	**Responsibility**
User	The User element represents the client side browser, which consists of the user interface and handles all client-side logic of the web application. This element will make requests to the Server Interface element in order to fetch useful data.
Server Interface	The Server Interface provides endpoints of communication for the user to access the server-side's services. This interface will ensure that there is separation and encapsulation between the front-end and back-end. This type of structure, where all business logic resides on the server-side, helps the system to address the targeted design concepts of defining an explicit interface of communication, between the client-side and server-side.
Authentication	The Authentication element's responsibility is to act as the middleware between the server interface and the server's contents. It makes sure that the request is authorized by the user's authentication level. Finally It forwards the request to the business logic component of the server-side.

Business Logic	The Business Logic element contains all logical components that are central to the functional requirements of the system. It is the part of the server-side which encodes the real-world business rules of the problem domain and determines how data is structured, created, stored, and changed.
Database	The Database element is responsible for storing all the persistent data within the system to be used.

Figure 2.2 represents the domain objects that were instantiated for the Use Case model

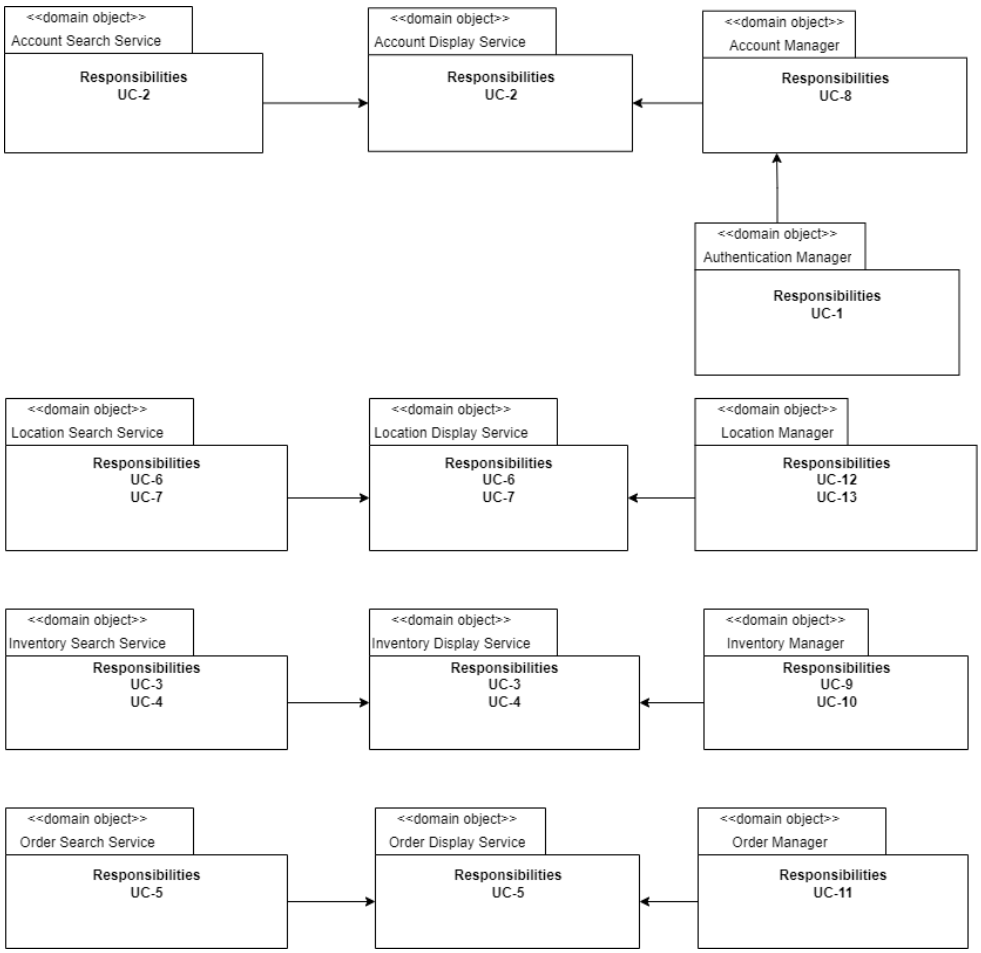

Figure 2.2 Domain Objects Associated with the Use Case Model

Figure 2.3 illustrates a sketch of a module view with modules that are derived from the business objects and associated with the primary use cases.

