Ain Shams University

Faculty of Engineering

Computer and Systems Engineering Department

CSE232:Advanced Software Engineering (Fall 2022)

Cinemat – Project Document

Team Members

Kareem Wael Hasan	2001151
Shady Emad Sabry	20P7239
Karim Bassel Samir	20P6794
Ahmed Tarek Aboelmakarem	20P6897
Fady Fady Fouad Fahim	20P7341
Matthew Sherif Shalaby	20P6785
Mina Fadi Mohsen	20P6996
Omar Hisham Gouda	20P6484
Osama Saad Farouk	20P3943

Table of Contents

Table of Figures and Tables

1. Introduction

1.1. Purpose

The purpose of this document is to discuss the ‘Cinemat’ project, With detailed explanation about what functionality is expected from the software alongside the perspective of this project from different point of views, A general description of the project including both its capabilities and constraints, The characteristics of the different user roles provided in the application, And what environment is best suited for this software.

This document is intended for developers and software engineers seeking details on how to create a project from scratch and design their document, As well as the sequential steps of design needed to be taken to create a fully functional project.

1.2. List Of Definitions

No unknown terms, acronyms or abbreviations were used in this document.

1.3. Scope

The desktop application is a cinema booking app. It allows the user to create his own account, as owner, customer, or to continue as a guest without login. Specific users can login as administrators. Cinema owners can add his cinema, its location, its working hours, edit movies available, show times, prices, offers, number of seats available, different cinema halls (Standard, IMAX, Gold, Kids, etc.), snacks menu and prices, report a problem. Customers can book his customized ticket, with the desired cinema, movie, hall, and snacks. He also can compare between cinemas upon prices and ratings. He is allowed to view his booked tickets’ details. Before purchasing, he must enter his billing info. Additionally, he may rate movies and/or cinemas. While the guest can only view cinemas available, as well as movies, but without prices. Administrators have full functionality through the application. He can add or drop owners with their cinemas and ban customers. He acts as the owner of all cinemas as well.

1.4. List of References

[1] Creately, "UML Diagram Types," 28 September 2022. [Online]. Available: https://creately.com/blog/diagrams/uml-diagram-types-examples/.

[2] Wikipedia, "Object-oriented analysis and design," 23 April 2022. [Online]. Available: https://en.wikipedia.org/wiki/Object-oriented_analysis_and_design.

[3] M. Gadhavi, "Top 5 Software Architecture Patterns: The Ultimate Guide," 5 July 2022. [Online]. Available: https://radixweb.com/blog/software-architecture-patterns.

[4] H. Dhaduk, "10 Best Software Architecture Patterns You Must Know About," 4 July 2020. [Online]. Available: https://www.simform.com/blog/software-architecture-patterns/.

[5] Wikipedia, "Booch method," 30 November 2022. [Online]. Available: https://en.wikipedia.org/wiki/Booch_method.

[6] OpenGenus IQ, "Rumbaugh, Booch and Jacobson Methodologies," 22 January 2022. [Online]. Available: https://iq.opengenus.org/rumbaugh-booch-and-jacobson-methodologies/.

[7] A. Luie, "Rumbaugh, Booch and Jacobson Methodologies," [Online]. Available: https://iq.opengenus.org/rumbaugh-booch-and-jacobson-methodologies/.

[8] Noteflix, "Jacobson methodology," [Online]. Available: https://bcanotes4u171863936.wordpress.com/jacobson-methodology/.

1.5. Overview

In the following document, You will find the general description for the ‘Cinemat’ project which includes Detailed capabilities, constraints, User characteristics, And more. In the sections below you will also find that we provided detailed diagrams for all the functions of this project showcasing how they are inter-connected together, Using UML Diagrams, CRC Cards, and different methodologies of object analysis and design. Finally, We will talk about the adopted methodologies that were used, Highlighting the benefit of each methodology in both the design and analysis phase. A Time plan will also be provided to showcase how we scheduled each of our tasks.

2. General Description

2.1. Product Perspective

An established ‘Cinemat’ database system stores the following information. :-

Account information :-

Includes user’s id, username, password, user type, vaccination status.

Cinema Details :-

Includes cinema’s name and cinema’s owner.

Movie Details :-

Includes movie’s name, average rating, duration, genre, cinemas which it is currently playing at, ticket price and whether it is available or not.

Menu Details :-

Includes the item’s name, price and the cinema which this menu is located at.

Booked Details :-

Includes user’s ID and the current movies that this user has booked.

2.2. General Capabilities

1- Create account
2- Login
3- Browse cinemas and movies
4- Book ticket(s)
5- Add ratings and reviews
6- Add/Remove cinema
7- Add/Remove movie
8- Edit show times
9- Edit ticket prices
10- Edit snacks and drinks prices

2.3. General Constraints

One of the constraints of our application is that it uses SQL to store data. This data includes account information, prices of movie tickets, prices of food, each Movie’s information, Each cinema’s information, etc. This amount of data will increase as time passes and the number of users increase, Therefore we will need a server which has a large size to store this database on it. Although SQL helps us organize our data in an efficient manner, Buying a server might be costly to some clients.

According to recent statistics, 49.78% percent of the total web visits are currently mobile, Compared to 50.22% coming from desktops. Which means that our application has another constraint, And it is that our application is developed only for windows. Which means that we will be missing a very large user-base by not deploying it for mobile phones as well.

2.4. User Characteristics

2.4.1. Guest

Guests who use this application will be able to freely view the available cinemas, As well as have the option to create a new account to be able to use the ‘User features’ or the ‘Cinema Owner features’

2.4.2. User

Users who use this application will have many functionalities. In terms of their account details, The user will be able to edit their account information. Additionally , The user will be able to view all the available cinemas and filter them based on prices or ratings. When the user selects a certain cinema they will we be able to view the current movies that cinema is playing. Upon selecting a movie, The user will be able to book tickets at their desired seat as long as it is not previously booked, The user will also have the option to pay online. The user will also be able to view their booked ticket details, Rate films that they have watched.

In case of any technical problem, The user will have the option to contact the developers easily using the click of a button.

2.4.3. Cinema Owner

Cinema Owners who use this application will also have many functionalities. The cinema owners can add their own cinemas to the application (To be shown to the users and guests), Alongside it's details (Location, working hours, etc..). Cinema Owners will also have the functionality to add and delete movies from their cinemas, Control ticket prices and seats available, Edit their working hours, And control their menu's prices. Cinema Owners who use our application will gain much more visibility to a wider audience.

In case of any technical problem, The user will have the option to contact the developers easily using the click of a button.

2.5. Environment Description

Our application will require the following environment for it to function as intended :-

1- A database
2- Operating System : Windows
3- Platform : Java
4- A server to host our database

2.6. Assumptions and Dependencies

Let's assume that this system is used in the following application:

1- A request for booking a ticket to a chosen movie, in the desired cinema, at a specific time, of course if the tickets are not fully booked, or the film is still available at the time chosen. Among the request, you can choose snacks and drinks through the system so that they are ready when you arrive in place.
2- Reviewing different movies and cinemas through our application.

3. Specific Requirements

3.1. Capability Requirements

- A Windows operating system : Required to run our application
- An established SQL Database : Required to store the account information of each user, Each cinema's movie list, Each movie's details, Prices of the different categories in our application (Food, Movie tickets) in an organized manner.

3.2. Constraint Requirements

To solve the first constraint, The client will require one of two things. Either the client already has a server on which he can run the database servers on, Or the client has the proper funding to purchase a server with a large size on which he can run the database servers on.

To solve the second constraint, The client will require proper funding to hire a team of mobile-application developers so that they can work on the mobile version of our application using the help of this document and the already established desktop application.

4. Use-Case Diagram and the Narrative Description of the use cases

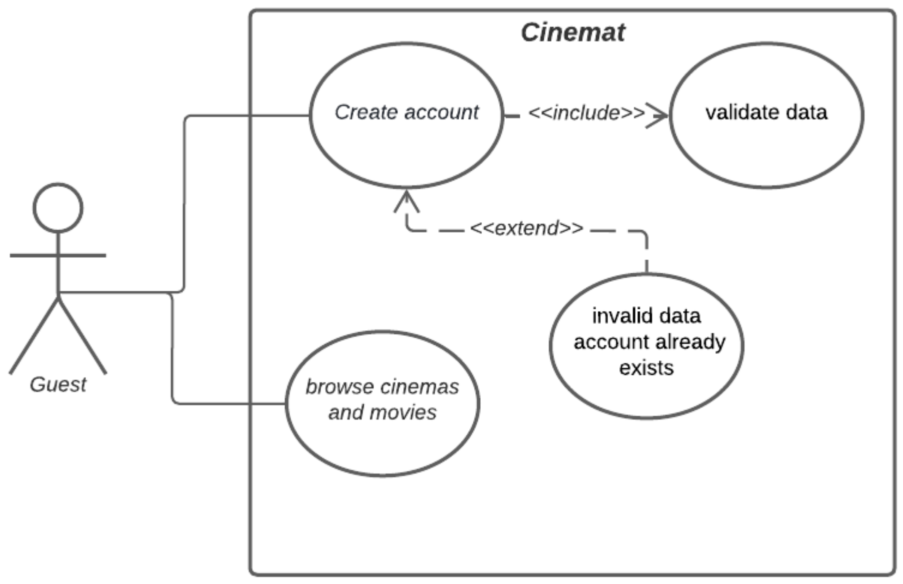

Cinemat
Create account
<<include>>
validate data
<<extend>>
invalid data account already exists
Guest
browse cinemas and movies

Figure 1 : Guest Use-case Diagram

4.1. Guest Use-cases descriptions

4.1.1. Create Account

Use Case name	Create Account
Goal In Context	A new or existing user requests a new account from the Administrator
Preconditions	None
Successful End Condition	A new account is created for the guest
Failed End Condition	The entered data is rejected by the Administrator
Primary Actors	Guest
Secondary Actor	None
Included Cases	Validate Data

Main Flow	Step	Action
	1	Administrator asks the system to create a new account.
	2	The administrator selects account type.
	3	The Administrator enters the user’s details.
	4	The entered details are compared with the database to make sure it is not identical to another user’s details.
	include::Validate Data	
	5	A new account is created.
Extensions	Step	Branching Action
	4.1	The user’s details match another user’s details in the database.
	4.2	The user’s entered data is rejected.

4.1.2. Browse Cinemas and Movies

Use Case name	Browse Cinemas and Movies
Goal In Context	User becomes able to view the available movies and cinemas
Preconditions	None
Successful End Condition	Cinema and Movie details are viewed by the Guest
Failed End Condition	User is not able to view the cinema or movie details
Primary Actors	Guest
Secondary Actor	None

Main Flow	Step	Action
	1	Movie Manager fetches the list of available movies and loads their details.
	2	Database Manager fetches the list of available cinemas and loads their details.
	3	List of available movies and cinemas are loaded on the guest’s screen.
Extensions	Step	Branching Action
	1.1	The Movie Manager fails to fetch the list of available movies.
	2.1	The Database Manager fails to fetch the list of available cinemas

4.1.3. Validate Data

Use Case name	Validate Data
Goal In Context	User's details need to be compared with existing users' details
Preconditions	None
Successful End Condition	The user's details are validated
Failed End Condition	The user's details are not validated
Primary Actors	Guest
Secondary Actor	None
Trigger	User's details are provided to the system for validation

Main Flow	Step	Action
	1	The details are provided to the system.
	2	The database manager compares the given details with the existing users' details
	3	The details are returned as validated by the Database Manager.

Extensions	Step	Branching Action
	2.1	The database manager does not validate the details.
	2.2	The details are returned as unverified.

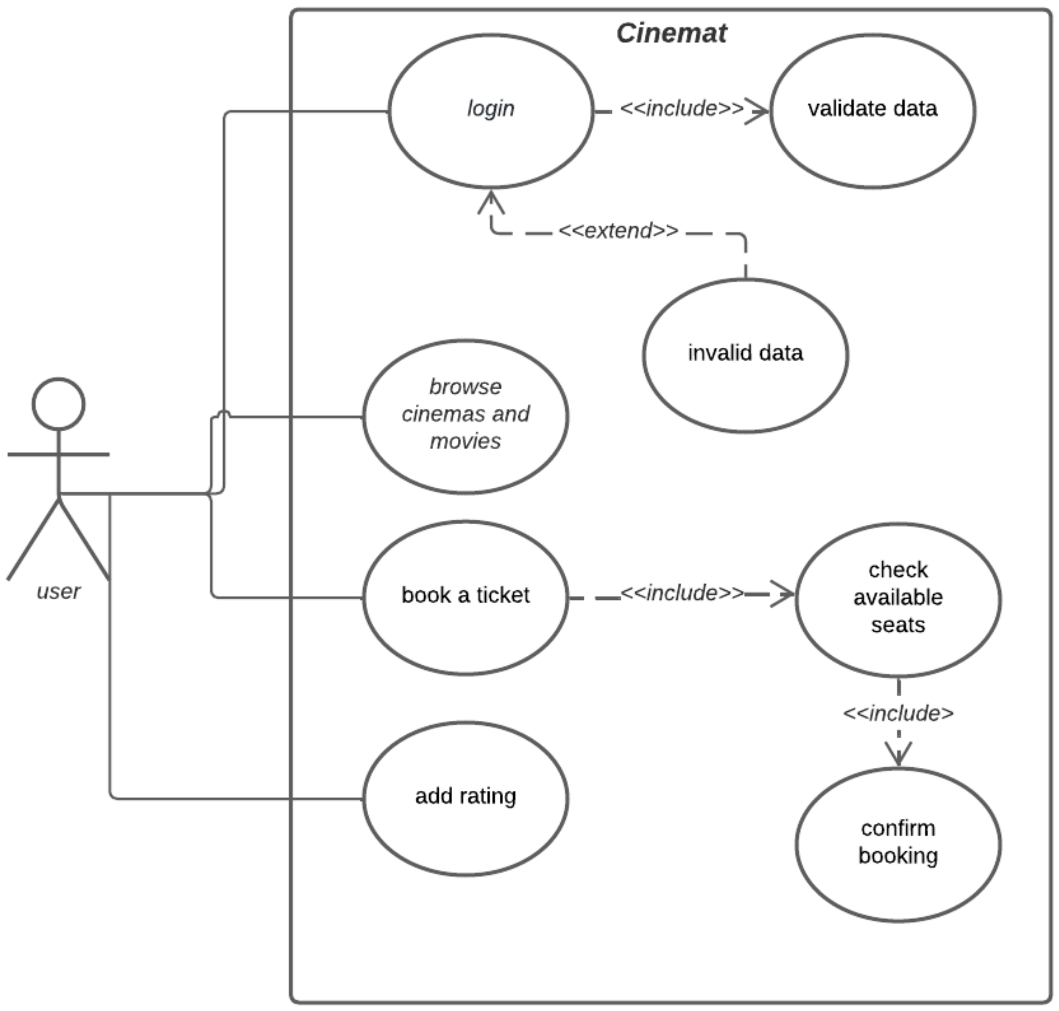


Figure 2 : User Use-case Diagram

4.2. User Use-cases descriptions

4.2.1. Login

Use Case name	Login
Goal In Context	User logs in
Preconditions	None
Successful End Condition	User is logged in successfully
Failed End Condition	User is not logged in
Primary Actors	User
Secondary Actor	None

Main Flow

Step	Action
1	User chooses to log in using their username and password
2	User enters their username and password in the presented fields for each one respectively
3	Database Manager validates user's username and password.

include::Validate Data

4	User logs in successfully and a menu of various options is displayed.

Extensions

Step	Branching Action
3.1	Database Manager notifies the user that their password is invalid.
3.2	The system returns back to its initial state.

4.2.2. Validate Data

Use Case name	Validate Data
Goal In Context	User's details need to be compared with existing user's details.
Preconditions	None
Successful End Condition	The user's details are validated
Failed End Condition	The user's details are not validated
Primary Actors	User
Secondary Actor	None
Trigger	User's details are provided to the system for validation

Main Flow	Step	Action
	1	The details are provided to the system.
	2	The database manager compares the given details with the existing users' details
	3	The details are returned as validated by the Database Manager.
Extensions	Step	Branching Action
	2.1	The database manager does not validate the details.
	2.2	The details are returned as unverified.

4.2.3. Browse Cinemas and movies

Use Case name	Browse Cinemas and Movies
Goal In Context	User becomes able to view the available movies and cinemas
Preconditions	None
Successful End Condition	Cinema and Movie details are viewed by the Guest
Failed End Condition	User is not able to view the cinema or movie details
Primary Actors	User
Secondary Actor	None

Main Flow	Step	Action
	1	Movie Manager fetches the list of available movies and loads their details.
	2	Database Manager fetches the list of available cinemas and loads their details.
	3	List of available movies and cinemas are loaded on the guest's screen.
Extensions	Step	Branching Action
	1.1	The Movie Manager fails to fetch the list of available movies
	2.1	The Database Manager fails to fetch the list of available cinemas.

4.2.4. Book a ticket

Use Case name	Book a ticket
Goal In Context	Tickets for a movie are booked
Preconditions	Seats available, Movie currently playing in theaters, User is logged in
Successful End Condition	User obtains a ticket ID.
Failed End Condition	User does not obtain a ticket ID.
Primary Actors	User
Secondary Actor	None

Main Flow	Step	Action
	1	User selects a cinema from list of cinemas.
	2	User selects a movie.
		Include::check available seats
	3	A list of available seats is shown to the user
	4	User selects seat.
	5	Uses books a ticket for the selected seat.
	6	Ticket Manager reserves the selected seat and prevents it from being booked by another user.

Extensions	Step	Branching Action
	2.1	User does not find their desired movie currently playing.
	2.2	User does not book a ticket.

4.2.5. Add Rating

Use Case name		Add Rating
Goal In Context		User adds a rating to a certain movie
Preconditions		User watched the movie being rated
Successful End Condition		Selected movie is rated successfully by the user.
Failed End Condition		Selected movie is not rated.
Primary Actors		User
Secondary Actor		None
Main Flow	Step	Action
	1	User selects a movie from list of movies.
	2	User submits a rating to the selected movie.
	3	Movie Manager stores the user’s rating and re-evaluates the overall rating of the movie.
Extensions	Step	Branching Action
	1.1	User selected a movie that they have not watched.
	1.2	User cannot add a rating to the movie.
	1.3	The selected movie is not rated

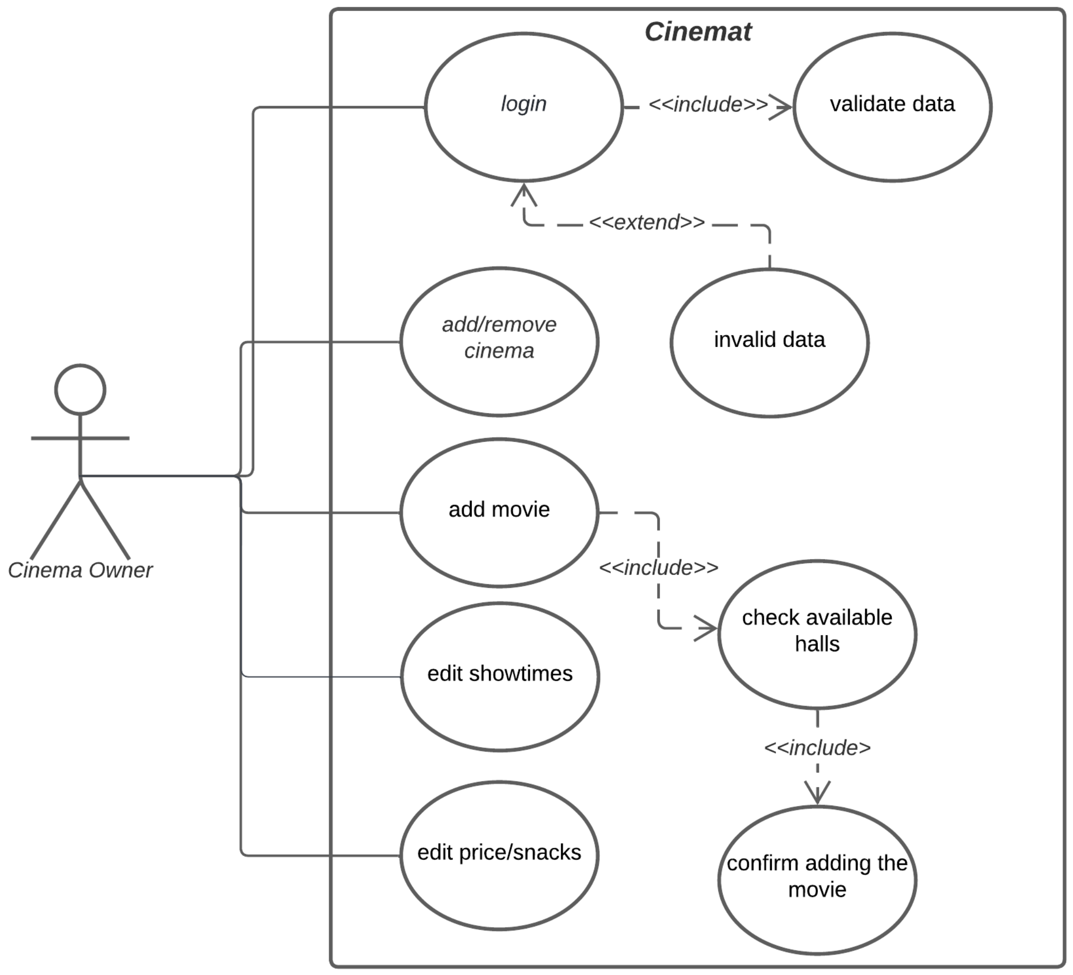


Figure 3 : Cinema Owner Use-case Diagram

4.3. Cinema Owner Use-cases descriptions

4.3.1. Login

Use Case name		Login
Goal In Context		User logs in
Preconditions		None
Successful End Condition		User is logged in successfully
Failed End Condition		User is not logged in
Primary Actors		Cinema Owner
Secondary Actor		None
Main Flow	Step	Action
	1	User chooses to log in using their username and password
	2	User enters their username and password in the presented fields for each one respectively
	3	Database Manager validates user's username and password.
	include::Validate Data	
	4	User logs in successfully and a menu of various options is displayed.
Extensions	Step	Branching Action
	3.1	Database Manager notifies the user that their password is invalid.
	3.2	The system returns to its initial state.

4.3.2. Validate Data

Use Case name		Validate Data
Goal In Context		User's details need to be compared with existing users' details
Preconditions		None
Successful End Condition		The user's details are validated
Failed End Condition		The user's details are not validated
Primary Actors		Cinema Owner
Secondary Actor		None
Trigger		User's details are provided to the system for validation.
Main Flow	Step	Action
	1	The details are provided to the system.
	2	The database manager compares the given details with the existing users' details
	3	The details are returned as validated by the Database Manager.
Extensions	Step	Branching Action
	2.1	The database manager does not validate the details.
	2.2	The details are returned as unverified.

4.3.3. Add Movie

Use Case name		Add Movie
Goal In Context		Cinema Owner adds a movie to their cinema
Preconditions		User logged in as Cinema-Owner
Successful End Condition		A movie is added to the Cinema Owner's cinema
Failed End Condition		A movie is not added to the Cinema Owner's cinema
Primary Actors		Cinema Owner
Secondary Actor		Movie Manager
Main Flow	Step	Action
	1	Cinema Owner requests their list of cinemas from Movie Manager
	2	Movie Manager shows the list of cinemas to the Cinema owner
	3	Cinema Owner selects a cinema from the shown list
	4	Cinema Owner sends movie's details to the Movie Manager
	5	Movie Manager stores the received details in the database and updates list of Movies for selected cinema.
Extensions	Step	Branching Action
	1.1	Cinema Owner does not have any cinemas created
	1.2	System returns to its initial state.

