**PROJECT REPORT**

Zinema – Management Software

Distributed System

**Students**

Andrei Cioanca – 266105

Claudiu Rediu – 266129

Dominika Kubicz – 266148

Nikita Roskovs – 266900

Stefan Harabagiu – 266116

**Supervisors**

Jakob Knop Rasmussen

Jan Otto Munch Pedersen

**Acknowledgements**

The team wishes to formally acknowledge VIA University College’s active implication in the evolution and completion of this project work. All documents contained or referred to from this project are based on VIA University College’s templates and designs. All information present in this document, while original, could not have been made possible without VIA’s resources.

**Abstract**

Over the years, Danish cinemas have experienced a steady decrease in activity and popularity. While the total number of cinemas and digital screens within them has increased

Zinema is an up and coming business in need of help. It wants to alleviate some of the issues in the cinema business brought by the steadily declining market in Denmark. The company employed the authors of this report to come up with an IT solution for their new cinema.

The system has been developed in both Java and C# and incorporates a robust and secure database. It fulfils most of the Product Owner’s requirements and stands as a solid and easily manageable system that can automate the cinema’s most time consuming endeavors.

Based on the Product Owner’s requirements the system has been built to be split amongst different employees of the company. It can create movies, rent them, create a schedule and present it to the customer accessing the Zinema’s official website.

A database has also been implemented and connected to the different parts of the system in order to easily store critical information and not lose any essential data regarding both customer and company.

As per the stakeholder’s wish, each part of the system also implemented a Graphic User Interface, made simple and clean in order to ease the employees’ work and make it easier for them to use it.

In the end the Product Owner received a satisfactory product made specifically based on his business requirements.

Contents

[1. Introduction 5](#_Toc532907623)

[2. Requirements 6](#_Toc532907624)

[3. System Analysis 8](#_Toc532907625)

[4. System Design 17](#_Toc532907626)

[5. System Implementation 28](#_Toc532907627)

[6. Testing 37](#_Toc532907628)

[7. Results and Discussion 43](#_Toc532907629)

[8. Conclusion 43](#_Toc532907630)

[9. Project Future 43](#_Toc532907631)

[10. List of Appendices 44](#_Toc532907632)

[11. Sources of Information 44](#_Toc532907633)

[**Webography** 44](#_Toc532907634)

# Introduction

The film industry has experienced a steady growth in recent years, as reported by the Motion Picture Association of America (Motion Picture Association of America, 2017), with the global box office in 2016 for all films increasing by one percent from 2015. In 2016 the number of cinema screens has also increased by eight percent showing an upward trend in what concerns the cinema business. In Denmark there have been reported 11.9 million admissions to movies in 2017 alone (Danish Film Institute, 2017). As the average ticket price for a movie can be estimated at about 9.6 euros the Danish box office revenue for 2017 was approximately 114 million euros. The annual budget of the Danish Film Institute (DFI), for that same year, was 66.9 million euros, the annual accounts reaching a total of 7.4 million euros. The operating costs for the DFI was roughly 1.6 million euros, 26 percent of which was attributed to properties, IT systems, etc.

Over the years, Danish cinemas have experienced a steady decrease in activity and popularity. While the total number of cinemas and digital screens within them has increased since 2011, overall, there has been a decline in the total number of tickets sold, despite the constant sprouting of new cinemas all over the country.

In 2011 there were 12.4m tickets sold in Danish cinemas (155 cinemas) (Danish Film Institute, 2012), in 2016 (Danish Film Institute, 2017) there were 13m tickets sold (163 cinemas) while in 2017, 11.9M (Danish Film Institute, 2018) (166 cinemas). The number of tickets sold per capita fluctuates between 2.1 and 2.4 with 2017 coming in at 2.1.

This overall decline in recent years (Nielsen, 2018) can be attributed to many reasons but the rise in popularity of streaming platforms such as Netflix and HBOGo that offer, subscribed users, movies and tv shows on demand has been one of the main reasons why Danes no longer go to the cinema as often as they did. Netflix launched in 2012 (media.netflix.com, 2012) in Denmark and since then it quickly became the most popular entertainment streaming platform in the country.

Zinema is an up and coming business in need of help. It wants to alleviate some of the issues in the cinema business brought by the steadily declining market in Denmark. Its main purpose is to revive the Danish films in a new modern way. Some of the biggest maintenance costs go into the IT systems. Their software has an old interface that is costly to maintain and built in a way that no further features can be added to it. The customers and administrators have reported that it’s hard to handle and crashes often, without having saved what the user has done. This has resulted in a lot of money lost for the starting company.

POStive Cinema, a company that offers management solutions to cinemas, states that their product helps their clients “achieve better results and improve operational efficiency” (POSitive Cinema, 2018). Knowing that software has a big influence on the future success, Zinema wants to provide their clients with the best possible software that will not only “enhance the cinema experience” (Vista Cinema, 2018), but also emphasize Zinema’s unique features.

The cinema business is in search of a way to improve the user experience and the functionality while also lowering the costs of maintenance.

# Requirements

1. **Functional Requirements**

**Movie Creator**

1. A user should be able to create movies and store them in the database.
2. A movie consists of a title, year of creation, release date, price for renting, name of the studio, director name, description, main actor name, rented status.

**Movie Manager**

1. A user should be able to retrieve a list of movies that are not rented from the database.
2. A user should be able to rent a movie from the acquired list.

**Movie Scheduler**

1. A user should be able to make a schedule and store it in the database.
2. A schedule consists of a list of scheduled movies.
3. A schedule movie consists of a movie, a room, a time and day at which it’s booked and a list of seats.
4. A user should be able to get a list of rooms.
5. A user should be able to get a list of rented movies.
6. A room consists of number of seats and a description.
7. A user should be able to add and remove rooms from the database.

**Ticket Booker**

1. A user should be able to book tickets for the scheduled movies.

\*Rent = change the rented status to true

1. **Non-Functional Requirements**
2. The system will have a 3-tier architecture
3. The system will be written in C# and Java
4. The system will have a GUI for each client
5. The system will use a protocol for sockets

# System Analysis

The system makes use of several different applications communicating with each other to achieve some sort of ecosystem. These programs are being controlled by the employees of Zinema and their customers are able to interact with the system through their official website.

Starting from the bottom, the Movie Creator creates the movies and stores them in the database.

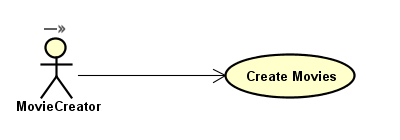


Figure 1 - Movie Creator Use Case

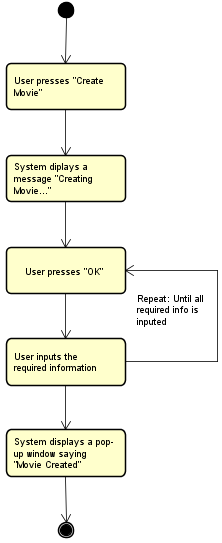


Figure 2 - Create Movie Activity Diagram

The employee that operates the application can also list the current movies from the database in order to ensure that his actions were successful without relying on a different part of the system.

The Movie Manager is the next part in the chain. This one can rent movies that have been created by the previous link, as long as they have not already been rented. In broad terms, it functions quite similarly to the Movie Creator but has a more restricted access to the data it works with.

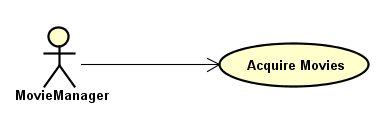


Figure 3 - Movie Manager Use Case

An employee is able to retrieve a list of the currently unrented movies from the database and then simply insert the id of the movie he wants to rent in the application and the system will store it as rented. Once this is complete, the list will be automatically updated and the rented movie removed from the list.

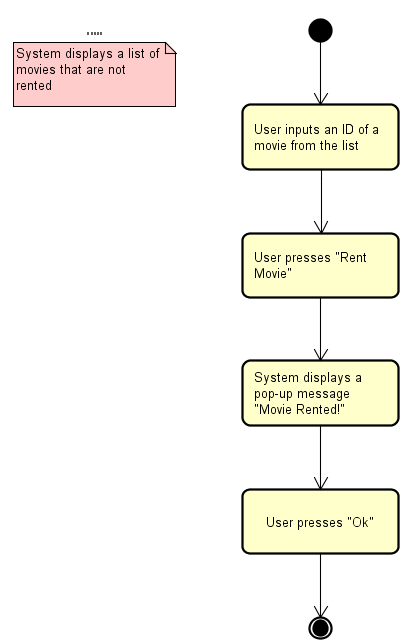


Figure 4 - Rent Movie Activity Diagram

The third program is the Movie Scheduler. It is definitely the most complex one since it can do a lot more than the others combined. The scheduler can create and manage rooms for the cinema as well as take the movies that the previous employee had rented and schedule them for screening using the available rooms.

