**Cristina Doroftei, Gheorghe Marian Mocanu,
Krisztian Szabo, Małgorzata Weronika Witkowska**
aka “Team Two”

Version 1.0

RUP Vision Document for the BioTrio Cinema Management System

GitHub Repository: https://github.com/SzaboKrisztian/BioTrio

Document pertains to commit 4cb8d3b38b619dd559ddba72edd349a777d57093 on master branch

Document History

Version	Date	Author	Notes
0.1	14-05-2019	Chris	Basic document outline
0.1.1	15-05-2019	Chris	Added the first few sections
0.2	16-05-2019	Team	Added ITO material
0.3	19-05-2019	Malgo	Added half of the use case casuals
0.4	21-05-2019	Cristina, Marian	Added the rest of the use case casuals
0.5	28-05-2019	Team	Use case refinement, small content additions
0.6	29-05-2019	Chris	General formatting, polished some sections
0.8	31-05-2019	Team	Added the missing diagrams
1.0	01-06-2019	Chris	Finishing touches. Final version

Table of Contents

1. Introduction....5
   a. Purpose (Chris)....5
   b. Scope (Chris)....5
   c. Definitions and abbreviations (Chris)....5
2. Business Analysis....6
   a. Problem statement (Chris)....6
   b. Feasibility study (Malgo)....6
      Executive summary....6
      Operational feasibility....7
      Technical feasibility....7
   c. Gantt Chart (Chris)....8
   d. Risk analysis (Marian)....9
   e. SWOT analysis (Cristina)....10
4. Problem domain model....12
   a. Functional requirements (Team)....12
   b. “Nice to have” requirements (Team)....12
   c. Implied requirements: (Team)....12
   d. Domain model (Cristina)....13
   e. Stakeholder and user descriptions (Chris)....13
      Stakeholders....13
      Users....14
   f. User environment (Chris)....14
   g. Use case list (Team)....15
5. Product overview....17
   a. Vision (Chris)....17
   b. Summary of capabilities (Chris)....17
   c. Assumptions and dependencies (Chris)....17
   d. Security aspects (Chris)....18
   e. Packaging (Chris)....18
   f. Cost and pricing (Chris)....19
   g. Licensing and installation (Chris)....19
6. Product Features....20
   a. Use case descriptions (Team)....20
      UC01 – View movie list....20
      UC02 – View movie details....20
      UC03 – Add new movie....20
      UC04 – Update movie....21
      UC05 – Delete movie....22
      UC06 – View upcoming movies list....22
      UC07 – Add movie to the upcoming movies list....23
      UC08 – Remove movie from the upcoming movies list....23
      UC09 – View theaters list....23
      UC10 – Add new theater....24
      UC11 – Update theater....24

UC12 – Delete theater....25
UC13 – View upcoming screenings....25
UC14 – View screenings list....25
UC15 – Add new screening....26
UC16 – Update screening....28
UC17 – Delete screening....29
UC18 – View employee list....30
UC19 – Add new employee....30
UC20 – Update employee....30
UC21 – Delete employee....31
UC22 – View user list....31
UC23 – Add new user....32
UC24 – Update user....32
UC25 – Delete user....33
UC26 – Sell ticket....33
UC27 – Void ticket....36
UC28 – Redeem booking....37
UC29 – Create booking....38
UC30 – Cancel booking....40
UC31 – View technologies list....41
UC32 – Add new technology....42
UC33 – Delete technology....42
UC34 – Delete past screenings....42
b. Database structure (Team)....44

1. Introduction

a. Purpose (Chris)

The purpose of this document is to define and collect the high level needs of the stakeholders on the one hand, and the features of the developed solution that are meant to fulfill those said needs on the other. The document also serves, at the same time, as a window into the thought process of the development team, showcasing the journey through the different phases of the unified process, from gauging the feasibility of such a project, through the actual development, all the way to deploying it in the customer's production environment.

b. Scope (Chris)

The BioTrio web application is meant to serve firstly as a portal through which the cinema employees are to track and manage their day-to-day activities and the related resources, and secondly as a portal through which movie goers are to more easily access some of the services provided by the cinema. The team will develop the system with round the clock availability, user friendliness, responsiveness, and robustness in mind, and it will be accessible and usable from any of the major modern browsers.

c. Definitions and abbreviations (Chris)

Starting here and throughout the rest of this document, the following terms will be used as defined below:

Screening – The projection of a movie in a particular cinema theater, at a particular scheduled point in time.

