University of Bahrain
Department of Computer Science
ITSE302 – 2nd Semester 2022-2023
Section 01

Virtual Garage System

Table of Contents:

Business Case

The virtual garage system is created to provide several services for all vehicles from an online platform. Additionally, it allows the customer to register their needs at the comfort of their homes. All-whilst avoiding traffic, weather conditions, inconveniences, making the system a time and cost-efficient tool to run your car's maintenance and service.

The system is made up of 5 use cases which fall under 4 categories: registration in relation to user account, reservation, service, and payment. The virtual garage system starts registering the user's account. If the customer is a first-time user, they would need to create an account but if it were a registered user, they would sign in with their exclusive userID and password. The system has an easy and simple structure to allow a smooth flow for both the user and the staff. It is opted for flexibility where users can update and make changes to their account information.

Requirements Description

The user is then going to book an appointment with reference to location, time, vehicle type, date availability to their desired service of choice. Subsequently, a check-up will be performed to inspect the condition of the vehicle, and this would require a downpayment which charges 15% of the total price of service. The user is given a wide range of services to choose from such as paint job, tint application, tire change and alignment, oil change, body makeover, filter changes, brake inspections, etc.

After the service has been confirmed and finalized, the user is given the final payment bill. The receipt of the full-service payment will be displayed excluding the downpayment price. The user will be given a choice of whether they want to print or send a copy of the receipt to their email address.

A complementary service of useful information is also provided by the system to the users, a self-service page is available to the users. This page has a list of problems that could be a potential issue a customer is facing with their vehicle alongside a solution on how the customer could fix this problem.

The most effective method to face a problem in an online service is customer service. Therefore, communication is made possible to users by customer support –a live chat between the customer and the support staff- in case of any latent complications and inconveniences the customer is facing, the customer would not feel lost or hopeless when they are to communicate with a staff member. Not only does this help resolve the issue but it creates significant impact on the customer's relationship with company. Finally, any information that needs to be noted down and updated is stored in a permanent database storage where the designated employee can add the changes to. shown in Figure 5.1 represents the UML case diagram for virtual garage system.

3. System Requirements

3.1 UML Use Case Diagram

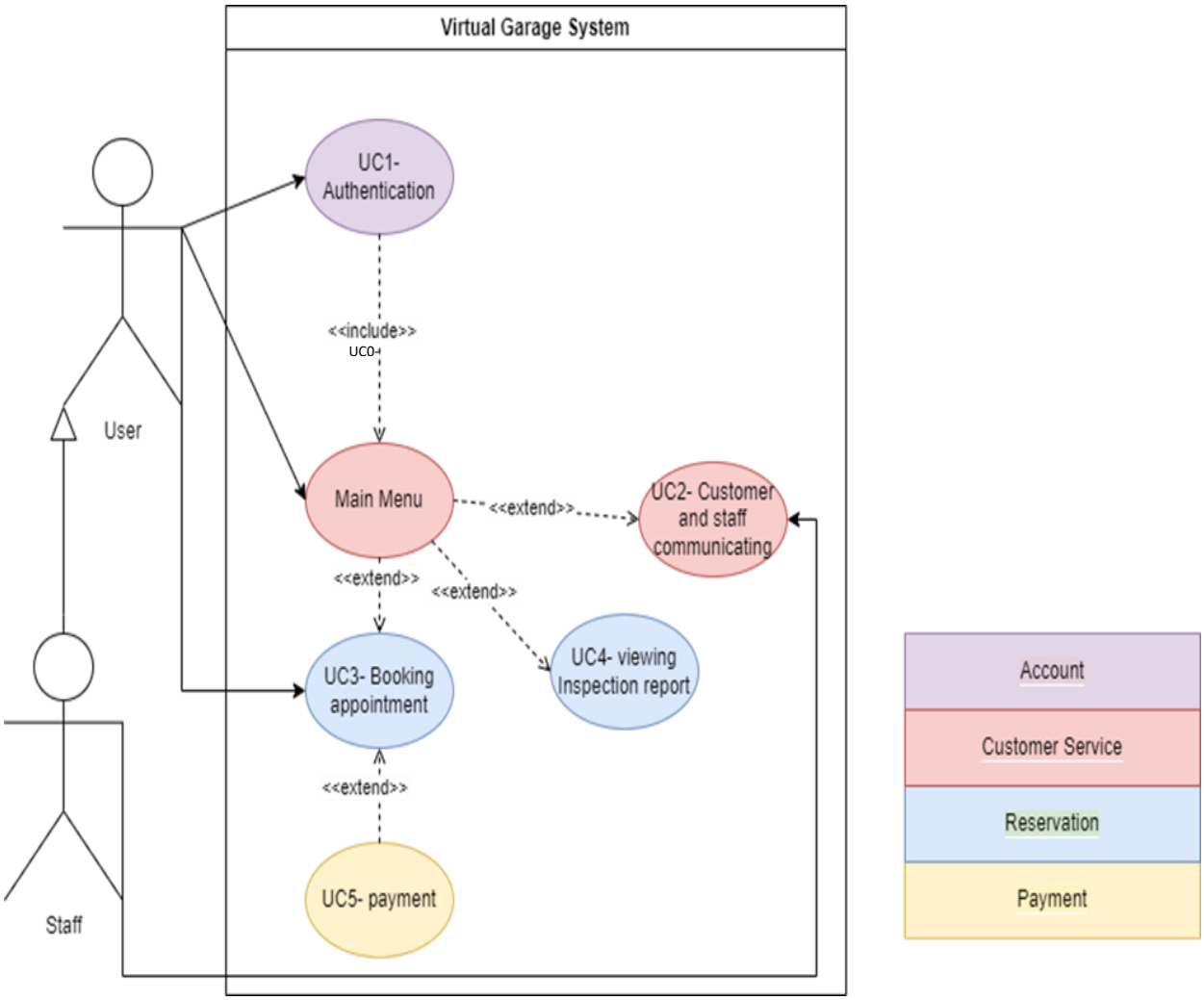


Figure 1.0 UML **USE CASE DIAGRAM** FOR VIRTUAL GARAGE SYSTEM

3.2 Use Case Descriptions

Use Case Name	Description
UC-1: Authentication	Initially, the customer is required to create an account if the customer is a first-time user in the system. To Create an account, the user must have a distinctive userID, a preferred strong password, email address, and phone number. The customer logs into the system using their registered information, after which they can select their service of choice. The system gives customer flexibility by allowing the user to update their data, changes are even possible for contact information like phone number and email.
UC-2: Customer and staff communicating	A page is dedicated for customer self-service. The user will be able to see specific problems with solutions on how to fix the issue. Customers are helped through a live chat with staff members to avoid inconveniences and resolve any difficulty they might be facing.
UC-3: Book appointment	The customer can specify the appropriate time and date to make a reservation for his vehicle to check the vehicle or to perform other services on it. After completing the vehicle inspection and receiving the report, the customer can choose the type of service he/she needs through several options provided by the system.
UC-4: Viewing Inspection report	After conducting the vehicle inspection, the staff sends the inspection report to the customer by uploading it to the system, including the exact inspection details. After completing the vehicle inspection and receiving the report, the customer can choose the type of service he/she needs through several options provided by the system.
UC-5: Payment	The user is asked to pay a downpayment of 15% of the total service provided. The user will be provided with a payment portal total amount of the car service, excluding the downpayment once the service is confirmed and done. Once the user has paid, the system displays their receipt with a print or emailing option.

Table 1.0 Description of each use case

3.3 Quality Attribute Scenarios

ID	Quality Attribute	Scenario	Related Use Case	Priority
QA-1	**Performance**	When the user requests a normal operation, the system should not take more than 4 seconds to perform the transaction.	**All**	**High** (Core requirement) (M, H)
QA-2	**Security**	A fired employee attempts to access and publish a list of clients' financial details to destroy the app's reputation. The system maintains immediate responses to any anonymous hack and prevents hackers from accessing the data.	**All**	**High** (Core requirement) (H, H)
QA-3	**Availability**	When the system crashes at runtime, the system will notify the operator and then jump to the backup application server or backup database server and continues to operate without any downtime.	**All**	**High** (Core requirement) (H, H)
QA-4	**Usability**	The customer can easily navigate through the system during runtime and use it productively after ten minutes of trial, because the system's interface is easy to use.	**All**	**High** (Core requirement) (H, M)
QA-5	**Testability**	The unit tester completes a code unit during development and performs a test sequence whose results are captured and that gives 85% path coverage within 3 hours of testing.	**All**	**High** (Core requirement) (M, H)

Table 2.0 Breakdown of the quality attributes with the scenarios

3.3.1 Utility tree

A utility tree is designed to enhance the quality attributes and its reference to each other in order of importance. Prioritizing them by labelling their significance as high or medium consequently.