5. Swimlane Diagrams of all use cases

User / Cinema Owner

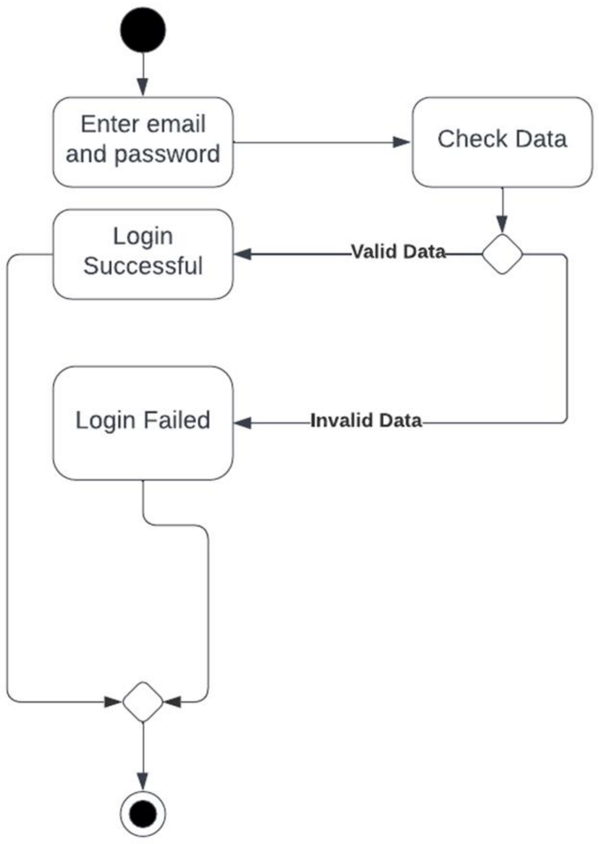


Figure 4: User/Cinema Owner Login Activity Diagram

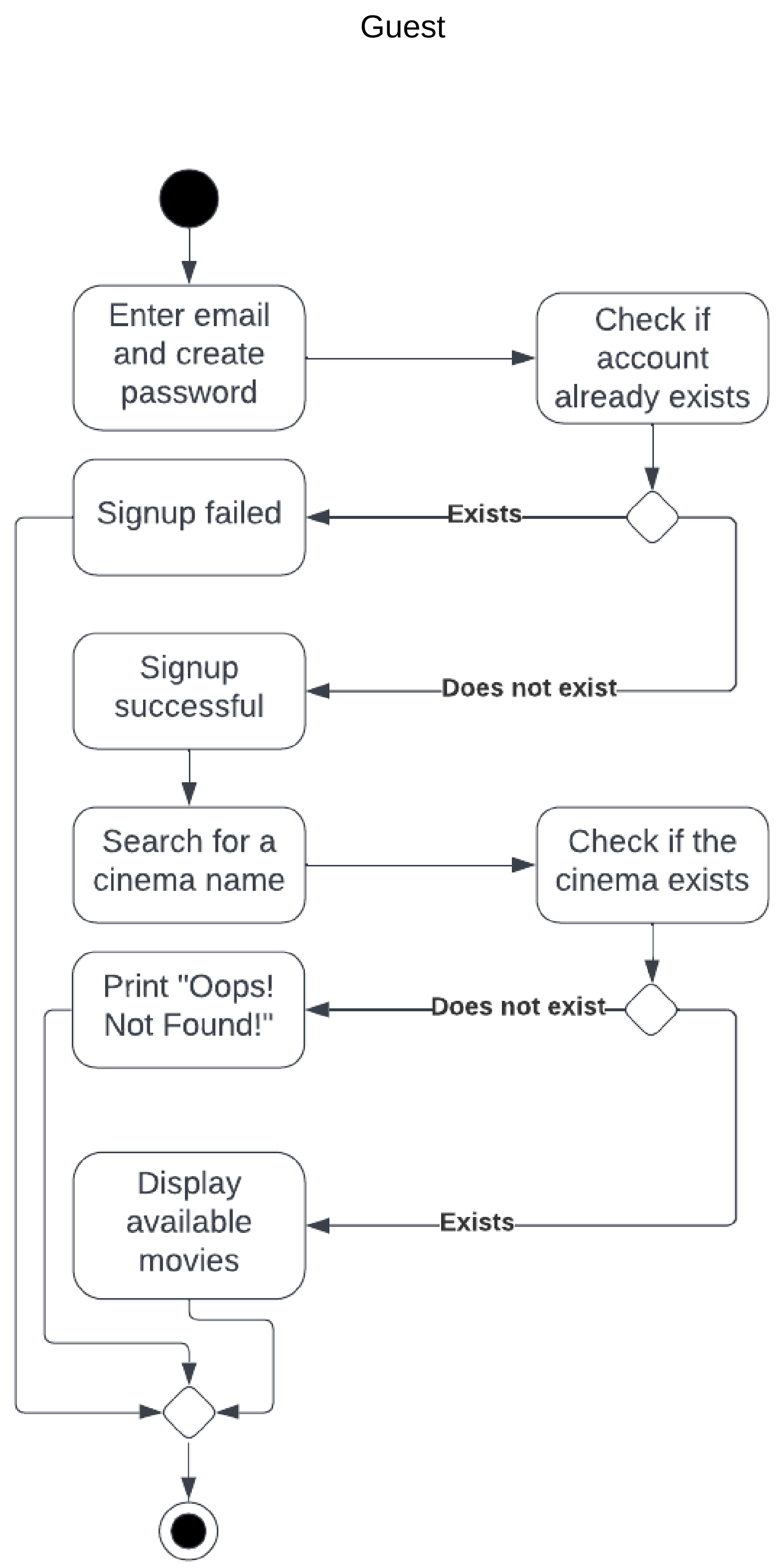


Figure 5: Guest Activity Diagram

User

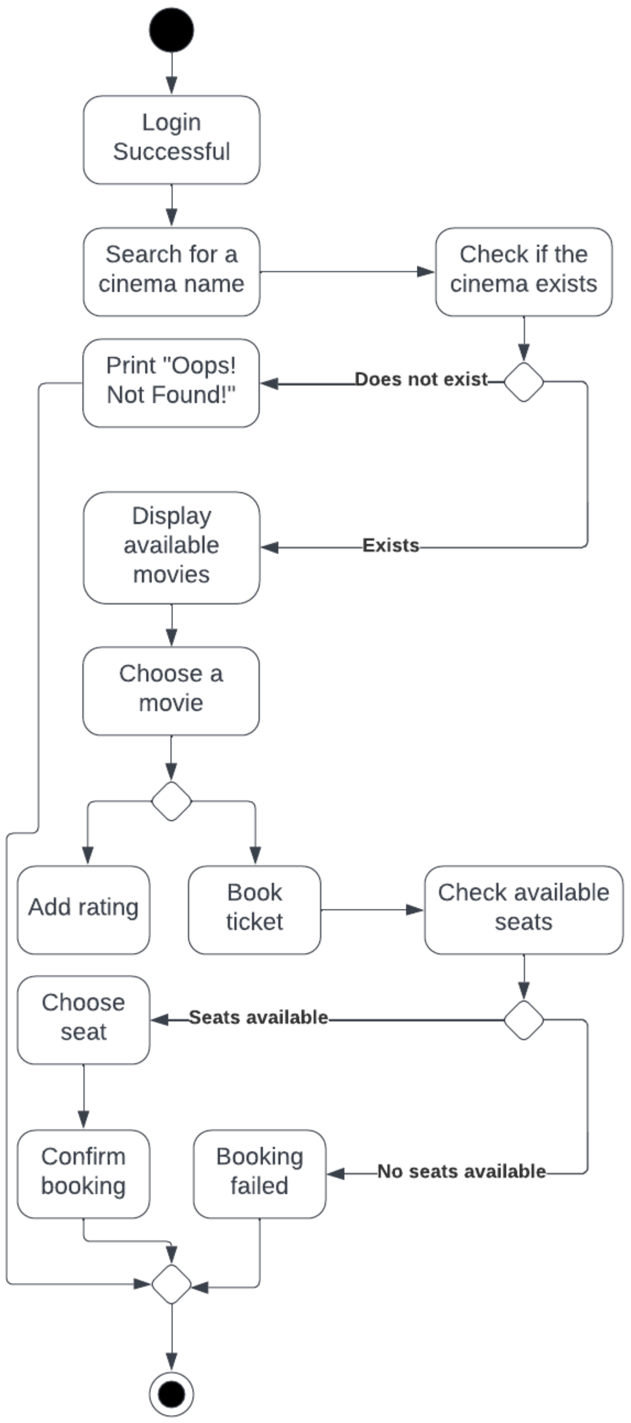


Figure 6: Browsing Cinemas, Movies and booking tickets Activity Diagram

Cinema Owner

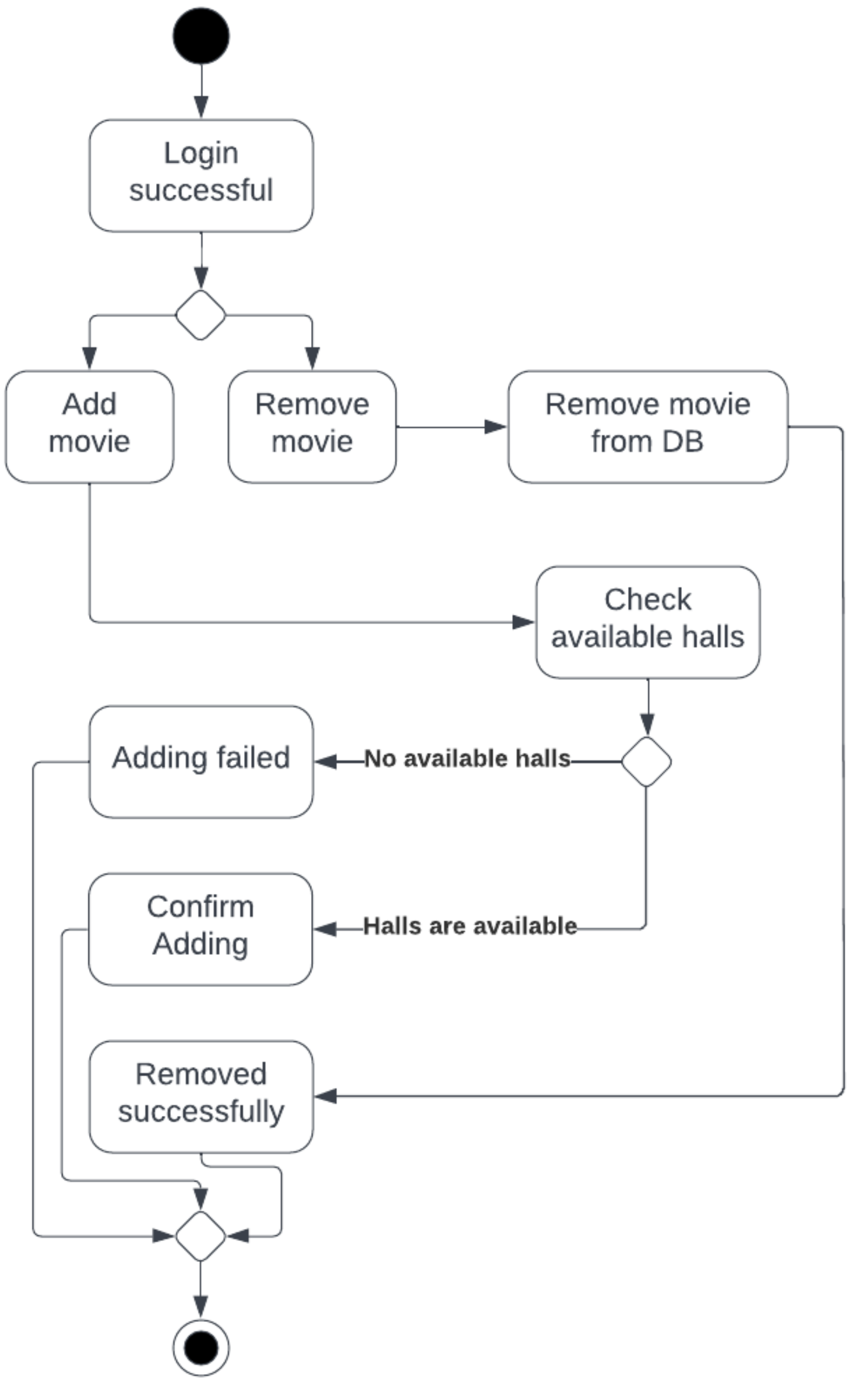


Figure 7: Add/Remove Movie(s) Activity Diagram

6. Interaction Diagrams

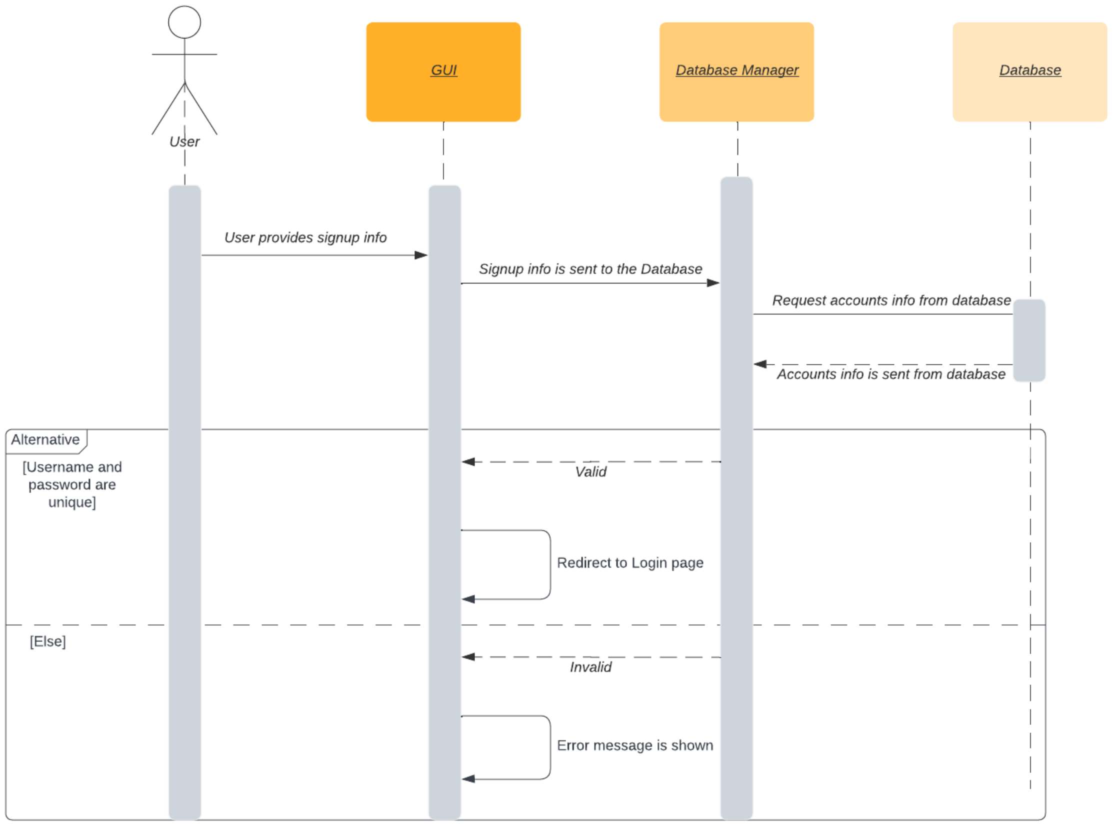

User
GUI
Database Manager
Database
User provides signup info
Signup info is sent to the Database
Request accounts info from database
Accounts info is sent from database
Alternative
[Username and password are unique]
Valid
Redirect to Login page
[Else]
Invalid
Error message is shown

Figure 8: Register Sequence Diagram

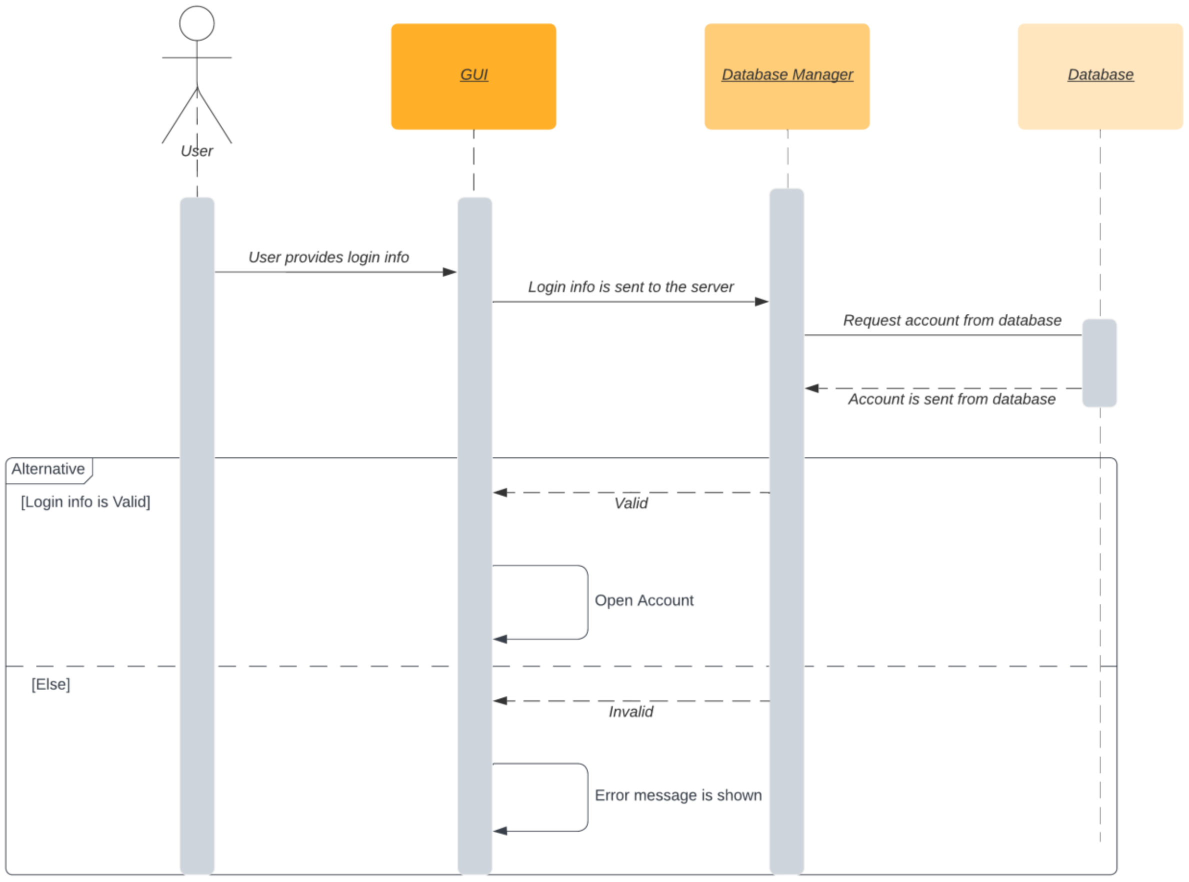


Figure 9 : Login Sequence Diagram

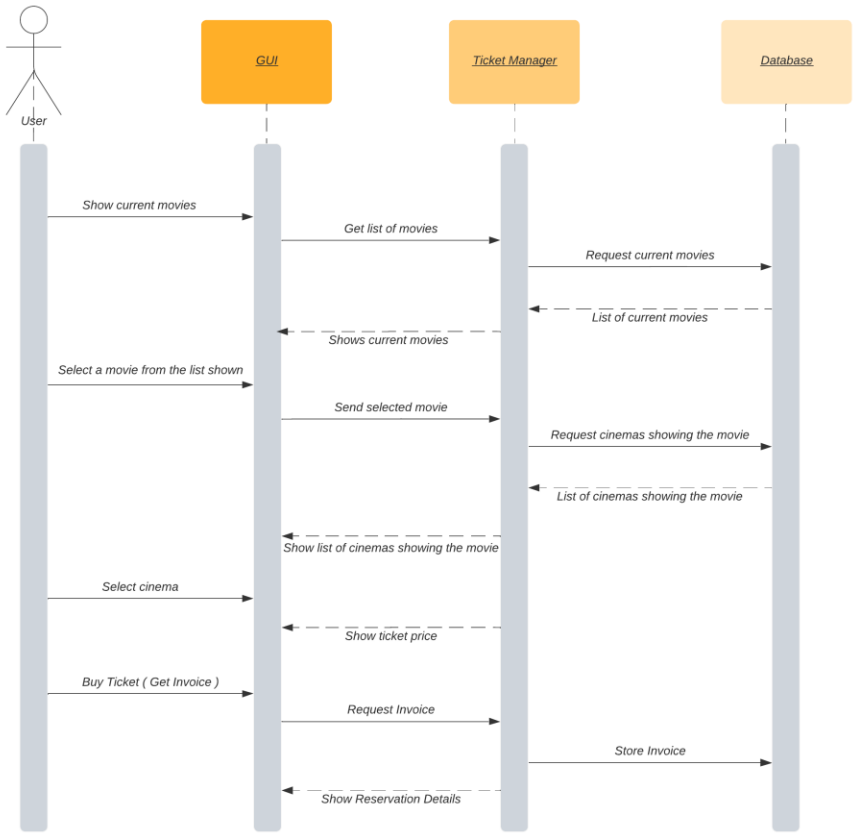


Figure 10 : Ticket Purchasing diagram

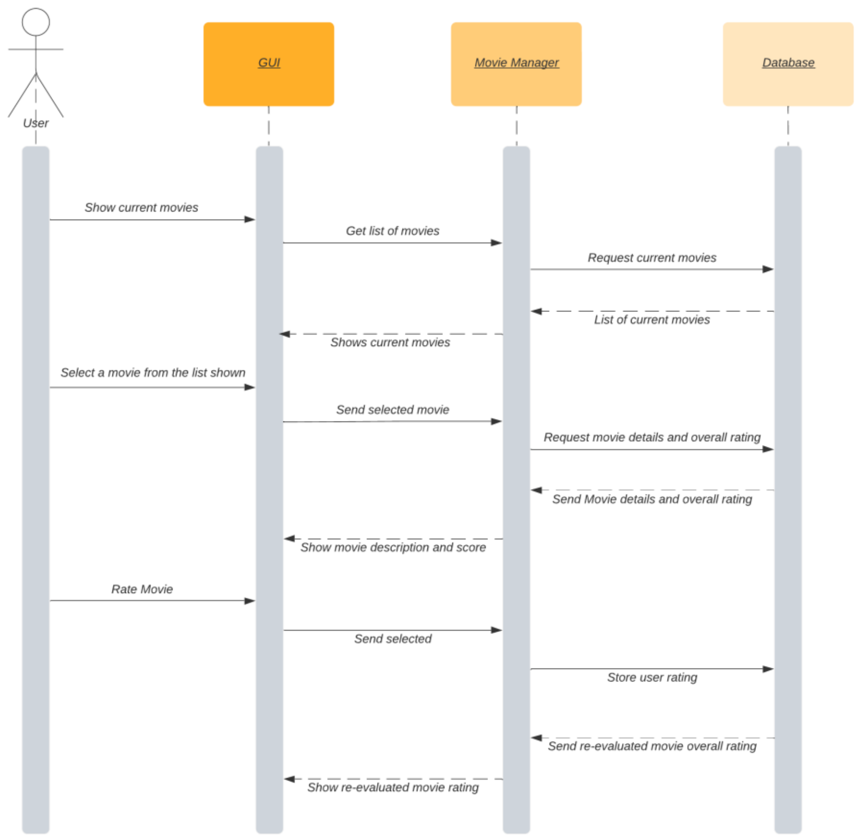

GUI
Movie Manager
Database
User
Show current movies
Get list of movies
Request current movies
List of current movies
Shows current movies
Select a movie from the list shown
Send selected movie
Request movie details and overall rating
Send Movie details and overall rating
Show movie description and score
Rate Movie
Send selected
Store user rating
Send re-evaluated movie overall rating
Show re-evaluated movie rating