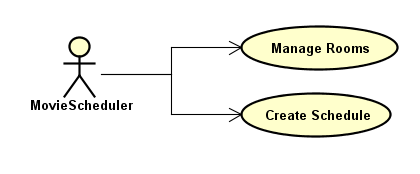


Figure 5 - Movie Scheduler Use Case

The employee working as the scheduler can also send the scheduled movies available for the week to the next and last component of the system, the one that handles customer interaction. This part of the system can also retrieve and manipulate information from the database such as getting the list of available movies to be scheduled or even deleting rooms from the system.

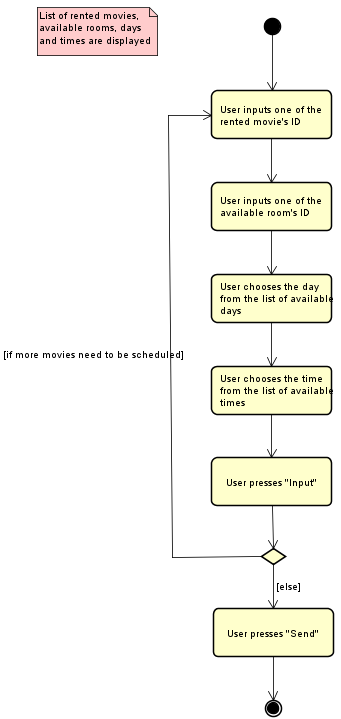


Figure 6 - Create Schedule Diagram

The last part is called the Ticket Booker and is the only application in the system that handles and allows for customer interaction. Using the information from the previous

link in the chain, the program allows the customer to input data through the cinema’s official website.

The customer can see this week’s scheduled movies and can select one from the list. Once selected he will be able to see all available seats in the room as well as the ones that have been taken. He can then pick the number of a seat and with a single click of a button the system will register his booking for that seat.

All that is left to do for the customer is to give his personal e-mail and telephone number so that the information can be sent to him and then he needs to show up thirty minutes before the screening to receive his ticket.

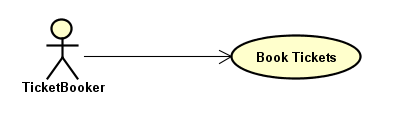


Figure 7 - Ticket Booker Use Case

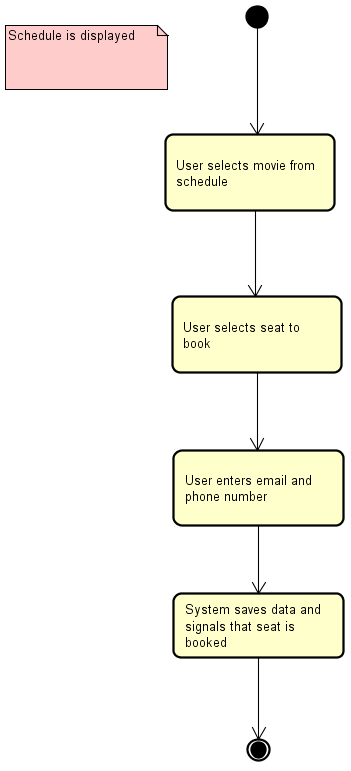


Figure 8 - Book Ticket Activity Diagram

As seen from these figures, the system is easy to understand and more importantly, easy to use for both customers and employees of the cinema, a trait highly appreciated by both consumers and stakeholders.

**Security**

In order to make the system secure, possible risks and dangers that come with the project were discussed. All goals, means and powers of an attacker were taken under consideration. Matters such as threat frequency and effect, preventive and corrective measures are examined in the next part of the document.

The threat of spoofing identity endangers the authentication property. A possible goal of the attacker might be to impersonate a different user. This is possible to achieve by preforming the man in the middle attack. The criminal can intercept the messages that the two users are exchanging and steal the signature of one of the users. A masquerade attack is an active type of attack usually performed by external attackers on the network. To overcome this threat implementing a login system and encrypting the signature can be a solution. This way the attacker won’t be able to get a hold of the user’s signature, nor send data as this specific user, because of the required password.

Tampering puts the integrity property in danger. A goal of such an attack is to modify the data while it is transferred from one user to another user. That means that the information a user is receiving is not the original that was sent. An external attacker can modify the message (which is an active attack) either online, offline or on the network. Mechanisms such as signing packets or checksums can prevent tampering. By signing we are sure that the message was sent from the right user, and thanks to checksums, once the information is coded it cannot be reverted. Because of that, the perpetrator either cannot access the data to modify it, or he cannot change the signature.

Repudiation threatens the confirmation property. In this case the attacker is denying the actions that they performed on the system. Such a violation can be performed in a system that is not keeping track of the user’s activity. Repudiation is an active attack that can be performed by an internal user, either offline, online or on the network. To prevent repudiation digital signatures or logging of user activity can be implemented. Thanks to such mechanisms the attacker will not be able to deny their action, because all his activity will be associated with themselves.

Information disclosure can be an attacker’s goal and threatens the confidentiality property of data used in our system. An attacker being able to disclose information means that they had access to data they should not have been able to see. This goal can be achieved by a number of attacks that our system may be vulnerable to. Some of those attacks could be passive, such as eavesdropping on the information sent, or active, for example blocking a message from arriving and then looking at the information. By using the EINOO model we can determine where and by whom our system could be attacked.

External attackers could perform network attacks, such as the ones mentioned above, in order to gain access to unauthorized information and internal attackers which may be able to perform off-line attacks so that they could steal information. A feasible approach to this threat would be to use cryptographic encryption in order to secure the data in our system. This way if the data is intercepted by an external attacker using passive or active attacks or if an internal attacker carries out successfully an off-line attack the information they get is encrypted and therefore secure. However, if an attacker does manage to achieve information disclosure then the mistake we made would be at the Mechanism level, suggesting that the vulnerability comes from what encryption we use.

Denial of service type attacks represent a danger to the availability of the data used in our system. An attacker could be able to deny a user’s access to our system and such a goal could be accomplished through some attacks that the system may be vulnerable to. A denial of service attack is an active one and can be carried out by both external and internal attackers; such attacks usually occurring on the network. A login system is susceptible to DoS attacks by the use of bots that input wrong information for a large amount of times. In order to prevent this the system should ask the user to perform an action only a human would be able to before verifying the login credentials. The most widely used solution for this is the reCAPTCHA security service. However, attacks that have the goal of denying service to the user may take many forms and if such an attack does succeed we should take another look at the Threat Model and see what types of attacks we dismissed or we did not think of.

An attacker may also have elevation of privileges as his goal, such attacks resulting in them being able to receive higher rights in our system than they should. Attacks directed at achieving elevation of privileges could be active ones and may occur on the network; such attacks being most usually carried out by external attackers. In order to prevent unauthorized access a login system and verification of input may be useful in order to deter this threat. Another idea would be to ask for the user’s credentials again before authorizing them to perform an important action (e.g.: permanently deleting some files). Omissions in the Threat Model may, however, permit attackers to succeed in elevating their privileges.

# System Design

The first step in designing the system was making an overall view of the system and its components and then expanding on that with details by using UML diagrams.

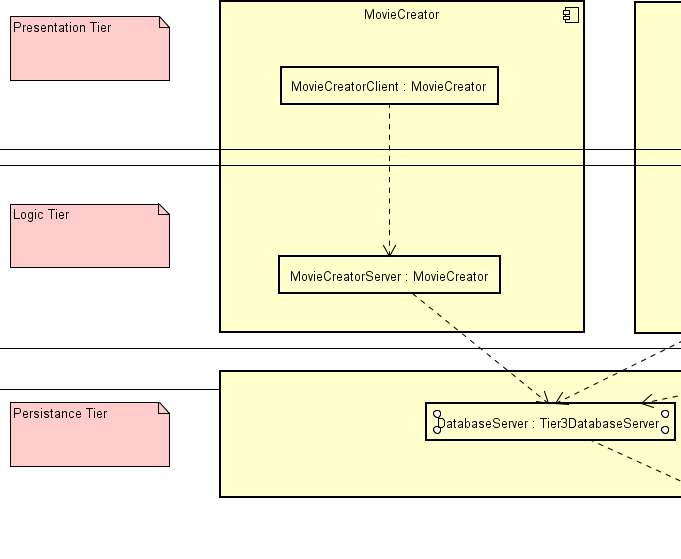


Figure 9 - The three tiers

The design of the system follows a 3-tier architecture that helps separate the user interface, business logic and data storage layers.

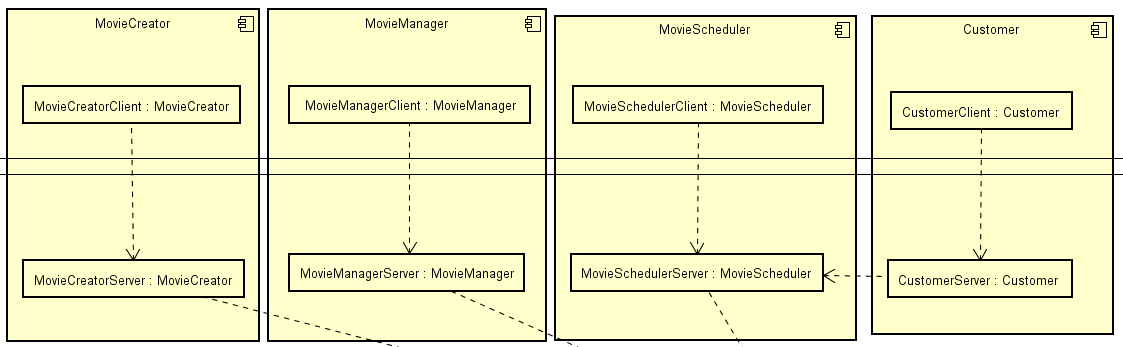


Figure 10 – Overview of all four components

The system is structured into four different components that communicate with each other; every component having specific responsibilities. For the first three components the first and second tiers are implemented in Java and for the last component tier 1 and 2 in C#.