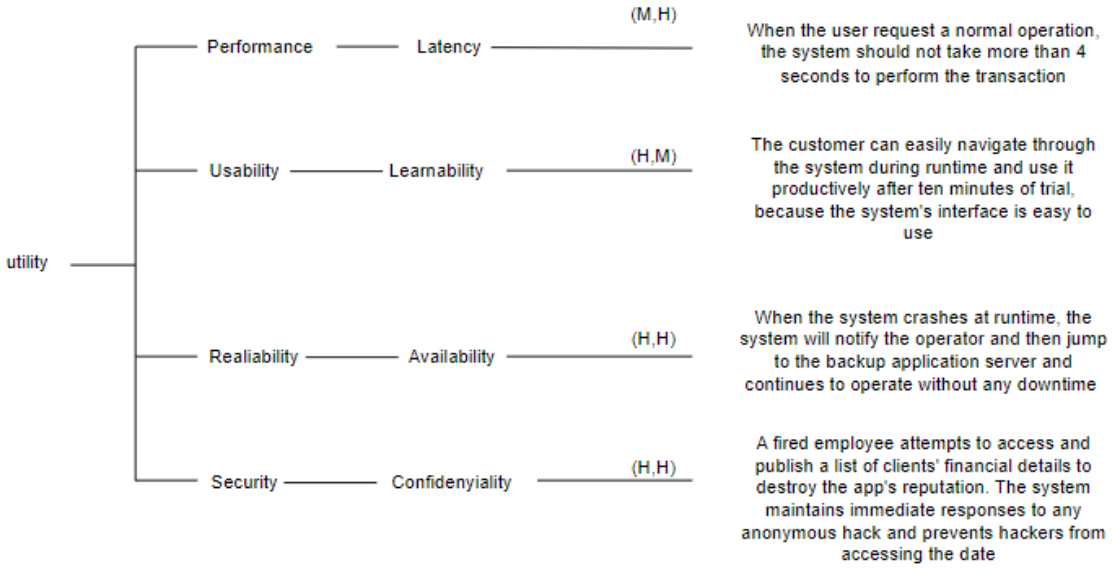


Figure 2.0 **Utility tree** representing the importance of the QA and its functions

3.3.2 Six-part diagrams

The six-part diagram demonstrates six-part scenario to test the quality of the attributes. This scenario consists of source of stimulus, the stimulus itself, the artifact, response to it and measured time of response.

