SOFE 3650U: Software Design & Architecture

Identifying Structures to Support Primary Functionality - ADD Iteration 2

Date: November 19, 2021

Group Members:

Foram Gandhi (100699245)

Kinjal Shah (100743551)

Rutvi Shah (100747171)

Danial Shaikh (100698628)

Iteration 2: Identifying Structures to Support Primary Functionality

Step 2: Establish Iteration Goal by Selecting Drivers

The goal of this iteration is to address the general architecture concern of identifying structures to support primary functionality. The architect considers CRN-3 which is the allocation of team members, as well as the following primary use cases:

- UC-1: System Availability
- UC-6: Date Selection
- UC-7: Ticket Selection
- UC-8: Seat Selection
- UC-9: Payment Options

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

The elements to be refined in this iteration are the modules in the reference architecture model from Iteration 1. The modules are located in the different layers and provide support of functionality for the system by collaborating different components associated with the modules.

Step 4: Choose One or More Design Concepts That Satisfy the Selected Drivers

Design Decision and Locations	**Rationale and Assumptions**
Create a **Domain Model** for the application	Before starting a functional decomposition, it's necessary to create an initial Domain Model for the system, identifying relationships and major entities in the domain. There are no good alternatives and the domain model must eventually be created or it will emerge in a suboptimal fashion, this leads to an complex ad hoc architecture that's hard to understand and maintain.
Identify **Layers** that map to functional requirements	Each layer of the application needs to have a distinct and specific responsibility. The interactions between all the layers have to be highly constrained with only unidirectional dependencies between them. One possible alternative is to consider domain objects from the start to eliminate the risk of not considering a requirement.
Decompose **Layers** into general and specialized **Components**	Each self-contained and coherent responsibility within a layer is categorized as a separate domain object (modules that can be developed independently). These modules are "components" in this pattern. Their specialization is associated with the layers where they are located,

	such as UI Modules.

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 Locations	**Rationale and Assumptions**
Create initial domain model	To accelerate this phase of design, only an initial domain model is created using the participating entities in the primary use cases.
System use cases mapped to layers	Analyzing the system use cases to initially identify domain objects.
Identify layer-specific modules with an explicit interface by decomposing the domain objects across the layers	Ensure that all modules that support the functionalities are identified.
Remove the Helpers and Utilities module from the Data Layer	Extra functionalities that are common to other modules are not needed in the system.

After identifying the architectural elements and interfaces in this step we can move on to recording design decisions in the next step.

Step 6: Sketch Views and Record Design Decisions

Initial Domain Model:

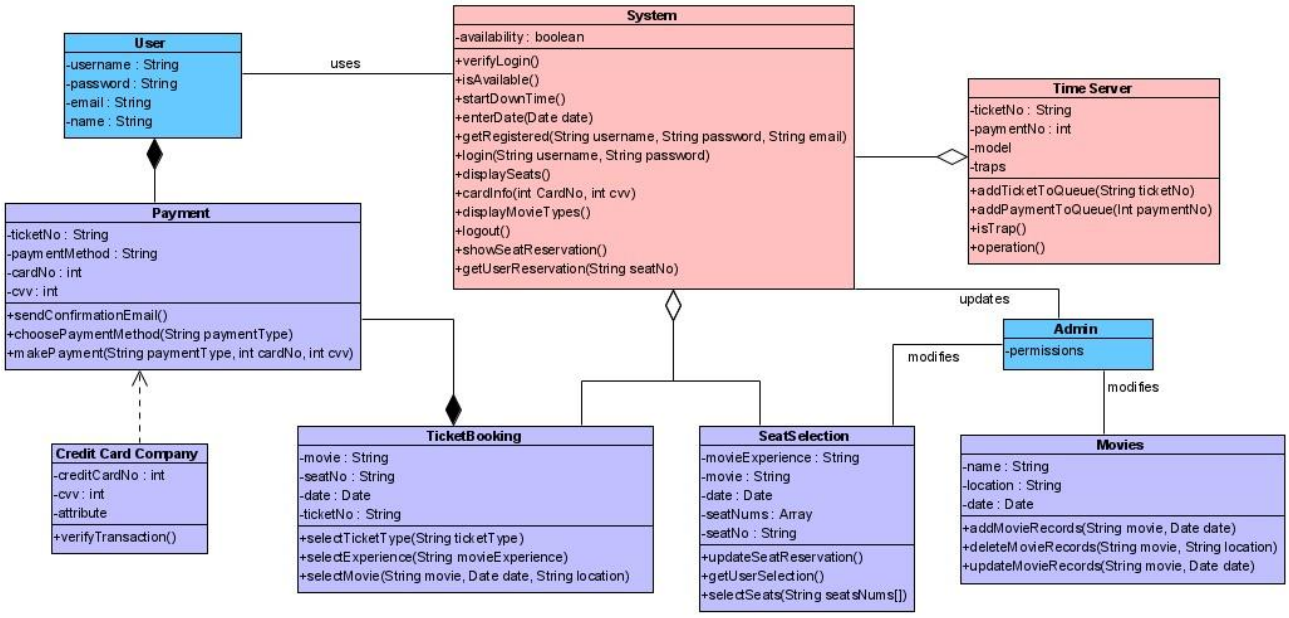

Fig 3: The initial domain model for the system supports the entities that participate in the primary use cases and their relationships.

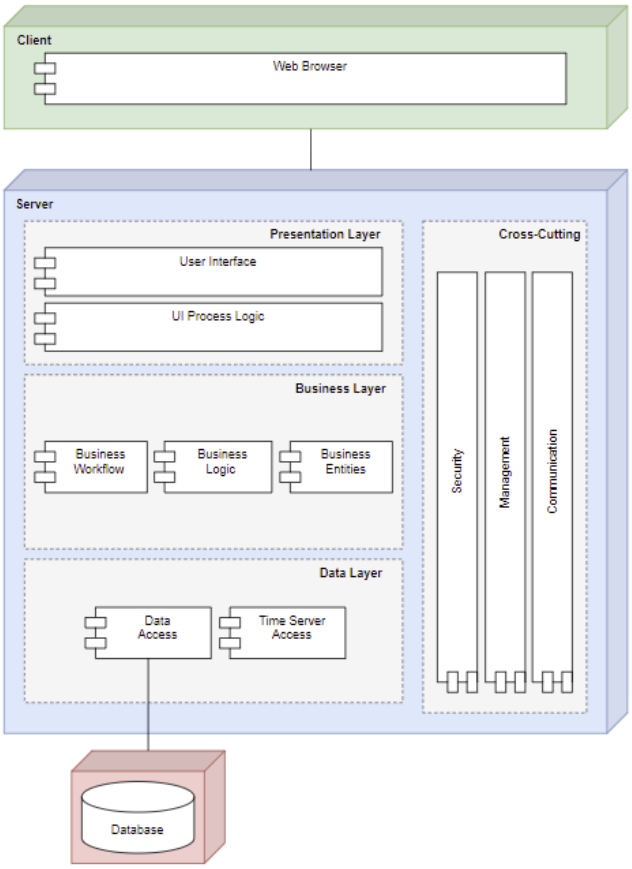

Fig 4: Web Application Model that supports the primary use cases.

The responsibilities for the elements identified in Figure 4 are summarized in the following table:

Layer	**Module**	**Responsibility**
Client	Web Browser	Movie Ticket Booking Application runs on the client's web browser.
Server - Presentation Layer	User Interface	Receives user interactions and presents the information to the users. It contains UI elements as buttons and text fields as well as a form in the ticket checkout page.
Server - Presentation Layer	UI Process Logic	Manages the control flow of the application’s use cases and provides data coming

		from the business layer to the user interface components. It is also responsible for data validation. An example is the Login/Signup Page.
Server - Business layer	Business Workflow	Involves the execution of multiple use cases (such as ticket selection and seat selection since seats selected must be equal to number of tickets previously selected)
Server - Business Layer	Business Logic - Payment - Ticket Booking	Retrieves and processes application data and applies business rules on the data (such as seat booking page and checkout page).
Server - Business Layer	Business Entities - User - Movie Information - System	Represents the entities from the business domain and the associated business logic.
Server - Data Layer	Data Access - MySQL Database - OMDB API	Provides the common components needed to retrieve and store information. It uses the OMDB API for retrieval of movies and movie info, as well as retrieval from the database to verify/check login/signup info.
Server - Data Layer	Time Server Access	This module isolates and abstracts operations to support communication between different types of time servers partially addressing QA-4.
Server - Cross-Cutting Layer	Security - User authentication - Password encryption/decryption	Handles security aspects such as user authorization and authentication, as well as password encryption/decryption