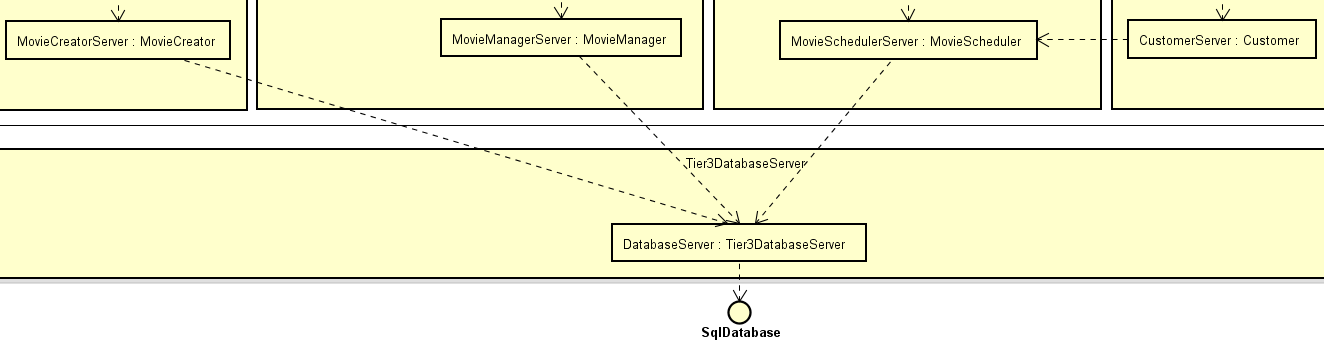


Figure 11 - Tier3 overview

The chosen method of data storage is a database. In order to benefit from the Entity Framework Core data access technology, the third tier was implemented in C#.

The communication between tiers is made through sockets using the TCP protocol. In both Figure 7 and Figure 8 communication implemented by the team through sockets is represented by the dotted arrows. To easily send data that can be interpreted by programs written in both Java and C#, the data sent between tiers is serialized with Json.

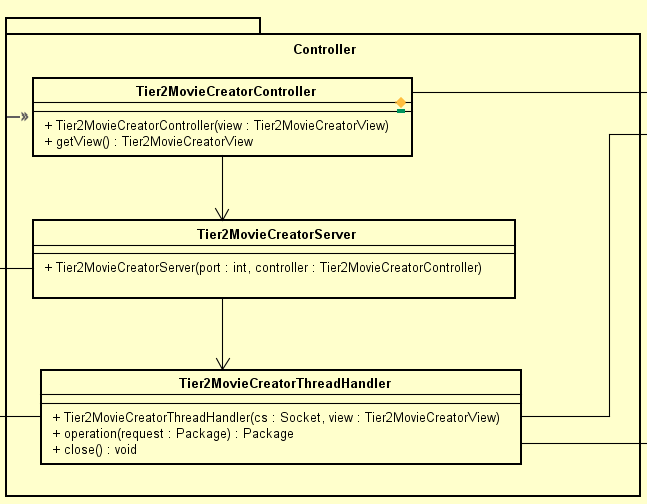


Figure 12 - Server Diagram

All servers in the system follow the structure shown in Figure 9. When the controller is instantiated through its constructor it will instantiate a Tier2MovieCreatorServer by giving it a port and injecting the controller. The server class will then open a socket on the given port and listen for connections. When a connection is made, the client socket will be injected in the ThreadHandler class, which runs on a thread, that handles the communication between the client and the server.

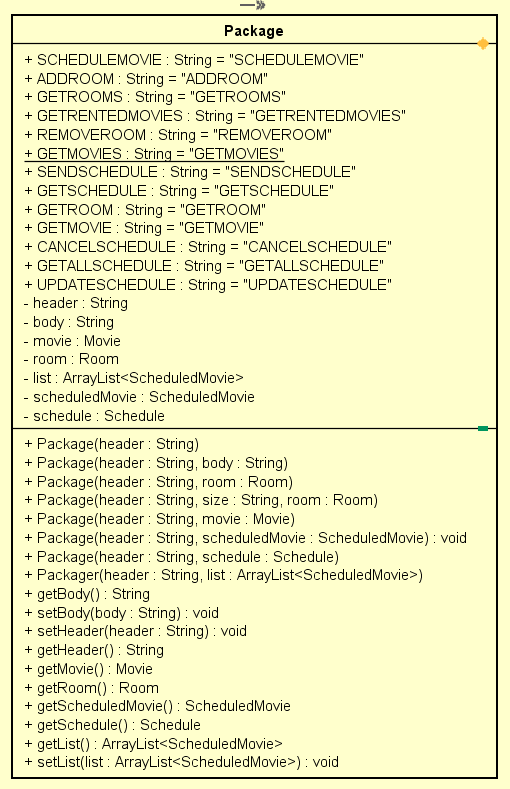


Figure 13 - Package class for sending data

The data sent between tiers is formatted by using a custom-made package. The Package class contains a header field so that the client or server receiving can identify what it needs to do with the data inside the package. This way, all objects that need to be sent through a TCP connection will be encapsulated in a package and then serialized into Json.

The third tier is common for all for components and it acts as a server that receives requests. According to those requests it runs queries on the database to either retrieve or save data. This part of the system uses EF Core to easily store objects into the database.