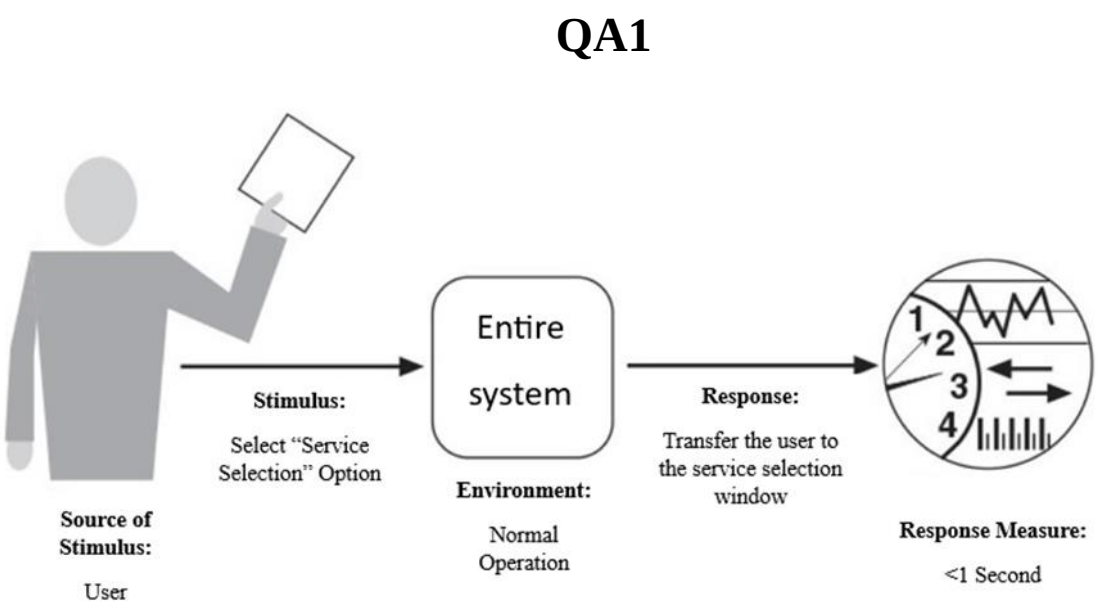


Figure 3.0 QA-1

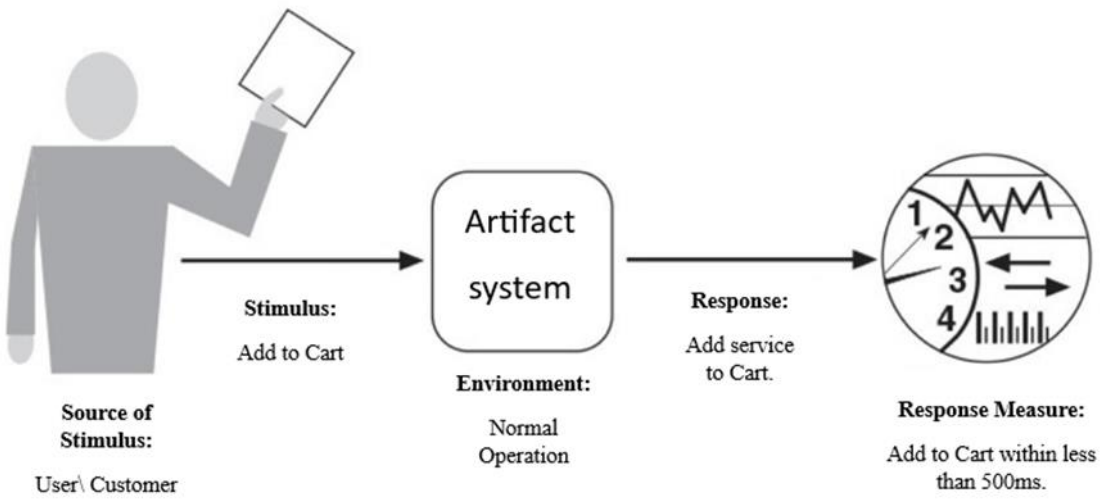


Figure 3.1 QA-2

QA3

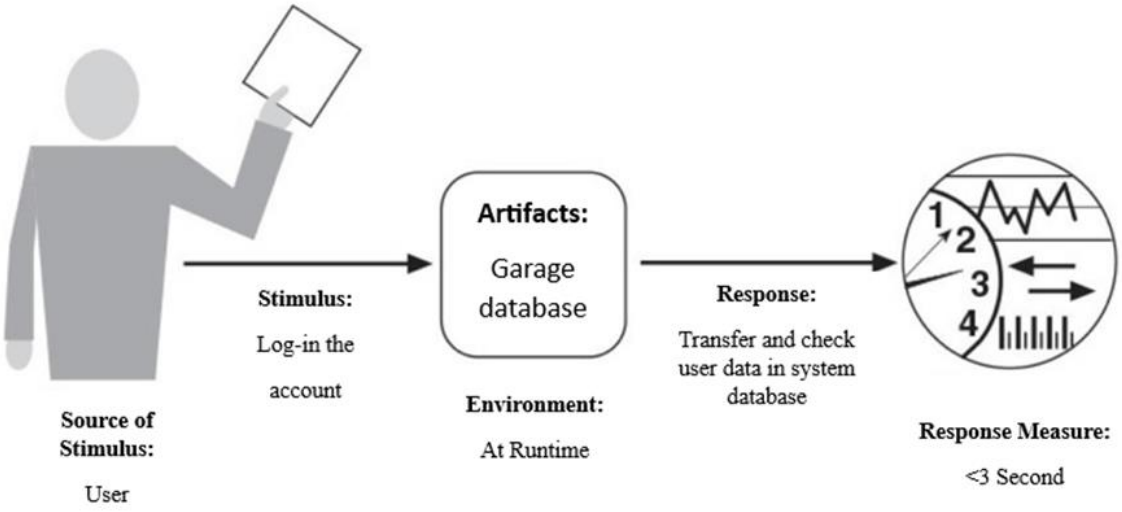


Figure 3.2 QA-3

QA4

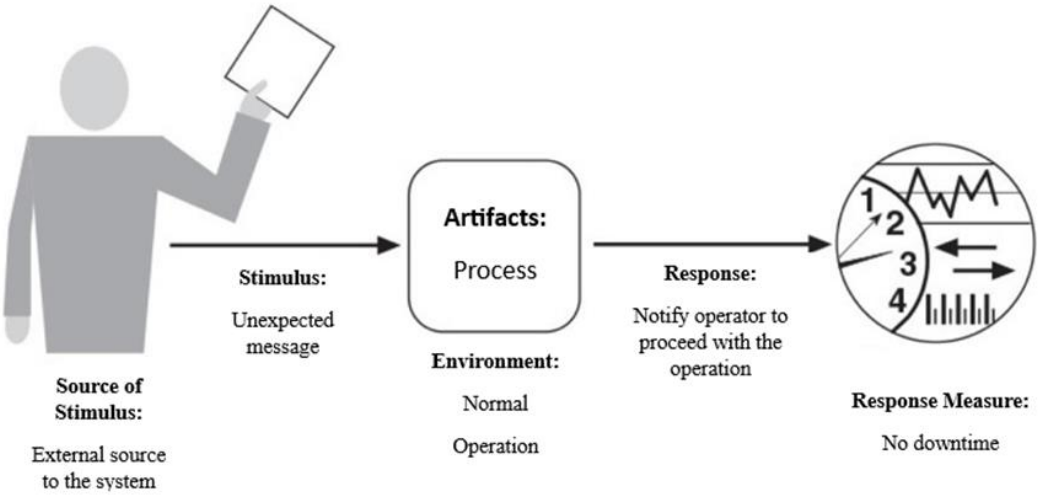


Figure 3.3 QA-4

4. Constraints

ID	CONSTRAINTS
CON-1	The system must run on IOS, Android and Windows platform.
CON-2	The system must only provide services to Customers who are inside Bahrain.
CON-3	Payment data from the last 40 days must be stored and the customer can view it.
CON-4	The Java programming language will be used to design the system, then the execution will be under windows and Linux.
CON-5	Within six months, the system must be completed, evaluated, and implemented.
CON-6	A minimum of 100 simultaneous users must be supported.
CON-7	An existing relational database server must be used. This server cannot be used for other purposes than hosting this database.

Table 3.0 The seven constraints as part of the architectural drivers

5. Concerns

ID	Concerns
CRN-1	Establishing an overall initial system structure.
CRN-2	Leverage the team's knowledge about Java technologies, including Spring, JSF, Swing, Hibernate, Java Web Start and JMS frameworks, and the Java language.
CRN-3	Allocate work to members of the development team.
CRN-4	The system must only provide services to Customers who have membership.

Table 4.0 Architectural concerns as part of the architectural drivers

6. The design process

We now ready to make the leap from the world of requirements and business concerns to the world of design. This is perhaps the most important job for an architect—translating requirements into design decisions. Of course, many other decisions and duties are important, but this is the core of what it means to be an architect: making design decisions with far-reaching consequences.

6.1 ADD Step 1: Review Inputs

The first step of the ADD method involves reviewing the inputs and identifying which requirements will be considered as drivers (i.e., which will be included in which requirements will be considered as drivers (i.e., which will be included in the design backlog).

Design purpose

This is a greenfield system from a mature domain. the purpose is to sufficiently detailed design to support the construction of the system.

Primary Functional Requirement

From the use cases presented in the project report the primary ones were determined to be:

Functions
Authentication
Book appointment
Payment

- All this functions directly supports the core business of the project.

Quality Attribute Scenarios:

The scenarios were described in Quality Attributes part. They have now been prioritized as follows:

Scenario ID	Importance to the customer	Difficulty of implementation according to the architect
QA-1	Medium	High
QA-2	High	Medium
QA-3	High	High
QA-4	High	High

- From the list QA-1, QA-2, QA-3, and QA-4 are selected as drivers.

Constraints:

- From the constraints we choose CON#2 and CON#4 as drivers.

Architectural Concerns:

- From the Architectural concerns we choose CNR#1, CNR#2, and CNR#3 as drivers.

6.2 Iteration 1: Establishing an overall system structure

Establishing an overall system structure This section presents the results of the activities that are performed in each of the steps of ADD in the first iteration of the design process.

6.2.1 STEP 2: Establish iteration goal by selecting drivers

Establish iteration goal by selecting drivers This is the first iteration in the design of a greenfield system, so the iteration goal is to achieve the architectural concern CNR#1 establishing an overall system structure. Although this iteration is driven by a general architectural concern, the architect must keep in mind all of the drivers that may influence the general structure of the system. In particular, the architect must be mindful of the following:

- QA-1: Performance.
- QA-2: Security.
- QA-3: Availability.
- QA-4: Usability.

CON-2: Leverage the team's knowledge about Java technologies, including Spring, JSF, Swing, Hibernate, Java Web Start and JMS frameworks, and the Java language.

CNR-3: Allocate work to members of the development team.

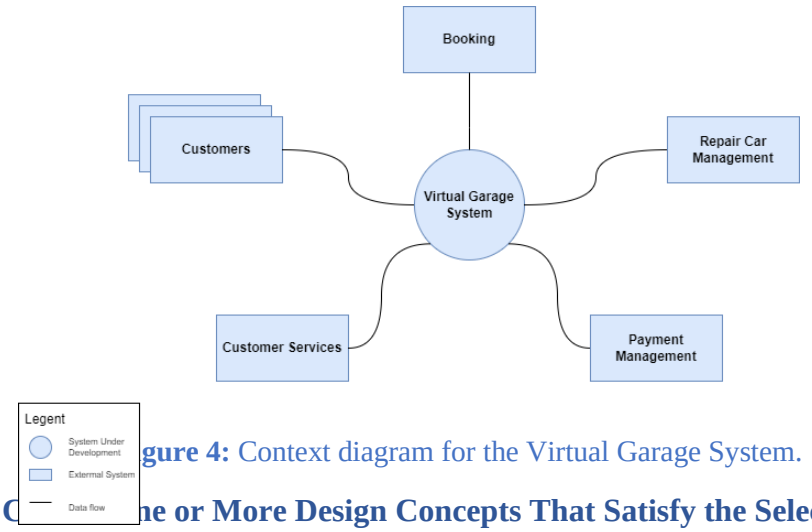
CON-1: The system must run on IOS, Android and Windows platform.

CON-2: The system must only provide services to Customers who are inside Bahrain.

CON-4: The system must only provide services to Customers who have membership

6.2.2 STEP 3: Choose One or More Elements of the System to Refine

Choose One or More Elements of the System to Refine This is a greenfield development effort, so in this case the element to refine is the entire Virtual Garage system, which is shown in **Figure 4**. In this case, refinement is performed through decomposition.



gure 4: Context diagram for the Virtual Garage System.

6.2.3 STEP 4: C ne or More Design Concepts That Satisfy the Selected Drivers

Choose One or More Design Concepts That Satisfy the Selected Drivers In this initial iteration, given the goal of structuring the entire system, design concepts are selected according to the roadmap presented in **Figure 5.** The following points summarizes the selection of design decisions.

Logic/ System Structure (Reference Architecture): Web application **(Figure 6)**

- The system requires portability of the user interface as CON-1 describe that it must run be in different platforms.
- The system needs to be accessible over the Internet.
- The system should use a minimum of client-side resources because it is safer for this system for the resources to be in the server side.
- rich user interface is not needed. it is not required to deploy the application by installing anything on the client machine because everything could be accessible through the browser.

Discarded Alternative 1: Rich client application

- it is not required to deploy the application by installing anything on the client machine because everything could be accessible through the browser.

- it is more complicated to update and deploy it.

Discarded Alternative 2: Mobile Application

- executed on handheld device and works in collaboration and this type of device was not considered for accessing the system.

Physical Structure (Deployment Pattern)

Three – tier deployment pattern. Since the system must be accessed from web browser and an existing database server must also be used (CON-7), a three tier deployment is appropriate.

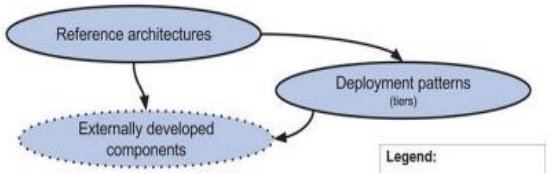
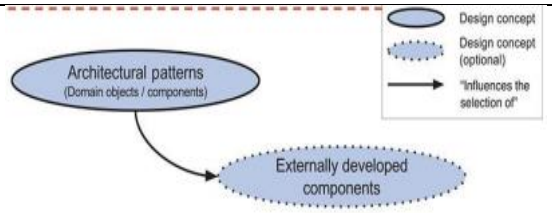
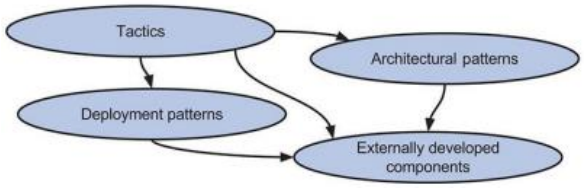
Iteration	Goal	Design Concept
1	Establishing an initial overall system structure.	
2	Identify structures to support primary functionality	
3.. n	Refining previously created structure to fully address the reaming drivers.	

Figure 5: roadmap for Virtual Garage System.