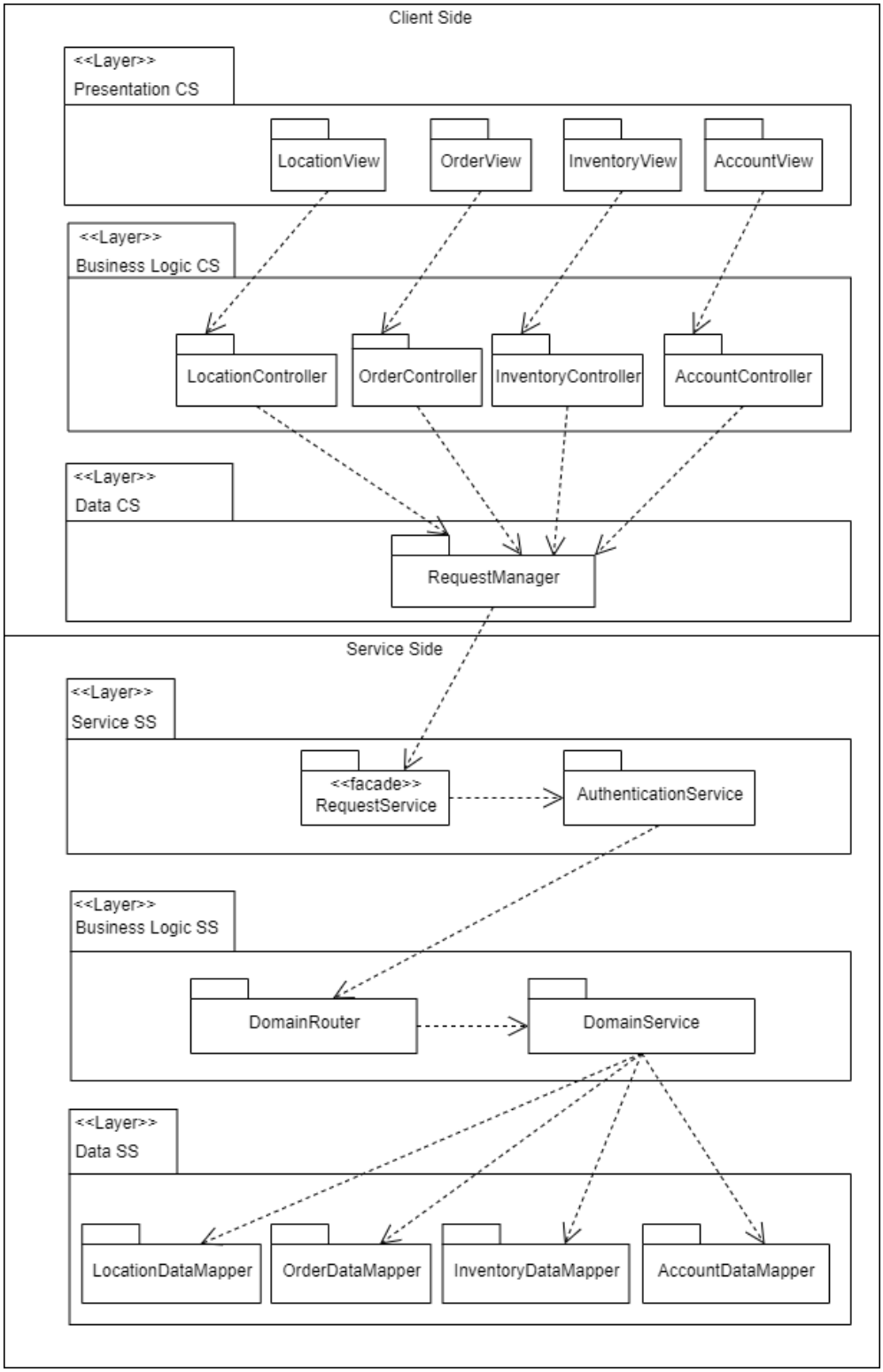

Figure 2.3 Domain Objects Associated with the Use Case Model

The responsibilities of the elements found in Figure 2.3:

Element	**Responsibility**
LocationView	Displays the Location (Store, Warehouse) representation and updates the LocationView based on server-side requests.
OrderView	Displays the Order representation and updates the OrderView based on server-side requests.
InventoryView	Displays the Inventory representation and updates the InventoryView based on server-side requests.
AccountView	Displays the Account representation and updates the AccountView based on server-side requests.
LocationController	Responsible for providing the necessary information to the LocationView for displaying the location(s) representation.
OrderController	Responsible for providing the necessary information to the OrderView for displaying the order(s) representation.
InventoryController	Responsible for providing the necessary information to the InventoryView for displaying the inventory(ies) representation.
AccountController	Responsible for providing the necessary information to the AccountView for displaying the account(s) representation.
RequestManager	Responsible for making requests to the server side.
RequestService	Provides a facade that receives requests from the clients.
AuthenticationService	Responsible for verifying the coming request based on authentication level, password and username.
DomainRouter	Responsible for mapping coming requests from authentication service to domain service.
DomainService	Contains business logic related to selected domains that came from the client side.

LocationDataMapper	Responsible for basic CRUD operations related to locations (Store, Warehouse etc.)
OrderDataMapper	Responsible for basic CRUD operations related to orders.
InventoryDataMapper	Responsible for basic CRUD operations related to inventories.
AccountDataMapper	Responsible for basic CRUD operations related to accounts (Employee, Customer etc.)

2.1.6 Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
		CON-5	No relevant decisions made
		CON-7	No relevant decisions made
QA-2			No relevant decisions made
	QA-3		Some components and interfaces elicited during this iteration will help to protect data from being accessed without authorization.
	QA-6		Certain components and interfaces derived will allow for enough separation in the software to allow for this quality attribute to come to fruition, but details have not been defined.
		CRN-1	No relevant decisions made
		CRN-3	The Svelte and Django frameworks have been chosen to support this constraint.
		CRN-4	Certain domain objects, such as the DataMapper components, have been instantiated that will help to address the relevant considerations associated with this concern.
		UC-3	The InventoryView object across the client-side's presentation layer and the DomainService object across the business logic layer, have been derived to support this use case.

		UC-4	The InventoryView object across the client-side's presentation layer and the DomainService object across the business logic layer, have been derived to support this use case.
		UC-5	The OrderView object across the client-side's presentation layer and the DomainService object across the business logic layer, have been derived to support this use case.
		UC-9	The InventoryView object across the client-side's presentation layer and the DomainService object across the business logic layer, have been derived to support this use case.
		UC-10	The InventoryView object across the client-side's presentation layer and the DomainService object across the business logic layer, have been derived to support this use case.
		UC-11	The OrderView object across the client-side's presentation layer and the DomainService object across the business logic layer, have been derived to support this use case.

3.1: ADD Iteration 3 - Addressing Quality Attribute Scenario Driver (QA-3)

Essential structural decisions and design choices were made in the first 2 iterations of the ADD process. The third iteration will begin to discuss the accomplishment of a more crucial quality attribute, which will play a key role in the design of the system.

3.1.1 Establish Iteration Goals by Selecting Drivers

The third iteration of the ADD process will primarily focus on the third quality attribute scenario, QA-3: A user attempts to access information or make changes which their account is not authorized to. Their request will be denied and their action will be restricted 100% of the time. This iteration will focus on addressing the third quality attribute scenario (QA-3) and making decisions regarding the system's architecture in order to handle the fulfilment of this quality attribute.

3.1.2 Choose One or More Elements of the System to Refine

For this security scenario, the elements of the system which have been selected for revision is the Authentication element as defined in the initial domain model of the system architecture.

3.1.3 Choose One or More Design Concepts That Satisfy the Selected Drivers

Design Decisions and Locations	Rationale
Introduce the **Authorize Actors** security tactic by implementing an additional authorization component into the authentication element.	The authentication module of the system will, as before, verify that the username and password which has been entered is correct via passing that request to the business logic element, which will cross reference that information with the database. The design decision to include additional refinement within the authentication module is the introduction of the authorize actors tactic by implementing an authorization forwarding component in the authentication module.