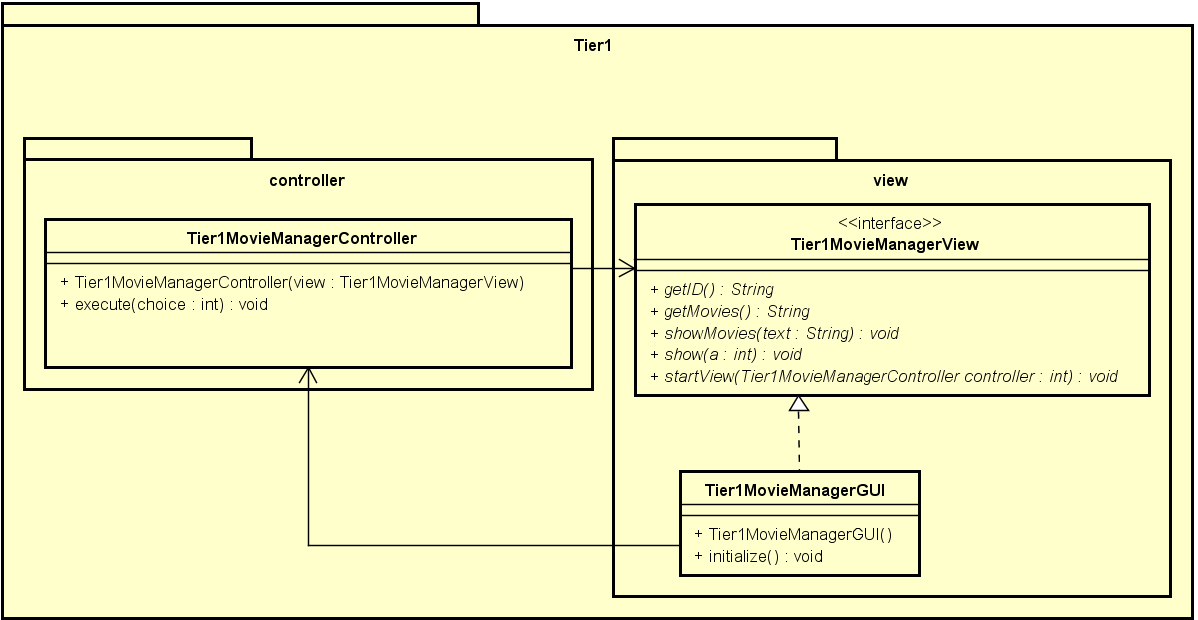


Figure 14 - Tier1 First Component

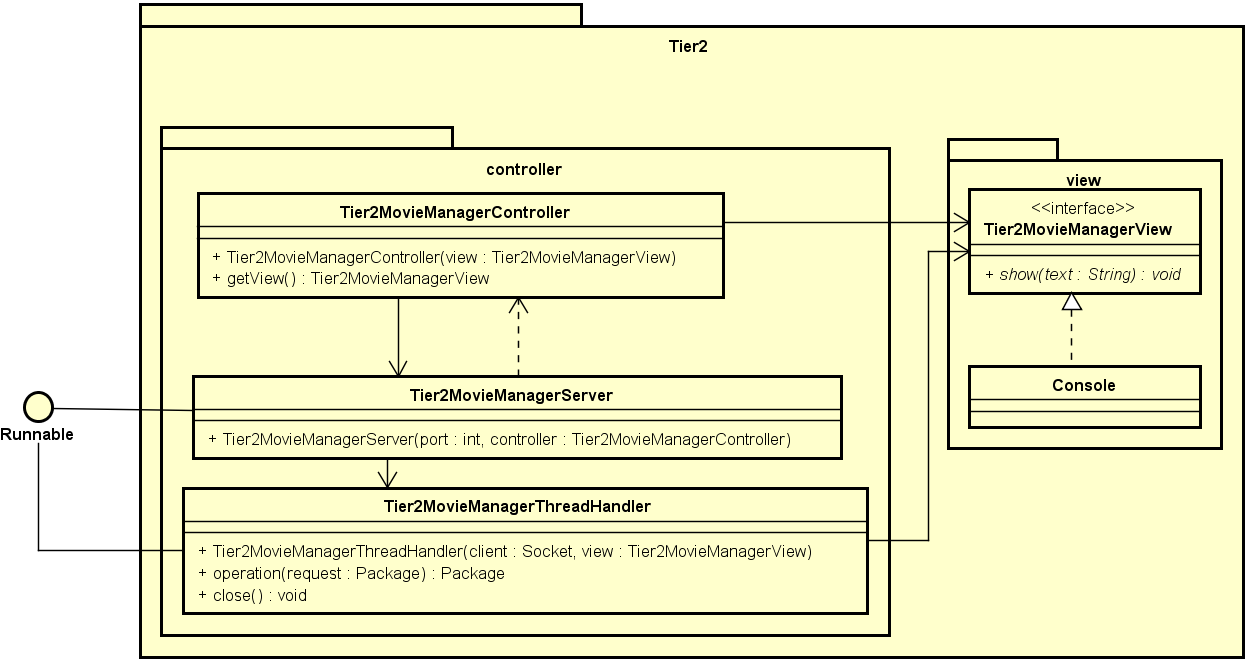


Figure 15 - Tier2 First Component

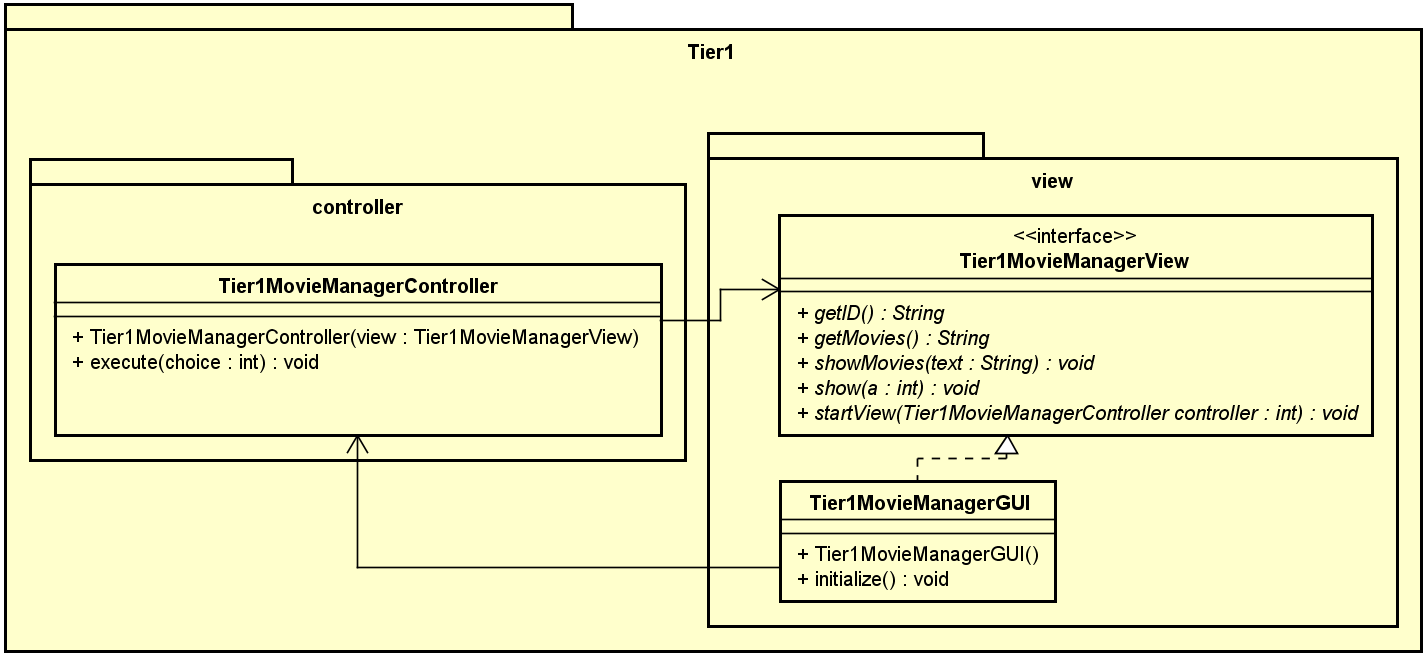


Figure 16 - Tier1 Second Component

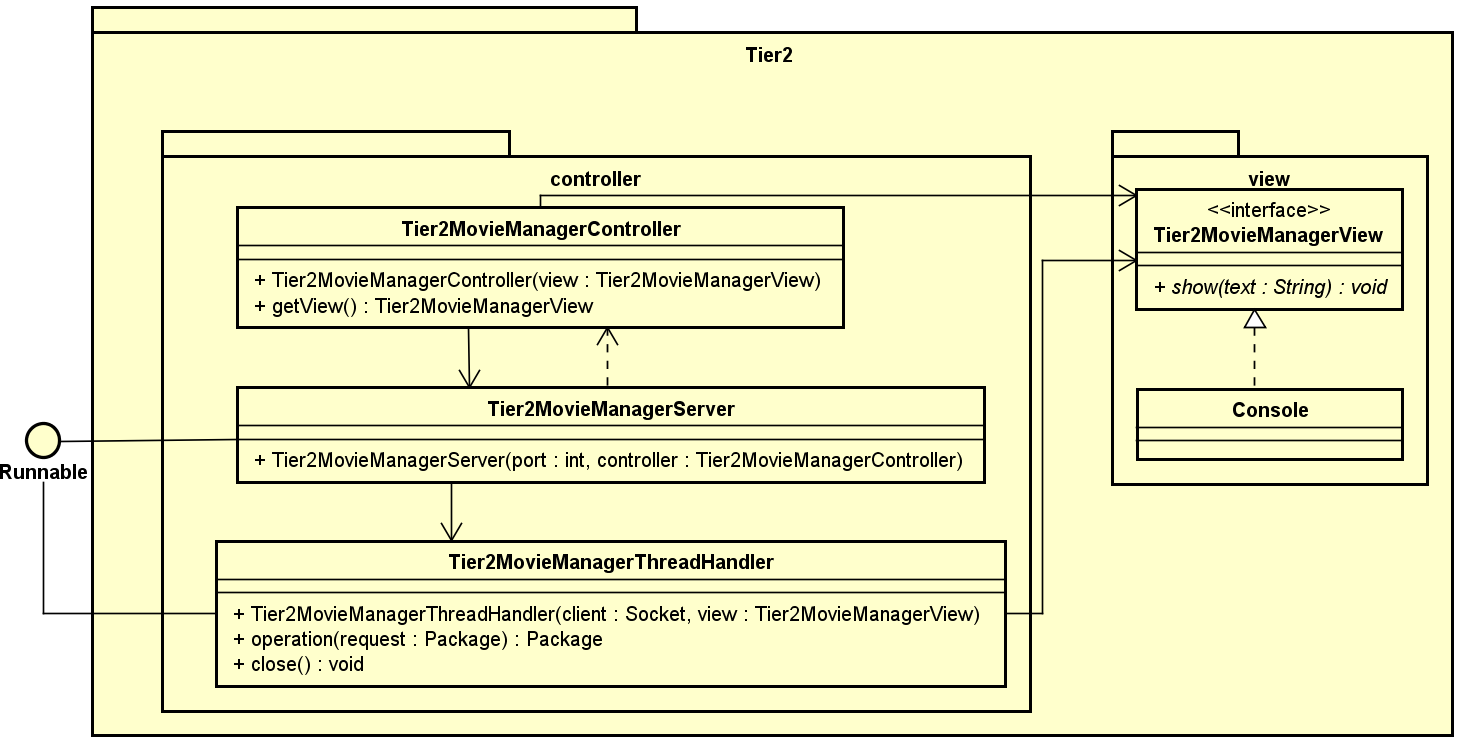


Figure 17 - Tier2 Second Component

The first two components are quite similar to each other in terms of functionality, so they are presented together. The first tier is responsible for displaying a GUI to the user and taking input from them.

The second tier acts as a server that receives requests from the first tier, interprets them and then it replies. The second tier also acts as a client for the third tier, as it sends data to tier 3 so it can save it into the database. Furthermore, on this level the input from the user is validated before sending it to the third tier.