6.2.4 STEP 5 Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces.

Practicing Code Reuse & Using Web-Application Frameworks:

Practicing code reuse and using web application frameworks can greatly improve both productivity and time market, reusing extremally developed components ca allows an to reap the above benefits.

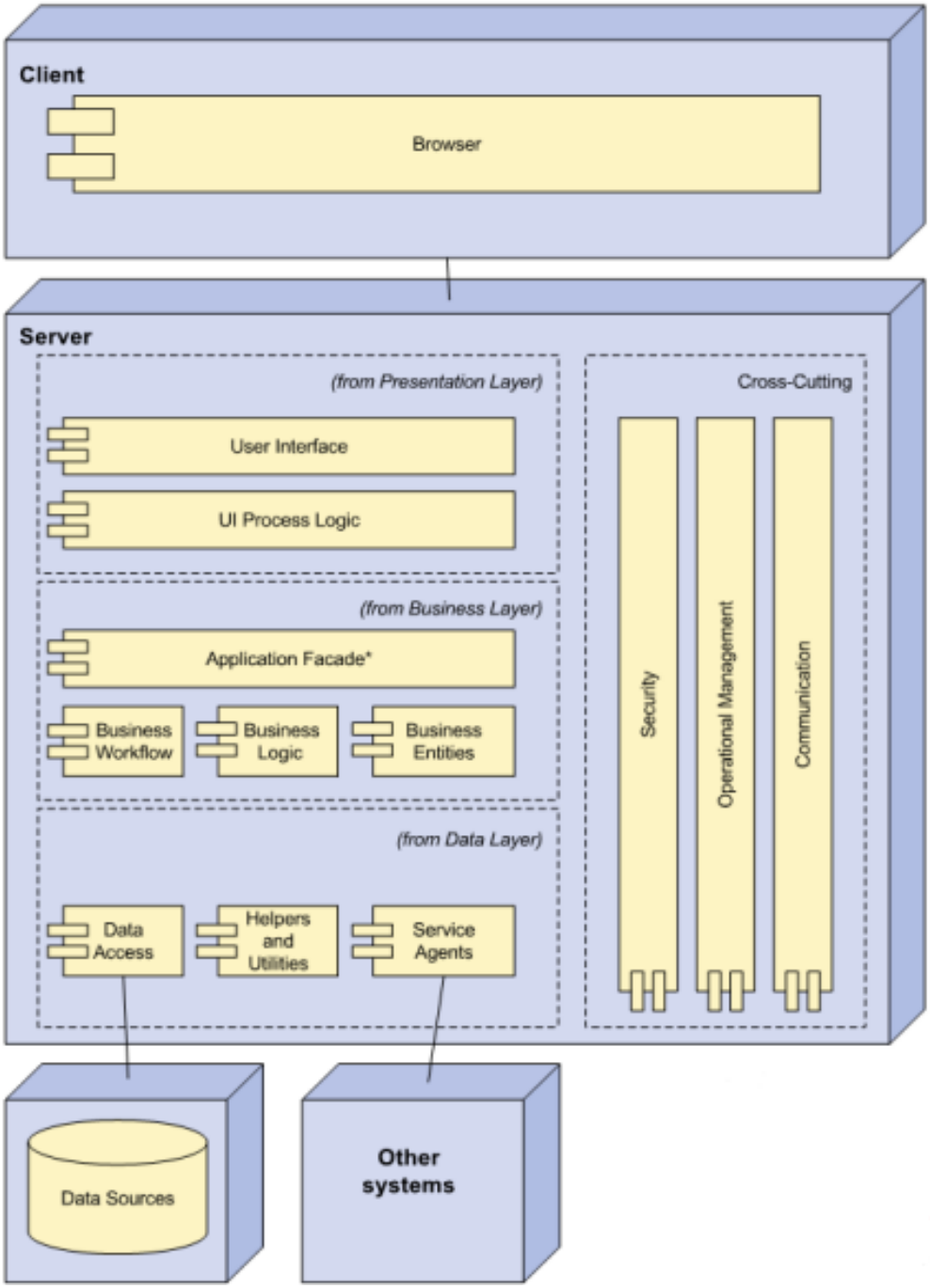
The results of these instantiation decisions are recorded in the next step. In this initial iteration, it is typically too early to precisely define functionality and interfaces. In the next iteration, which is dedicated to defining functionality in more detail, interfaces will begin to be defined.

STEP 6: Sketch Views and Record Design Decisions.

The diagram in **Figure 6** shows the sketch of a module view of the reference architectures that were selected for the web applications. These have now been adapted according to the design decisions we have made.

This web application is typically initiated from a web browser that communicates with a server using the HTTP protocol. The bulk of the application resides on the server, and its architecture is typically composed of three layers: the presentation, business, and data layers. The presentation layer contains modules that are responsible for managing user interaction.

Figure 6: web application reference architecture.



The following table summarizes the responsibilities of the components present in this reference architecture:

Component	Responsibility
Browser	A web browser running on the client machine.
User Interface	These components are responsible for receiver Interactions and presenting information to the users. They contain Ul elements such as buttons and text fields.
UI process logic	These components are responsible for managing the control flow of the application's use cases. They are responsible for other aspects such as data validation, orchestrating interactions with the business logic, and providing data coming from the business layer to the user interface components.
Application facade	This component is optional. It provides a simplified interface (a facade) to the business logic components.
Business workflow	These components are responsible for managing (long-running) business processes, which may involve the execution of multiple use cases
Business logic	These components are responsible for retrieving and processing application data and applying business rules on this data.
Business Entities	These components represent the entities from the business domain and their associated business logic.
Data access	These components encapsulate persistence mechanisms and provide common operations used to retrieve and store Information.
Helpers and Utilities	These components contain functionality common to other modules in the data layer but not specific to any of them.
Service Agents	These components abstract communication mechanisms used to transfer data to external services.
Security	These components include cross-cutting functional it that handles security aspects such as authorization and authentication.
Operation management	These components include cross-cutting functionality such as exception management, logging, and instrumentation and validation.
Communication	These components include cross-cutting functionality that handles communication mechanisms across layers and physical tiers.

The deployment diagram in **Figure 7** sketches an allocation view that illustrates where the components associated with the modules in the previous diagram will be deployed.

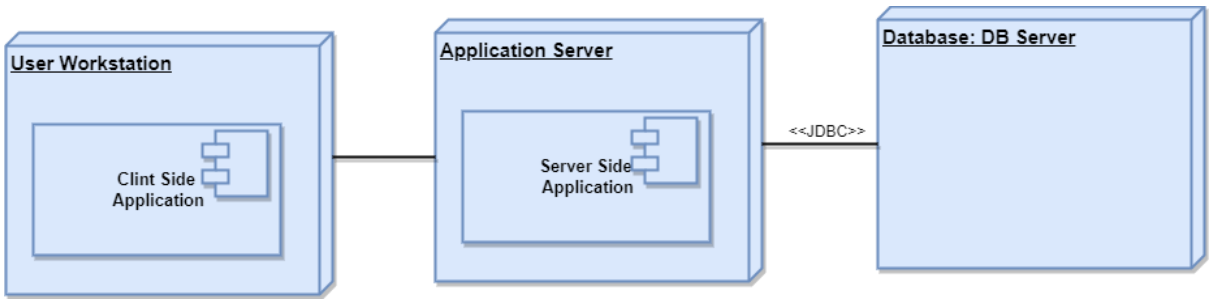


Figure 7: Initial deployment diagram for the Virtual Garage System (Key: UML).

The responsibilities of the elements are summarized here:

Element	Responsibility
User Workstation	The user's PC, which hosts the client-side logic of the application.
Application Server	The server that hosts server-side logic of the application and serves web pages.
Database Server	The server that hosts the legacy relational database.
Management Control	Manager side that controls and updates the system.

6.2.6 STEP 7 Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose.

Not Addressed	Partially Addressed	Completely Addressed	Design Decision Made During the Iteration
	UC-1		Selected reference architecture establishes the modules that will support this functionality.
	UC-3		Selected reference architecture establishes the modules that will support this functionality.
	UC-5		Selected reference architecture establishes the modules that will support this functionality.
QA-1			No relevant decision made, as it is necessary to identify the elements that participate in the use case that is associated with the scenario.
	QA-4		The details of this components and its interfaces have not been defined yet
QA-3			No relevant decision made, as it is necessary to identify the elements that participate in the use case that is associated with the scenario
	CON#6		Structuring the system using 3 tiers will allow multiple clients to connect to the application server. Decision regarding concurrent access have not been made yet.
		CON#4	Use of java and java script to build the website, Since the web application is being programmed, this supports execution Under Windows and Linux.
		CON#7	Physically structure the application using 3-tier deployment pattern and isolate the database by providing database access

			components in the data layer of the application server.
CON#3			No relevant decision made.
CON#6			No relevant decision made.
		CRN#1	Selection of reference architectures and deployment pattern.
		CRN#4	No relevant decision made.
	QA-3		Identification of the elements derived from the deployment pattern that will need to be replicated.