3.1.4 Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Design Decisions and Locations	Rationale
Implement an authorization forwarding component within the authentication element in order to filter server-side accessibility to the user.	After successful verification of a user's credentials, all requests made by the user through the server interface must now also pass through this additional component. The authorization forwarding component will essentially filter certain aspects of the server-side's functionality such that if a user's level of authorization does not allow for access to certain information/operations, this component will either deny access or permit access to certain pages and functions, as well as hide sensitive data by not passing particular requests to the business logic element.

3.1.5 Sketch Views and Record Design Decisions

Figure 3.1 shows a refined domain model that is specific to the authentication module that was previously defined in the overall system's initial domain model

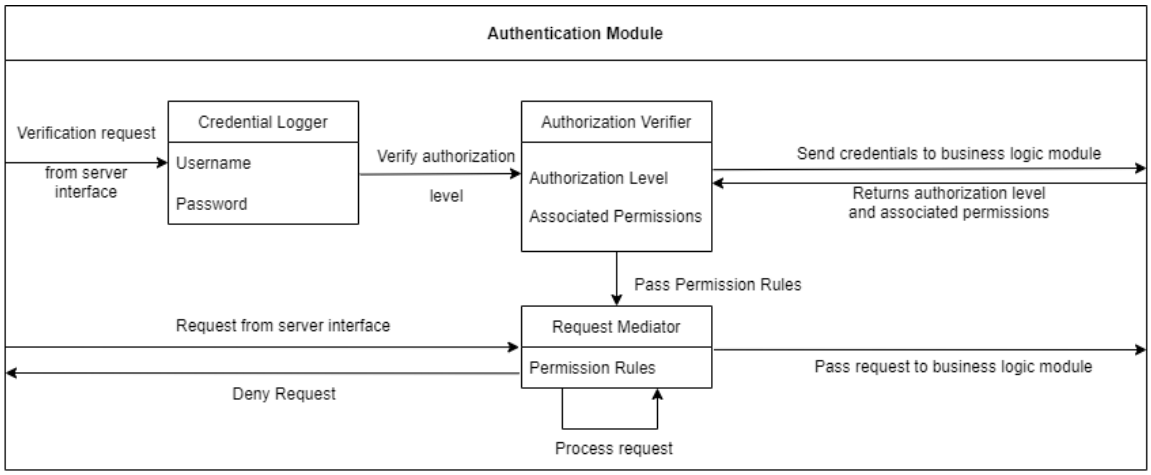

Figure 3.1 Domain model specific to the refined Authentication module

The responsibilities of the elements found in Figure 3.1:

Element	Responsibility
Credential Logger	This element will parse and prepare the username and password data that has been received from the server interface such that it matches the format stored within the database, and will pass that data to the Authorization Verifier element.
Authorization Verifier	After retrieving that username and password data, this element will send the credentials to the business logic module, where the Business Logic module will be responsible for using the credentials to query the authorization and permission data, where it will return that data, when received, to the Authorization Verifier in the Authentication module. Afterwards, this element will pass the fetched permission rules to the Request Mediator.
Request Mediator	After receiving the correct permission rules, the Request Mediator will act as a broker for all requests coming through the server interface. If this element finds that a request is within the authorization of the user, it will pass that element to the Business Logic module, which will handle all further processing. However, if this element finds that the request is outside of the user's authorization, the Request Mediatior will deny that request and not pass anything forward.

Figure 3.2 shows the initial domain model after making some revisions in order to include messages pertaining to the refined Authentication module defined in Figure 3.1