Figure 11 : Movie rating Diagram

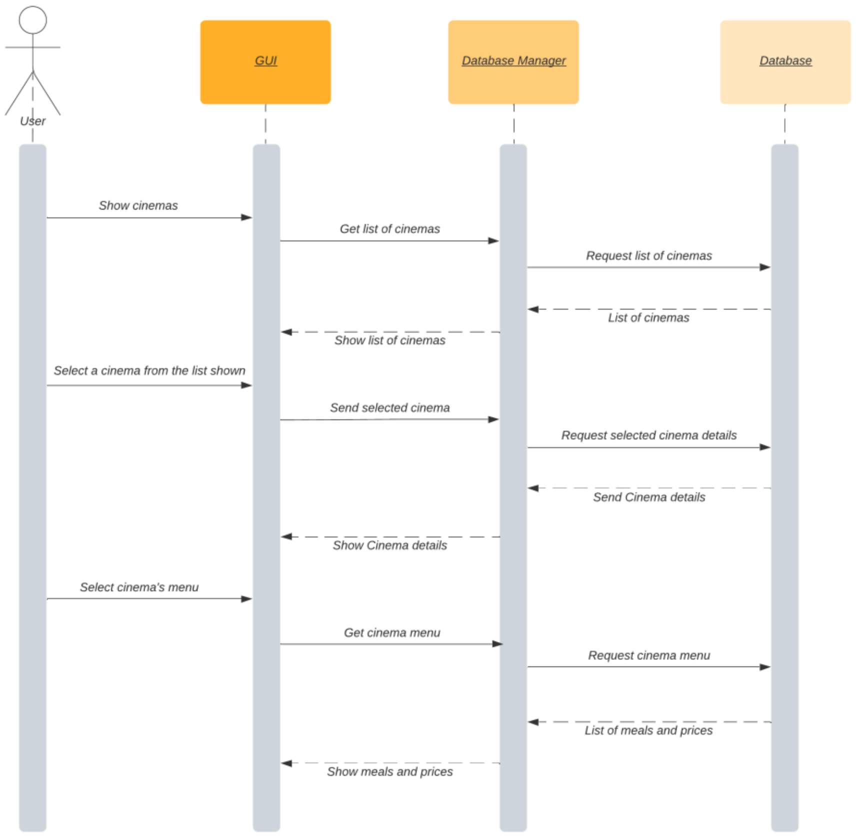


Figure 12 : Show menu Diagram

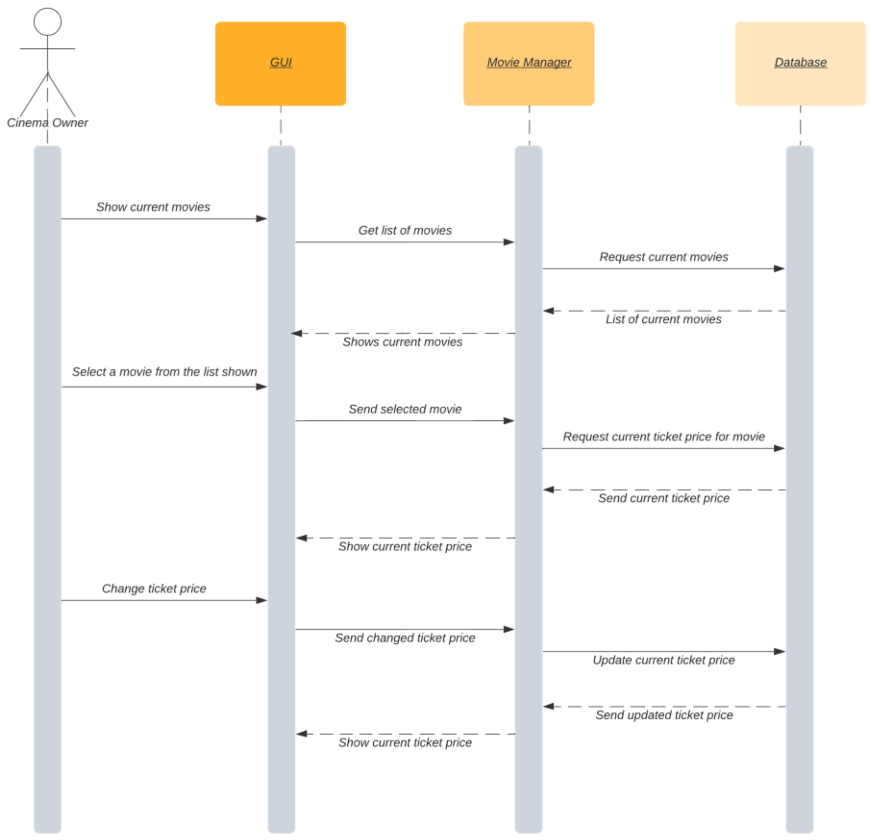

Cinema Owner
GUI
Movie Manager
Database
Show current movies
Get list of movies
Request current movies
List of current movies
Shows current movies
Select a movie from the list shown
Send selected movie
Request current ticket price for movie
Send current ticket price
Show current ticket price
Change ticket price
Send changed ticket price
Update current ticket price
Send updated ticket price
Show current ticket price

Figure 13 : Change ticket price Diagram

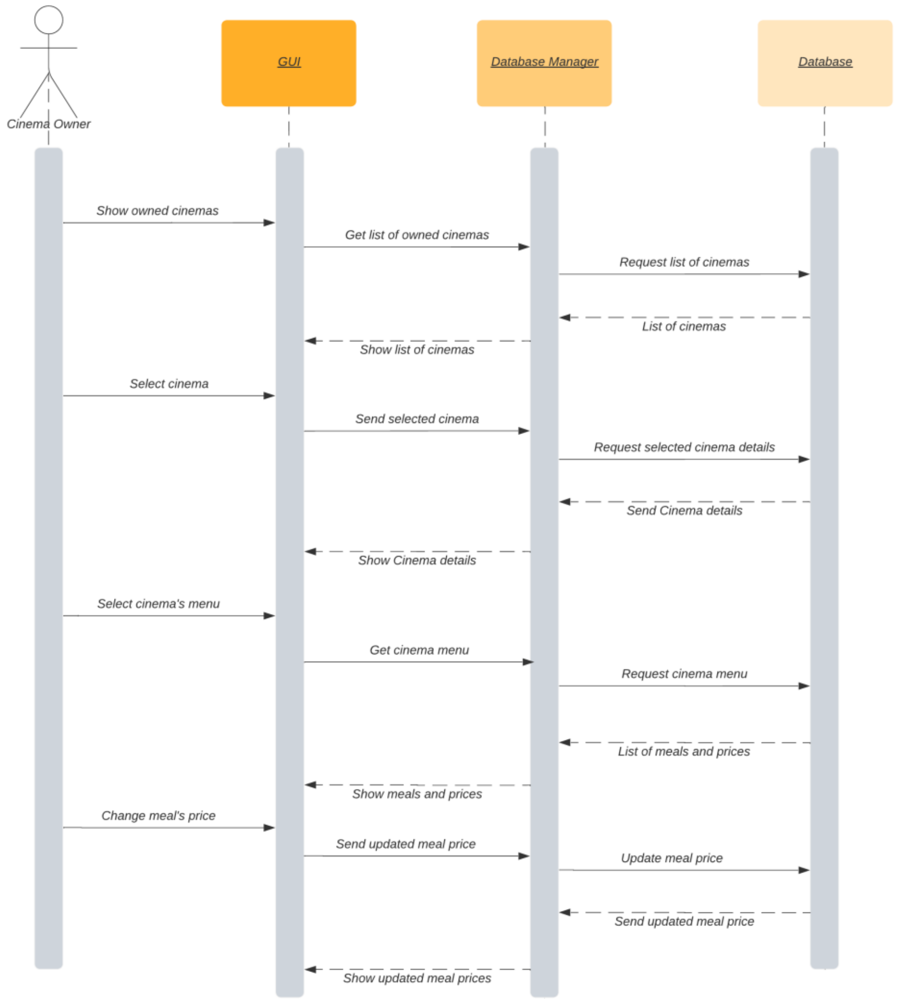


Figure 14 : Update menu price Diagram

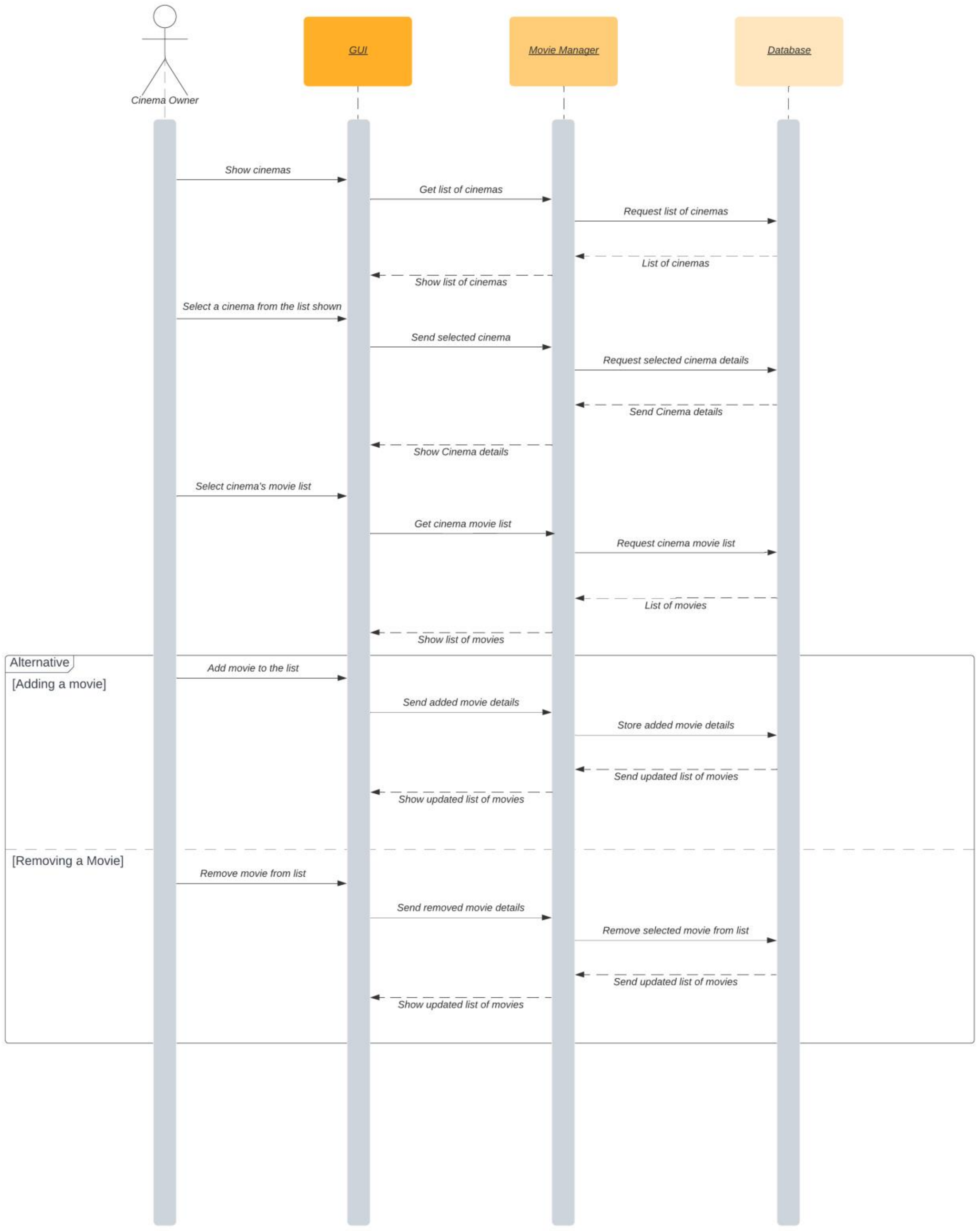


Figure 15 : Add\Remove movies Diagram

7. Noun Extraction and CRC Cards

Using the Scope section, We can perform the noun extraction process

7.1. All General Description Nouns

All the nouns contained in the general description are:

1. Cinema
2. Person
3. Movie
4. Ticket
5. Food
6. User
7. Branch
8. Cinema Owner
9. Reservation
10. Hall
11. Clients
12. Information
13. Cinema Number
14. Services
15. Location
16. Crew
17. Time
18. Password
19. Duration

20.Price

21.Snack

22.Drink

23.Size

24.Actor

7.2. Extracted Nouns

After removing the general nouns and renaming the nouns

1. Cinema
2. Person
3. User
4. Cinema Owner
5. Snack
6. Movie
7. Drink
8. Food
9. Branch
10. Actor
11. Crew

7.3. CRC Diagrams

7.3.1. First iteration

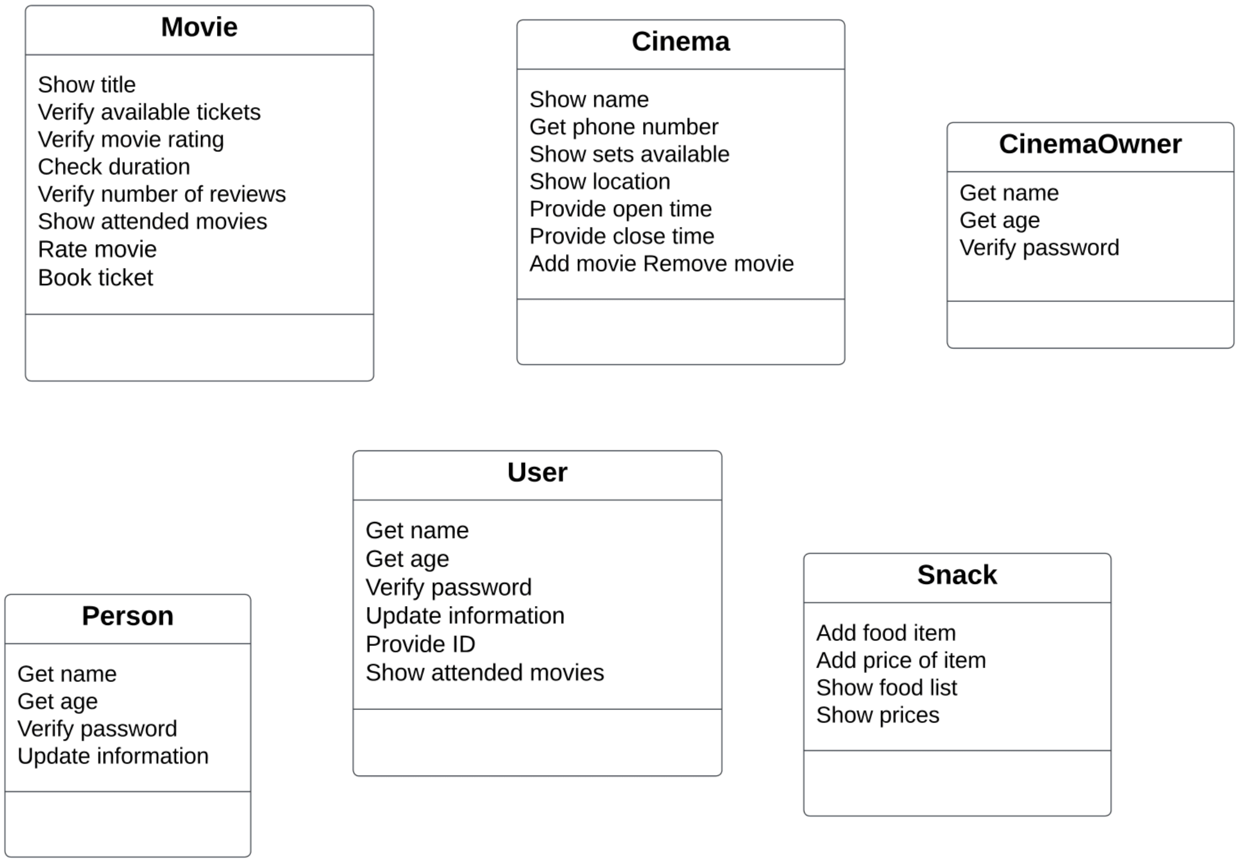


Figure 16 : CRC Diagrams First iteration

7.3.2. Second iteration

Movie
Show title Verify available tickets Verify movie rating Check duration Verify number of reviews Sends message to **User** to show attended movies
Class **User**

Cinema
Show name Get phone number Show sets available Show location Provide open time Provide close time Add movie Remove movie Sends message to **CinemaOwner** to get name Sends message to **Movie** to show title Sends message to **Movie** to check duration Sends message to **Movie** to verify number of reviews Sends message to **Movie** to verify available tickets Sends message to **Movie** to verify movie rating Sends message to **User** to show attended movies Sends message to **Snack** to show snack list Sends message to **Snack** to show prices
Class **CinemaOwner** Class **Movie** Class **User** Class **Snack**

CinemaOwner
Sends message to **Person** to get name Sends message to **Person** to get age Sends message to **Person** to verify password Sends message to **Cinema** to show name Sends message to **Cinema** to add movie Sends message to **Cinema** to remove movie Sends message to **Cinema** to provide open time Sends message to **Cinema** to provide close time Sends message to **Cinema** to show sets available Sends message to **Cinema** to show location Sends message to **Snack** to add snack item Sends message to **Snack** to add price of item
Class **Person** Class **Snack** Class **Cinema**

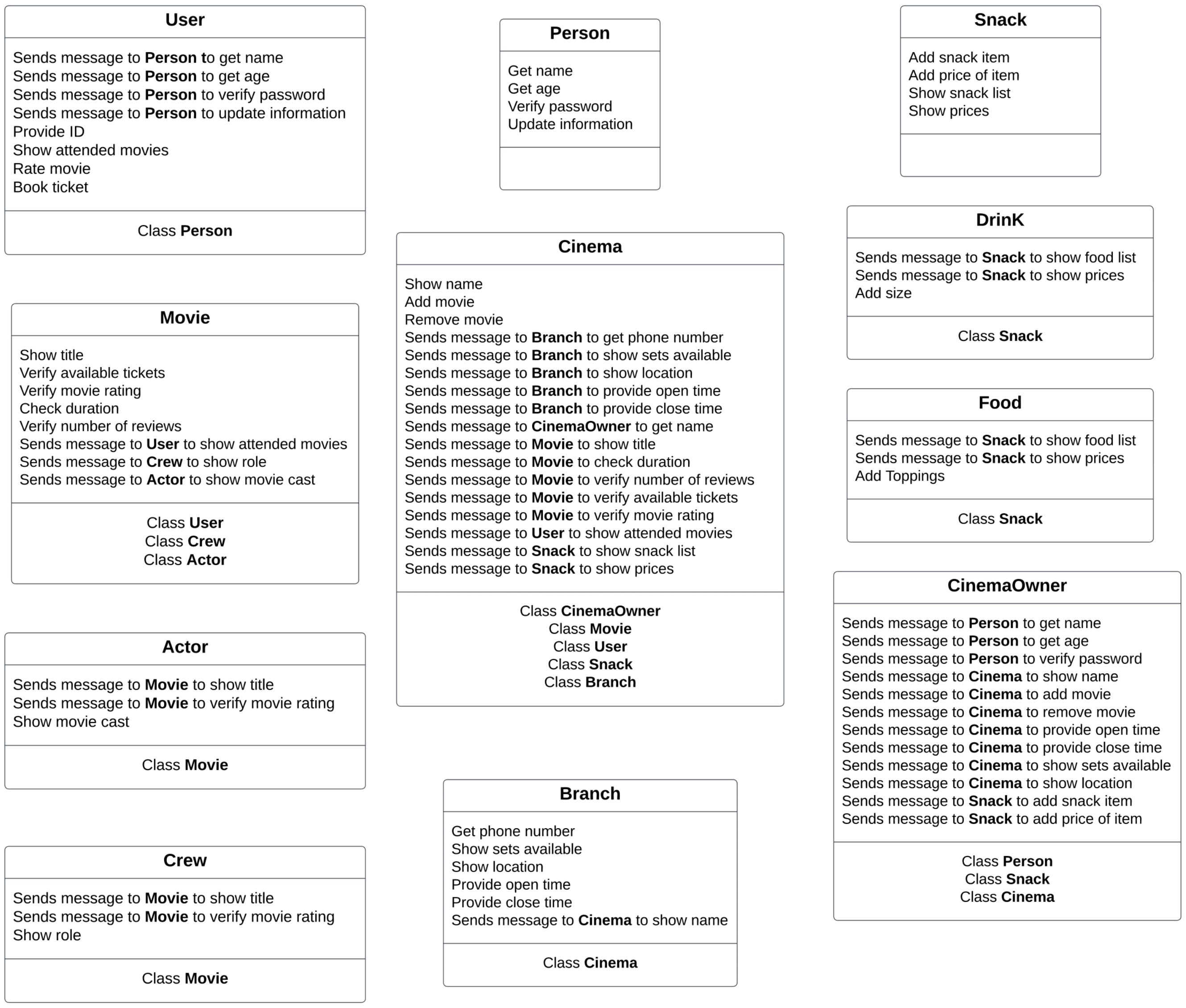
User
Sends message to **Person t**o get name Sends message to **Person** to get age Sends message to **Person** to verify password Sends message to **Person** to update information Provide ID Show attended movies Rate movie Book ticket
Class **Person**

Snack
Add snack item Add price of item Show snack list Show prices

Person
Get name Get age Verify password Update information

Figure 17 : CRC Diagrams Second iteration

7.3.3. Third iteration



User

Sends message to **Person** to get name
Sends message to **Person** to get age
Sends message to **Person** to verify password
Sends message to **Person** to update information
Provide ID
Show attended movies
Rate movie
Book ticket

Class **Person**

Person

Get name
Get age
Verify password
Update information

Snack

Add snack item
Add price of item
Show snack list
Show prices

DrinK

Sends message to **Snack** to show food list
Sends message to **Snack** to show prices
Add size

Class **Snack**

Cinema

Show name
Add movie
Remove movie
Sends message to **Branch** to get phone number
Sends message to **Branch** to show sets available
Sends message to **Branch** to show location
Sends message to **Branch** to provide open time
Sends message to **Branch** to provide close time
Sends message to **CinemaOwner** to get name
Sends message to **Movie** to show title
Sends message to **Movie** to check duration
Sends message to **Movie** to verify number of reviews
Sends message to **Movie** to verify available tickets
Sends message to **Movie** to verify movie rating
Sends message to **User** to show attended movies
Sends message to **Snack** to show snack list
Sends message to **Snack** to show prices

Class **CinemaOwner**
Class **Movie**
Class **User**
Class **Snack**
Class **Branch**

Movie

Show title
Verify available tickets
Verify movie rating
Check duration
Verify number of reviews
Sends message to **User** to show attended movies
Sends message to **Crew** to show role
Sends message to **Actor** to show movie cast

Class **User**
Class **Crew**
Class **Actor**

Food

Sends message to **Snack** to show food list
Sends message to **Snack** to show prices
Add Toppings

Class **Snack**

CinemaOwner

Sends message to **Person** to get name
Sends message to **Person** to get age
Sends message to **Person** to verify password
Sends message to **Cinema** to show name
Sends message to **Cinema** to add movie
Sends message to **Cinema** to remove movie
Sends message to **Cinema** to provide open time
Sends message to **Cinema** to provide close time
Sends message to **Cinema** to show sets available
Sends message to **Cinema** to show location
Sends message to **Snack** to add snack item
Sends message to **Snack** to add price of item

Class **Person**
Class **Snack**
Class **Cinema**

Actor

Sends message to **Movie** to show title
Sends message to **Movie** to verify movie rating
Show movie cast

Class **Movie**

Branch

Get phone number
Show sets available
Show location
Provide open time
Provide close time
Sends message to **Cinema** to show name

Class **Cinema**

Crew

Sends message to **Movie** to show title
Sends message to **Movie** to verify movie rating
Show role