The client – "BioTrio" cinema company that commissioned the development of the system

User – An employee of "BioTrio" that will use the system in their daily activities

Customer – A movie goer that frequents the "BioTrio" cinema and potentially uses the web application

2. Business Analysis

a. Problem statement (Chris)

The provincial cinema “BioTrio” would like to have developed a web application to serve as a management interface for its resources and employees’ activities. The system is to, first of all, allow tracking and managing of movies, theaters, screenings, employees, tickets, and bookings, from the business’ point of view, and secondly to allow customers to book tickets in advance for screenings, as this both increases user satisfaction by granting them autonomy and control, and reduces the workload for the employees that would normally be tasked with taking phone bookings.

Additionally, the system is to be developed in a flexible manner, so as to accommodate future expansion of the cinema, or technological upgrades.

b. Feasibility study (Malgo)

Executive summary

Based on "Introduction to Information Systems: Supporting and Transforming Business" By R. Kelly Rainer, Efraim Turban, there are six elements to a feasibility study:

1. Operational feasibility – an analysis of who are the system’s users, how the system is going to be used and how it will support the business.
2. Technical feasibility – an analysis of the software and hardware status and needs, technological know-how, and available technologies. Risk analysis can be included here.
3. Schedule feasibility – an analysis of the project development timeline including activities and deadlines. (to be found in section **2.c Gantt Chart**)
4. Legal and contractual feasibility – an analysis of legal issues and binding contracts.
5. Political feasibility – an analysis of key stakeholders’ view on the system.
6. Economic feasibility – a financial assessment of the project. Tools that can be used include cost-benefit analysis, present value and return on investment calculations.

The BioTrio system is our proposition for first year's final project, therefore our feasibility study is limited to data that we were given in the task description. That is the reason why our focus is on technical feasibility and operational feasibility.

Operational feasibility

The main reason for creating the BioTrio management system is increasing seat booking by customers. An increase in this behavior results in faster service in contrast to the present marginal booking via phone, which results in queues and slower service. The system must be intuitive, so visitors are not discouraged to use it.

The system in the form of a web application will be used externally to book seats by customers and internally to manage those bookings by the employees.

We aim at four levels of access to the system:

- For customers to view screening and movie information, and to create and cancel bookings
- For employees to handle ticketing and booking
- For managers to handle screenings and movies
- For administrators to handle employee information, user accounts, and theater related information

From the organization's point of view the system will be a part of daily routines for the ticket office to sell booked tickets and get an overview on available tickets. Furthermore, the web application will serve as an information provider about the current and upcoming movies in the cinema. Lastly the system will be designed to manage the cinema and its employees.

Technical feasibility

Hardware status:

Without having actual data from a real company we have to assume that the cinema has checked the hardware in house and either has resources to either buy/lease new computers or have enough well working equipment for all the users: cashiers in the ticket office and management in all levels. Enough hardware will ensure that responsible employees will access the system according to their privileges to overview or make changes. On the customer point of view a computer or smartphone is necessary to make a booking through the system.

Software status:

The system will be built using technologies that we got familiar with during the first year of computer science program. Those technologies include:

Java for back-end code, Spring Boot web framework, Thymeleaf for templates, SQL for databases, GitHub and Git for version control, and Bootstrap and CSS for responsive front-end development.

Expertise status – this part is based on technical risk diagram and takes in consideration 3 factors:

- Project complexity: BioTrio does not have any management system thus far, so we need to build it from scratch – it is a low structure project.
- Project size: It is a medium size project with possibility to develop further. It is meant to be completed in 5 weeks.