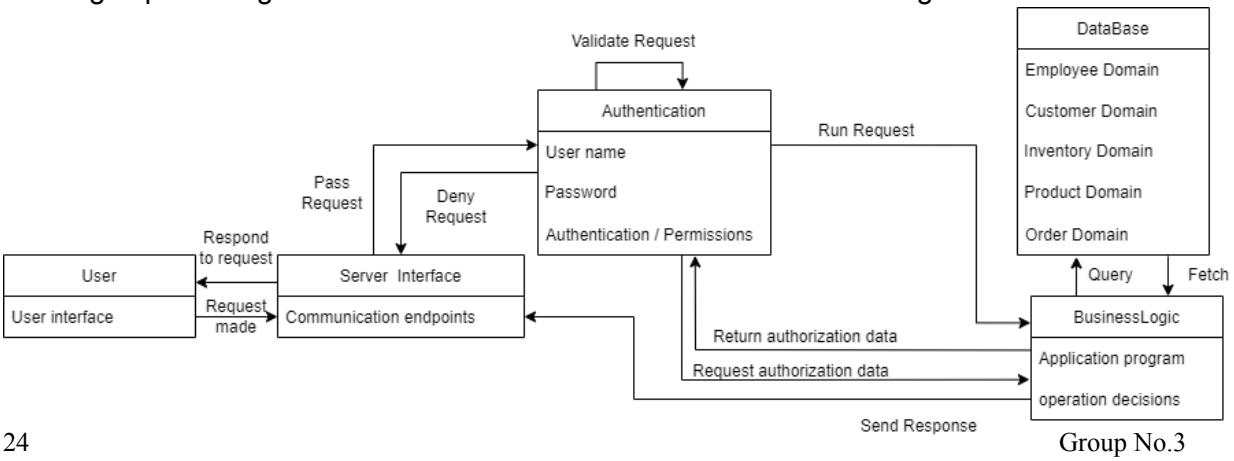

Figure 3.2 Refine System Domain Model

The UML sequence diagram shown in Figure 3.3 illustrates the order of events which occur as the various elements defined in the refined Authentication module take in and process a request. This sequence diagram will show the operations from a viewpoint as it were inside the Authentication module.

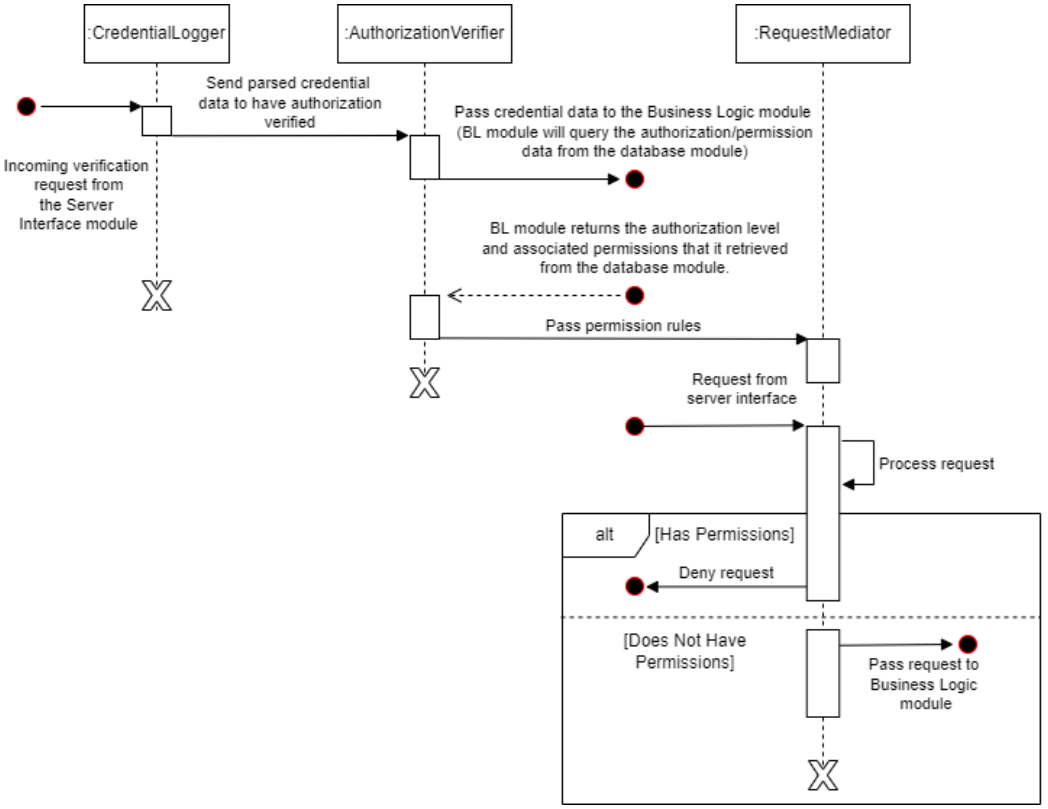

Figure 3.3 Sequence diagram illustrating the process of taking in and processing a request from inside the Authentication module

The UML sequence diagram shown in Figure 3.4 represents the same sequence which occurred in Figure 3.2, from an external view, where all the elements from the refined domain model are present from a high level view.