Class **Movie**

Figure 18 : CRC Diagrams Third iteration

8. Class Diagram

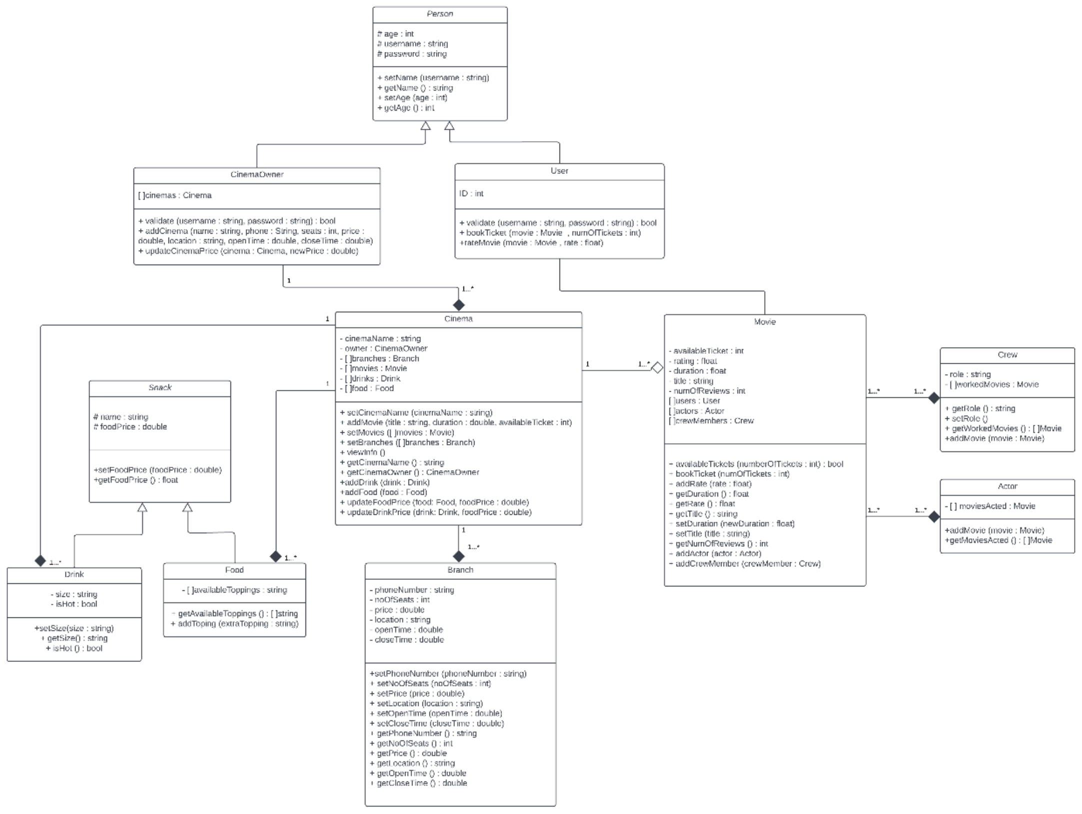

Person
# age : int
# username : string
# password : string
+ setName (username : string)
+ getName () : string
+ setAge (age : int)
+ getAge () : int
CinemaOwner
[ ]cinemas : Cinema
+ validate (username : string, password : string) : bool
+ addCinema (name : string, phone : String, seats : int, price : double, location : string, openTime : double, closeTime : double)
+ updateCinemaPrice (cinema : Cinema, newPrice : double)
User
ID : int
+ validate (username : string, password : string) : bool
+ bookTicket (movie : Movie , numOfTickets : int)
+rateMovie (movie : Movie , rate : float)
1
1...*
Cinema
- cinemaName : string
- owner : CinemaOwner
- [ ]branches : Branch
- [ ]movies : Movie
- [ ]drinks : Drink
- [ ]food : Food
+ setCinemaName (cinemaName : string)
+ addMovie (title : string, duration : double, availableTicket : int)
+ setMovies ([ ]movies : Movie)
+ setBranches ([ ]branches : Branch)
+ viewInfo ()
+ getCinemaName () : string
+ getCinemaOwner () : CinemaOwner
+addDrink (drink : Drink)
+addFood (food : Food)
+ updateFoodPrice (food: Food, foodPrice : double)
+ updateDrinkPrice (drink: Drink, foodPrice : double)
Movie
- availableTicket : int
- rating : float
- duration : float
- title : string
- numOfReviews : int
[ ]users : User
[ ]actors : Actor
[ ]crewMembers : Crew
- availableTickets (numberOfTickets : int) : bool
- bookTicket (numOfTickets : int)
- addRate (rate : float)
- getDuration () : float
- getRate () : float
- getTitle () : string
- setDuration (newDuration : float)
- setTitle (title : string)
- getNumOfReviews () : int
- addActor (actor : Actor)
- addCrewMember (crewMember : Crew)
Crew
- role : string
- [ ]workedMovies : Movie
+ getRole () : string
+ setRole ()
+ getWorkedMovies () : [ ]Movie
+addMovie (movie : Movie)
Actor
- [ ] moviesActed : Movie
+addMovie (movie : Movie)
+getMoviesActed () : [ ]Movie
Snack
# name : string
# foodPrice : double
+setFoodPrice (foodPrice : double)
+getFoodPrice () : float
Drink
- size : string
- isHot : bool
+setSize(size : string)
+ getSize() : string
+ isHot () : bool
Food
- [ ]availableToppings : string
+ getAvailableToppings () : [ ]string
+ addToping (extraTopping : string)
Branch
- phoneNumber : string
- noOfSeats : int
- price : double
- location : string
- openTime : double
- closeTime : double
+setPhoneNumber (phoneNumber : string)
+ setNoOfSeats (noOfSeats : int)
+ setPrice (price : double)
+ setLocation (location : string)
+ setOpenTime (openTime : double)
+ setCloseTime (closeTime : double)
+ getPhoneNumber () : string
+ getNoOfSeats () : int
+ getPrice () : double
+ getLocation () : string
+ getOpenTime () : double
+ getCloseTime () : double

Figure 19 : Class Diagram

- Classes **User** and **CinemaOwner** represents the target actors of the software, they both inherit from the abstract class ***Person*** (represented in italics in the diagram) that contains the shared attributes and methods between them.
- Class **Cinema** represents the heart of the system and contains lists of objects from classes **Movie**, **Branch**, **Food**, and **Drinks** provided in each cinema.
- Class **Branch** contains information about each branch in the cinema.
- Classes **Food** and **Drinks** represents available snacks in each cinema, they both inherit from the abstract class ***Snack*** (represented in italics in the diagram) that contains the shared attributes and methods between them.
- Class **Movie** contains info about the movie showed in cinemas as well as the users that reserved a ticket for it, it also includes information about crew members in the movie divided into actors and crews (with roles like director and producers).
- Classes **Actor** and **Crew** represents movie's crew members.

9. State Diagram

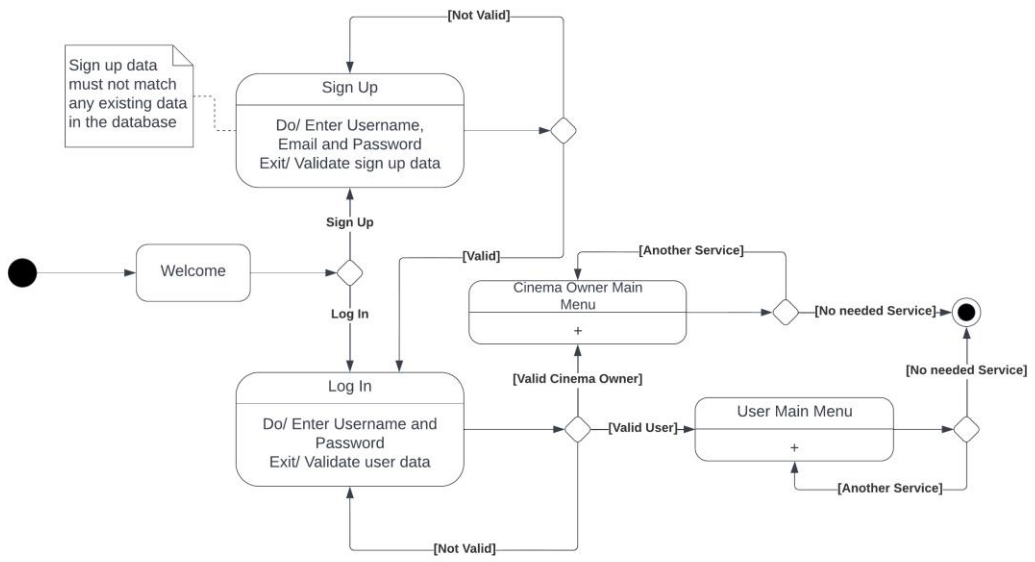


Figure 20 : Title screen State Diagram

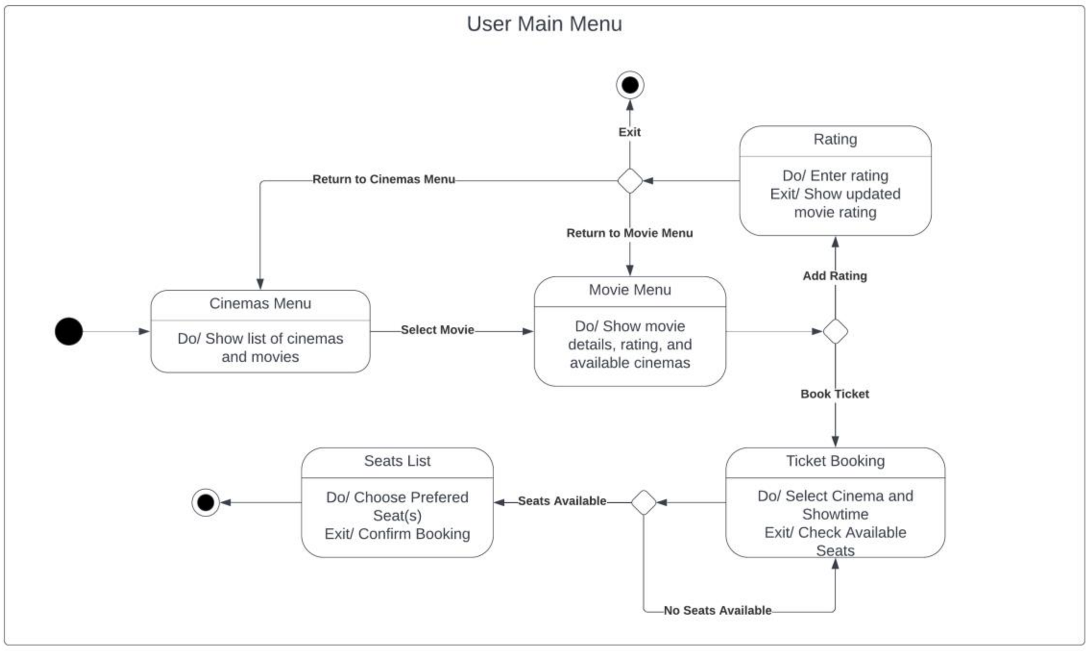


Figure 21 : User State Diagram

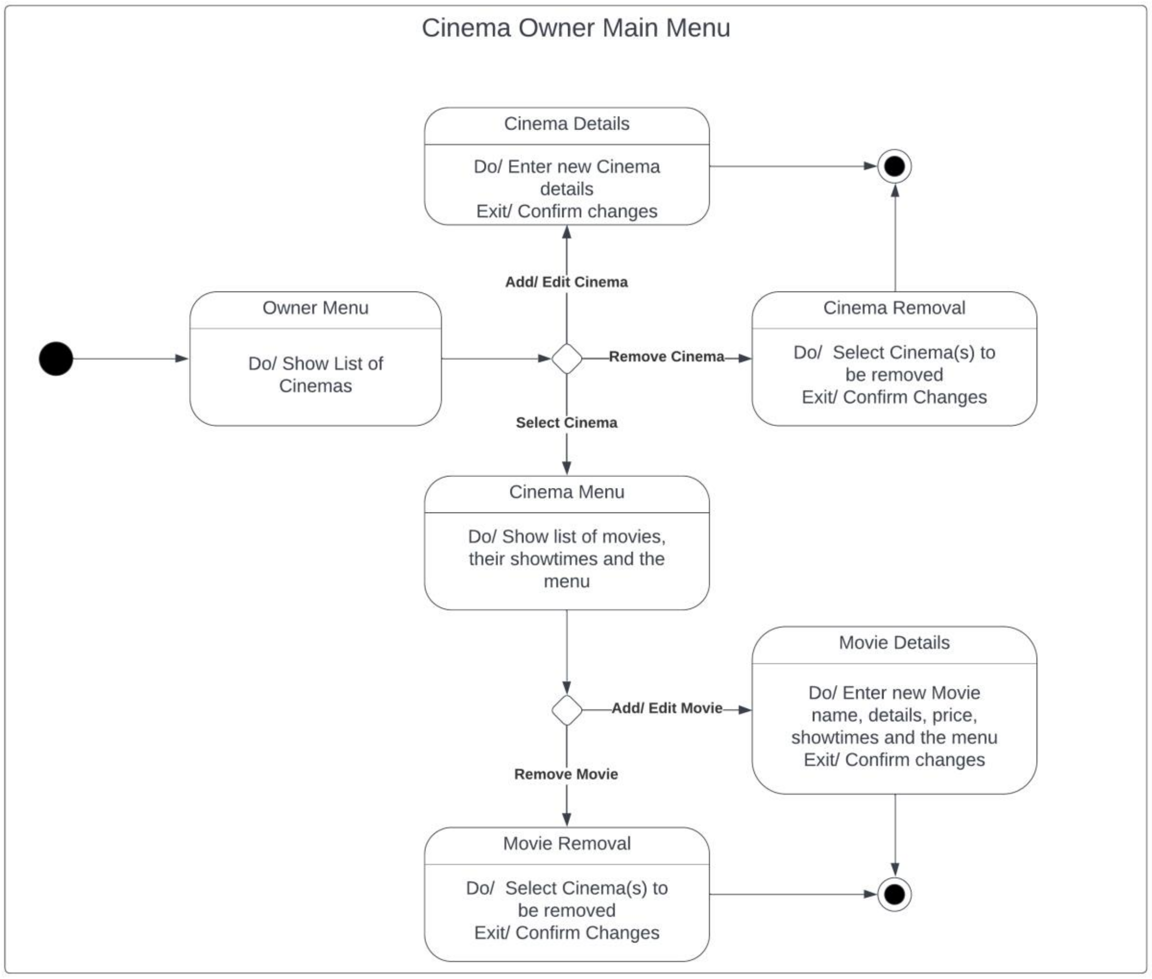


Figure 22 : Cinema-Owner Screen State Diagram

10. Client-Object Relation Diagram

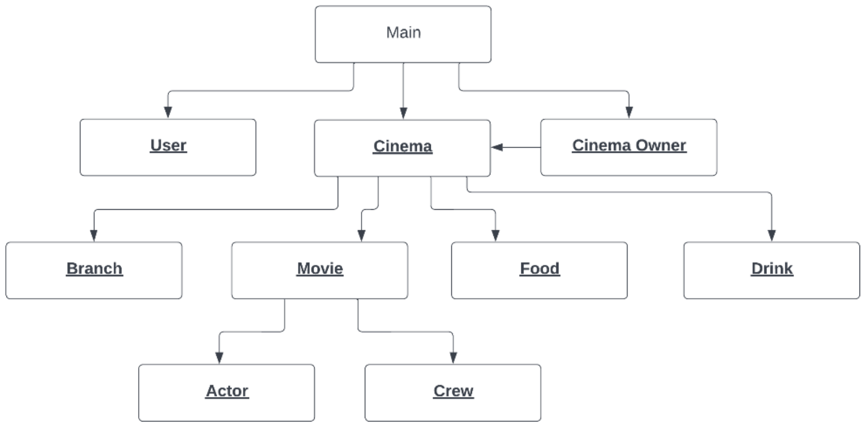


Figure 23 : Client-Object relation diagram

- The Diagram is based on the CRC cards, abstract classes are ignored since the diagram represents relations between objects.
- The main instantiates an object from class User and CinemaOwner during login/signup process, objects from class Cinema are instantiated to let the user browse movies inside them.

11. OOAD Methodologies

11.1. OOA and OOD

Divided into two main phases, the OOAD mainly consists of first the Object-Oriented Analysis phase, then the Object-Oriented Design phase.

First, Object Oriented Analysis. The OOA is based on three types of modeling:

1- Use case modeling, where it describes how the system is interacted by external entities. See **Use-Case Diagram and the Narrative Description of the use cases**.

2- Class modeling, which determines the classes and their attributes. check **Extracted Nouns** and **After removing** the general nouns and renaming the nouns

1. Cinema
2. Person
3. User
4. Cinema Owner
5. Snack
6. Movie
7. Drink
8. Food
9. Branch
10. Actor
11. Crew

3- **CRC Diagrams**. The final **Class Diagram** can be found here.

4- Dynamic modeling, where we define the actions performed by each class. Look at **State Diagram** here to see a sequence of states an object can assume during its lifetime.

Second, Object Oriented Design. Here we have four steps:

1- Interaction diagrams, the constructing of interaction diagrams for each scenario involves sequence diagrams and collaboration diagrams which you can find here: **Interaction Diagrams**.

2- Detailed class diagram, check **Class Diagram**.

3- Design product in terms of clients of objects, see **Client-Object Relation Diagram**.

4- Performing detailed design where we implemented the software system.

11.2. OMT Methodology

OMT describes a method for the analysis, design, and implementation of a system using an object-oriented technique, and consists of four phases:

1- Analysis: here we identified objects and relationships between them. The main goal of this phase is to build a model of real-world situations. This phase is concerned with preparation of correct and accurate modeling of the real world. It starts with setting a goal, finding the problem statement which is then extended into models (object model, dynamic model, functional model):

1- Object model: it focuses on classes and associations with attributes and operations. Check **Class Diagram**.

2- Dynamic model: it focuses on states(of major objects), and transitions between states. It also included the events to move from one state to another Done to target operations in object. See **State Diagram**.

3- Functional model: see which data that should be in the system . It focuses on processes, data store and data flow. See **Appendix 1 : DFD**.

2- System design: Determines the whole system architecture using subsystems High-level architecture divides the system into sub-systems and packages.

3- Object design: Implementation plan is established Classifies classes attributes, operations and associations Documentation of detailed objects and dynamic and functional models.

4- Implementation: Matter of translating the design into a programming language constructs (software).

11.3. The Booch Methodology

Booch methodology covers the analysis and design phases of systems development.

It's broken into two processes:

1. The macro development process, which consists of the following:

   1.1. Conceptualization: we established the code requirements of the system. Setting the goals and developing a prototype to prove the concept.

1.2. Analysis and development of the model: here we use the **Class Diagram** to describe the roles and responsibilities of objects.

1.3. Design or create the system architecture: here we use the model diagram to map out where each class object should be declared.

1.4. Evolution or implementation: where we produce a stream or software implementations, each of which is refinement of the prior one.

1.5. Maintenance.

2. The micro development process, including:

   2.1. Identifying classes and objects: We define the problem, identify what is and what is not part of our problem domain.

   2.2. Identifying class and object semantics: We proceeded with our list of things from the first step and develop the semantics of our classes and objects. Here we ended up with interface files for our classes.

   2.3. Identifying class and object relationships: We formalized in our diagrams the relationships between classes and objects. Associations are between classes.

   2.4. Identifying class and object interfaces and implementation: we do just enough of this to help show gaps and lacks with the design. Eventually the implementation becomes more and more real.

The following are the diagrams used in the Booch methodology:

1. Class diagram: see **Class Diagram**.
2. State transition diagram: see **State Diagram**.
3. Interaction diagram: see **Interaction Diagrams**.
4. Module diagram: see **Component Diagram**.

12. Comparative Analysis of the Output of the Adopted Methodologies

The analysis and design of the application was mainly developed using **The Booch Methodology**. For the following reasons:

1- It covers static structure and dynamic behavior efficiently.
2- It is very suitable for producing detailed Object oriented design models.
3- It divides the process of development of the project into micro and macro processes. This helped us start big with a fully documented object oriented code which made the process of development mush easier.
4- The main advantage here is that the methodology is cyclical and accounts for incremental improvements that are made in the evolution of a product.

We also used some methodologies from the **OMT Methodology** because:

1- As they are easy and cover functionality.
2- Like in the Booch methodologies, cover the static structure and dynamic behavior.
3- Using the OMT methodology we managed to simulate entities before construction.
4- In addition, it was easy for us to reduce the complexity through visualization.

Although we could have used the Jacobson Methodology, we didn't. as:

1- Its main focus is on use cases, and use cases were very clear and not complicated at all. (See **Use-Case Diagram and the Narrative Description of the use cases**).
2- The problem with this methodology lies in its complexity to introduce in a company since it covers the whole software life cycle.
3- Also, the transition from analysis objects to design blocks (and the blocks classes) is not described in a clear way which makes it difficult to maintain the traceability. There are two main reasons, although it could have provided robustness to the software design, were more than enough for us to not use it.

13. Architectural Model

Building a software system needs a study of several architecture styles to wisely choose the most suitable style that correctly defines the system's structure, The choice in this project was to implement a merged architectural style to combine the benefits of several architectural styles.

The chosen styles were Data-Centered, Object-Oriented architecture style, Model-View-Controller (MVC).

13.1. Data Centered Architecture Style

The data is centralized and accessed frequently by the other software components that communicate with each other by reading, writing and modifying data in the shared database so they are relatively independent.

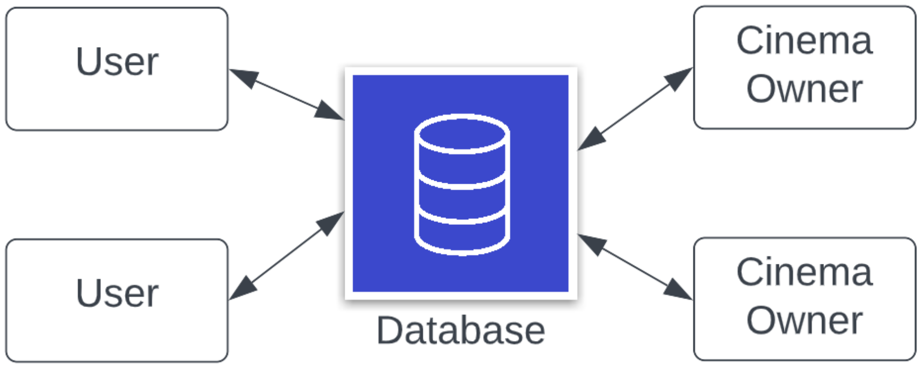


Figure 24 : Data centered Architecture

Benefits:

- Provides data integrity, backup and restore features.
- Provides scalability and reusability of agents, the user and the cinema owner do not have direct communication with each other.
- Reduces overhead of transient data between software components.

13.2. Object Oriented Architecture Style

The responsibilities of the system are divided into individual reusable and self-sufficient objects. Thus, viewing the software system as a collection of entities known as objects. Object oriented is based on modeling real-world objects.

See **Class Diagram** for the illustration of this architectural style.

Benefits:

- Object-Oriented architecture maps the application to real world objects for making it more understandable.

- It is easy to maintain and improves the quality of the system due to reusability.
- This architecture provides reusability through polymorphism and abstraction and improve testability through encapsulation.
- It has ability to manage the errors during execution. (Robustness)
- It has ability to extend new functionality and does not affect on the system.
- Object-Oriented architecture reduces the development time and cost.

13.3. Model-View-Controller (MVC) Architecture Style

The logic, user interface, and data storage are isolated. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response.

Model: The lowest level of the pattern which is responsible for maintaining data.

View: This is responsible for displaying all or a portion of the data to the user.

Controller: Logic Code that controls the interaction between the Model and View.