Server - Cross cutting Layer	Management	Handles validation, exception management and logging across all layers.
Server - Cross-Cutting Layer	Communication - Data layer communicates with database - Time Servers	Handles communication mechanisms across layers and physical tiers
External Data Sources	Database	Stores all data from the system, later accessed by the DataAccess module in the Data Layer (using MySQL to store databases).

UC-1: System Availability

Fig 5 shows an initial sequence diagram for UC-1 (System Availability). As shown, the technician makes sure the system is available at all times and that the time servers are not running into traps. The interaction starts with a Timeserver sending a trap, which is received by the Technician . If TimeServers.trap==true, then the technician initiates downtime for the system.

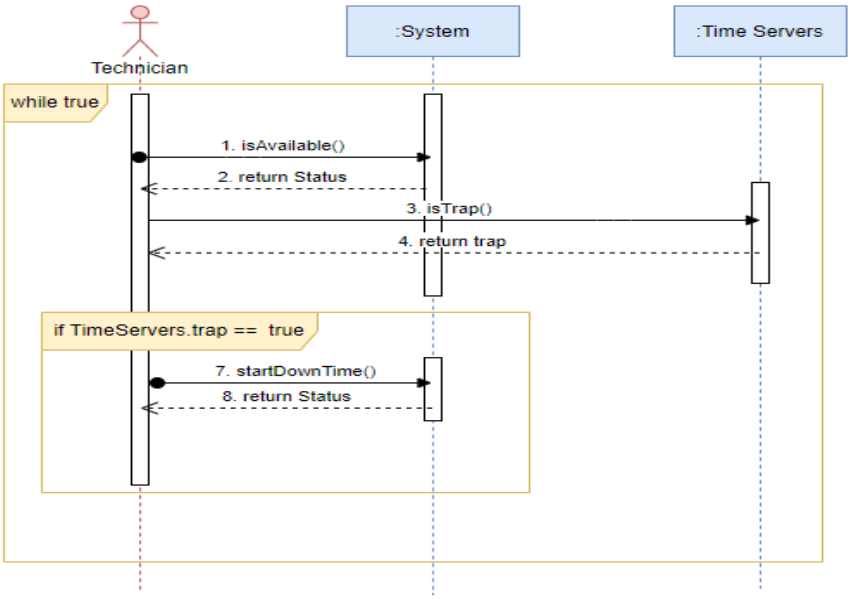

Fig 5: UML Sequence Diagram for UC-1

From the interactions identified in Fig 5, initial methods for the interfaces of the interacting elements are identified in the following table:

Element	Method Name	Description
System	boolean isAvailable()	Infinitely checks if all the components of the system are available and returns a status boolean
	boolean startDownTime()	When called the system in under downtime, is can be due to traps that are addressed immediately by developers
Time Servers	boolean isTrap()	Checks if a trap is detected and return the status as a boolean 'trap'

UC-6: Date Selection

The UML sequence diagram shown in Fig 6 illustrates how EnterDate() and SelectMovie() let the user interact with the UI of the system in order to select the date that they want to view movies by asking the user to enter a specific date. The user can then select a movie of interest and proceed forward according to date availability for the selected movie.