6.3 ITERATION 2 Identifying Structures to Support Primary Functionality

This section presents the results of the activities that are performed in each of the steps of ADD in the second iteration of the design process for the Virtual Garage system. In this iteration, we move from the generic and coarse-grained descriptions of functionality used in iteration 1 to more detailed decisions that will drive implementation and hence the formation of development teams.

This movement from the generic to the specific is intentional and built into the ADD method. We cannot design everything up front, so we need to be disciplined about which decisions we make, and when, to ensure that the design is done in a systematic way, addressing the biggest risks first and moving from there to ever finer details. Our goal for the first iteration was to establish an overall system structure. Now that this goal has been met, our new goal for this second iteration is to reason about the units of implementation, which affect team formation, interfaces, and the means by which development tasks may be distributed, outsourced, and implemented in sprints.

6.3.1 STEP 2 Establish Iteration Goal by Selecting Drivers.

The goal of this iteration is to address the general architectural concern of identifying structures to support primary functionality in this second iteration the architect considers the system's primary use cases:

- **UC-1:** Authentication
- **UC-3**: Book appointment
- **UC-5:** Payment

6.3.2 STEP 3: Choose One or More Elements of the System to Refine.

The elements that will be refined in this iteration are the modules located in the different layers defined by the reference architecture from the previous iteration. In general, the support of functionality in this system requires the collaboration of components associated with modules that are located in the different layers.

6.3.3 STEP 4: Choose One or More Design Concepts That Satisfy the Selected Drivers.

In this iteration, several design concepts—in this case, architectural design patterns—are selected from the book Pattern Oriented Software Architecture, Volume 4. The following points summarizes the design decisions:

Create a Domain Model for the Application:

Before starting a functional decomposition, it is necessary to create an initial domain model for the system, identifying the major entities in the domain, along with their relationships. There are no good alterative. A domain model must eventually be created, or it will emerge in a suboptimal fashion, leading to an ad hoc architecture that is hard to understand and maintain.

Identify Domain Objects that Map to Functional Requirements:

Each distinct functional element of the application needs to be encapsulated in a self-contained building block—a domain object. One possible alternative is to not consider domain objects and instead directly decompose layers into modules, but this increases the risk of not considering a requirement.

Decompose Domain Objects into General and Specialized Components:

Domain objects represent complete sets of functionalities, but this functionality is supported by finer-grained elements located within the layers. The “components” in this pattern are what we have referred to as modules.

6.3.4 STEP 5: Instantiate Architectural Elements, Allocate Responsibilities and Define Interfaces.

The instantiation design decisions made in this iteration are summarized in the following table:

Design Decision and Location	Rational
Create only an initial domain model	The entities that participate in the primary use cases need to be identified and modelled but only an initial domain model is created, to accelerate this phase of design.

Map the system use cases to domain object	An initial identification of domain objects can be made by analysing the system's use cases. To address CRN#3, domain objects are identified for all of the use cases.
Decompose the domain objects across the layers to identify layer-specific modules with an explicit interface	This technique ensures that modules that support all of the functionalities are identified. The architect will perform this task just for the primary use cases. This allows another team member to identify the rest of the modules, thereby allocating work among team members. Having established the set of modules, the architect realizes the need to test these modules, so a new architectural concern is identified here: CRN#4: A majority of modules shall be unit tested. Only "a majority of modules" are covered by this concern because the modules that implement user interface functionality are difficult to test independently.
Connect components associated with modules using Spring	This framework uses an inversion of control approach that allows different aspects to be supported and the modules to be unit-tested (CRN#4).
Associate frameworks with a module in the data layer	ORM mapping is encapsulated in the modules that are contained in the data layer. The Hibernate framework previously selected is associated with these modules.

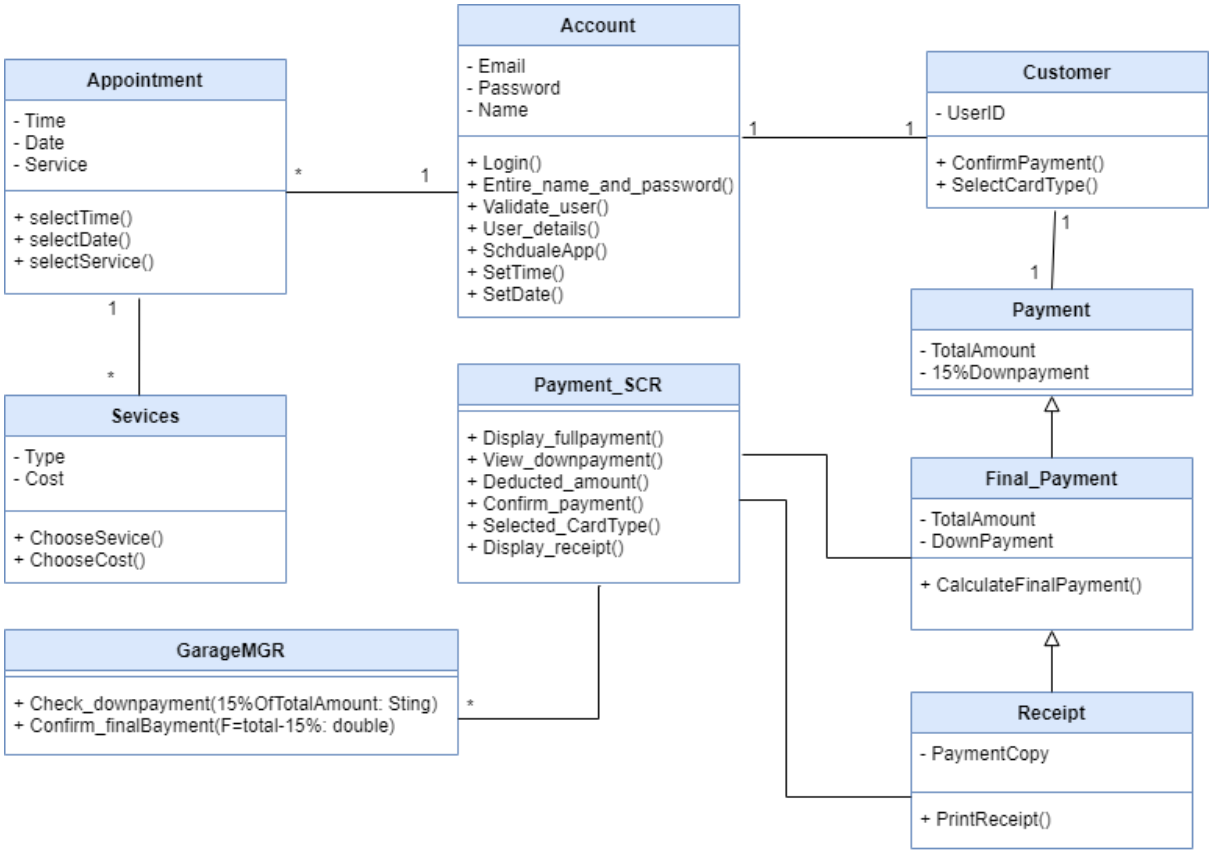
While the structures and interfaces are identified in this step of the method, they are captured in the next step.

6.3.5 STEP 6: Sketch Views and Record Design Decisions

As a result of the decisions made in step 5, several diagrams are created.

- Figure shows an initial domain model for the system.
- Figure shows the domain objects that are instantiated for the use case.
- Figure shows a sketch of a module view with modules that are derived. From the business objects and associated with the primary use cases.