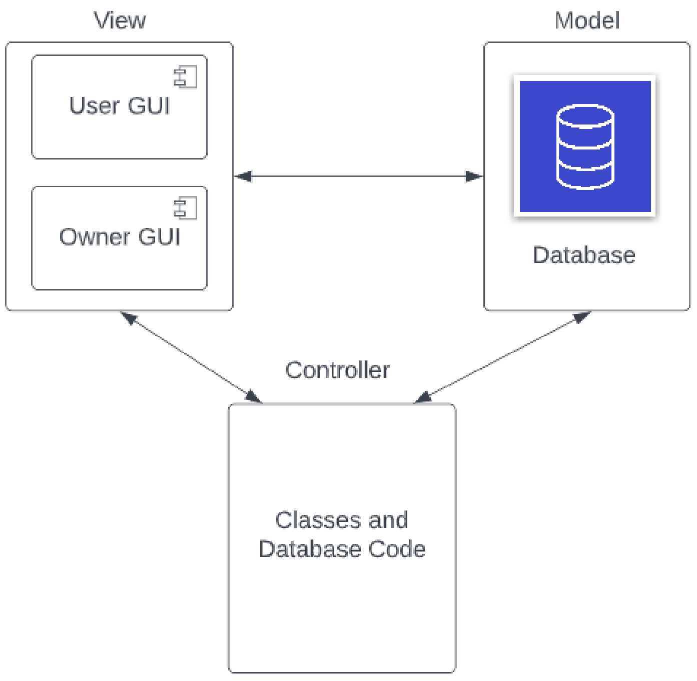


Figure 25 : Model-View-Controller (MVC) Architecture

Benefits:

- Accelerates development and make best use of the development team’s skills, those who are skilled in Front-End-Development were assigned to the View, those who are skilled in Databases were assigned to Model, the rest implemented the logic in Controller.
- Supports multiple views, for User and for Cinema Owner, each with its own data format.
- Easy planning, modifications, and maintenance.

13.4. Merged Architecture Style

Combining the benefits of the three architectural styles, the design choice is a merged architectural style centered on MVC style, with a data-centered style for the system model and object-oriented style accompanied with the database code in the Controller, and two views for the User and Cinema Owner.

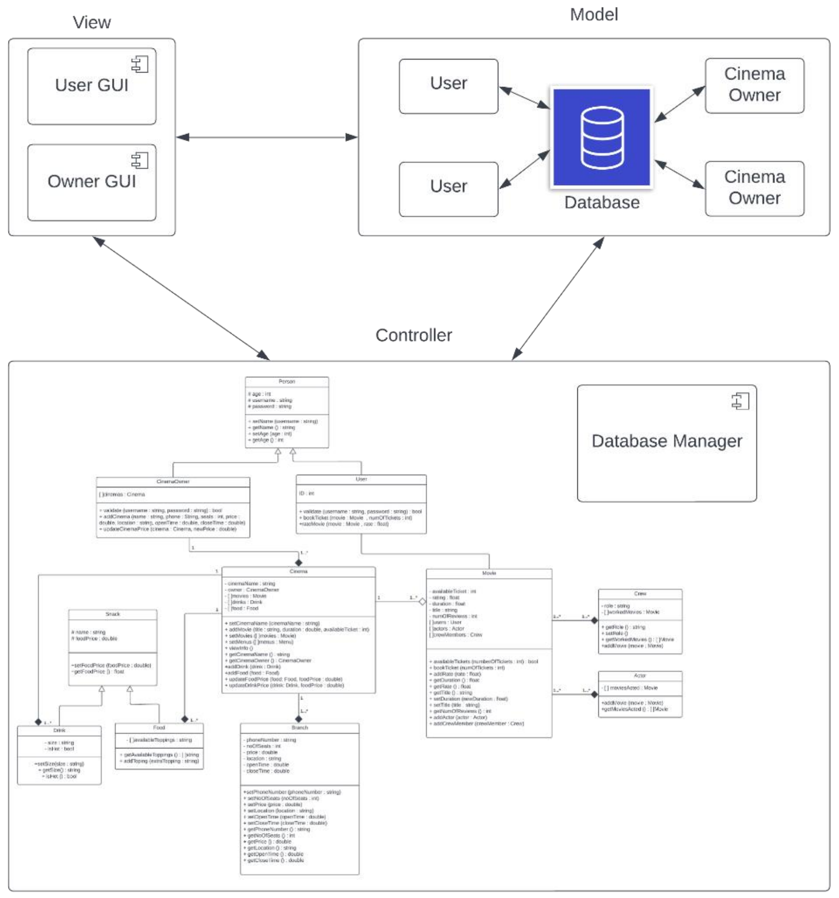


Figure 26 : Merged Architecture

14. Component Diagram

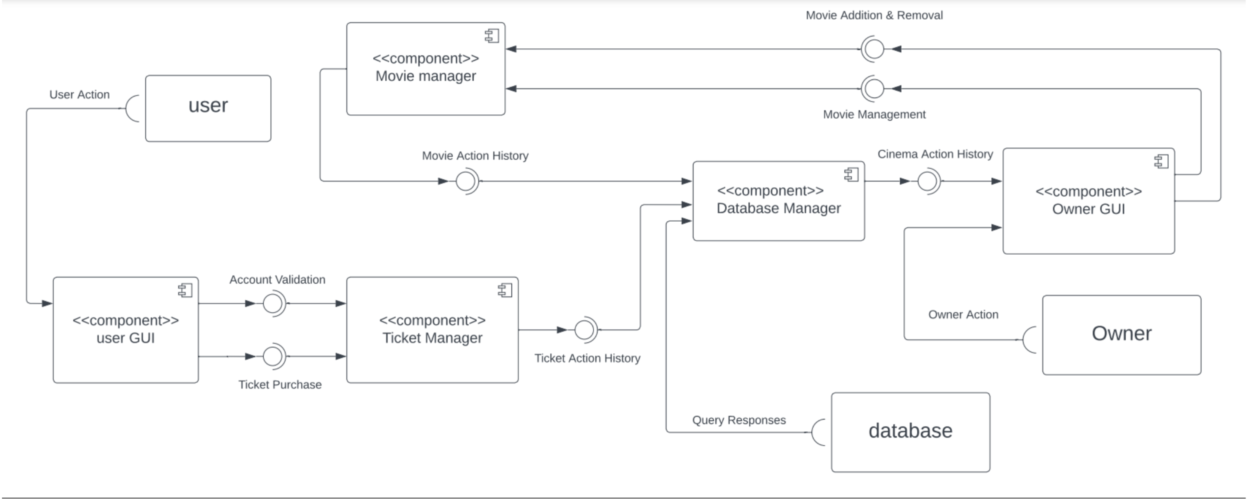


Figure 27 : Component Diagram

15. Time Plan

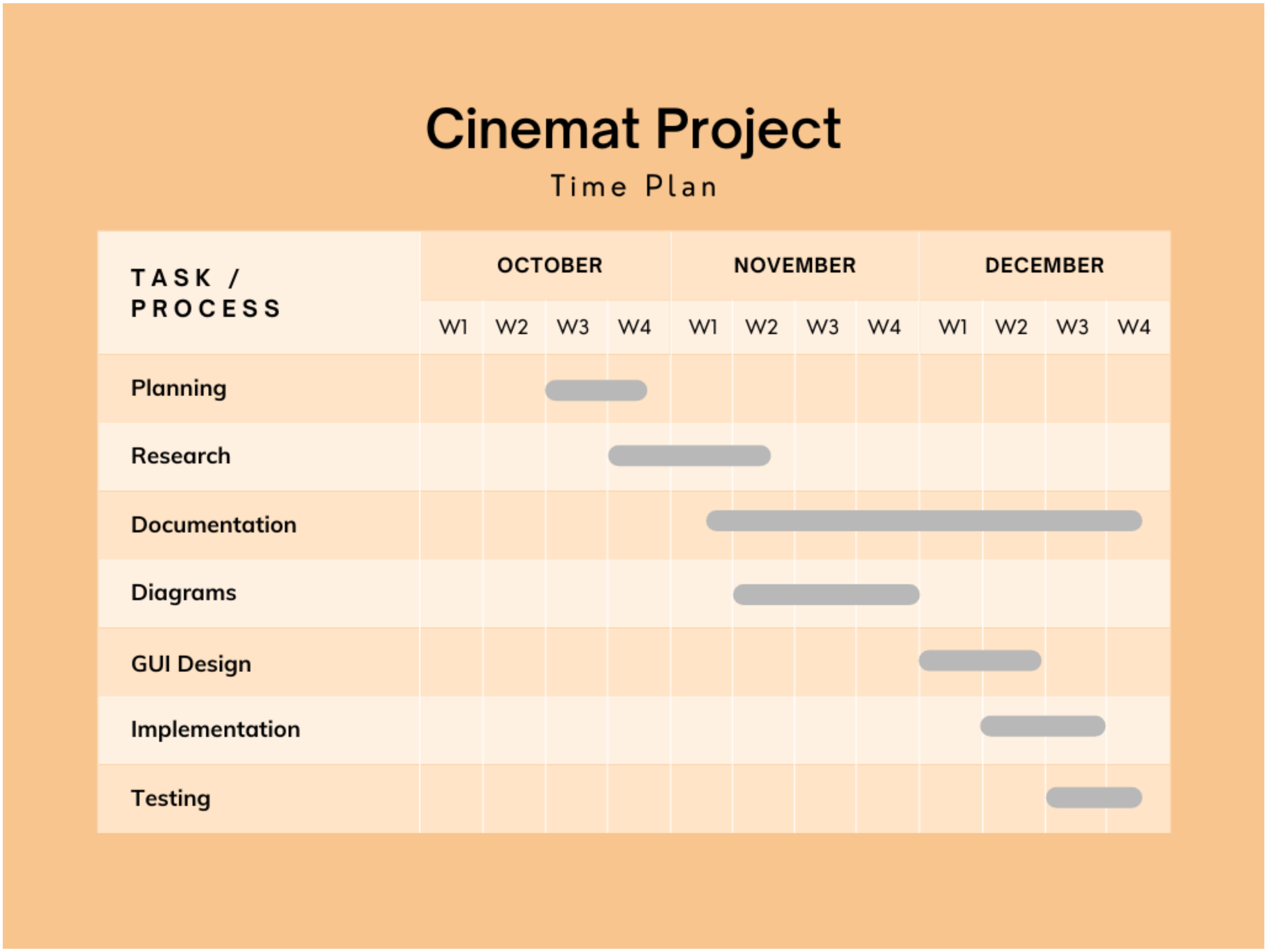


Figure 28 : Time plan Gantt chart

16. User Guide

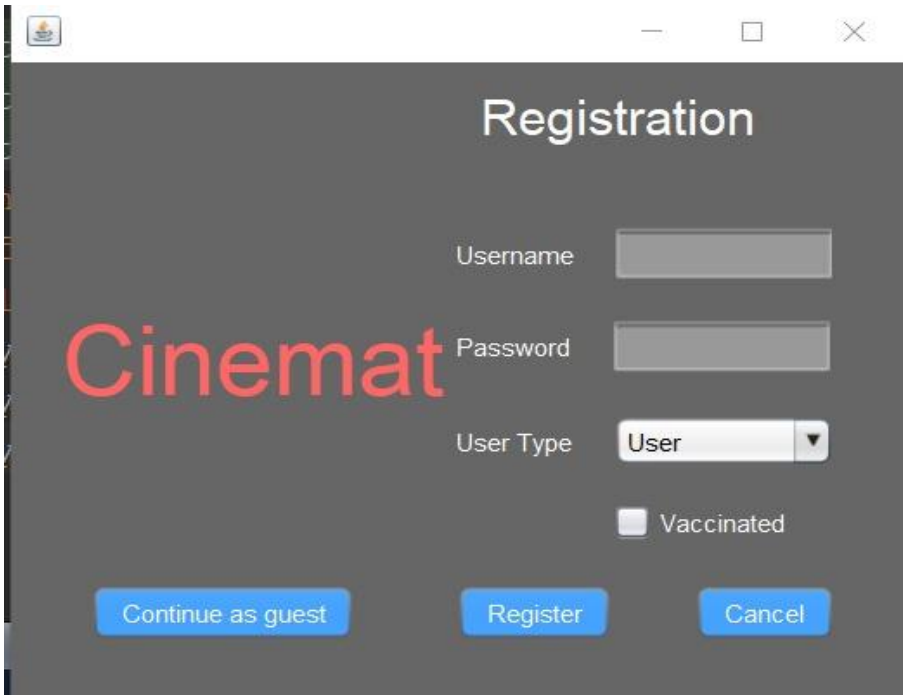


Figure 29 : Registration Screen

16.1. Guest User Guide

If user chooses to continue as a guest, they can choose a cinema to get its info

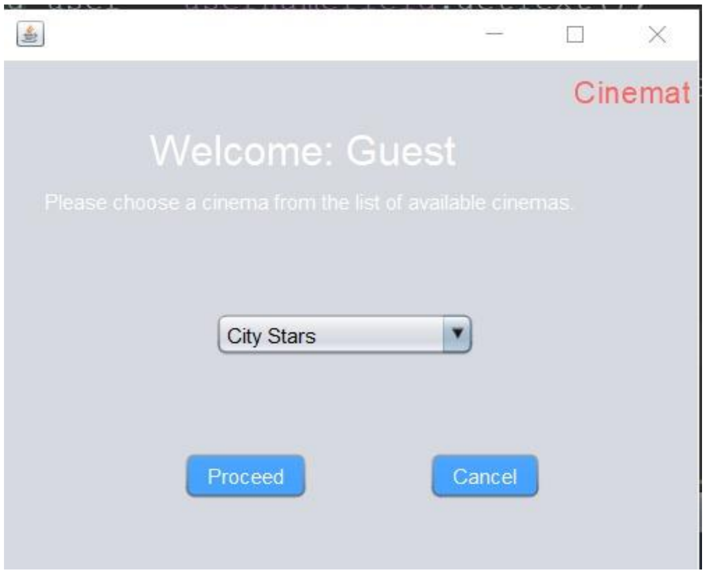
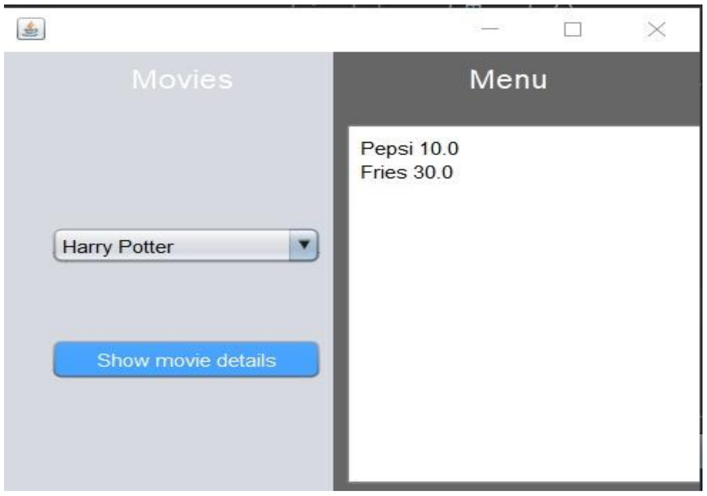


Figure 30 : Guest user interface (1)

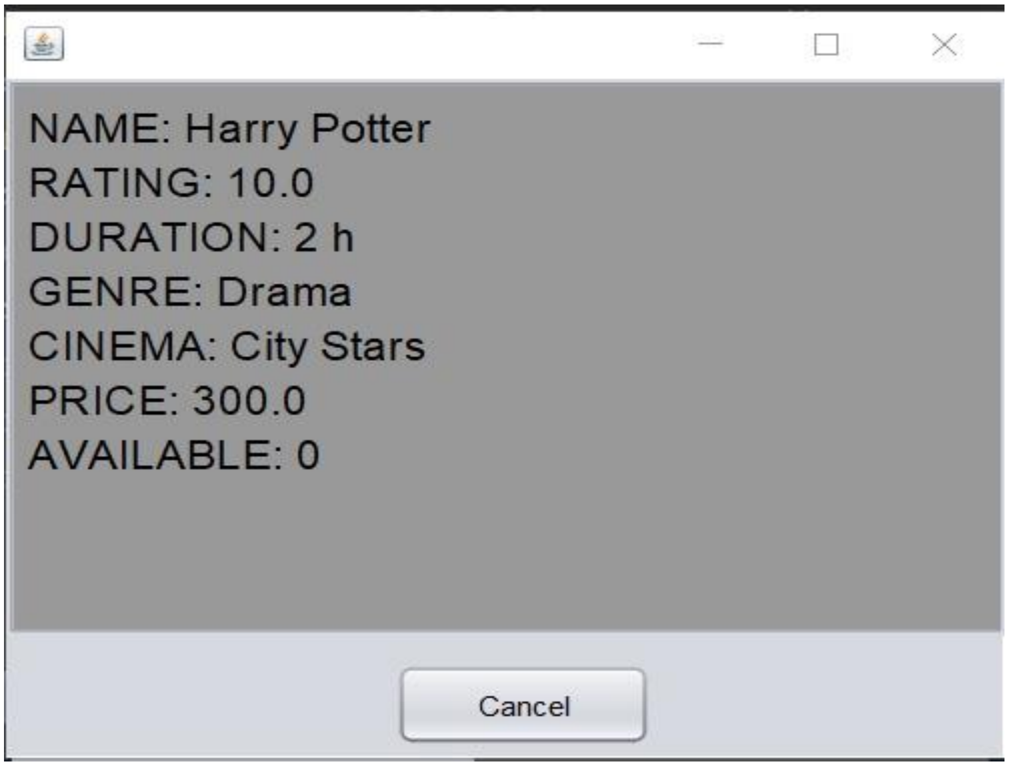
Menu and snacks available at the chosen cinema are viewed and the guest can select a movie to view its description.



Pepsi 10.0
Fries 30.0

Figure 31 : Guest user interface (2)

Various movie details are shown. Name, rating, duration, Genre, Cinema, ticket price, and number of available tickets.



NAME: Harry Potter
RATING: 10.0
DURATION: 2 h
GENRE: Drama
CINEMA: City Stars
PRICE: 300.0
AVAILABLE: 0

Figure 32 : Movie details Screen

16.2. Cinema Owner User Guide

If the user chooses to login he will be provided with the following screen :-

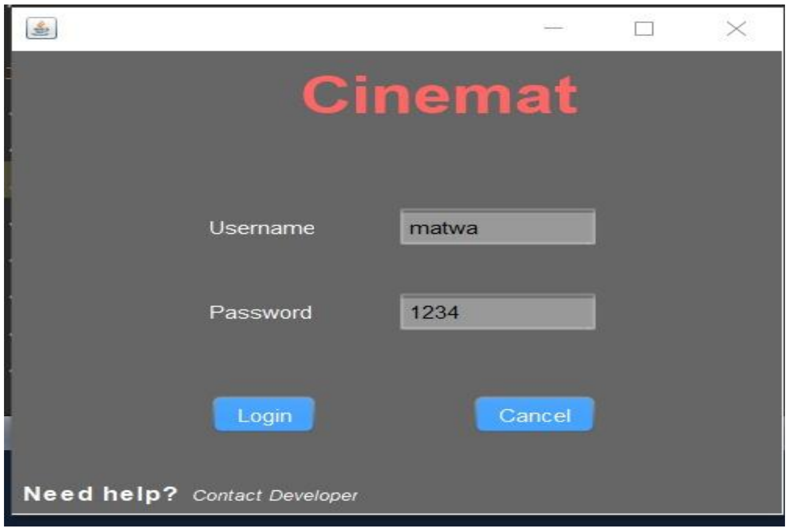


Figure 33 : Login screen

If the user (registered as a Cinema owner) logs in this interface is shown and he can choose what action he would like to do.

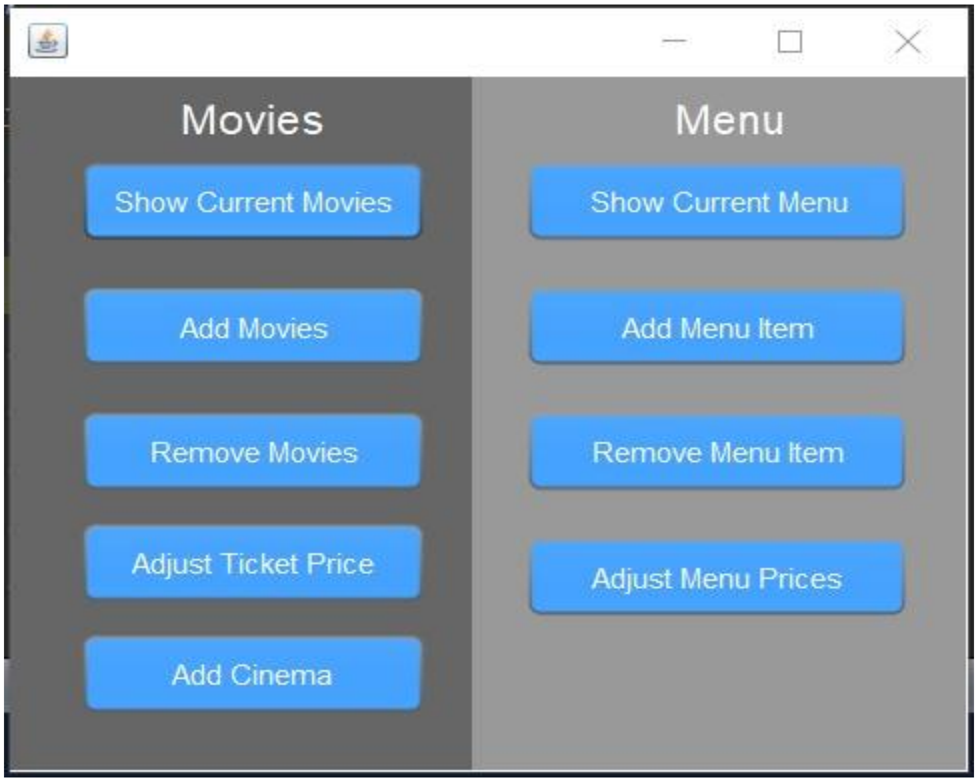
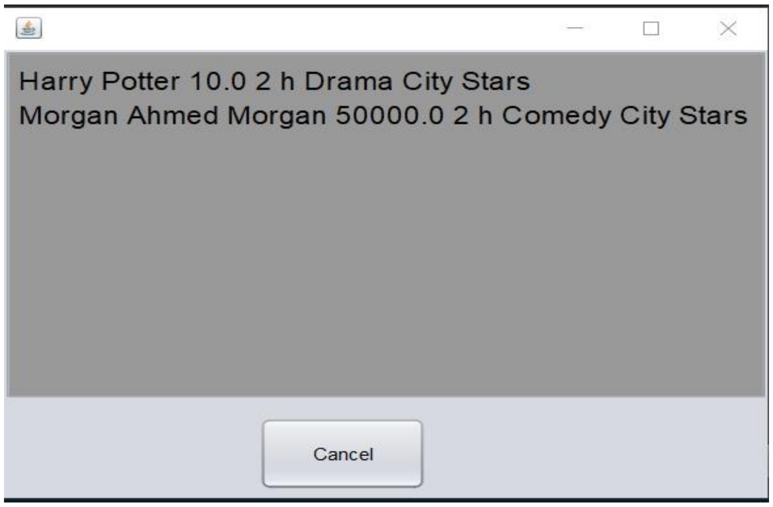


Figure 34 : Cinema Owner user interface

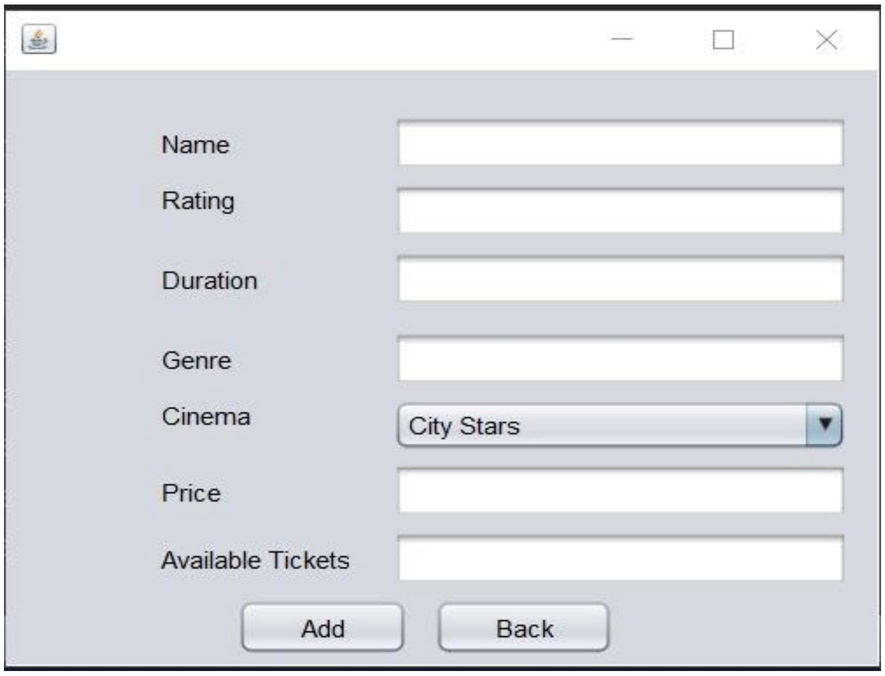
If the owner clicks on show current movies, all movies present in his owned cinemas are shown.



```
Harry Potter 10.0 2 h Drama City Stars
Morgan Ahmed Morgan 50000.0 2 h Comedy City Stars
```

Figure 35 : Show Current Movies screen

If the owner clicks on add movies he chooses the cinema in which he wants to add that movie, enters the movie details and then clicks on the add button.