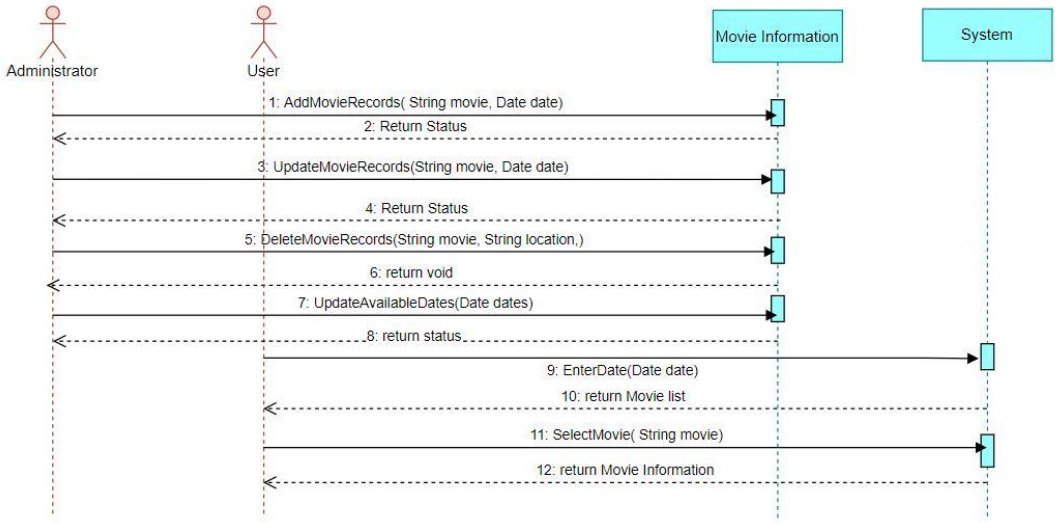

Fig 6: UML Sequence Diagram for UC-6.

From the interactions identified in Fig 6, initial methods for the interfaces of the interacting elements are identified in the following table:

Element	**Method Name**	**Description**
Movie Information	boolean AddMovieRecords() boolean UpdateMovieRecords() boolean DeleteMovieRecords() boolean UpdateAvailableDates()	The admin has authority to add, update and delete movies as well as update the available movie dates from the ticket booking system which returns a boolean status value indicating if their action was successful or not.
System	boolean EnterDate(Date dates) String SelectMovie(String movie)	The system allows the user to select the date that they want to view movies by asking the user to enter a specific date. The user can then select a movie of interest and proceed forward according to date availability for the selected movie.

UC-7: Ticket Selection

The UML sequence diagram shown in Fig 7 shows the system and payment components satisfying UC-7 (Ticket Selection) where the User interacts with the system to select the type of ticket they want (Regular, 3D, D-Box, IMAX, Atmos). The User is asked to GetRegistered() if they are not registered and already do not have a Login(). After the user has selected their movie and seats approved by the administrator through the **System**, they then proceed to **Payment** where they MakePayment() and then Logout() once everything is approved.