- Familiarity with technology: In the past we completed several projects, but they were on a much smaller scale or they were not applying all the mentioned technology. It is a first project with Spring Boot of such complexity therefore our technological know-how is low to medium as a starting point.

Based on those 3 factors we evaluate this project to have a high technical risk.

Below is a technical feasibility risk diagram (source: Hoffer J. A, George J. E, Valacich J. S. (1999), “Modern systems analysis & design”), expanded by a “medium” column for the project size.

		Low structure*	High structure*
High familiarity with technology or application area	Large project	(1) Low risk (susceptible to mismanagement)	(2) Low risk
	Medium project	(3)	(4)
	Small project	(5) Very low risk (susceptible to mismanagement)	(6) Very low risk
Low familiarity with technology or application area	Large project	(7) Very high risk	(8) Medium risk
	Medium project	(9) **High risk**	(10)
	Small project	(11) High risk	(12) Medium-low risk

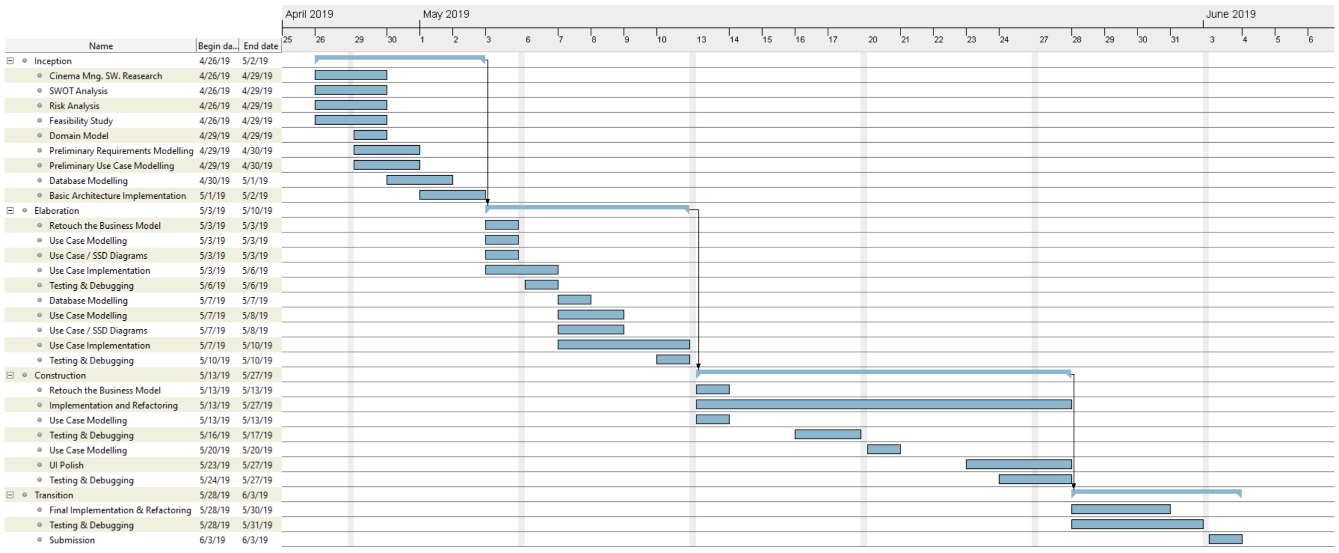
*Structure: New system or renovation of existing system? Many or few organizational changes resulting from the system? User and management commitment to the system? The degree of information needed from users (source: ITO presentation “Feasibility” by James Brink).

Conclusion:

After considering operational, technical and schedule factors we conclude that the project is feasible. We keep in mind that our expertise is the weakest part but since the purpose of this project is learning, we put a strong focus to gain new knowledge and complete the project in a timely manner.