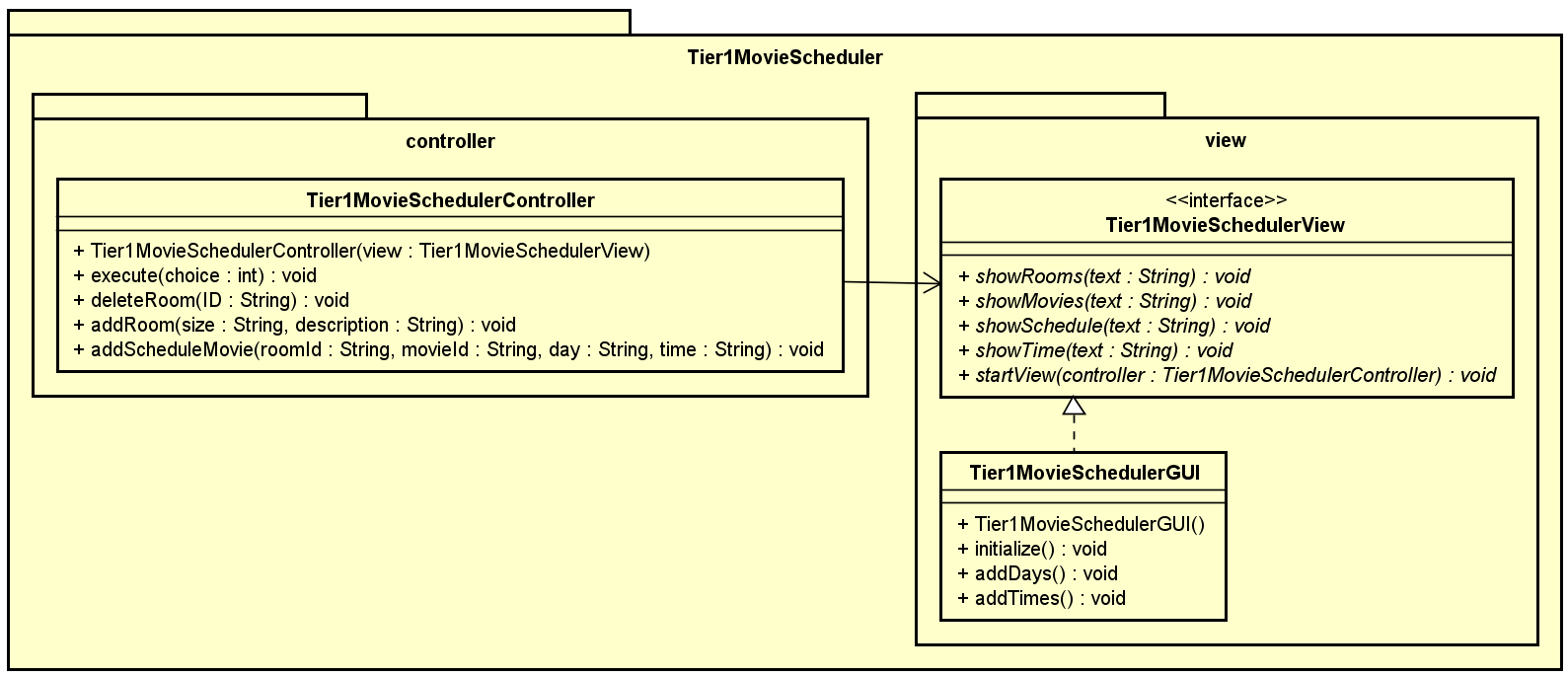


Figure 18 - Tier1 third component

The third component is responsible for creating rooms in which movies can be displayed and for creating the schedule of the cinema.

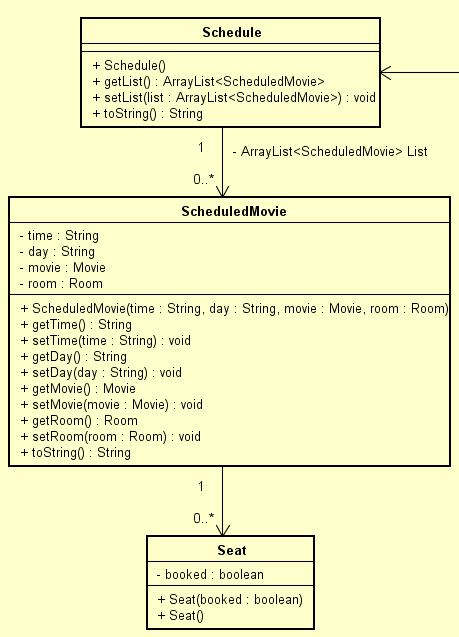


Figure 19 - MovieScheduler Schedule Model

A schedule consists of multiple Movie – Time – Day – Room – Seats pairings. The ScheduledMovie class is used to represent one pairing and the Schedule class contains a list of ScheduledMovie objects. The size of the Seats array is dependent on the size attribute of the Room the movie is displayed in.