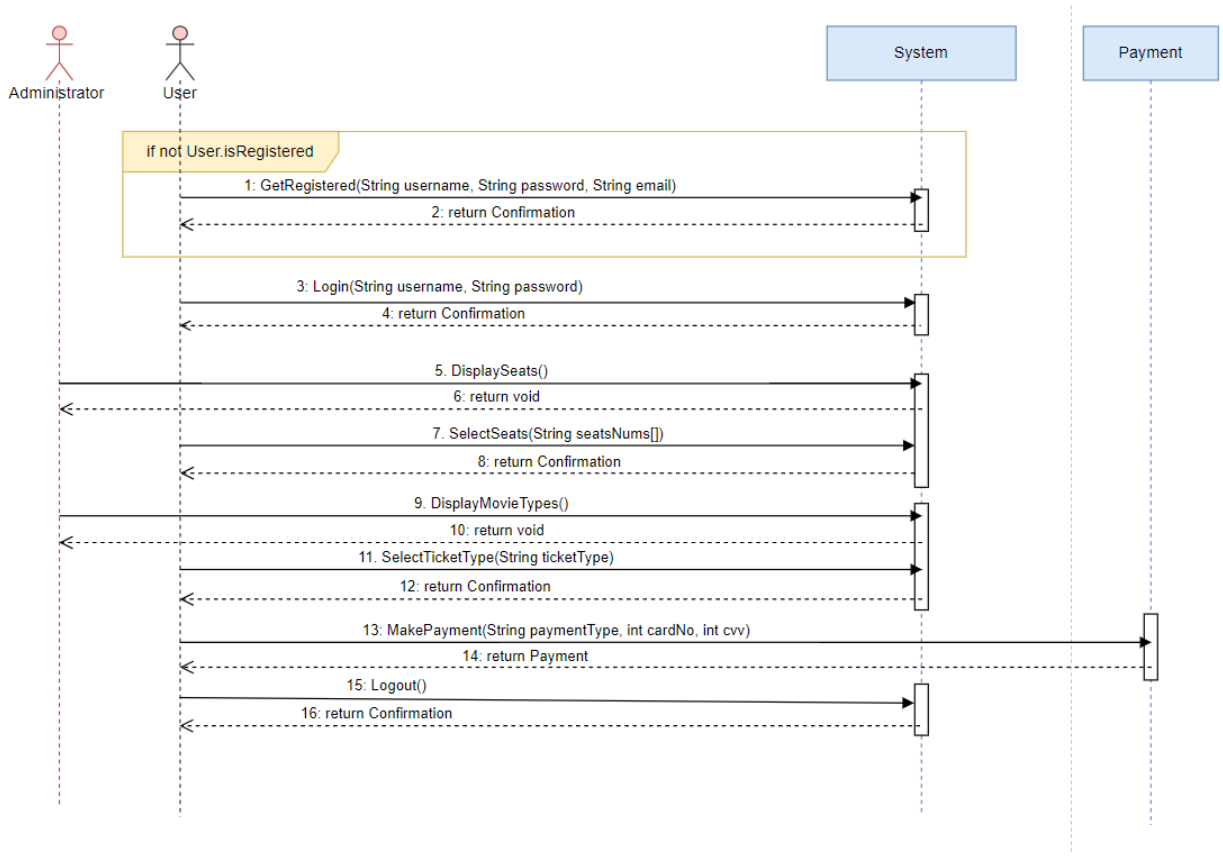

Fig 7: UML Sequence Diagram for UC-7

From the interactions identified in Fig 7, methods for the interfaces of the interacting elements are identified in the following table:

Element	**Method Name**	**Description**
System	void Login(String username, String password)	Logs users into the system allowing them access to book movies.
	void GetRegistered(String username, String password, String email)	If the user is trying to book a ticket and is not already registered into the system they are prompted to register by entering their desired username, password, and email.
	void Logout()	Logs out the user denying them access to book tickets.

	DisplaySeats() SelectSeats(String seatNums[]) DisplayMovieTypes() SelectTicketType(String ticketType)	Displays all the available seats in the theatre. Select seat(s) in the theatre and return a confirmation. Display all the movie types available for movie and selected theatre. Select ticket type and return confirmation.
Payment	boolean MakePayment(String paymentType, int cardNo, int cvv)	For the user to make a payment they must enter the type of payment they'd like to make the payment with plus they must also enter their card number and cvv. This function returns a confirmation status upon verifying the payment.

UC-8: Seat Selection

The UML sequence diagram shown in Fig 8 shows the exchange of messages between the elements to support UC-8 (Seat Selection), which is associated with QA-7 (Usability). The purpose of this diagram is to illustrate the communication that occurs between the physical nodes. Here, the user reserves their preferred seat location in the movie theatre through the Select Movie, Select Movie Experience & Seat Booking components respectively.