Figure 8: Initial domain model (Key: UML)



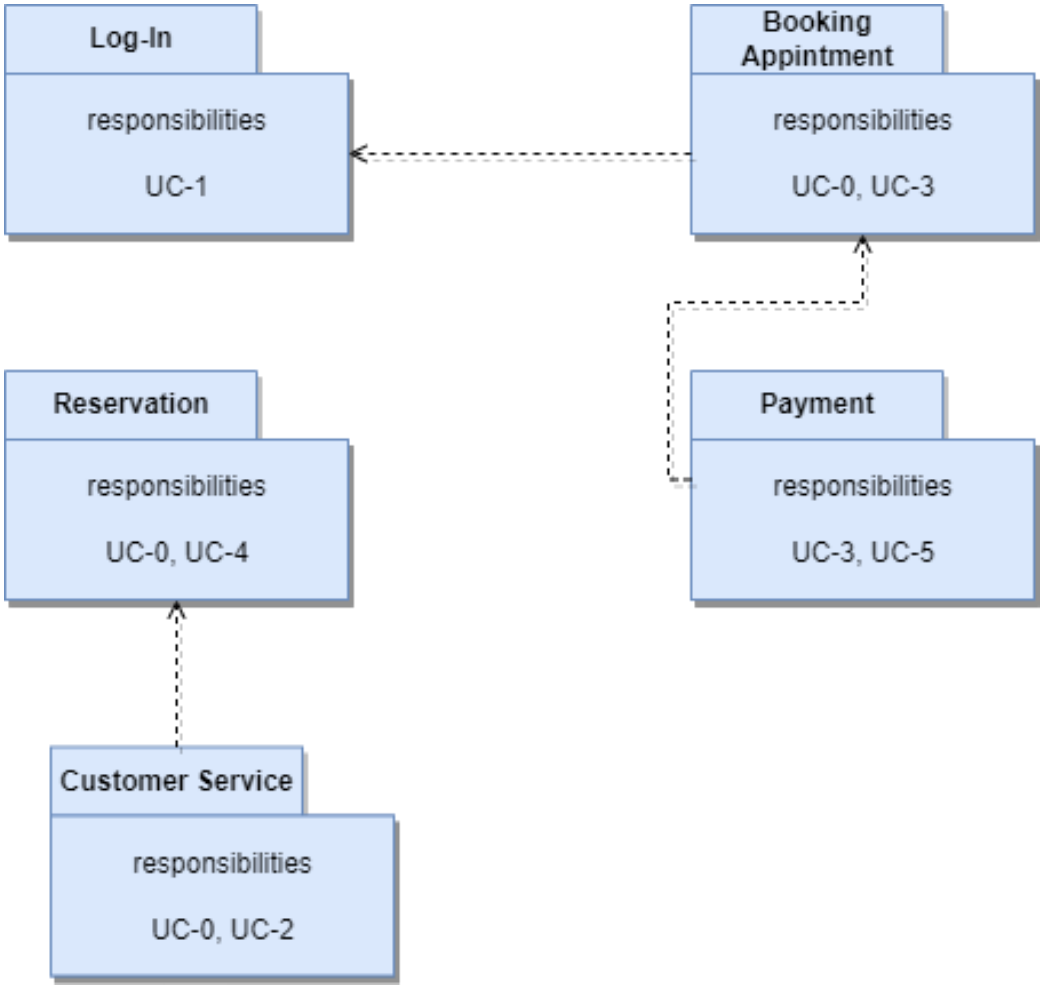


Figure 9: Domain objects associated with the use case model (Key: UML).

Figure 10: shows a sketch of a module view with modules that are derived from the business objects and associated with the primary use cases.

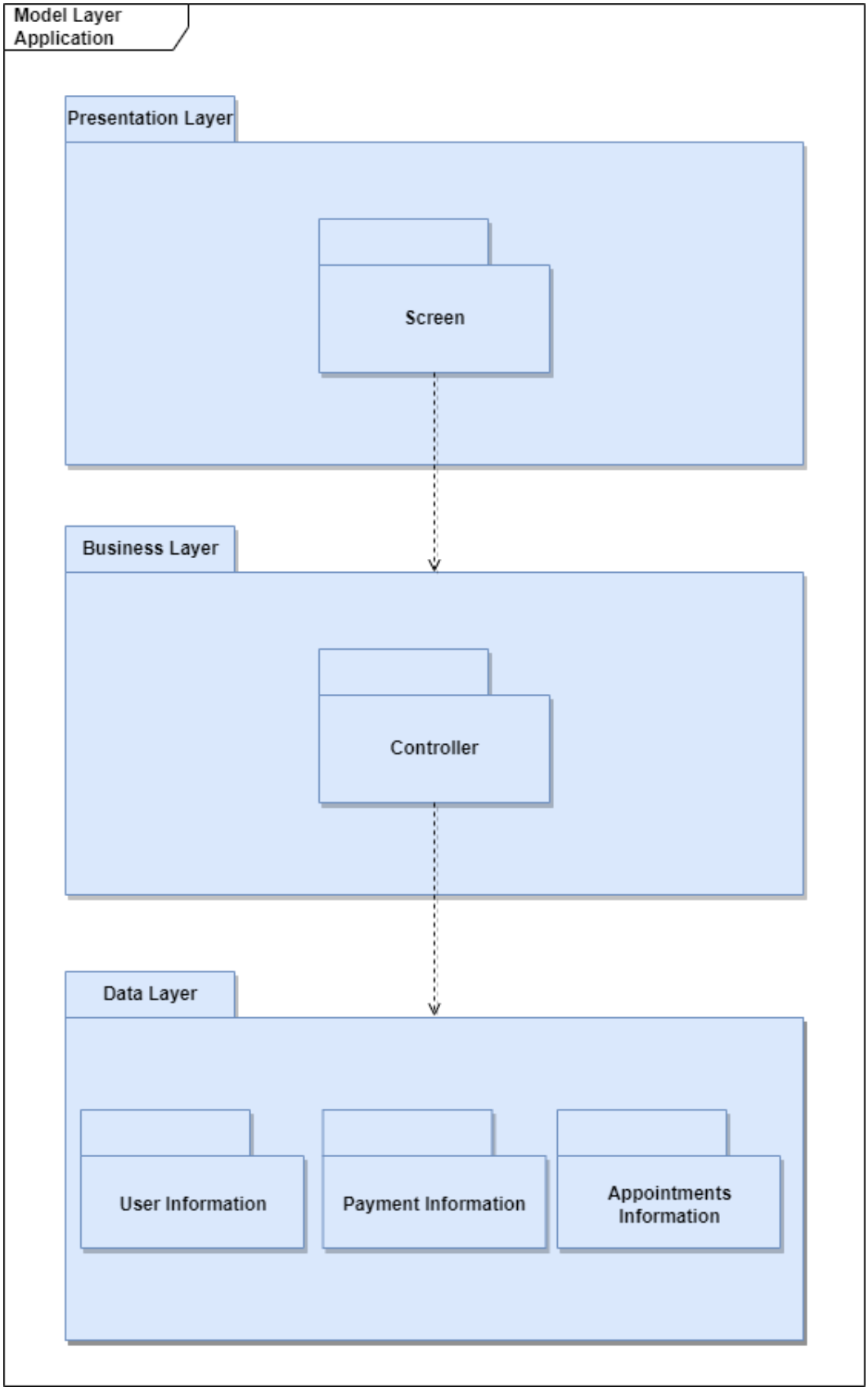


Figure 10: Modules that support the primary use cases (Key: UML).

The responsibilities for the elements identified in **Figure 10** are summarized in the table below:

Element	Responsibility
Screen	Display the network representation and update it when events are received.
Controller	Responsible for providing an information to the presentation layer for displaying the network representation.
User information	Contains the user information.
Payment information	Contains the payment information for each user and the payment status.
Appointments information	Contains the appointment assigned for each user.

The following sequence diagrams for **Authentication, Book appointment** and **Payment** were created in the previous step of the method to define interfaces:

UC-1: Authentication:

Figure 11 shows an initial sequence diagram for (Authentication) and there is a scenario beside that shows the interactions:

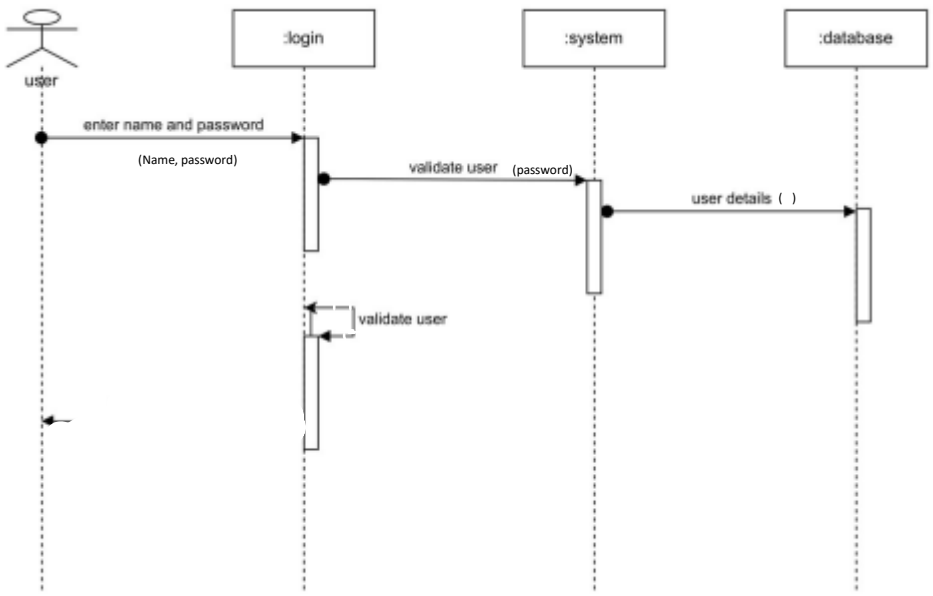


Figure 11: Sequence diagram for UC-1 (Authentication) (Key UML)

Authentication Use Case Scenario:

- Start the application. Ahmed asked to enter the username, e-mail, mobile number, and password.
- Ahmed enters the username, e-mail, mobile number, and password.
- System does authentication.
- Main screen is displayed.