Name	
Rating	
Duration	
Genre	
Cinema	City Stars
Price	
Available Tickets	

Figure 36 : Add movies screen

If the owner clicks on the remove movie button, he selects a movie, clicks the erase button then this movie is removed from all the cinemas he owns

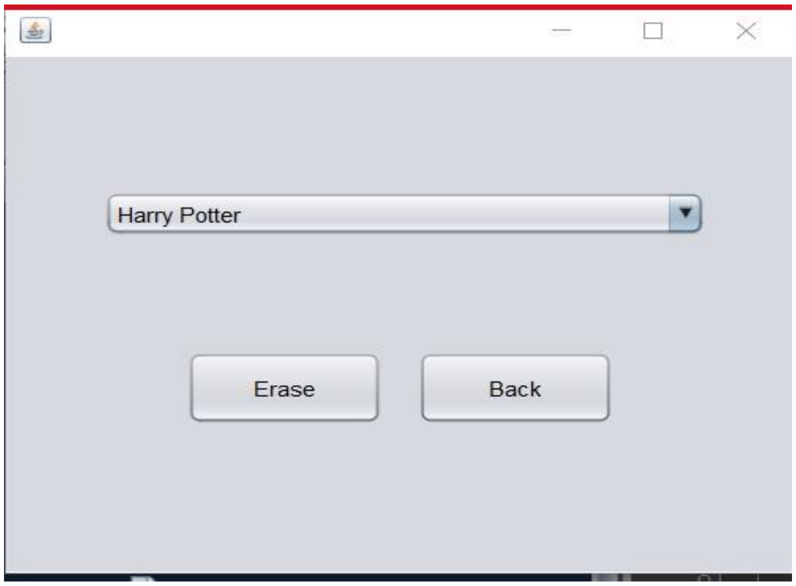


Figure 37 : Remove movie screen

If the owner wants to adjust ticket prices they click on adjust ticket price, choose a movie, then click on get film and enters the new price in text field and then clicks on update.

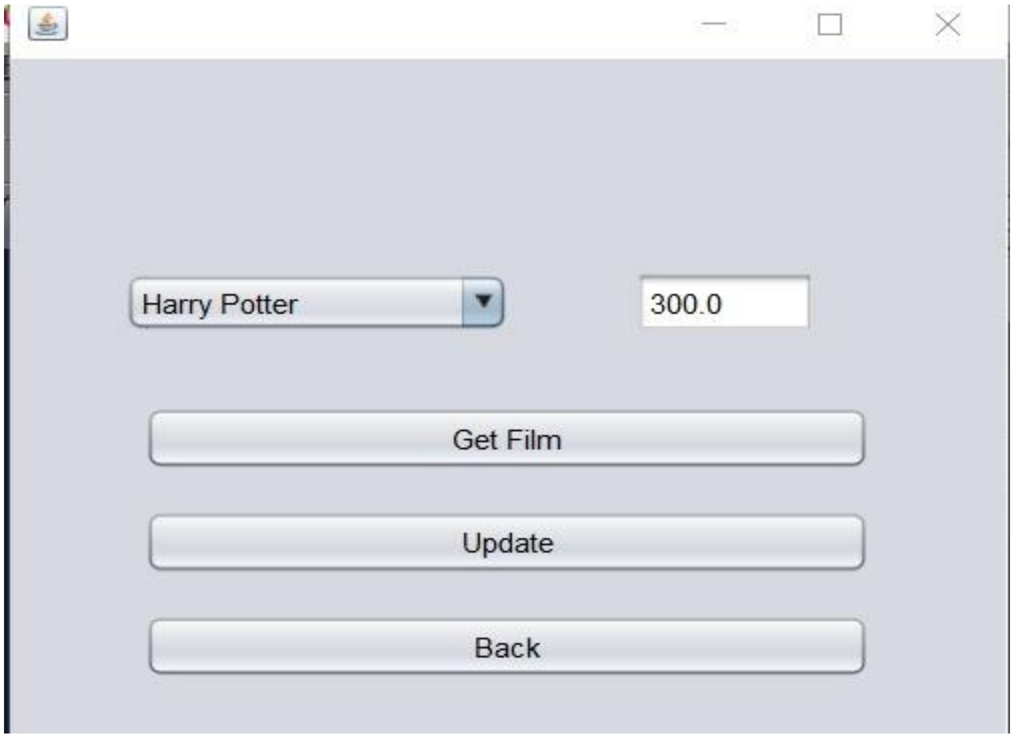


Figure 38 : Adjust ticket price screen

If the owner wants to view the current menu items, he clicks on show current menu, chooses which cinema, then clicks on get items. And all menu items available in that cinema will be shown in a text field

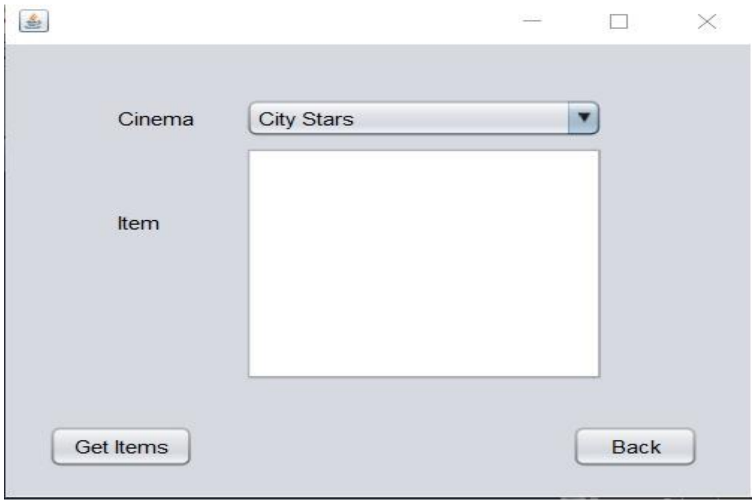


Figure 39 : Show current menu screen

If the owner wants to add a menu item, they click on add menu item, enter its name and price and chooses which cinema they want to add the item in, then click the add button.

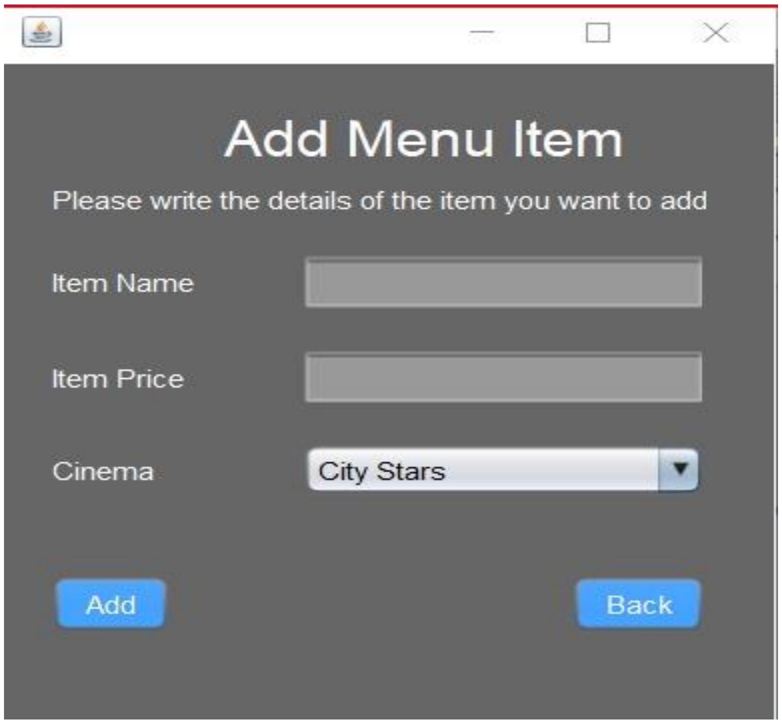


Figure 40 : Add Menu Item screen

If the Owner wants to remove a menu item, he clicks on remove menu item, chooses which cinema he wants to remove it from, and clicks on get items. Hence all available menu items in the chosen cinema will be shown in a combo box. Then the Owner selects the item he wants to remove and then clicks remove button

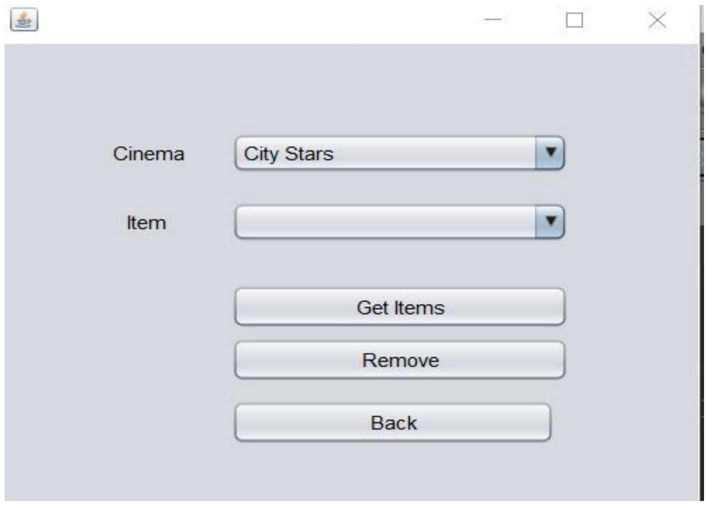


Figure 41 : Remove menu item screen

If the owner wants to adjust price of a menu item. they click on adjust menu price, chooses from which cinema they want to adjust the item price, and clicks on get items. Hence all available menu items in the chosen cinema will be shown in a combo box. Then the owner selects the item he wants to adjust the price for and then clicks update button after inserting a new price.

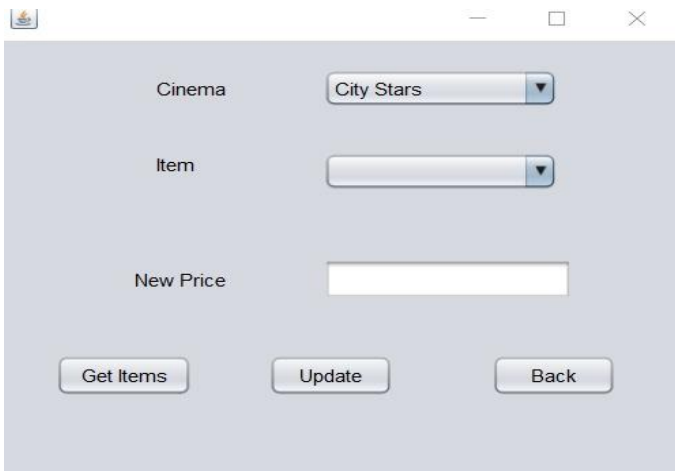


Figure 42 : Adjust menu prices screen

If the owner wants to add a cinema, they click on the add cinema button then enter the cinema’s name in the text field then clicks the add button.

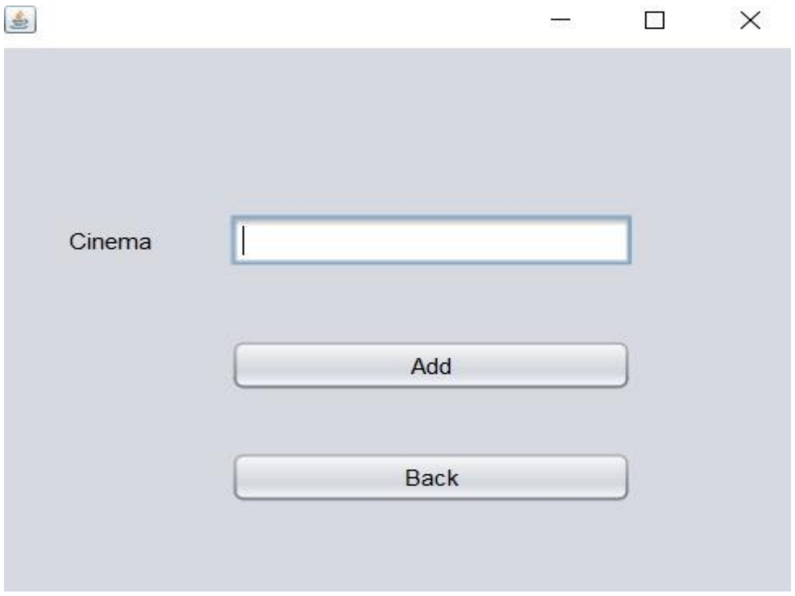


Figure 43 : Add cinema screen

16.3. ‘Standard user’ User Guide

If user (registered as user) logs in, This user interface is shown, and they can choose which cinema they want and then click the proceed button.

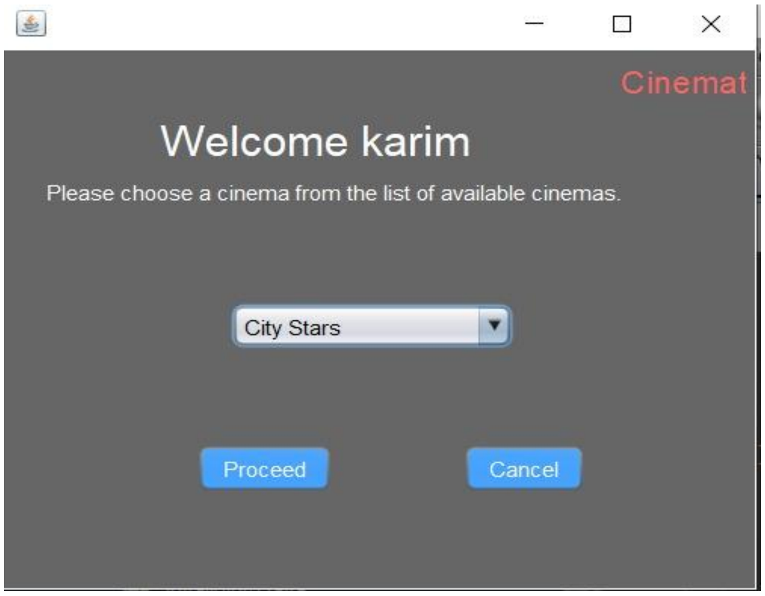
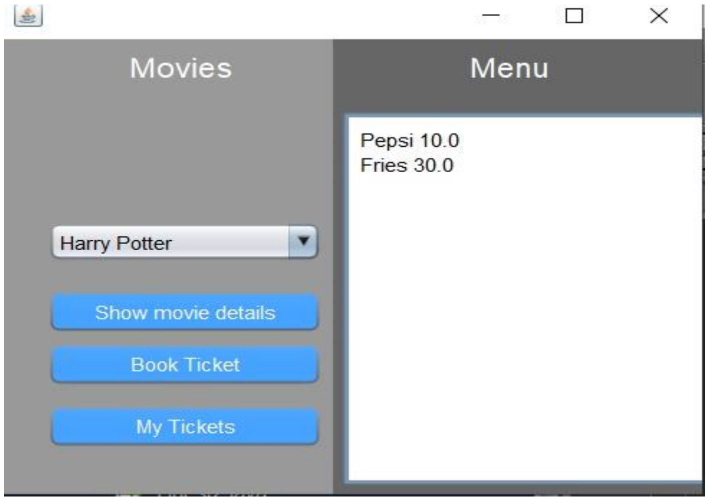


Figure 44 : 'Standard user' user interface (1)

Menu and snacks available at chosen cinema are viewed and user can select a movie to view its description or book a ticket to that movie

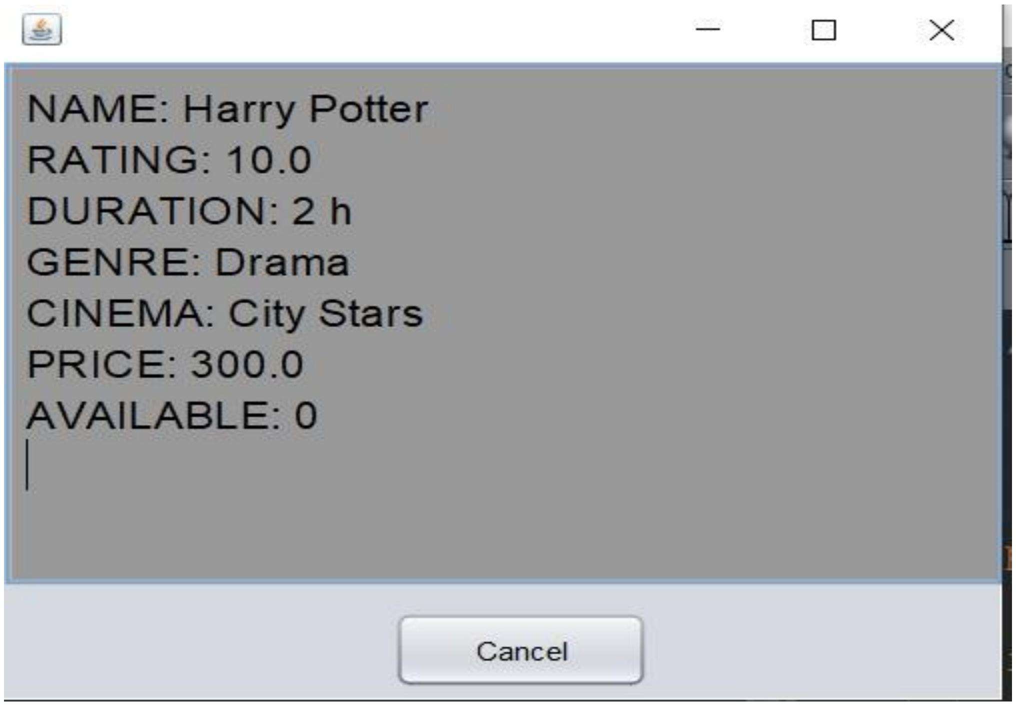


Pepsi 10.0
Fries 30.0

Figure 45 : 'Standard user' user interface (2)

If the user clicks on show movie details, All of the movie details are shown.

Its Name, rating, duration, Genre, Cinema where it is available, ticket price, and number of available tickets.



NAME: Harry Potter
RATING: 10.0
DURATION: 2 h
GENRE: Drama
CINEMA: City Stars
PRICE: 300.0
AVAILABLE: 0

Figure 46 : Chosen movie details screen

If the user wants to book a ticket to desired movie they can click on book ticket and this screen is shown to the user upon successful booking.

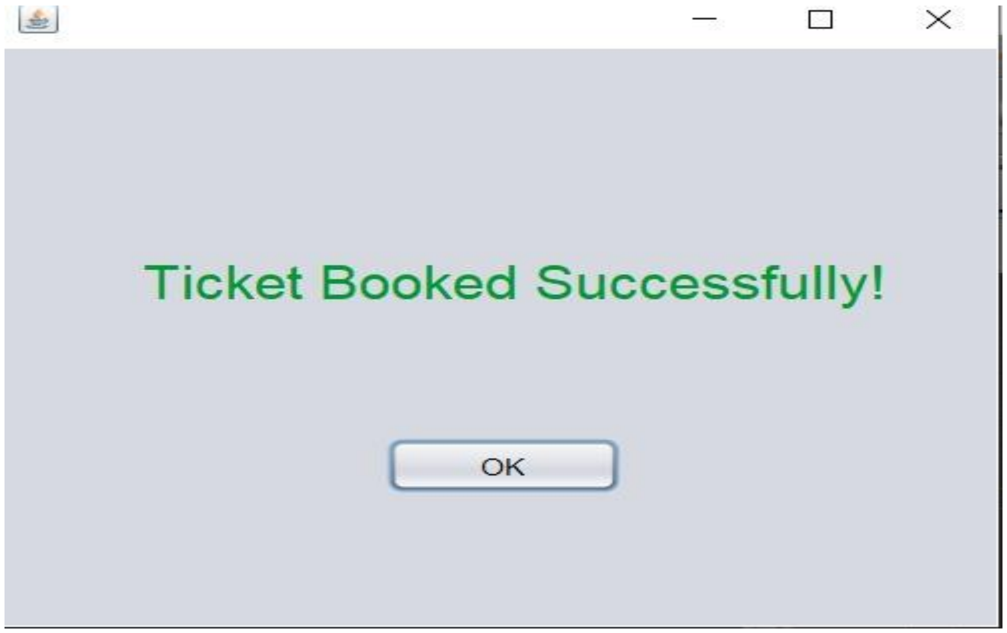

Ticket Booked Successfully!

Figure 47 : Successful booking screen

If the user wants to enquire about their tickets, They click on My tickets and their booked movie tickets will be shown.

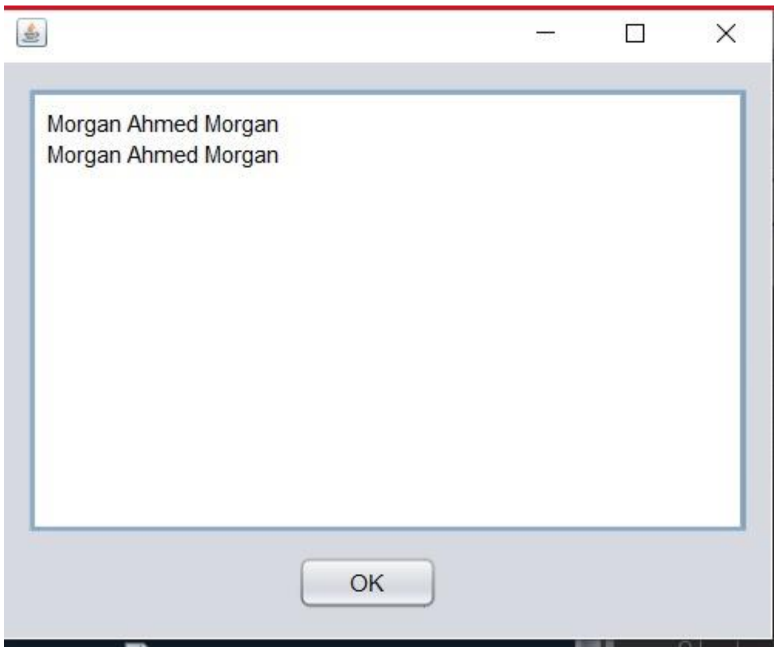

Morgan Ahmed Morgan
Morgan Ahmed Morgan

Figure 48 : My tickets screen

17. Research report

17.1. The Rumbaugh et al. OMT

It was developed about 1991 by as a method to support OOP also to help in the development of OO systems.

The Object Modeling Technique methodology is a prescriptive method for the construction of object-oriented design. It provides extensive guidance for the construction of object-oriented design

17.1.1. Phases of OMT

It consists of three main phases that progress a design from early requirements analysis to detailed design until the implementation. These three phases can be performed iteratively which are:

17.1.1.1. Analysis Phase

The analysis phase is about understanding and modelling the application. OMT suggests that the initial input to the analysis phase is a problem statement that describes the problem to be solved and provides an overview of the proposed system. This problem statement may be a textual description, or a formal description Problem statement is further divided into object, dynamic and functional model.

17.1.1.2. The Design Phase

It's divided into two subphases which are the system design and the object design The System Design which determine the overall architecture of the system. The Object design which attempts to produce a practical design and classify the objects into different classes with each attributes and necessary operations needed which is moving the focus from conceptual objects to the implementation.