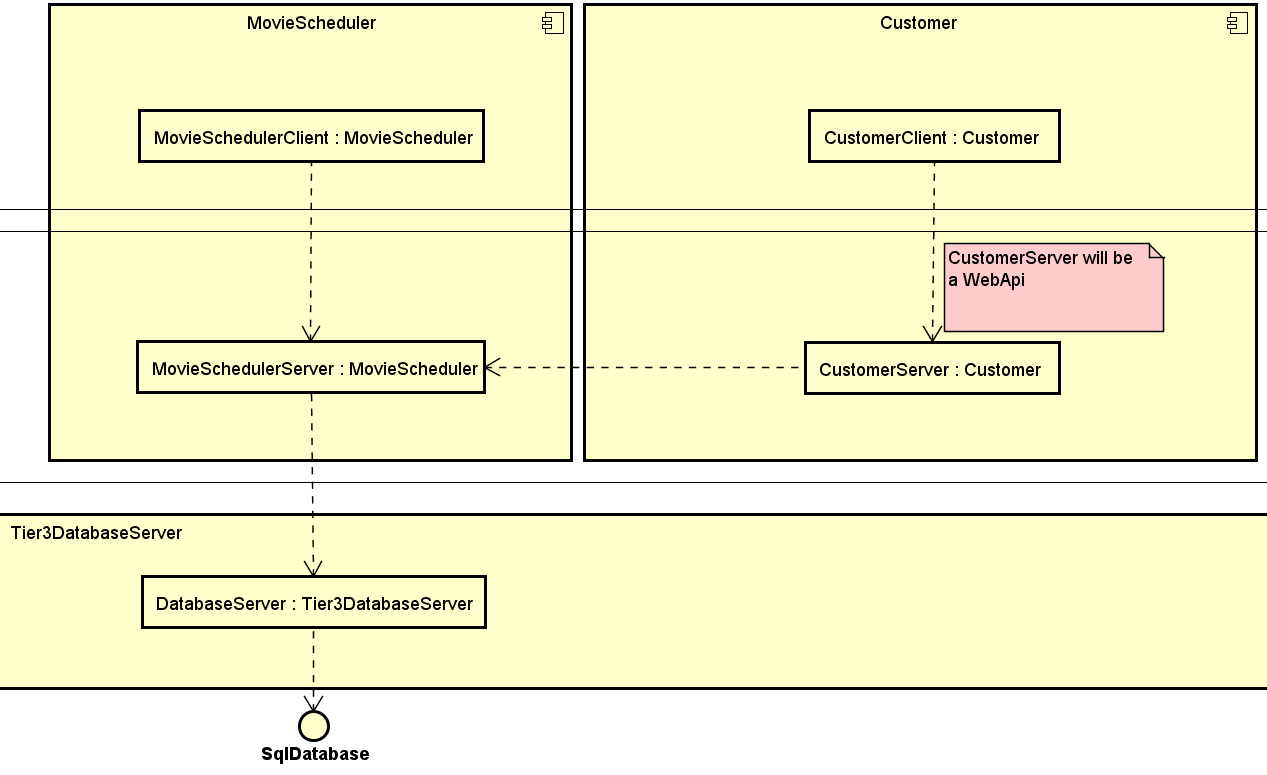


Figure 20 - Communication between last two components

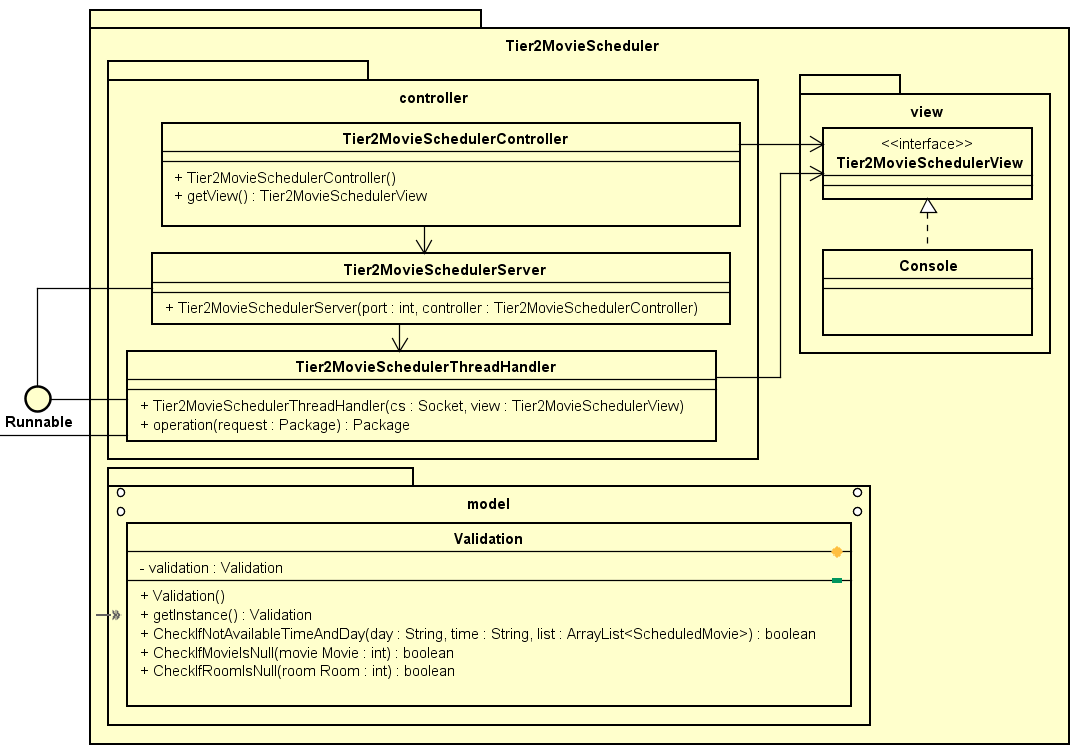


Figure 21 - Tier2 third component

First of all, the tier 2 in this component acts as a client to the third tier just like the other components, however, the server on tier 2 can receive requests from the tier 1 in the MovieScheduler component, which is implemented in Java, and from the WebApi from the fourth component, implemented in C#. This design makes it so that the WebApi does not have direct access to the data storage layer. Secondly, the input from the user on tier 1 is validated on this level through the use of the Validation class. The methods in this class are used to check for mistakes the user could have made when writing.

The last component was designed to be used by the clients of Zinema, therefore, it uses a web application to grant ease of access to the users. The WebApi once it starts will send a request to the server on the second tier of the previous component, in order to get the movie schedule for the cinema. The user will interact with this component through a web page designed with Razor Pages, that depending on the user’s input will send HTTP requests to the WebApi. The WebApi can return the schedule to the user on a GET request and can book a seat for a specific screening on a PUT request.

**Security**

Taking into consideration the design of the system, not too many security mechanisms are required to be implemented as potential losses are not particularly high. The last component uses a website and Webapi and this does expose the system to certain risks.

After the client chooses what seat to book they will be asked for the email address and phone number in order to be able to receive the ticket. Even though that data is not stored by our system, the message sent by the client to the Webapi could be intercepted and thus, information disclosure would be achieved by the attacker. As this scenario would best be avoided, SSL certificates are used as a proactive measure to deter such attacks. This grants the data both confidentiality and authenticity, but because anyone can access the website and there are no authentication protocols set in place this last component is still vulnerable to DoS attacks and it would be preferable to implement some prevention mechanisms.

The first three components send data that is not protected between tiers and this makes it vulnerable to attacks. However, the information sent by the system is not that confidential so disclosing it does not pose much of an issue. Tampering, though, is a more dangerous threat as it is not desirable for corrupted data to reach either tier 2 or 3.

It would be preferable that digital signatures would be used on the messages sent from tier 1 to ensure authenticity. Coupling that with public key encryption of the messages would give a good layer of security to the communication between the first two tiers.

The second tier sends data to the third tier to be stored in the database. Because there would already be some mechanisms implemented on the first tier, some less tight security mechanisms could be implemented just in case the previous mechanisms fail.

Because there is a fixed number of servers on tier 2 and only one server on tier 3, symmetric key encryption could be used to communicate between those two tiers. As there are only three components that communicate with the third tier, the server would only need to store three keys. In order to also provide some form of authentication but not in the shape of digital signatures, the messages sent could also contain a MAC that the tier 3 server would verify when received.

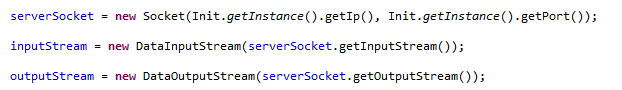
# System Implementation

The implementation uses the diagrams and specifications mentioned in the Design to create the whole system. The system was created using a 3-tier architecture and two different programming languages. The focus of this chapter will be to explain what facilitates the communication between the tiers.

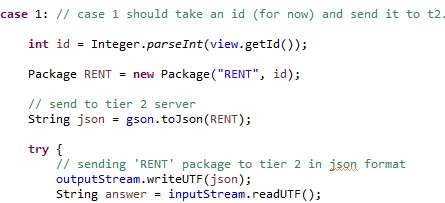
The Movie Creator and Movie Manager components are similar in the methods they employ. Explaining how the communications is done in the Movie Manager will also cover the Movie Creator.

The communication protocol that is used in the system is TCP (transfer control protocol). Once two sockets are connected, they can be used to transmit data in both directions.

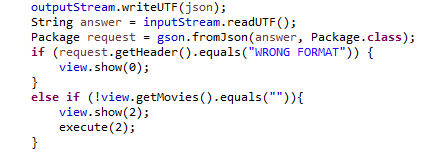
The communication in Movie Manager between tiers is achieved through TCP sockets. Tier 1 only acts as a client, sending out request to the server that is Tier 2 in order to receive data. First, it connects to the server socket, that is listening for clients, through a specific port and opens an Input Stream, which is used as a channel to send the requests, and an Output Stream, which is used as channel to receive the response from the server.



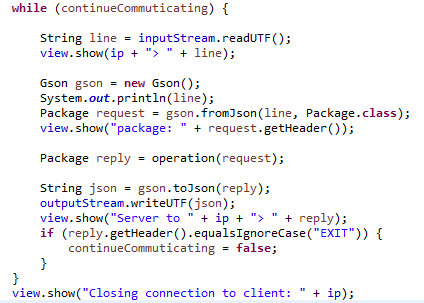
In the case of renting, when the user asks for the request to be sent, the system translates it into Json, sends in through the InputStream to the server that is waiting to handle request from the user.



The next of the step is waiting for the answer, decoding it from Json to the Package class and reading the contents. The view will act according to the response it receives and display the movies that are available to rented if it has been successful.



Tier 2 in Movie Creator acts both as a client and a server. As a server, it waits for the input, translates it from Json, does the specific actions to the header of the package sent and send the response to the client.

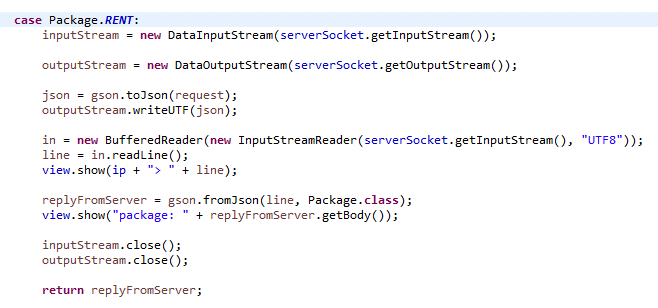


As a client, it acts when it needs to persist data or receive data that was already persisted on Tier 3. In the case of communication between the Tier 2 and Tier 3, the connection is opened on a per request basis. In between Tier 1 and Tier 2, continuous communication is desired to ensure a better user experience, but in between Tier 2 and Tier 3, there is no need for that. It doesn’t keep the database waiting more than is necessary.

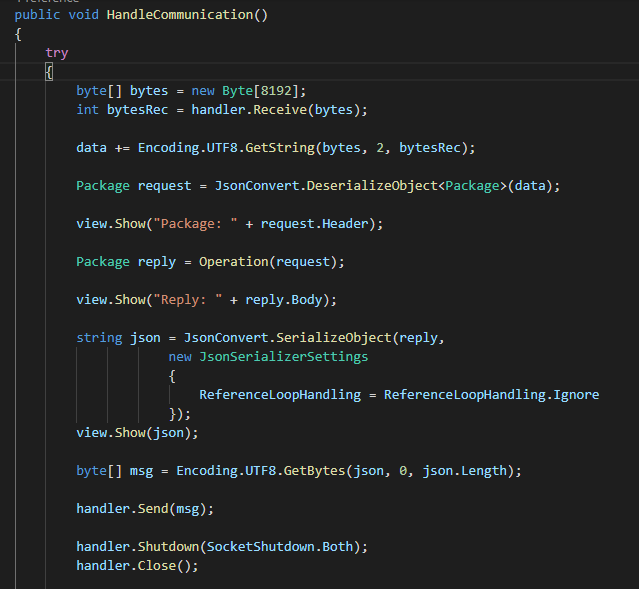
Tier 3 is written in C#, meaning that Tier 2 needs to handle the communication in a different way. Json in Newtonsoft serializes nulls by default, but Gson in java doesn’t. The Gson builder was adjusted to ensure that both tiers can understand each other’s message.