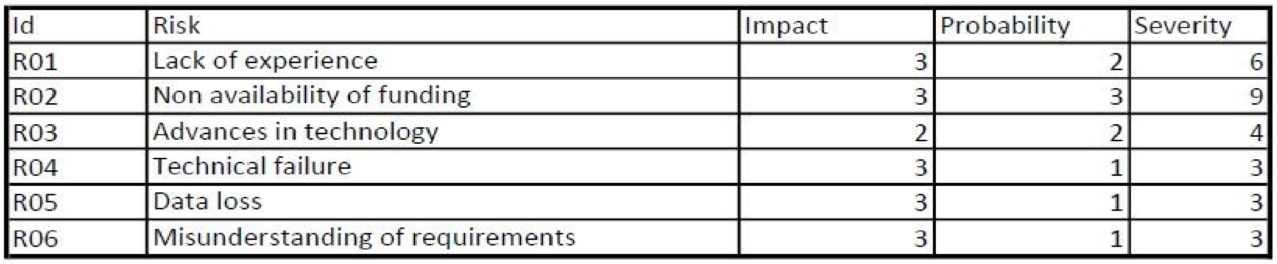
c. Gantt Chart (Chris)

We have planned to develop the system according to the rational unified process, which is an iterative software development process. Its defining characteristic is that it is in fact an adaptable process framework, and as such, we have taken the elements that we felt will suit the task at hand, our capabilities, and the technological context of the development process. Right in the beginning of the inception phase we planned to divide the allocated time as follows:

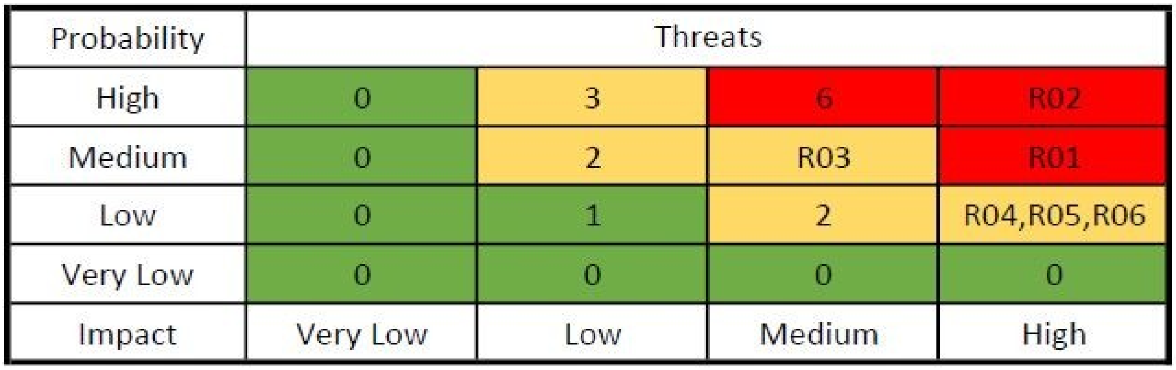


Name	Begin da...	End date
Inception	4/26/19	5/2/19
Cinema Mng. SW. Reasearch	4/26/19	4/29/19
SWOT Analysis	4/26/19	4/29/19
Risk Analysis	4/26/19	4/29/19
Feasibility Study	4/26/19	4/29/19
Domain Model	4/29/19	4/29/19
Preliminary Requirements Modelling	4/29/19	4/30/19
Preliminary Use Case Modelling	4/29/19	4/30/19
Database Modelling	4/30/19	5/1/19
Basic Architecture Implementation	5/1/19	5/2/19
Elaboration	5/3/19	5/10/19
Retouch the Business Model	5/3/19	5/3/19
Use Case Modelling	5/3/19	5/3/19
Use Case / SSD Diagrams	5/3/19	5/3/19
Use Case Implementation	5/3/19	5/6/19
Testing & Debugging	5/6/19	5/6/19
Database Modelling	5/7/19	5/7/19
Use Case Modelling	5/7/19	5/8/19
Use Case / SSD Diagrams	5/7/19	5/8/19
Use Case Implementation	5/7/19	5/10/19
Testing & Debugging	5/10/19	5/10/19
Construction	5/13/19	5/27/19
Retouch the Business Model	5/13/19	5/13/19
Implementation and Refactoring	5/13/19	5/27/19
Use Case Modelling	5/13/19	5/13/19
Testing & Debugging	5/16/19	5/17/19
Use Case Modelling	5/20/19	5/20/19
UI Polish	5/23/19	5/27/19
Testing & Debugging	5/24/19	5/27/19
Transition	5/28/19	6/3/19
Final Implementation & Refactoring	5/28/19	5/30/19
Testing & Debugging	5/28/19	5/31/19
Submission	6/3/19	6/3/19

d. Risk analysis (Marian)