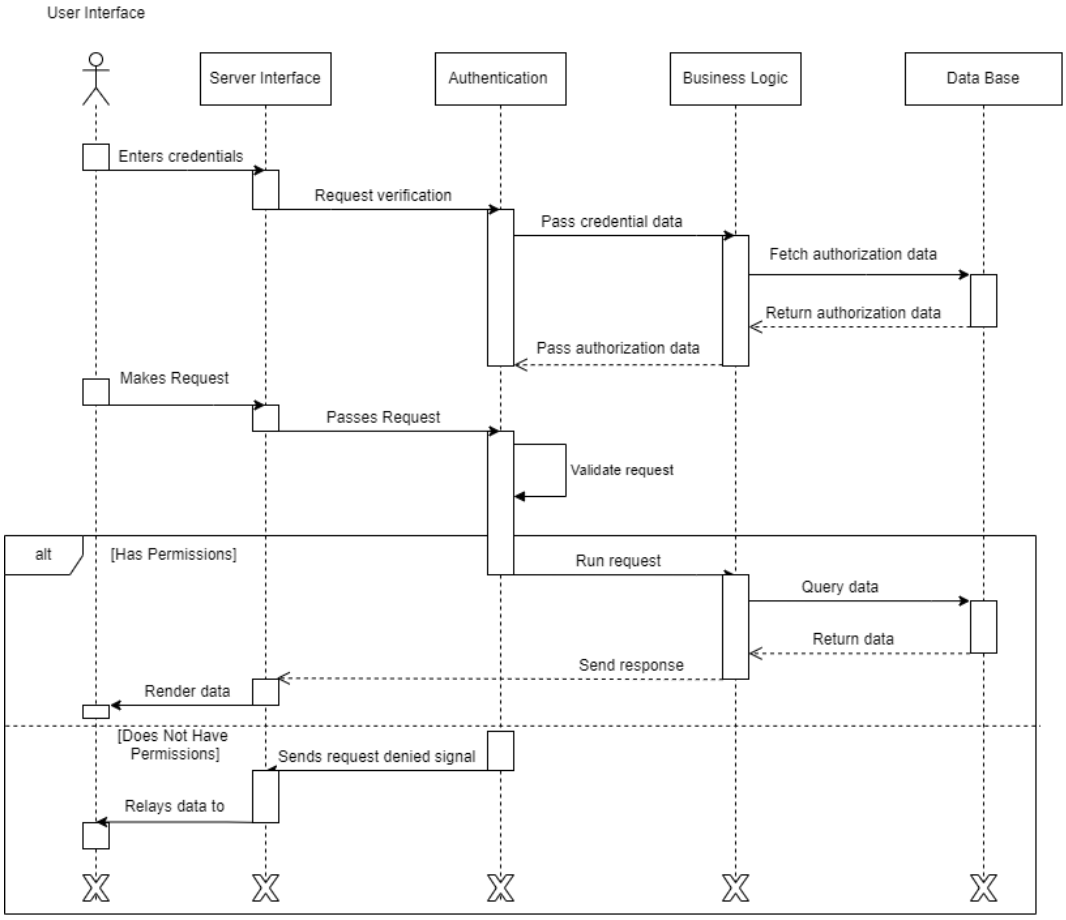

Figure 3.4 Sequence diagram illustrating the process of taking in and processing a request from a system level

3.1.6 Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
		CON-5	No relevant decisions made
		CON-7	No relevant decisions made
QA-2			No relevant decisions made
		QA-3	This quality attribute was completely addressed through the introduction of the Authorize Actors security tactic. The

			Authentication element of the system domain was refined to implement an authorization forwarding component in order to filter server-side accessibility to the user. By defining these refinements using updated domain models, specified domain models, and sequence diagrams, a concrete foundation has been built on how this quality attribute will be incorporated into the development of this system.
	QA-6		No relevant decisions made
		CRN-1	No relevant decisions made
		CRN-3	No relevant decisions made
		CRN-4	No relevant decisions made
		UC-3	No relevant decisions made
		UC-4	No relevant decisions made
		UC-5	No relevant decisions made
		UC-9	No relevant decisions made
		UC-10	No relevant decisions made
		UC-11	No relevant decisions made