In JAVA sockets, the communication is done through a special kind of UTF in which the first two bytes sent are about the size of the message. C# uses the default UTF-8 format in which the system waits for something to signal the end of stream or message. To solve this issue, the response from Tier 3 is handled with a buffered reader that waits for a Newline or a EOF (End of Stream) to know that all the message has been sent. In the case of the following code, it waits for an EOF.

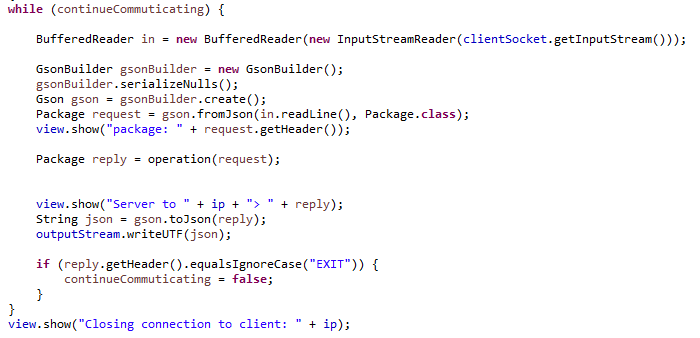


Tier 3 only acts as a server for Tier 2. For convenience, the message sent to the database was kept in the special UTF specific to JAVA. The solution to reading the message was to skip the first two bytes when the string in UTF 8 is decoded, so the message could be converted through Json.

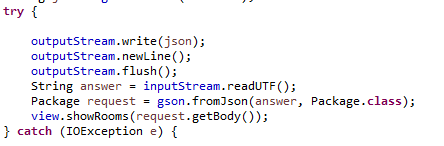


After the message is sent, the stream is closed so the Tier 2 receives the EOF.

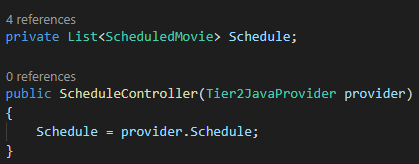
In order to introduce the last component, the Ticket Booker, the Tier 2 in Movie Scheduler needed some adaptations. It’s special because it has as clients both the C# web API and the JAVA Client. It could not receive the messages in the usual UTF from JAVA, so it was adapted to read with a BufferedReader. To know when the web API is done sending the message it receives a Newline.

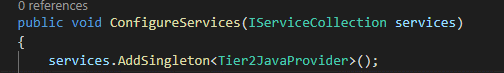


The client in JAVA had to be modified to also send a Newline at the end of each message and flush to make sure that are the bytes have been written because the program can be closed at any time by the user.

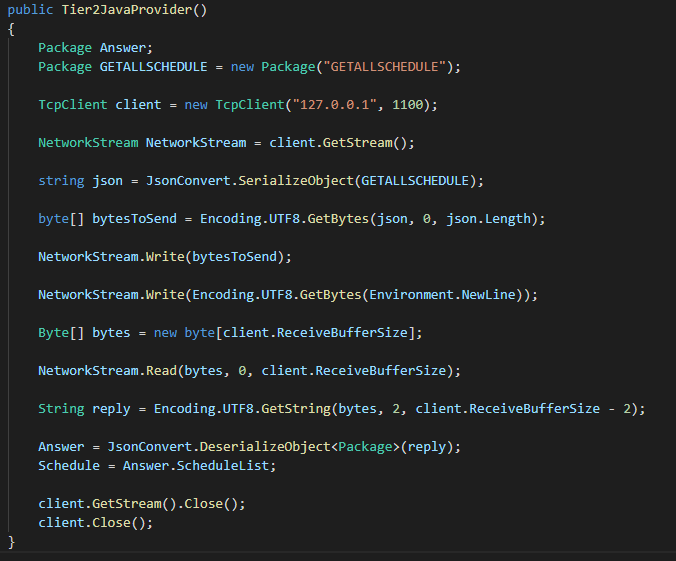


To receive the schedule, the controller uses a class through dependency injection that connects to the Tier 2 in JAVA to ask for the data. In this way, we make sure that the schedule is received only once at the start of the web API and not each time a user has a request.





The web API uses TcpClient to connect to Tier 2 in Movie Scheduler and creates a NetworkStream to be used for writing and reading messages.

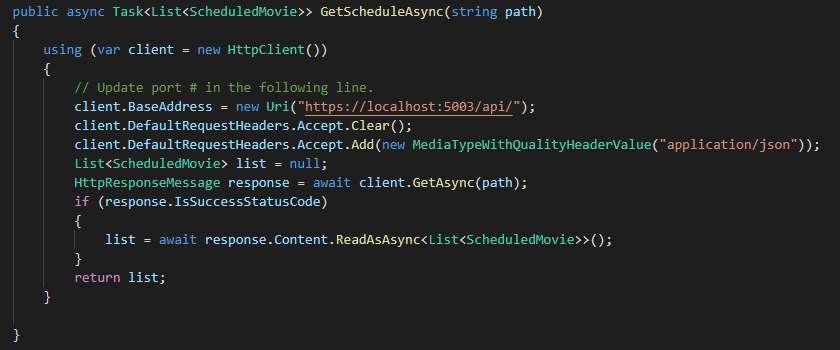


When receiving the reply, it skips the first two bytes again. To make sure that it doesn’t read more than it needs to, it has 2 subtracted from the total size of the message.

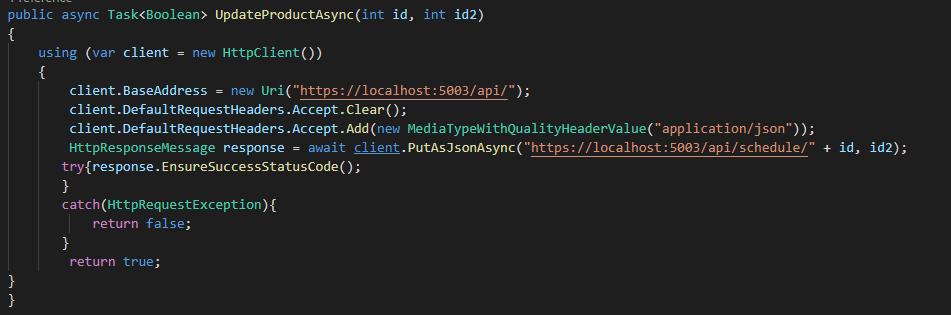
On the last part of the Implementation, the analysis of the communication on Tier 1 in the Ticket Booker will come into attention.

The way it receives data is through a GET request to the web API for the scheduled movies. To save the booked tickets, it uses a PUT request in which it introduces the link the desired movie and, the desired seat to book in the body. After the PUT request the web API sends a request for the schedule to be updated with the new seats.

On the GET request, the client sets as a header "application/json" so it knows that it wants to receive the information in Json. If the web API sends a success status code, then the response is read.



On the PUT request, the client sets the header again to know it sends json. If the request is successful, then it lets the client know by returning true and if it fails, it catches the exception and signals to the client to show an error page.



This concludes the explanation on how the communication was done and how it works in the current system. Next comes testing in which the system will be put through different scenarios to ensure that it behaves as expected.

# Testing

As the focus was put on communication and having a 3-tier architecture, there wasn’t much in the implementation of the model in each tier. The JUnit tests verify if the validation is done properly. Most of the system is tested using Test Cases that check the behaviour of the system. The JUnit tests can be found in Appendix F.