Id	Risk	Impact	Probability	Severity
R01	Lack of experience	3	2	6
R02	Non availability of funding	3	3	9
R03	Advances in technology	2	2	4
R04	Technical failure	3	1	3
R05	Data loss	3	1	3
R06	Misunderstanding of requirements	3	1	3



Probability	Threats			
High	0	3	6	R02
Medium	0	2	R03	R01
Low	0	1	2	R04,R05,R06
Very Low	0	0	0	0
Impact	Very Low	Low	Medium	High

In our process of creating this piece of software we might encounter a few risks of which we must be aware of while developing it.

First of all, our lack of experience can cause us to delay the delivery of the product. In order to avoid it, we should not only spend no time on small details until the deadline, but also concentrate on the main features of our software. This way, there is no need to rush in order to comply the deadline. Moreover, time can be spent on looking into those small details after the product is delivered.

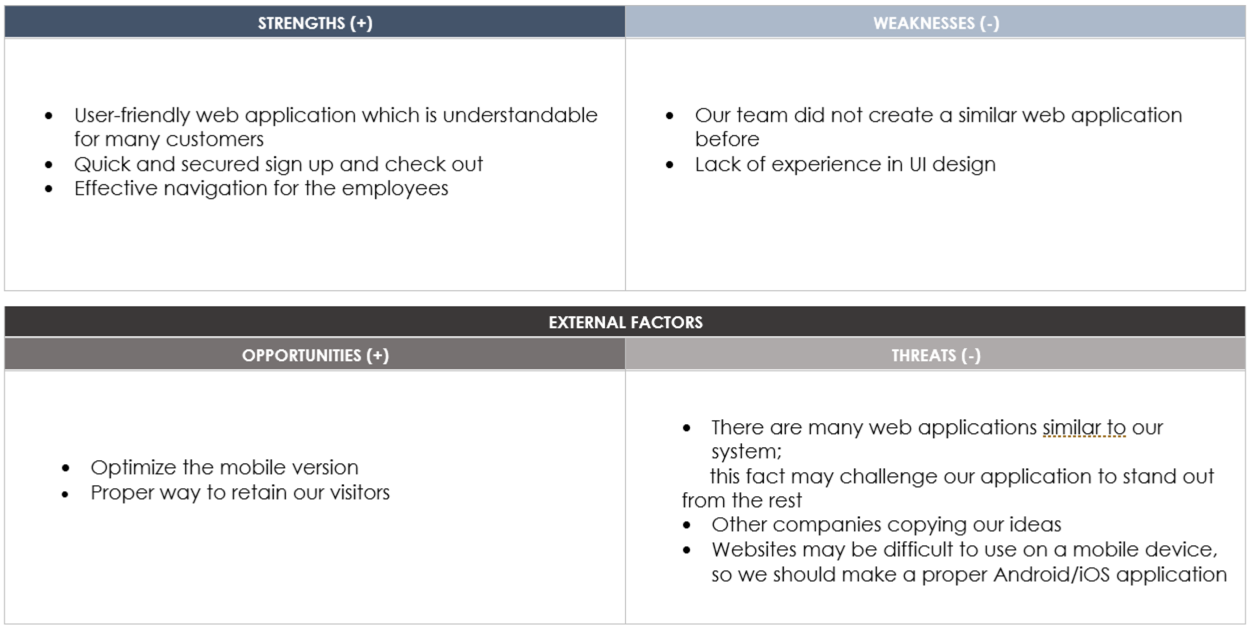
The fact that there is no funding for all the work put towards this project may cause us to feel less motivated, but the experience gained throughout working on it can be more valuable than monetary rewards.