From the interactions identified in the sequence diagram, initial methods for the interfaces of the interacting elements can be identified:

Method Name	Description
Element: Screen	
enter_name_and_password (name, password)	It will allow user to input his/her name and password of his/her account
Element: Controller	
validate_user (password)	It will get the user password from the customer to validate it.
Element: User Information	
User_details ()	This method is used to look up for the password in the system database to validate it, and to modify the account and store it in the database.

UC-3: Book appointment:

Figure 12 shows an initial sequence diagram for money transaction and there is a scenario beside that shows the interactions.

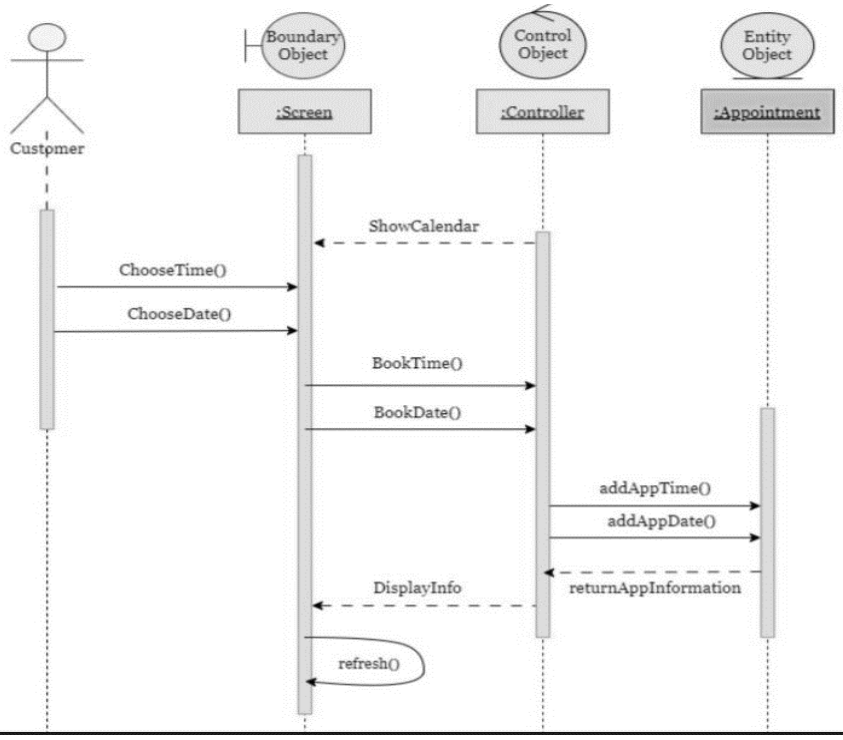


Figure 12, Sequence diagram for use case UC-3 (Book appointment) (Key UML)

Book appointment Use Case Scenario:

- The user clicks on Schedule an appointment option.
- The system display for the user options (Schedule an appointment, Choose service).
- The user clicks on show the calendar.
- System will display the calendar for the user with all available days and timing.
- User chooses available date and time for the appointment.
- The system will display for the user that the appointment booked successfully.
- The system will return appointment information for the user on the screen.

From the interactions identified in the sequence diagram, initial methods for the interfaces of the interacting elements can be identified:

Method Name	Description
Element: Screen	
ChooseTime ()	It allows user to choose a time from the calendar for the appointment.
ChooseDate ()	It allows user to choose a date from the calendar for the appointment.
Element: Controller	
BookTime ()	It will get the appointment date from the customer.
BookDate ()	It will get the appointment date from the customer.
Element: Appointments Information	
AddAppTime ()	It will add the appointment time into the appointment database.
AddAppDate ()	It will add the appointment date into the appointment database.

UC-5: Payment:

Figure 13 shows an initial sequence diagram for money transaction and there is a scenario beside that shows the interactions:

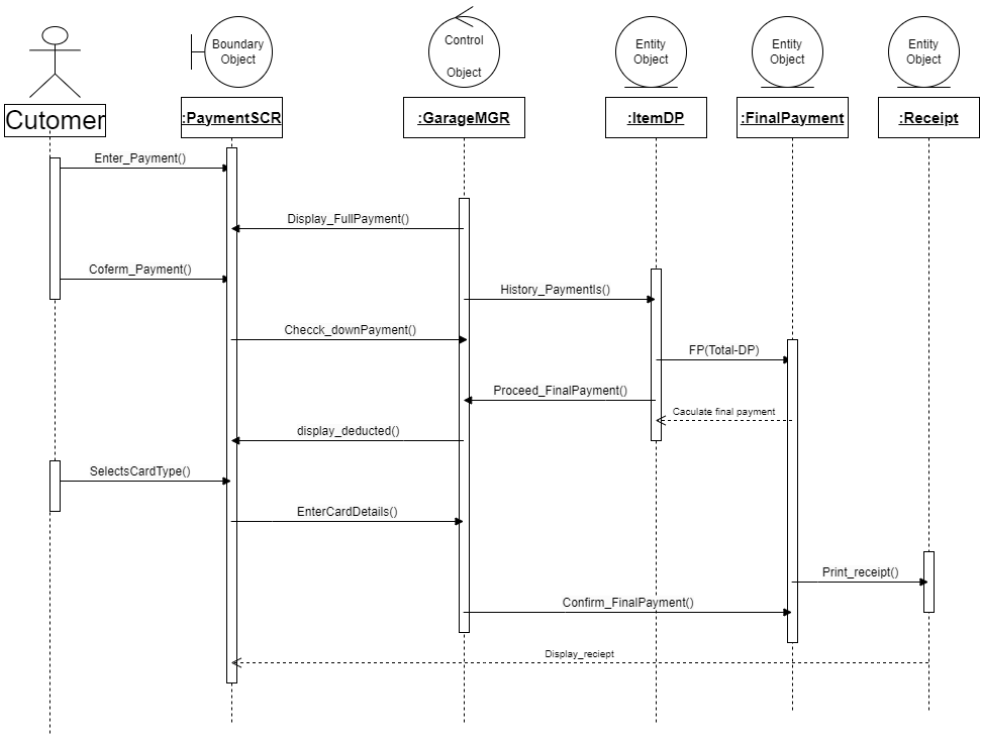


Figure 13: Sequence diagram for use case UC-5 (Payment) (Key UML)

Payment Use Case Scenario:

- Khalifa has previously paid 15% down payment of the service he has confirmed and completed.
- The system then displays the full and final payment of the service provided.
- The system then deducts 15% of the total amount to display final payment.
- Final payment = (total amount – down payment amount)
- The system will further display the rest of the amount that Khalifa needs to pay.
- Khalifa confirms the final payment and choses whether he would like to pay with a credit card or a debit card.
- Once payment is successful, the system provides Khalifa with a digital receipt.

From the interactions identified in the sequence diagram, initial methods for the interfaces of the interacting elements can be identified:

Method Name	Description
Element: Screen	
Enter_Payment ()	It is showing the payment page.
Coferm_Payment ()	It allows user to confirm the payment.
SelectsCardType ()	It allows user to select the type of card to complete payment process.
Element: Controller	
Display_FullPayment ()	It is displaying the full payment.
Checck_downPayment ()	It is checking for down payment.
display_deducted ()	It is displaying the deducted amount.
EnterCardDetails ()	It is allowing user to enter the card details.
Proceed_FinalPayment ()	It is showing the final payment page.
Element: Payment Information	
History_PaymentIs ()	It is search in the database about the user payment history.
FP(Total-DP)	It is deducting an amount of the final amount depend on the payment history.
Confirm_FinalPayment ()	It is adding the payment to the payment information database.
Print_receipt ()	It is displaying the final amount.

6.3.6 STEP 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose.

The decisions made in this iteration provided an initial understanding of how functionality is supported in the system. The modules associated with the primary use cases were identified by the architect, and the modules associated with the rest of the functionality were identified by another team member. From the complete list of modules, a work assignment table was created:

Not Addressed	Partially Addressed	Completely Addressed	Design Decision Made During the Iteration
		UC-1	Modules across the layers and preliminary interfaces to support this use case have been identified.
		UC-3	Modules across the layers and preliminary interfaces to support this use case have been identified.
		UC-5	Modules across the layers and preliminary interfaces to support this use case have been identified.
	QA-1		The elements that support the associated use case (UC-1) have been identified.
	QA-2		The elements that support the associated use case (UC-3) have been identified.
	QA-3		The elements that support the associated use case (UC-5) have been identified.
	QA-4		No relevant decisions made.
	CON#1		No relevant decisions made.
	CON#4		No relevant decisions made.
CRN#1			Selection of reference architectures and deployment pattern.
		CRN#4	No relevant decision made.
	QA-3		Identification of the elements derived from the deployment pattern that will need to be replicated.