**Test Specifications**

Test Cases were used to ensure that the Use Cases and the requirements have been respected. The following table displays all the test specifications:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nr | Description | Actor | Precondition | Expected Result | Steps | Result |
| 1 | The user creates a movie | Movie Creator | The system must be connected to the database | A movie is created and added to the list | 1. User presses on the button “CreateMovie”  2. User introduces title and submits  3. User introduces creation year and submits  4. User introduces release date and submits  5. User introduces price and submits  6. User introduces the name of the studio and submits  7. User introduces the director’s name and submits  8. User introduces the description and submits  9. User introduces the main actor’s name and submits | A movie is created and added to a list. |
| 2 | The user wants to create a movie but inputs the year of creation in the wrong format | Movie Creator |  | The system displays an error message “Input a creation year made by digits” | 1. User presses on the button “CreateMovie”  2. User introduces title and submits  3. User introduces creation year and submits | The system displays an error message “Input a creation year made by digits” |
| 3 | The user wants to create a movie but inputs the release date in the wrong format | Movie Creator |  | The system displays an error message “Please enter the date in this exact format: [DD/MM/YYYY]” | 1. User presses on the button “CreateMovie”  2. User introduces title and submits  3. User introduces creation year and submits  4. User introduces release date and submits | The system displays an error message “Please enter the date in this exact format: [DD/MM/YYYY]” |
| 4 | The user wants to create a movie but inputs the price in the wrong format | Movie Creator |  | The system displays an error message “Input a price made by digits” | 1. User presses on the button “CreateMovie”  2. User introduces title and submits  3. User introduces creation year and submits  4. User introduces release date and submits  5. User introduces price and submits | The system displays an error message “Input a price made by digits” |
| 5 | The user wants to create a room | Movie Scheduler | The system must be connected to the database | A room is created and added to a list | 1. User chooses to create a room  2. User enters size of the room  3. User enters description  4. User validates his choice  5. Room is created and added to the list | A room is created and added to a list |
| 6 | The user wants to create a room but inputs the size in the wrong format | Movie  Scheduler | The system must be connected to the database | The system displays an error message “You need to input a number” | 1. User chooses to create a room  2. User enters size of the room | The system displays an error message “You need to input a number” |
| 7 | The user wants to delete a room | Movie Scheduler | The system must be connected to the database | The selected room will be removed from the system | 1. System displays a list of all rooms  2. User selects a room from the list  3. User deletes the room  4. System removes room from the list | The selected room will be removed from the system |
| 8 | The user wants to delete a room but doesn’t input the ID in the field or inputs the wrong format | Movie Scheduler | The system must be connected to the database | The system displays an error message “You need to input a number in the ID field” | 1. System displays a list of all rooms  2. User selects a room from the list | The system displays an error message “You need to input a number in the ID field” |
| 9 | The user wants to create a schedule | Movie Scheduler | Rooms are stored in the system  Managed list of movies is stored in the system  The system must be connected to the database | Schedule created and added to the database | 1. System displays a list of available movies, rooms, days and times  2. User inputs the id of the movie.  3. User inputs the id of the room.  4. User selects the day from a list of available days.  5. User selects the time from a list of available times.  6. User presses a button with the text “Input”.  7. Steps 2 to 7 are repeated until the user decides he/she is done.  12. After he is done, the user can press the button with the text “Send” to send over to the database the schedule. | Schedule created and added to the database |
| 10 | The user wants to see the current schedule | Movie Scheduler | Rooms are stored in the system  Managed list of movies is stored in the system  A schedule in the system  The system must be connected to the database | Schedule data displayed | 1. User presses the button “show” | Schedule data displayed |
| 11 | The user wants to create a schedule but doesn’t input the ID of Room or/and Movie | Movie Scheduler | Rooms are stored in the system  Managed list of movies is stored in the system  The system must be connected to the database | The system displays a message “Wrong ID/Wrong format Inputted (Must be a number present in the list written with digits)” | 1. System displays a list of available movies, rooms, days and times.  4. User selects the day from a list of available days.  5. User selects the time from a list of available times.  6. User presses a button with the text “Input”. | The system displays a message “Wrong ID/Wrong format Inputted (Must be a number present in the list written with digits)” |
| 12 | The user wants to create a schedule but doesn’t a number as the ID of Room or/and Movie | Movie Scheduler | Rooms are stored in the system  Managed list of movies is stored in the system  The system must be connected to the database | The system displays a message “Wrong ID/Wrong format Inputted (Must be a number present in the list written with digits)” | 1. System displays a list of available movies, rooms, days and times  2. User inputs the id of the movie.  3. User inputs the id of the room.  4. User selects the day from a list of available days.  5. User selects the time from a list of available times.  6. User presses a button with the text “Input”. | The system displays a message “Wrong ID/Wrong format Inputted (Must be a number present in the list written with digits)” |
| 13 | The user wants to create a schedule but doesn’t input the ID of a Room or/and Movie from the list | Movie Scheduler | The system must be connected to the database | The system displays a message “Wrong ID/Wrong format Inputted (Must be a number present in the list written with digits)” | 1. System displays a list of available movies, rooms, days and times  2. User inputs the id of the movie.  3. User inputs the id of the room.  4. User selects the day from a list of available days.  5. User selects the time from a list of available times.  6. User presses a button with the text “Input”. | The system displays a message “Wrong ID/Wrong format Inputted (Must be a number present in the list written with digits)” |
| 14 | The user wants to rent a movie that is not rented | Movie Manager | Movies that are not rented in the database | A movie is rented by the Movie Manager | 1. System displays list of movies that are not rented  2. User inserts a movie id  3. System registers the movie as rented  4. The system displays a pop-up message informing the user of the successful operation  5. The system automatically updates the list of movies that are not rented | A movie is rented by the Movie Manager |
| 15 | The user wants to rent a movie, but the list is empty | Movie Manager | Empty list of movies that are not rented | System displays a pop-up message with “No Available Movies” | 1. System displays list of movies that are not rented | System displays a pop-up message with “No Available Movies” |
| 16 | The user wants to rent a movie that doesn’t exist or is rented already | Movie Manager | Movies that are not rented in the database | System displays an error message with “Invalid Input” | 1. System displays list of movies that are not rented  2. User inserts a movie id  3. User presses the button “Rent Movie” | System displays an error message with “Invalid Input” |
| 17 | The user wants to rent a movie and inputs something else instead of a number in the id field | Movie Manager |  | System disables the rent button | 1. System displays list of movies that are not rented  2. User inserts a movie id | System disables the rent button |
| 18 | The user rents a movie and then the list of movies that are not rented is empty | Movie Manager | Movies that are not rented in the database | System displays a pop-up message with „No Available Movies” | 1. System displays list of movies that are not rented  2. User inserts a movie id  3. System registers the movie as rented  4. The system displays a pop-up message informing the user of the successful operation  5. The system automatically updates the list of movies that are not rented | System displays a pop-up message with „No Available Movies” |

# Results and Discussion

In the end the Product Owner received a satisfactory product made specifically based on his business requirements. The system has been built using Java and C# and employs a three-tier architecture design. It can fulfil everything that the Product Owner requires and more depending on the company’s needs.

# Conclusion

The project followed the Unified Software Development Process and SCRUM. At the start the focus was on setting up why the system should be created. As it advanced the functional and non-functional requirements were defined and use cases were created accordingly.

A Domain Model was used to analyse the problem and understand how different objects would interact with each other. Next, the design of the system was defined using Design Class diagrams based on the existing Domain model.

The Design Class diagrams acted as blueprints for the implementation of a 3-tier architecture system. In the last phases of the project, the application was tested based on the Use Cases.

Once finalized, the results were subject to discussion and analysis to bring further improvements in the future.

# Project Future

Even though the system is robust and overall well done, it still could be improved by quite a lot.

First of all, it could be expanded by adding a currency system that would work with the Movie Manager and Ticket Booker components. The manager would use the currency to rent movies while the customer would replenish his balance by booking movie tickets.

Secondly, the customer side of the system could definitely benefit from a proper authentication system and as a last addition, using the same authentication as before, the clients could be separated into customers and advertisers, the latter part being able to, apart from booking tickets, also rent slots to show advertisements before or after the scheduled movie’s screening.

# List of Appendices

Appendix A: Project Description

Appendix B: Activity Diagrams, Use Cases

Appendix C: Design Class Diagram, System Architecture Diagram

Appendix D: Sequence Diagram

Appendix E: The Code

Appendix F: JUnit Tests

Appendix G: User Guide

Appendix H: JavaDoc

# Sources of Information

# **Webography**

Danish Film Institute, 2012. *dfi.dk.* [Online]   
Available at: https://www.dfi.dk/files/docs/2018-02/f\_and\_f\_2012\_screen\_singlepages%20%281%29.pdf  
[Accessed 20 September 2018].

Danish Film Institute, 2017. *dfi.dk.* [Online]   
Available at: https://www.dfi.dk/files/docs/2018-02/Facts%20and%20Figures%202017%20%281%29.pdf  
[Accessed 20 September 2018].

Danish Film Institute, 2018. *dfi.dk/.* [Online]   
Available at: https://www.dfi.dk/files/docs/2018-04/Facts-and-figures-2018\_0.pdf  
[Accessed 20 September 2018].

http://www.upedu.org/, 2018. *http://www.upedu.org/.* [Online]   
Available at: http://www.upedu.org/  
[Accessed 20 September 2018].

media.netflix.com, 2012. *media.netflix.com.* [Online]   
Available at: https://media.netflix.com/en/press-releases/netflix-launches-in-sweden-denmark-norway-and-finland-migration-1  
[Accessed 20 September 2018].

Motion Picture Association of America, 2017. *mpaa.org.* [Online]   
Available at: https://www.mpaa.org/wp-content/uploads/2017/03/MPAA-Theatrical-Market-Statistics-2016\_Final.pdf  
[Accessed 20 September 2018].

Nielsen, M. B., 2018. *dst.dk.* [Online]   
Available at: https://www.dst.dk/en/Statistik/emner/kultur-og-kirke/film-boeger-og-medier/biografer-og-film  
[Accessed 20 September 2018].

POSitive Cinema, 2018. *http://positivecinema.com/.* [Online]   
Available at: http://positivecinema.com/  
[Accessed 20 September 2018].

Scrum.org, 2018. *Scrum.org.* [Online]   
Available at: https://www.scrum.org/resources/what-is-scrum  
[Accessed 20 September 2018].

Vista Cinema, 2018. *Vista.* [Online]   
Available at: https://www.vista.co/en/vista-products/vista-products/cinema/  
[Accessed 20 September 2018].