Furthermore, since technology advances very fast nowadays, we need to keep our devices, developing software and personal knowledge up to date. Thus, the ability to work on both current and future projects will be enhanced, since most of the technical failures are avoided.

When it comes to preventing from losing our data, using an online repository (GitHub), which not only stores our data, but also lets us access different versions of it, might be the best option to work with.

Understanding of requirements for developing any project is probably the most important aspect. The development team is multicultural and of all ages, therefore, the life experiences of team members offer a wide perspective while designing any project.

e. SWOT analysis (Cristina)



STRENGTHS (+)	WEAKNESSES (-)
• User-friendly web application which is understandable for many customers • Quick and secured sign up and check out • Effective navigation for the employees	• Our team did not create a similar web application before • Lack of experience in UI design

EXTERNAL FACTORS	
OPPORTUNITIES (+)	THREATS (-)
• Optimize the mobile version • Proper way to retain our visitors	• There are many web applications similar to our system; this fact may challenge our application to stand out from the rest • Other companies copying our ideas • Websites may be difficult to use on a mobile device, so we should make a proper Android/iOS application

By doing a SWOT analysis for our project, our team tried to discover which are the resources and capabilities that guarantee our website's success and the potential weaknesses that can affect the project.

Firstly, the internal factors are represented by our strengths and weaknesses. These characteristics are usually within our control. One of the aspects of our project that is strong and give us a competitive advantage is the user-friendly web application. We find the user experience very important because it defines how easily the user interacts with the system and gives them a sense of value. We tried to make our web application very easy-to-navigate by separating all the functionalities into logical categories so the users can find their way around without any difficulties. Furthermore, our system provides a secured login system where all the users can access different functionalities based on their employee privileges.

Conversely, one of our weaknesses is the fact that we did not create a similar web application before and the lack of experience may cause delays when getting some tasks done, but our team tried to stay focused on the things that we found important. In addition, our user interface design is also a weakness. Even if our application is easy to use, the visual design may not be particularly impressive.

Secondly, the external factors are represented by our opportunities and threats. Unlike the internal factors, these ones are out of our control and can be represented by the marketplace.

One of the opportunities is represented by retaining our visitors. It is known that is better to have long-term customers than single-deal customers. Also, staying connected with the customers is easier when you already got to know their needs, and this usually happens with the long-term customers.

On the other hand, there are many web applications similar to our project. This threat may cause a loss to our number of customers, because they can also use another web application with the same functionalities as ours. This fact should challenge us to think about some out of the ordinary features that are not used by many web applications. Also, in the future we should consider making a proper iOS/Android application because clients usually prefer to access an application already installed on their phone instead of accessing a website on their mobile browser.

Making the SWOT analysis is a very important part of our project and it helped us understand how to compete in the market and ensure the success.

4. Problem domain model

a. Functional requirements (Team)

- REQ001: The system shall be able to manage the movies
- REQ002: The system shall be able to manage the movie theaters
- REQ003: The system shall be able to manage the screenings
- REQ004: The system shall be able to manage the employees
- REQ005: The system shall be able to manage the user credentials
- REQ006: The system shall be able to manage the ticket selling
- REQ007: The system shall be able to provide online ticket booking functionality

b. “Nice to have” requirements (Team)