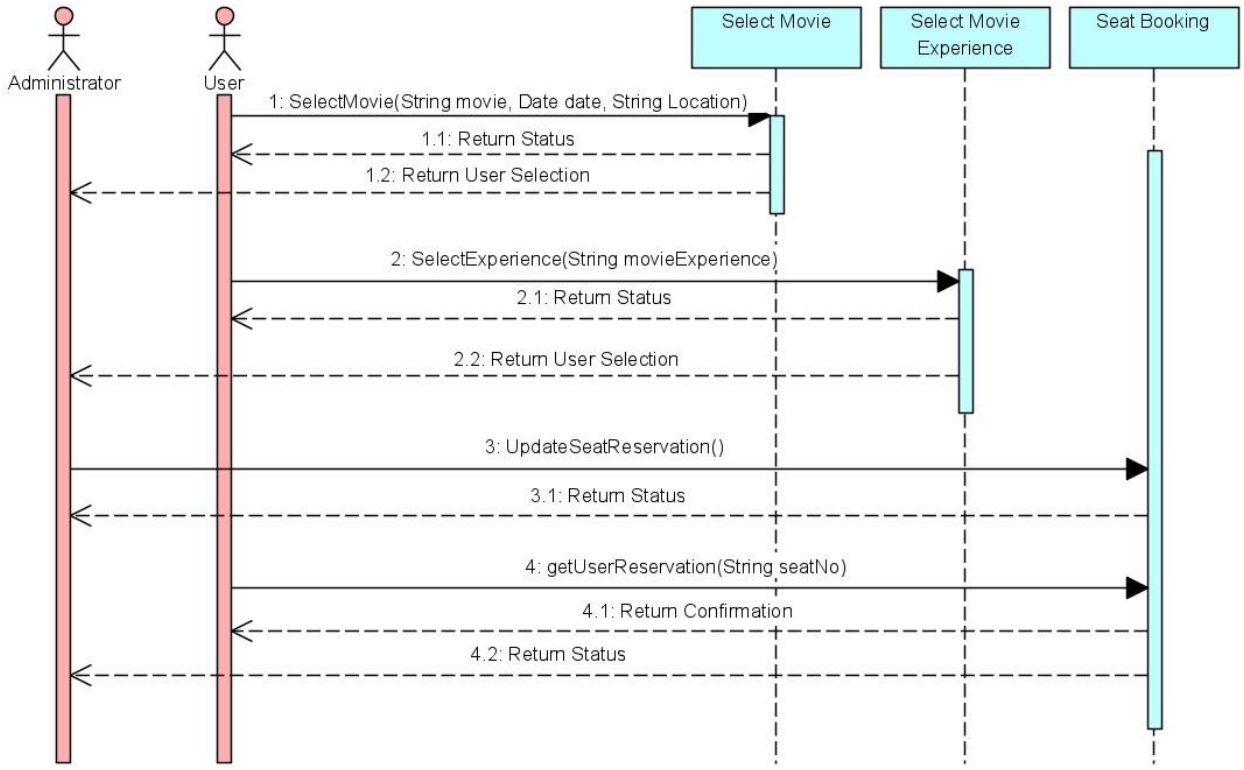

Fig 8: UML Sequence Diagram for UC-8

From the interactions identified in Fig 8, methods for the interfaces of the interacting elements are identified in the following table:

Element	**Method Name**	**Description**
Select Movie	String SelectMovie(String movie, Date date, String Location)	The user has the choice of what movie they want to reserve the seats for and according to the date, location and movie they will be shown the appropriate seats to reserve. The function returns the status of the movie selected to ensure the movie is available.
Select Movie Experience	String SelectExperience(String movieExperience)	The user must select the type of movie experience they want to book seats for, this could include regular, 3D, DBox, etc., depending on the location of the movie. The

		function returns the status to confirm the movie experience was selected.
Seat Booking	boolean updateSeatReservation() String getUserReservation(String seatNo)	The administrator must update the seat reservation page every time a user wants to reserve a seat. The function returns a status update back to the administrator. The user must select a seat number to reserve the respective seat for the movie of their choosing. The function returns confirmation to the user with the seat number.

UC-9: Payment Options

The UML sequence diagram shown in Fig 9 shows the system, payment & credit card company components satisfying UC-9 (Payment Options) where the developer must be able to create a system that can manage several payments being made at checkout from various users at the same time. Once the User selects the payment type and inputs the payment information through the **System**, the info is transferred to the **Payment** component where it is then verified with the **Credit Card Company**. Once the payment information is verified, the User is sent back a confirmation email.