6.4 ITERATION 3 Addressing Quality Attribute Scenario Driver

This iteration will present the outcome and effect through activities that are a resultant effect of the implementation of the ADD steps.

6.4.1 STEP 2 Establish Iteration goal by selecting drivers.

After building the fundamental structural decisions through iteration 1 and 2, In this iteration we will start to fulfil our primary Quality attributes. In this Third iteration we will focus in two primary quality attributes:

- **QA-3 (Availability):** When the system crashes at runtime, the system will notify the operator and then jump to the backup application server or backup database server and continues to operate without any downtime.
- **QA-1 (Performance):** When the user requests a normal operation, the system should not take more than 4 seconds to perform the transaction.

This document will show the deployment and sequence diagram for each quality attributes above.

6.4.2 STEP 3 Choose one or more elements to refine.

The elements that will be refined for both of the drivers chosen are the physical nodes that were identified during the first iteration:

- Application server.
- Database server.

6.4.3 STEP 4: Choose One or More Design Concepts That Satisfy the Selected Drivers.

The design concept used in this iteration are in the following points:

Using the Active Redundancy by Replicating Application Servers:

By replicating those core elements this will withstand the failure Which support availability (QA-3) and will increase the throughput and performance (QA-1).

Using Message Queue Technology:

Using trap receiver will make sure that failures have been processed in a fast way which increase availability (QA-3).

6.4.5 STEP 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces.

Deploy Message Queue on a Separate Node:

From using the tactic of active redundancy Deploying the message queue on a different node will make sure that traps are not lost.

Duplicating Application Server and the Use of Load Balancer:

There will be more than one application server, so for better performance, there should be a good load distribution between them.

Implement Load Balancing and Redundancy Using Technology Support:

Many technological options for load balancing and redundancy can be implemented without having to develop an ad hoc solution that would be less mature and harder to support.

6.4.6 STEP 6: Sketch Views and Record Design Decisions.

Figure 14 shows a refined deployment diagram that includes the introduction of redundancy in the system:

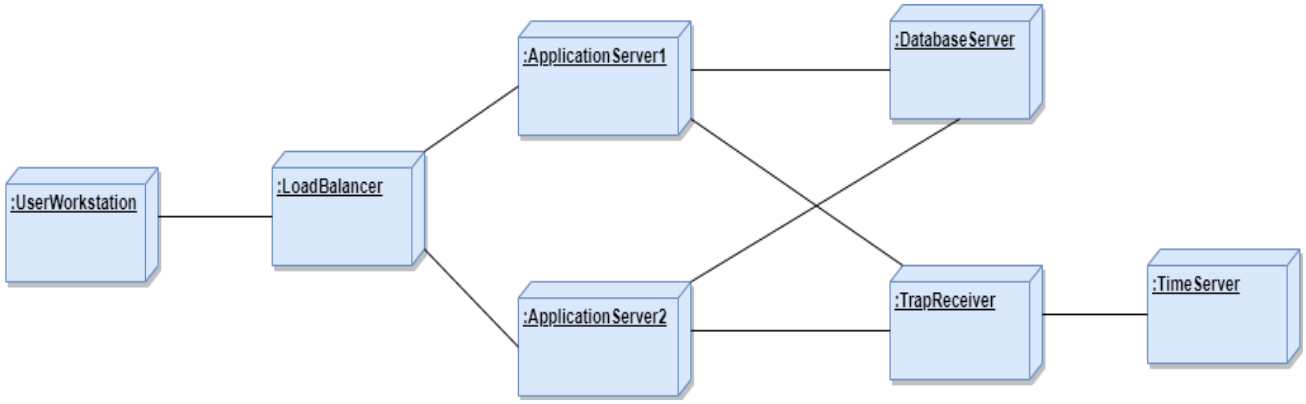


FIGURE 14: Refined deployment diagram (Key: UML)

The following table describes responsibilities for elements that have not been listed previously (in iteration 1):

Element	Responsibility
LoadBalancer	Receive client's requests and re-direct them to various servers according to their current load in order to balance them.
TrapReceiver	Receive traps from network devices converts them into events and puts these events into persistent message queue.

The UML sequence diagram shown in **Figure 15** illustrates how the TrapReceiver that was introduced in this iteration exchanges messages with other elements shown in the deployment diagram to support UC-3 (Booking Appointment), which is associated with both QA-3 (availability) and QA-1 (performance). As the purpose of this diagram is to illustrate the communication that occurs between the physical nodes, the names of the methods are only preliminary; they will be refined in further iterations.

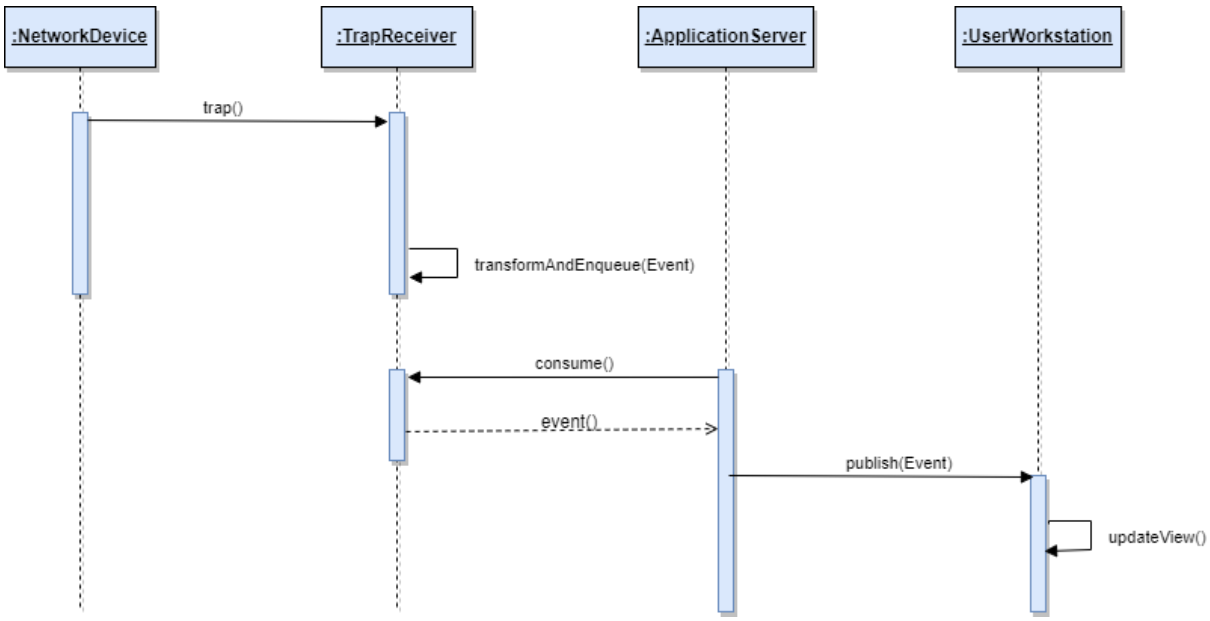


FIGURE 15: Sequence diagram illustrating the messages exchanged between the physical nodes to support UC-3 (Key: UML)

6.4.7 STEP 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose.

In this iteration, important design decisions have been made to address QA-3, which also impacted QA-1. The following table summarizes the status of the different drivers and the decisions that were made during the iteration. Drivers that were completely addressed in the previous iteration have been removed from the table.

Not Addressed	Partially Addressed	Completely Addressed	Design Decision Made During the Iteration
	QA-1		because trap reception is performed in a separate node, this approach reduces application server processing load, thereby helping performance. Because specific technologies have not been chosen, this driver is marked as "partially addressed".
		UC-3	Modules across the layers and preliminary interfaces to support this use case have been identified.
		UC-5	Modules across the layers and preliminary interfaces to support this use case have been identified.
	QA-5		No relevant decisions made.
	QA-3		y making the application server redundant, we reduce the probability of failure of the system. Furthermore, if the load balancer falls, a passive replica is activated within the required time period. Because specific technologies have not been chosen (message queue), this driver is marked as "partially addressed".
	CON-6		Replication of the application server and the use of a load balancer will help in supporting multiple user requests.
	CON-3		No relevant decisions made.
	CRN-2		No relevant decisions made.
	CRN-4		No relevant decision made.

7. Conclusion

The virtual garage system is built using various software design and architectural tools such as the UML diagram, establishing the use cases, approaching technical requirements –quality attributes-to test the software's properties and measure its quality. The QAs are critical in software design, and we further developed it through a utility tree and a six-part diagram for each quality attribute.

To ensure a successful design we addressed the architectural drivers: design purpose, quality attributes, primary functionality, architectural concerns and constraints and then executed the concept and decisions made. Abstraction and decomposition of the system has been broken down through the ADD steps.

Finally, the three iterations are implemented to promote the design and development, testing and implementation of the virtual garage system.