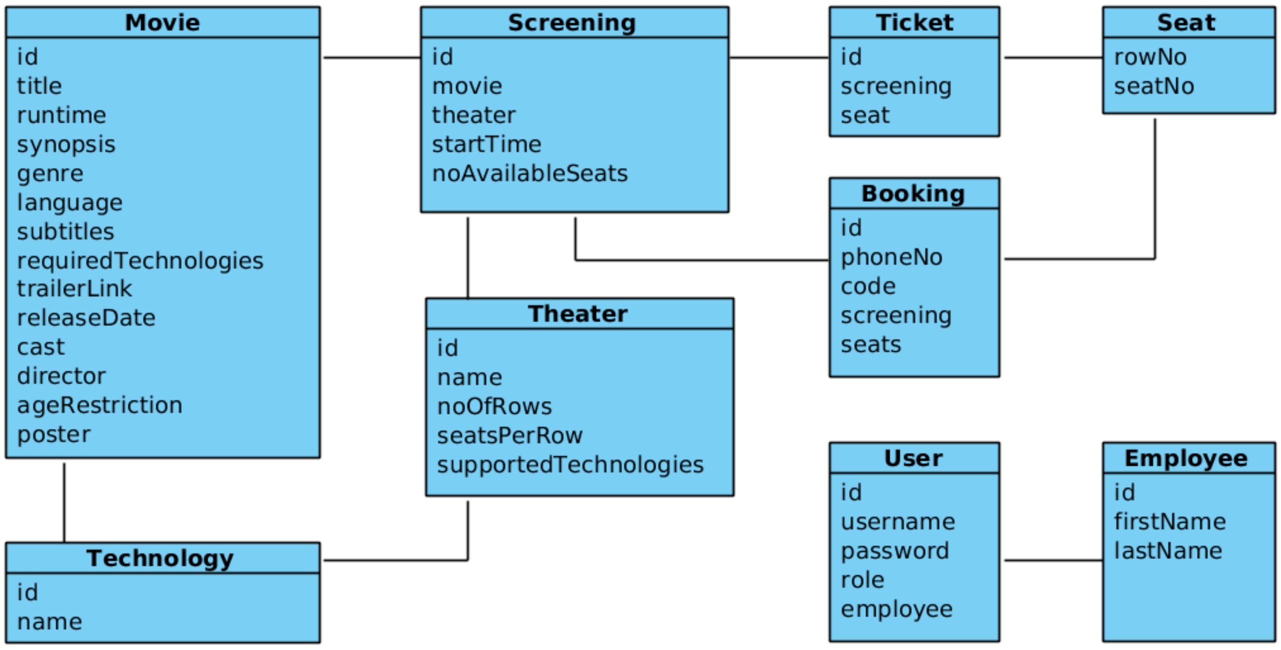
(To be implemented if development time constraints allow)

- REQ008: The system shall prevent users from creating screenings that overlap and, as such, are practically invalid
- REQ009: The system shall prevent users from selling more than one ticket per seat
- REQ010: The system shall prevent customers from double booking the same cinema seat
- REQ011: The system shall only allow the scheduling of screenings in theaters that support the technologies required by the movie (3D projection, Dolby Atmos, etc)
- REQ012: The system shall be able to manage a list of upcoming movies

c. Implied requirements: (Team)

- REQ012: The system shall be developed as a Java Spring Boot application
- REQ013: The system shall use a database for data persistence

d. Domain model (Cristina)



e. Stakeholder and user descriptions (Chris)

Stakeholders

Name	Description	Responsibilities
Team Two	The development team	Gather requirements, model use-cases to fulfill them, implement the resulting system, to a high standard, and write the present text to document the process.
BioTrio	The owners of the business	Provide relevant and accurate information about their business needs, and provide feedback when required, to aid the development process

Users	The employees of “BioTrio”	Test the usability and interface of the system, provide feedback for eventual tweaks to the system, to bring it in line with their daily workflow

Users

Name	Description	Responsibilities	Stakeholder
Administrator	Top management employee within “BioTrio”	Manage all of the cinema’s resources	User
Projections Manager	Management within “BioTrio”	Manage the movies, screenings, ticketing, and bookings	User
Employee	Employee of “BioTrio”	Manage ticketing and bookings	User
Customer	A movie goer	None	User

f. User environment (Chris)

The system is designed to run on a web server that can serve Java applications, and run a MySQL or MariaDB database management system for data persistence. The responsibility of setting up and securing the production environment falls on “BioTrio”, but advice on software choices, compatible versions, and eventual settings for a successful deployment will be provided by the development team.