17.1.1.3. The Implementation Phase

considers how the design should be implemented and then the design is translated into software

17.1.2. Modeling in OMT

OMT separates analysis phase to three parts depending on the following models:

17.1.2.1. Object Model

It represents the most stable and static phenomena which uses class diagram not complete class diagram as used in UML it only has the names and relations between classes

17.1.2.2. Dynamic Model

It represents the transition view using state diagrams same as UML

17.1.2.3. Functional Model

Functional Model in OMT describes the whole processes and actions using data flow diagram (DFD) same as UML. It focuses on how data is flowing, where data is stored and how it is processed by the different processes

17.1.3. Advantages and drawbacks

17.1.3.1. Advantages

The primary advantage of OMT is it allows a complete specification of the system by covering its static structure, dynamic behavior and functionality so it allows us to

- To test physical entity before construction of them.
- To make communication easier with the customers.
- To present information in an alternative way i.e. visualization.
- To reduce the complexity of software.

- To solve the real-world problems.

17.1.3.2. Drawbacks

The main problem with this approach is that each model of the three is designed independent from the other one so they need to be integrated together to get the whole picture but as mentioned before that each of them is designed independent so a lack of integration problem occurs coming after the lack of consistency between the object, dynamic and functional models.

17.2. The Booch Methodology

Booch methodology covers the analysis and design of systems

Its an iterative OO development process by Grady Booch about the year 1992 when he was working for Rational software back then now its acquired by IBM

17.2.1. Phases of Booch Methodology

It consists of two phases which are micro and macro development process each divided into sub phases

17.2.1.1. Macro Development Process

It identifies the activity life cycle of the following:

- Conceptualization: establish core requirements
- Analysis: develop a model with desired behavior
- Design: create an architecture
- Evolution: for the implementation
- Maintenance: for evolution after the delivery

17.2.1.2. Micro Development Process

It’s applied to the structures or behaviors that results from the macro development process and it consists of the following steps:

1. Identification of classes and objects
2. Identification of their semantics
3. Identification of their relationships
4. Specification of their interfaces and implementation

17.2.2. Diagrams Used in Booch Methodology

The Booch method consists of the following diagrams:

- Class diagrams
- Object diagrams
- State transition diagrams
- Module diagrams
- Process diagrams
- Interaction diagrams

We can see each diagram and its correspondence in the table below

Model	Type	Diagram	UML correspondence
Logical	Static	Class diagram	Class diagram
		Object diagram	Object diagram
	Dynamic	State transition diagram	State chart diagram
		Interaction diagram	Sequence diagram
Physical	Static	Module diagram	Component diagram
		Process diagram	Deployment diagram

17.2.3. Advantages and drawbacks

17.2.3.1. Advantages

It covers the analysis and design phases of an object-oriented system also defines different models to describe a system and it supports the iterative and incremental development of systems.

17.2.3.2. Drawbacks

It uses large set of symbols which Booch also was criticized for that. This is due to the detailed description level of the notation. Since the method uses a number of different diagrams there is also a risk that information can be fragmented across various diagrams. The working process of the method is uncleared defined which can make it difficult to use.

17.3. Jacobson's methodologies

Jacobson et al. methodologies cover the entire life cycle of the software system. It stresses on traceability between the different phases

17.3.1. Jacobson's Methodologies Phases

It can be broke down into five phases which are Requirements, Analysis, Design, Implementation and Testing model

Requirements: a problem domain object diagram then specify the use case diagrams

Analysis: Develop a model by defining object classes, their attributes, methods, and inheritance. Include associations, the dynamic part of a mod

Design: State transition diagrams and interaction diagrams

Implementation: implement the Software model

Testing: verify and validate by testing the model

17.3.2. Diagrams Used in Jacobson's methodologies

Use case Diagram is considered the heart of this methodology as the Use cases are scenarios for understanding system requirements. Also they capture the goal of the user and the responsibility of the system to its users

17.3.3. Advantages and drawbacks

Its main advantage that it introduced the use cases a new way to describe the system scenarios while one of its drawbacks was its transition from the analysis to the design was not clear enough

17.4. Software Architecture Patterns

It’s the design of individual components that are collected together to make the whole software and includes rules and guidelines for organizing the relationships between them.

17.4.1. Layered Software Architecture Pattern

Layered architecture is one of the most recognized and standard patterns because it’s easy to develop and maintain the purpose of it is to interconnect all the components without any dependability, also as each layer has a distinct role in the application and is marked as closed. It means that a request must pass through the layer right below it to go to the next layer each layer is only allowed to communicate with the layers below it but never above it this enables you to modify components within one layer without affecting the other layers

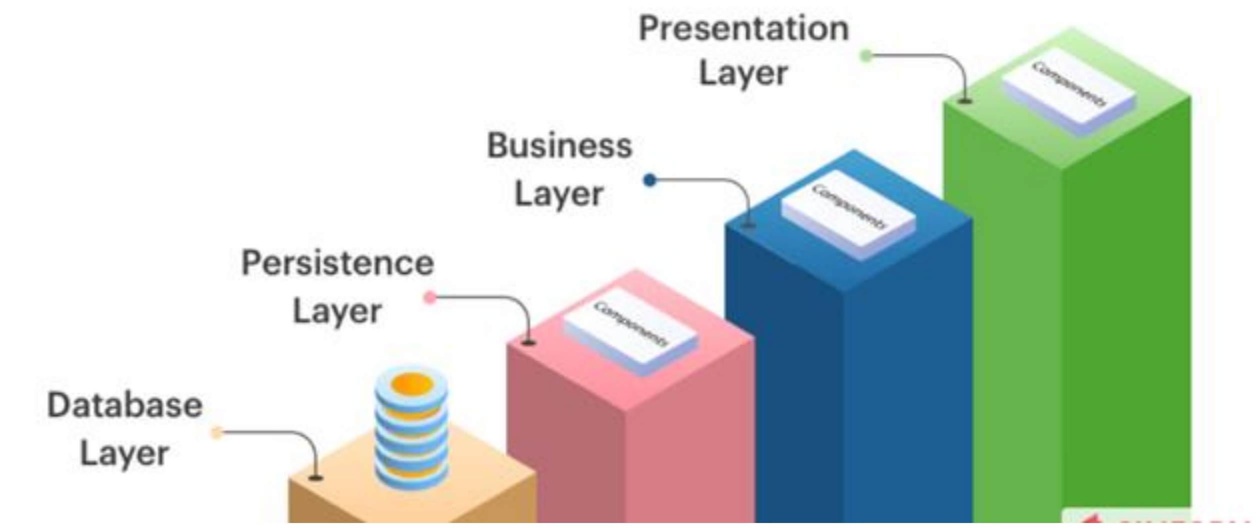


Figure 49 : Layered Software Architecture

17.4.1.1. Advantages

- The components in this model is not dependent on each other so that makes unit testing and functional testing easy
- It’s a tiered approach so any application created with this pattern will have the same format which make replacement of the components easier
- It’s a good pattern for applications that need to be built quickly

- Appreciate for teams with inexperienced developers and limited knowledge of architectural patterns

17.4.1.2. Disadvantages

- Large projects that requires the use of this pattern usually consume a lot of resources and its flexibility is low
- This model leads to development of a monolithic systems that minute changes on it may cause complete redeployment of it
- Skipping previous layers to create a tight coupling may lead to logical mess and complex interdependencies

17.4.2. Microservices Software Architecture Pattern

This pattern can break down a large system into smaller and more manageable components each one of them works individually and fosters independent communication with the others its seen also as an alternative to monolithic applications and service-oriented architectures as it enables developers to work on a particular component without the functionality of other components being affected so many of the developers consider it one of the best, Netflix is one of the early ones that used this pattern and they utilized it to serve millions of people

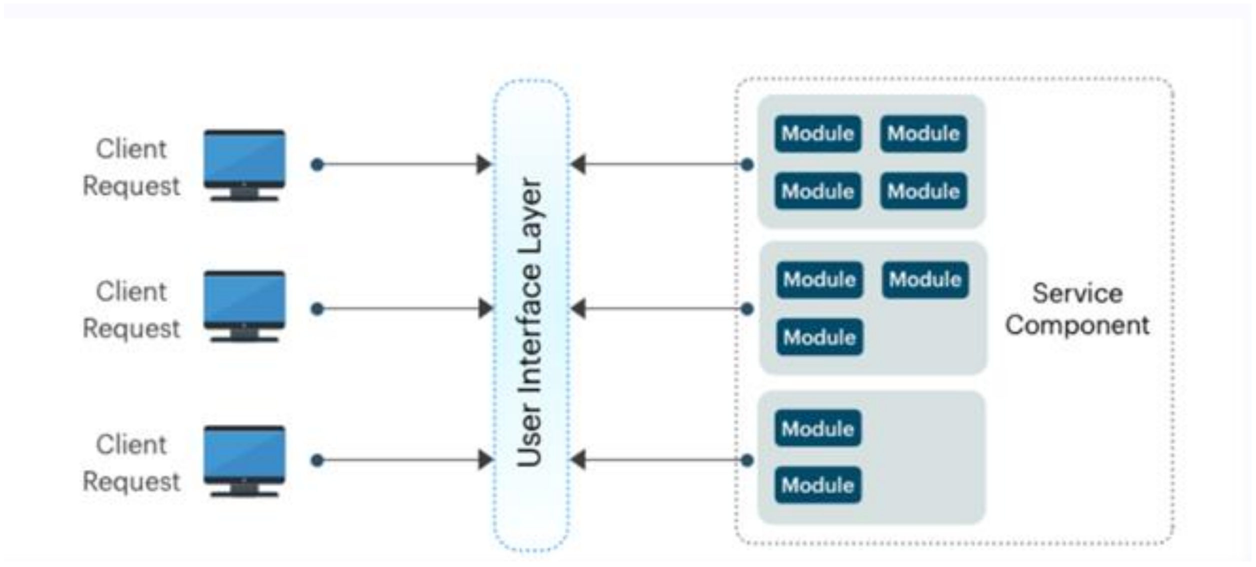


Figure 50 : Microservices Software Architecture

17.4.2.1. Advantages

- Systems built with this pattern tolerate faults if some microservices stopped working the entire software will not fall
- Applications built using it will be highly scalable as you can modify or update each component with updated stack of technology and different programming languages
- It allows you to integrate various services into your application based on the scope of the project
- Its suitable for modern software systems that requires rapidly growing data
- Its suitable for web apps with teams that work remotely and with defined boundaries

17.4.2.2. Disadvantages

- It can be difficult to manage each of the components individually and interlink them together
- Security breaches and data integrity will be challenging, and it will need high level developers
- Performance may be affected as the single task may be spread across different microservices
- Designing the right level of granularity of the microservice is very hard and considered as a challenge

17.4.3. Event-Driven Architecture Pattern

It's made of decoupled components each one of them is single-purpose event driven that asynchronously work together to receive and process events this patterns behavior is around the productions, detection and consumption of all events.

It consists of mediator and broker a mediator is used when many steps are needed to be done within an event bus through a central mediator while broker is used to chain events together

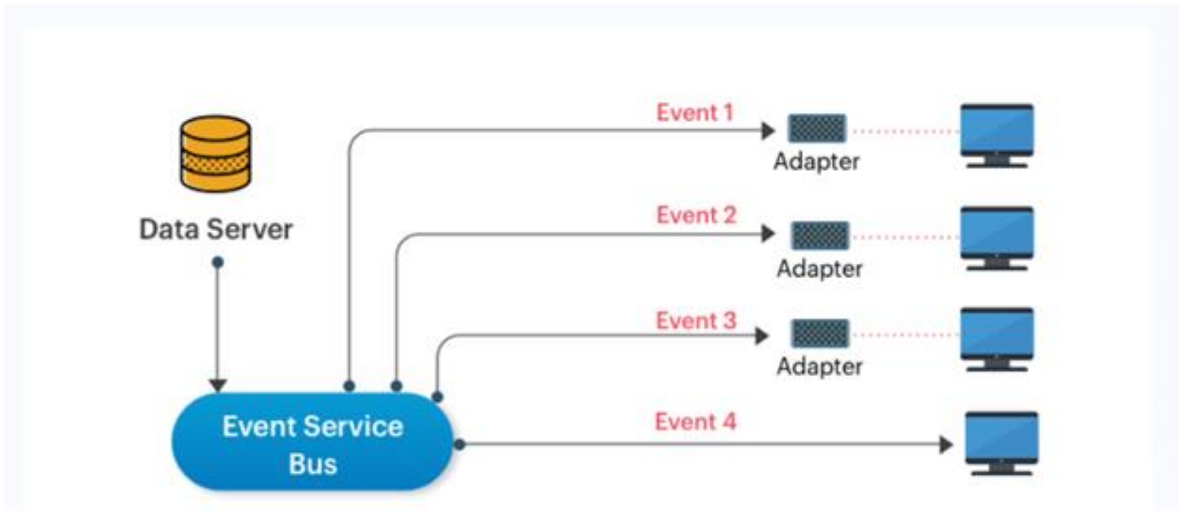


Figure 51 : Event-Driven Architecture

17.4.3.1. Advantages

- This pattern adds to the response time of the architecture which results in better business outcomes
- Its best to use when you need real time updates in your app with asymmetric data flow
- IoT devices that need to exchange data between service providers and customers in real time are suitable for it
- It helps with user interfaces
- It is suitable for applications where individual data blocks interact only with few modules

17.4.3.2. Disadvantages

- Its challenging to test it if many modules responsible for single event
- The transaction-based mechanism for consistency tends to be complicated because of the decoupling and mutually dependent components.
- Error handling is hard to structure when several modules are handling the same event
- Development wide data structures for events will be very hard if they have different needs

17.4.4. Space-Based Software Architecture Pattern

Space-based architecture pattern is unique for the ability to overcome intensive loading problems. Most of the patterns deal with databases that leads to app crashes when the database can’t handle the high load. while, this pattern brings forth the concept of tuple space – the distributed shared memory.

It comprises two main components a processing unit and a virtualized middleware. The processing unit consists of back-end business logic and web-based components. also, you can deploy smaller apps as a single unit and the large ones can be broken down into different processing units. also, the virtualized-middleware module contains components to manage data synchronization and handle requests.

The most important feature of the patterns is that it doesn’t need a database to work, the software receives a request and processes the bid with a timestamp. After that, it upgrades the bid’s data and sends it back to the browser. This is an effective example of how modern web development stacks work

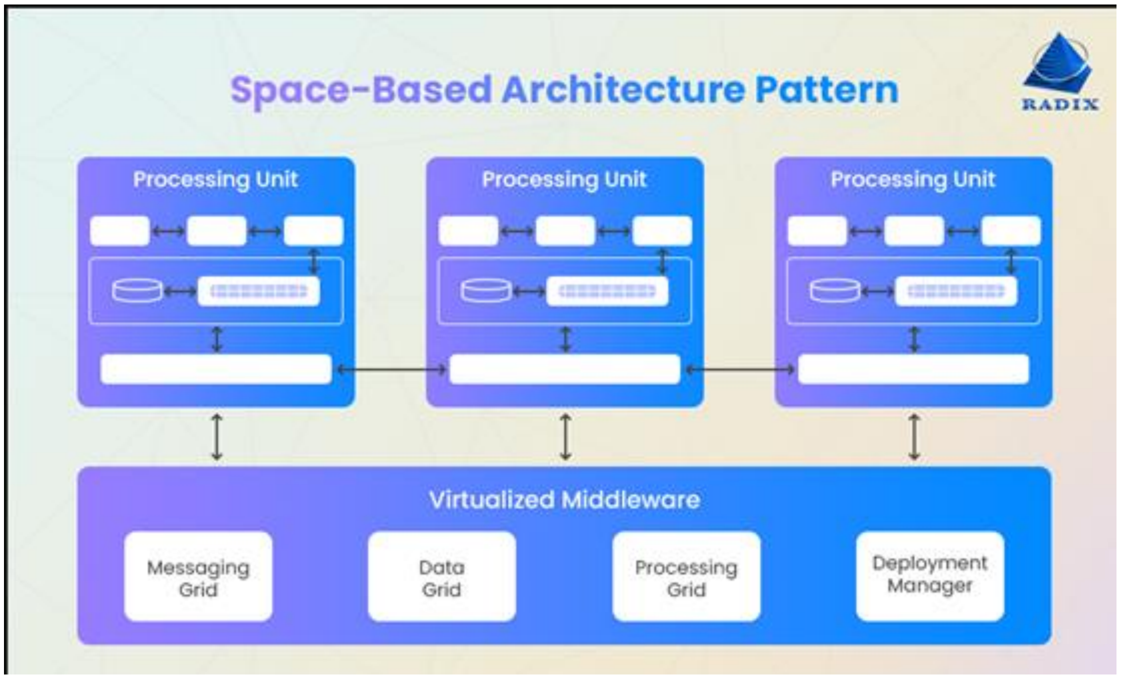


Figure 52 : Space-Based Architecture

17.4.4.1. Advantages

- It's the best architectural pattern to mitigate scalability and concurrency issues in your application.
- If you work with low-value data that you can afford to lose without having to face the high-risk consequences, this pattern might prove to be useful.
- Apps that must deal with unpredictable concurrent user loads will work seamlessly with this model
- Its useful with Software and apps that work with a huge userbase or constant volume of requests.

17.4.4.2. Disadvantages

- If you cache data for speed multiple copies may be destroyed
- It’s hard to test the app as you have to generate a heavy load

17.5. Conclusion

17.5.1. Layered Software Architecture Pattern

Factor	**Layered Pattern**
Agility	Low (Difficult to make changes)
Scalability	Low (Hard to scale)
Ease of Development	High (Easy to develop)
Testability	High (Presentation components can be mocked for testing)
Performance	Low (Not suitable for high-performance apps)
Ease of Deployment	High (Minor changes require redeployment of the whole system)

Table 1 : Layered Software Architecture Conclusion

17.5.2. Microservices Software Architecture Pattern

Factor	Microservices Pattern
Agility	High (Small app size makes it easy to change)
Scalability	High (Highly scalable)
Ease of Development	High (Isolated and smaller components make development easy)
Testability	High (Testing is easy due to isolated business logic)
Performance	Low (Distributed nature makes it comparatively slow)
Ease of Deployment	High (Deployment of services as individual units)

Table 2 : Microservices Software Architecture Conclusion

17.5.3. Event-Driven Pattern

Factor	Event-Driven Pattern
Agility	High (Making isolated changes is possible without depending on other components)
Scalability	High (Very easy to scale)**
Ease of Development	Low (Complex development for asynchronous feature)
Testability	Low (Specialized testing tools for generating events)
Performance	High (Perfect for high-performance apps)
Ease of Deployment	High (Decoupled nature makes it easy to deploy)

Table 3 : Event-Driven Pattern Conclusion

17.5.4. Space-Based Pattern

Factor	Space-Based Pattern
Agility	High (Loosely coupled modules make it easy)
Scalability	High (Highly scalable)

Factor	Space-Based Pattern
Ease of Development	Low (Complicated to build due to in-memory data grid and sophisticated caching)
Testability	Low (Testing is complicated and time-consuming)
Performance	High (Caching mechanism and in0memory data access make faster apps)
Ease of Deployment	High (Easy deployment for cloud-based tools)

Table 4 : Space-Based Pattern Conclusion

18. Appendix 1 : DFD

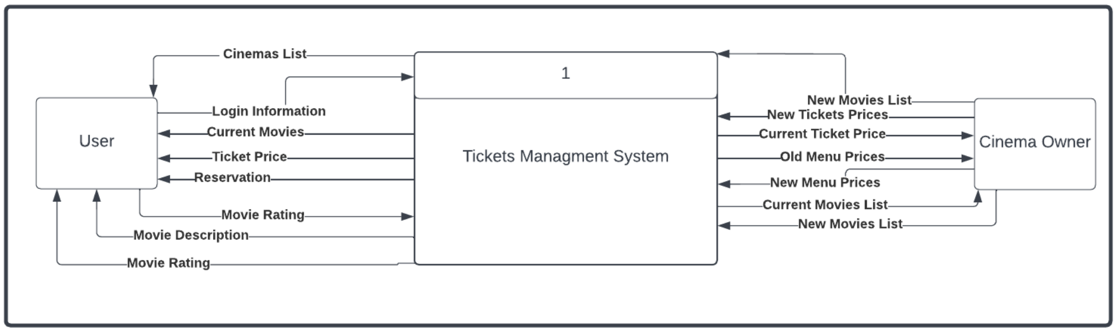


Figure 53 : Context-level DFD

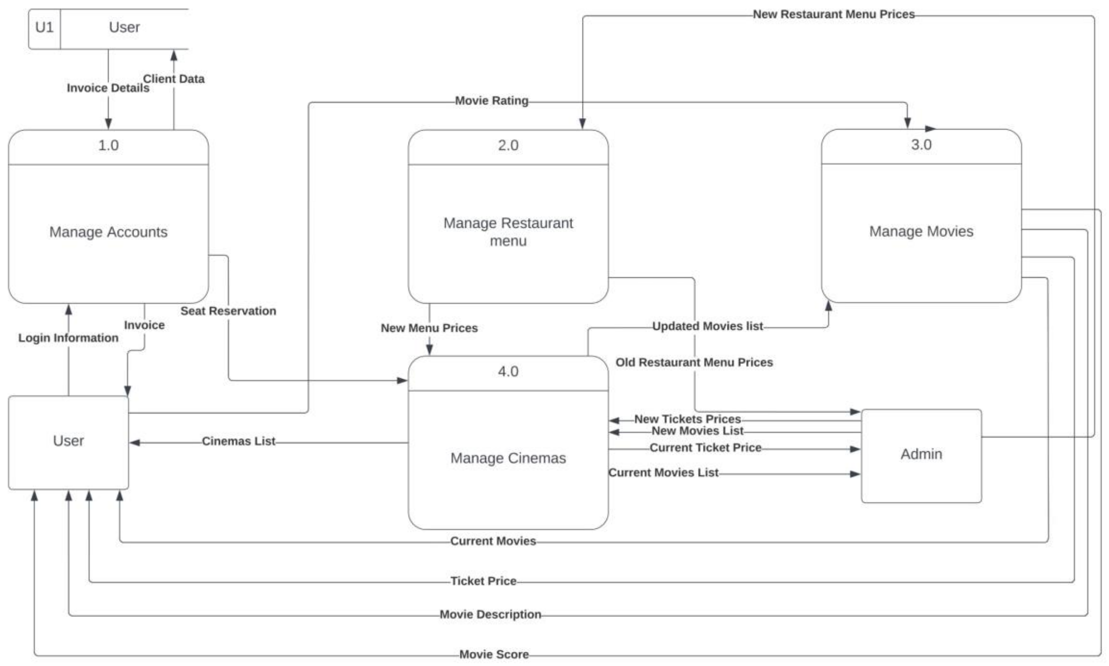


Figure 54 : Level 0 DFD Diagram

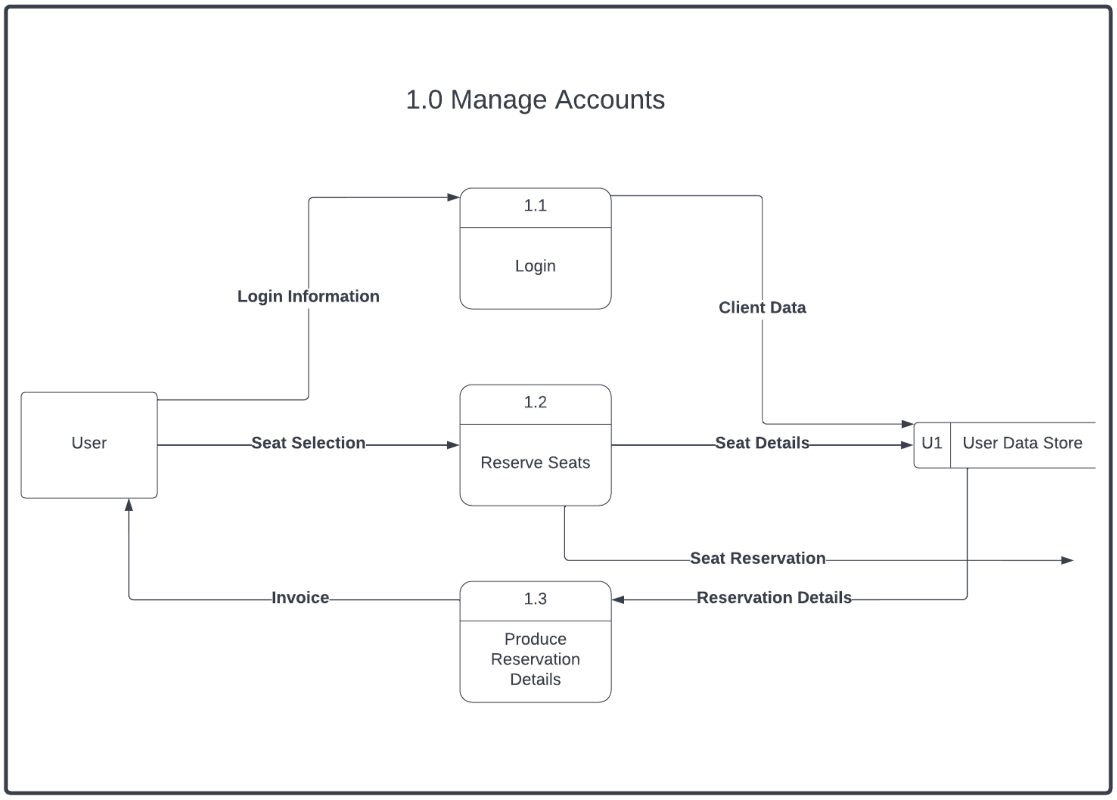


Figure 55 : Level 1 DFD Diagrams (Part 1)