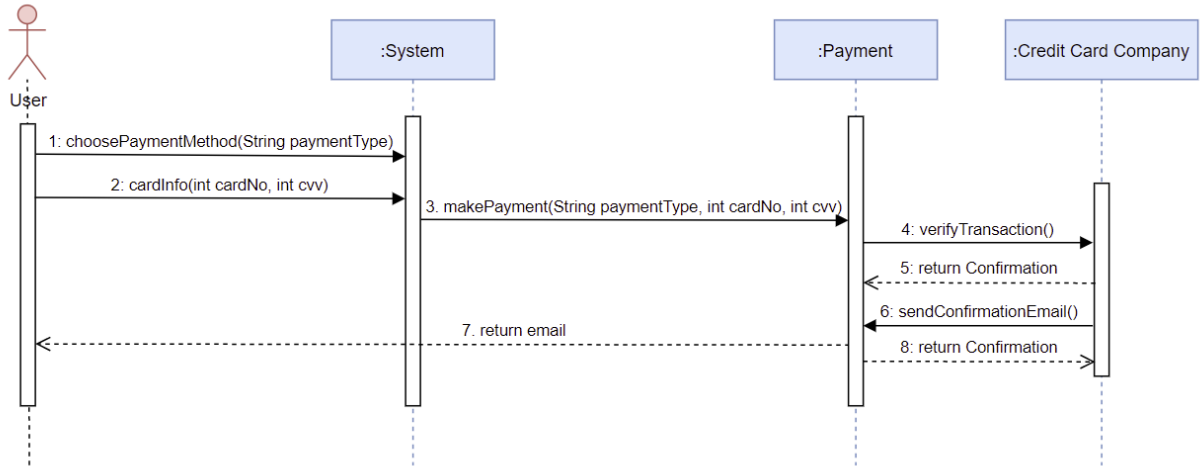

Fig 9: UML Sequence Diagram for UC-9

From the interactions identified in Fig 9, methods for the interfaces of the interacting elements are identified in the following table:

Element	**Method Name**	**Description**
System	void choosePaymentMethod(String paymentType)	The user must select their specific payment method in order to purchase tickets to the movie of their choice. Once this selection is made, the user will be prompted to enter their card information.
	boolean cardInfo(int cardNo, int cvv)	The user must enter their credit card/debit card information such as the card number and CVV in order to make a payment.
Payment	boolean MakePayment(String paymentType, int cardNo, int cvv)	For the user to make a payment they must enter their preferred payment method, as well as their card number and cvv. This function returns a confirmation email to the user upon a successful payment.
Credit Card Company	boolean verifyTransaction()	Once the user makes a payment, their credit card company will be alerted of the transaction and will have to verify it. This function returns a confirmation upon verification of payment to the system.
	void sendConfirmationEmail()	After the transaction has been confirmed/approved a confirmation will be sent to the credit card company indicating that the confirmation email has been sent successfully to the user.

Step 7: Perform Analysis of Current Design and Review Iteration

The selections taken in this iteration offered a basic grasp of how the system's functionality is supported. The status of the various drivers, as well as the decisions taken throughout the iteration, are summarised in the table below. The table has been cleared of drivers that were fully handled in the previous iteration.

Not Addressed	**Partially Addressed**	**Completely Addressed**	**Design Decisions Made During the Iteration**
		UC-1	Modules across the layers, supporting this use case have been identified.
	UC-6		Modules partially support this use case across the layers.
	UC-7		Modules partially support this use case across the layers.
	UC-8		Modules partially support this use case across the layers.
UC-9			No relative decisions made in this iteration for the use case.
		QA-2	The elements that support the associated use cases have been identified.
	QA-4		The elements that support the associated use cases have been identified.
QA-5			No relative decision has been made for this particular quality attribute
CON-1			Decisions for the work efficiency of the technician have not been made yet.
	CON-2		Decisions for the communication relationship between the developer and technician have not been made yet.
CON-3			Constraint is referenced by the

			architecture model although no relative decision has been made.
CON-6			No relative decision has been made for this constraint.
	CRN-2		Technologies and reference architecture that have been considered up to this point have taken into account the knowledge of the developers.
		CRN-3	Modules associated with all the use cases have been identified.