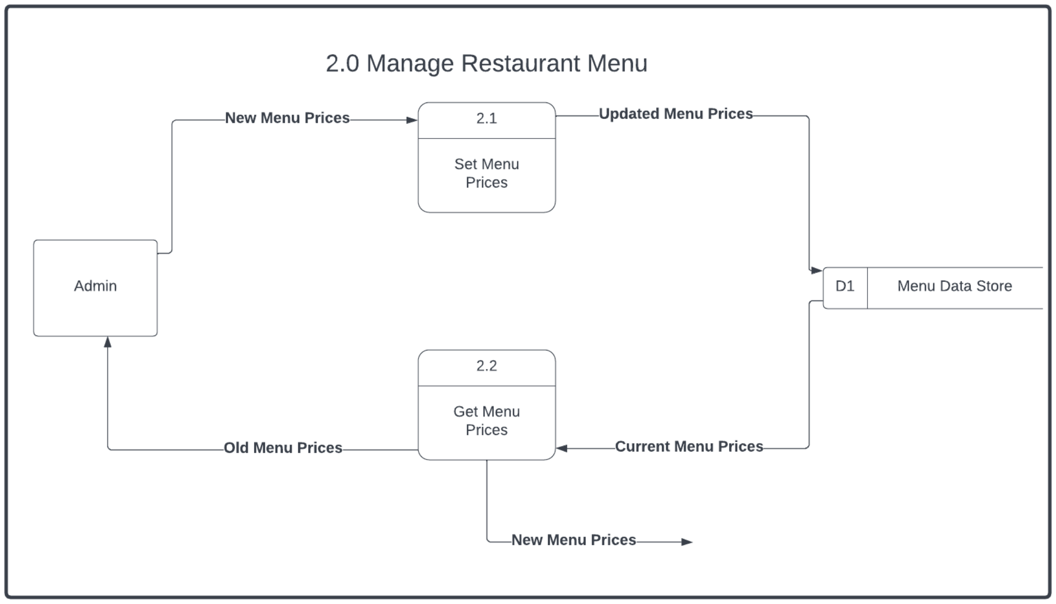


Figure 56 : Level 1 DFD Diagrams (Part 2)

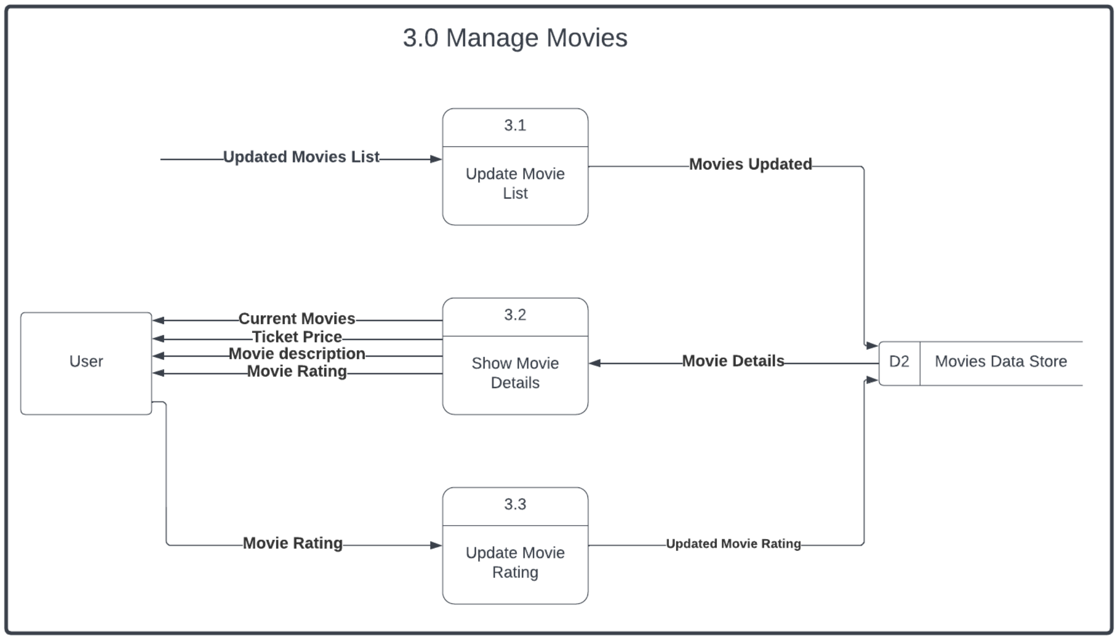


Figure 57 : Level 1 DFD Diagrams (Part 3)

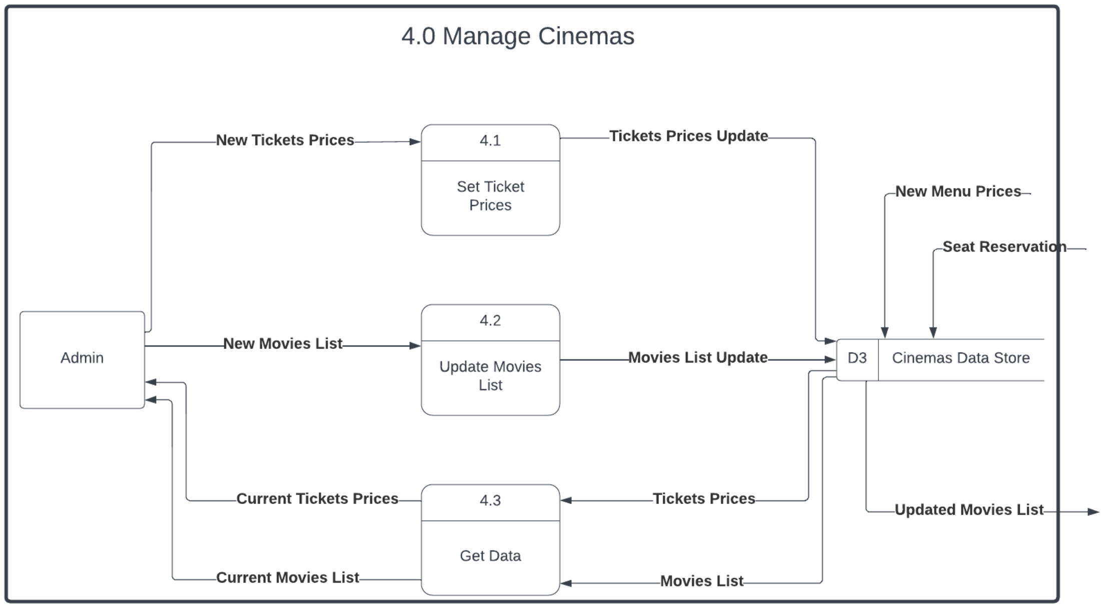


Figure 58 : Level 1 DFD Diagrams (Part 4)

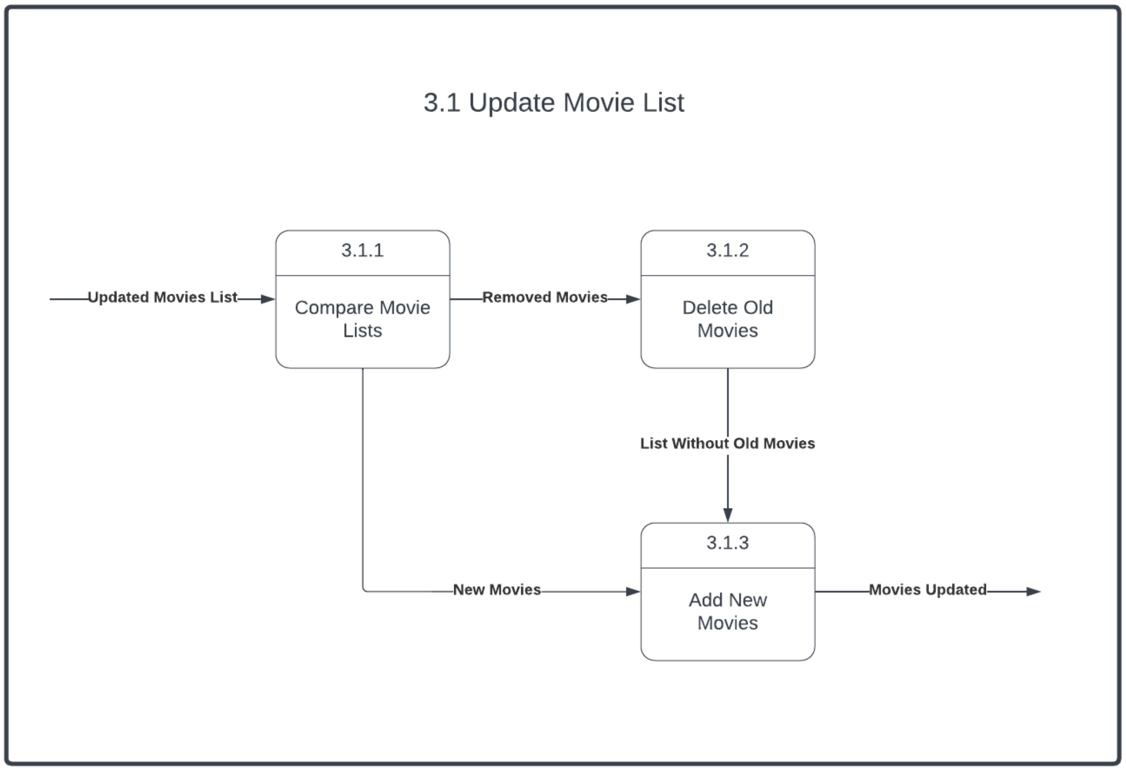


Figure 59 : Level 2 DFD Diagrams (Part 1)

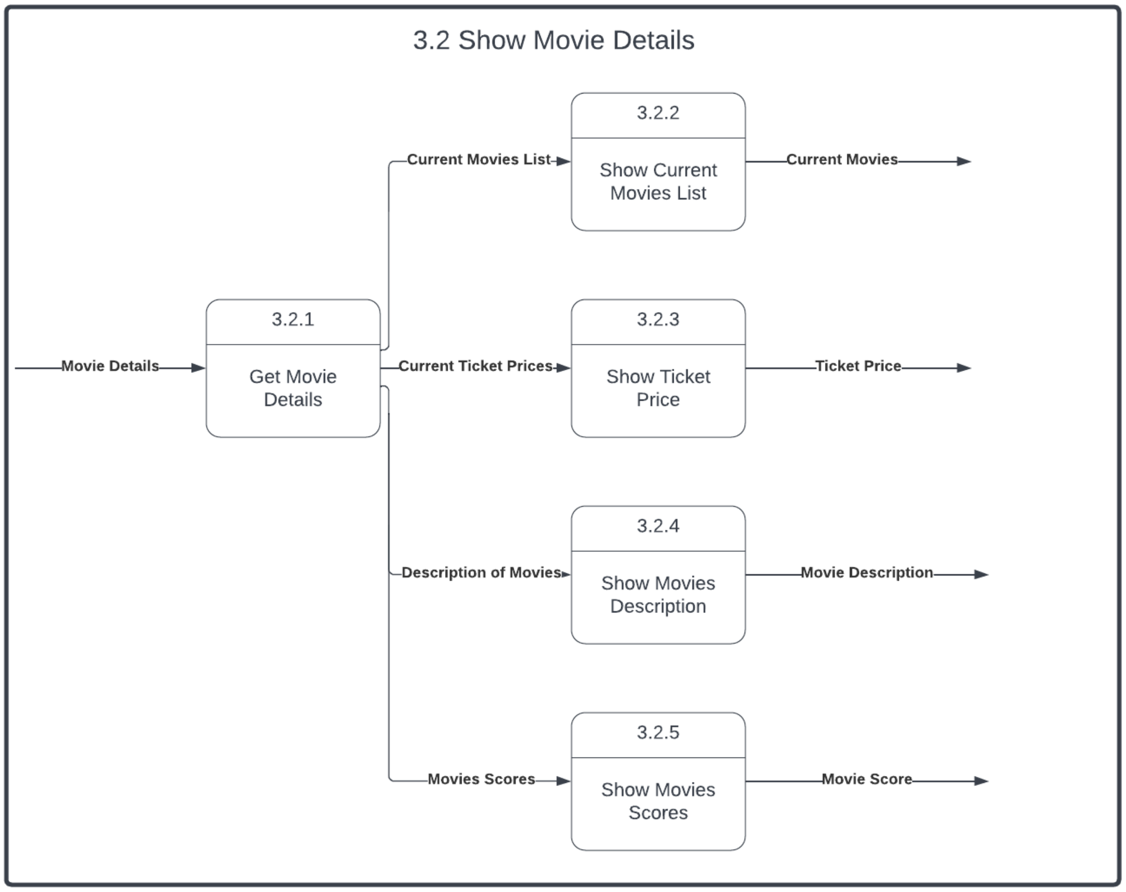


Figure 60 : Level 2 DFD Diagrams (Part 2)

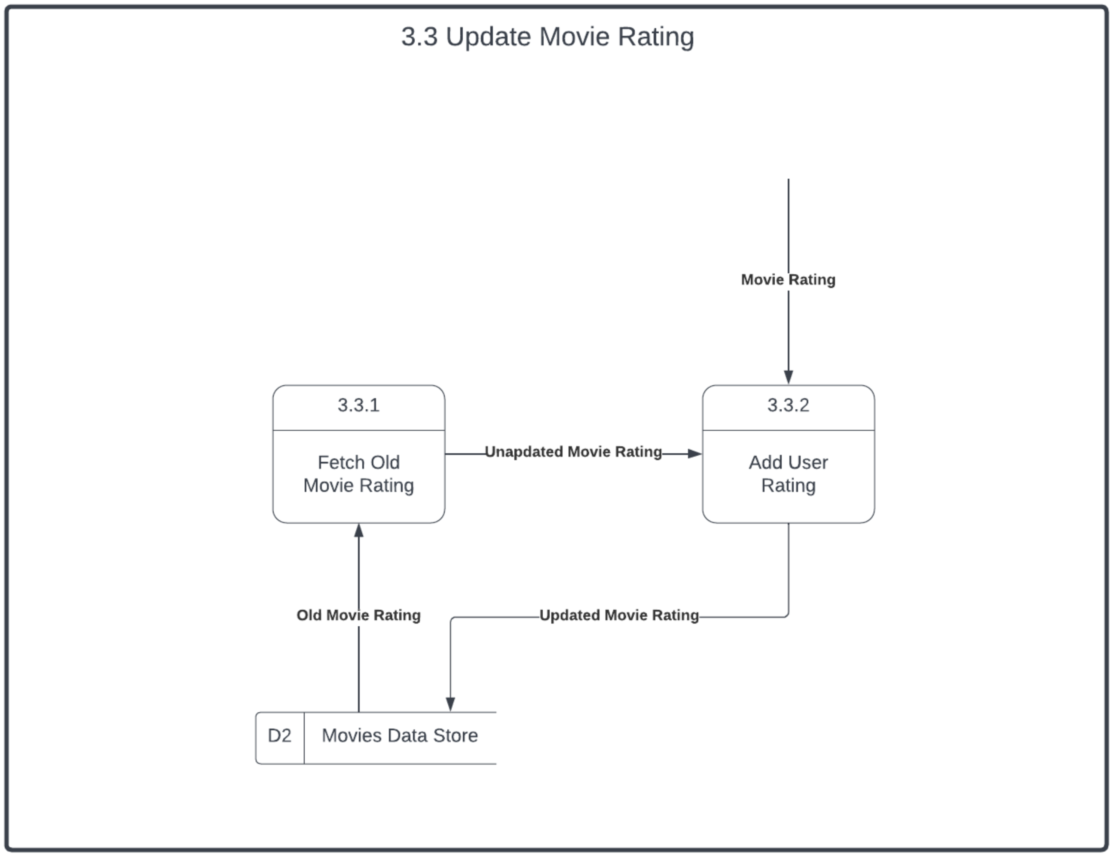


Figure 61 : Level 2 DFD Diagrams (Part 3)

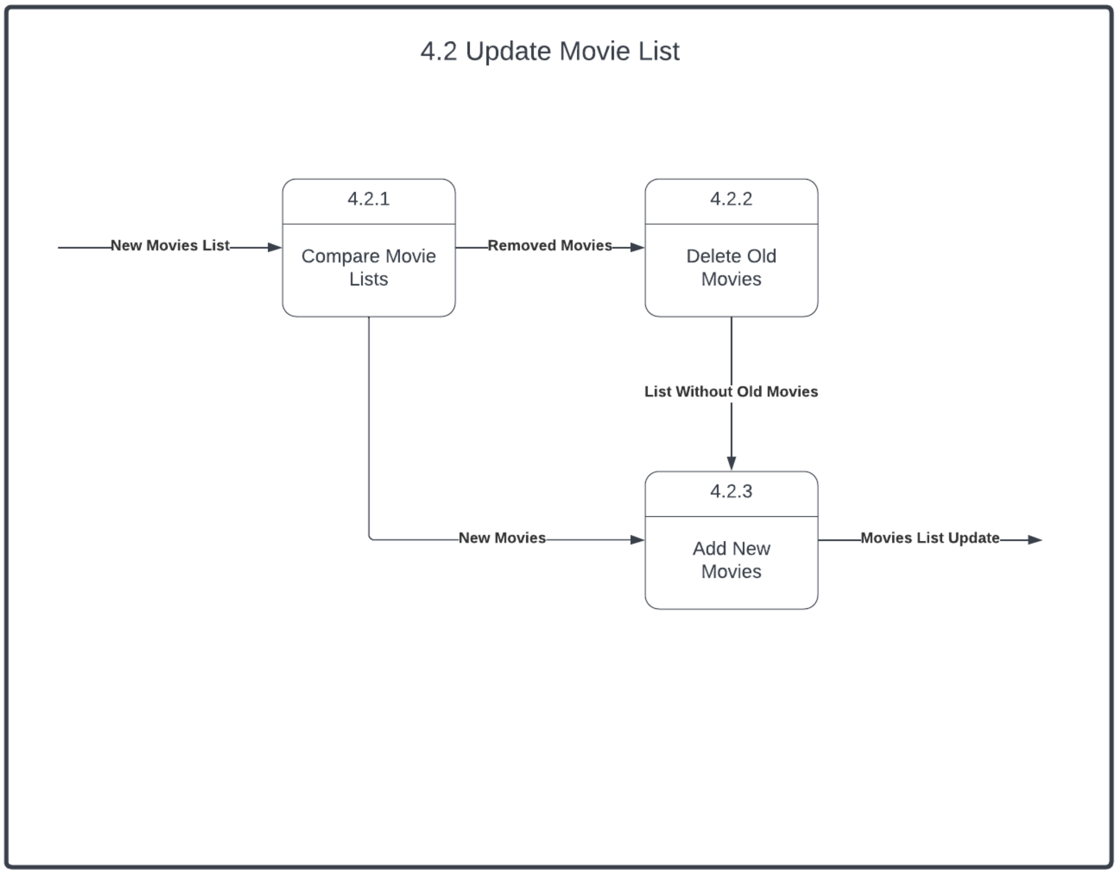


Figure 62 : Level 2 DFD Diagrams (Part 4)

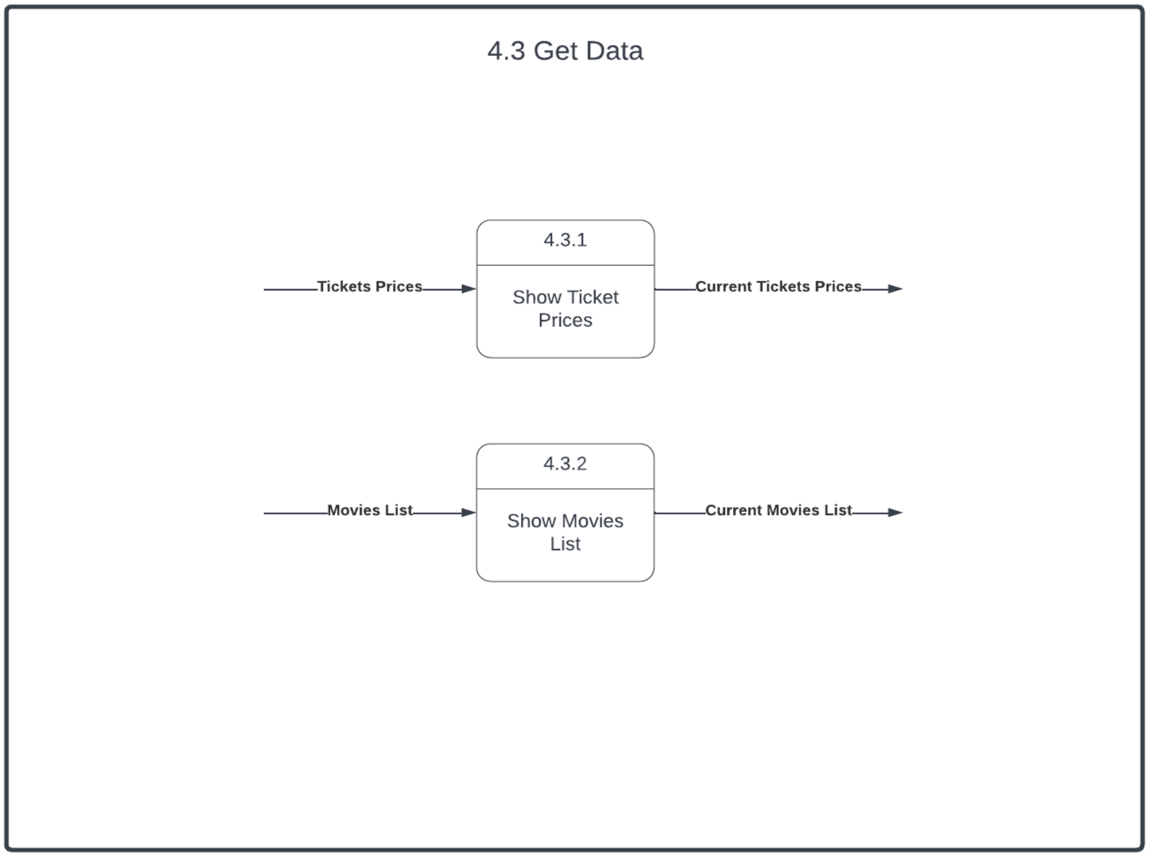


Figure 63 : Level 2 DFD Diagrams (Part 5)