As with most web applications, this system will naturally support multiple concurrent users by its very nature, but the scope will be limited for the near future, as the number of moviegoers is limited in the provincial town where “BioTrio” is located. This allows the web application to run on fairly limited hardware, with an average, or somewhat above average bandwidth internet connection.

In the case of future expansion, the system can be effortlessly moved to a more powerful server, with higher bandwidth.

g. Use case list (Team)

After having analyzed the list of requirements, we have identified the following use cases, that represent goals the users of the system will want to achieve:

- UC01 – View movie list
- UC02 – View movie details
- UC03 – Add new movie
- UC04 – Update movie
- UC05 – Delete movie
- UC06 – View upcoming movies list
- UC07 – Add movie to upcoming movies list
- UC08 – Remove movie from upcoming movies list
- UC09 – View theater list
- UC10 – Add new theater
- UC11 – Update theater
- UC12 – Delete theater
- UC13 – View upcoming screenings
- UC14 – View screening list
- UC15 – Add new screening
- UC16 – Update screening
- UC17 – Delete screening
- UC18 – View employee list
- UC19 – Add new employee
- UC20 – Update employee
- UC21 – Delete employee
- UC22 – View user list
- UC23 – Add new user
- UC24 – Update user

- UC25 – Delete user
- UC26 – Sell ticket
- UC27 – Void ticket
- UC28 – Redeem booking
- UC29 – Create booking
- UC30 – Cancel booking
- UC31 – View technologies list
- UC32 – Add new technology
- UC33 – Delete technology
- UC34 – Delete past screenings

5. Product overview

a. Vision (Chris)

We’re building the system to help both the employees carry out their day to day tasks with more ease, and simplify the customers’ interaction with the company, thus adding value to the company both internally and externally.

b. Summary of capabilities (Chris)

The system, being a Java Spring web application, provides round the clock, global access as it runs on a simple HTTP server. It exposes an interface with basic functionality to any moviegoers that access it, while providing all the management and administrative functions for the employees, behind a security layer implemented with the help of Spring Security. By its nature, the system is both responsive and stable, and the interface we designed is aimed to make it user friendly as well.

The next section (6. Product Features) will describe all the features that the system offers. These will include a description of the functionality, the actors, and any other related aspects, such as any assumptions and / or dependencies involved.

The features were implemented with some basic principles in mind:

- Single responsibility principle
- Don’t repeat yourself
- Extensibility
- Flexibility
- Keep it simple

c. Assumptions and dependencies (Chris)

The system assumes a fair level of technological literacy both from the point of view of the customer and the employees. It is assumed that they are capable of operating a modern browser to navigate to a web page, and use the stereotypical input controls of a web form to send data to the application. It is also assumed that a significant portion of the customers own some form of a personal computer or a modern mobile device that allows them to connect to the web application.

The system’s operation depends on a web server with a high up-time, than is correctly configured to serve Java Spring applications, a MySQL or MariaDB database for data persistence, and a decent bandwidth to keep the application responsive for the customers that connect remotely. The server can be a physical server on the premises of BioTrio, a rented server in some other location, or even a modern cloud solution.

d. Security aspects (Chris)

As previously mentioned, the web application uses the Java Spring Security libraries to provide authentication and authorization for the sections of the website that are meant for internal use only. Because the responsibilities of the users will not change unless the company adds new facilities, or does major alterations to the existing ones, the user roles have been hard-coded into the system as outlined in section 4. e. (User descriptions). If new functionality is required, modifying the source code is unavoidable, and thus new user roles, if any are warranted for, can be added at the same time.

e. Packaging (Chris)

The system's source code is grouped into several packages to help with managing the files. The structure is as follows:

- **dk.kea.stud.biotrio –** This is the base package. It only contains two classes that Spring requires to execute the application (BiotrioApplication which is the main entry point, and ServletInitializer), and two of our classes: AppGlobals, and MiscController. The former defines some system-wide variables and a couple of static methods, whereas the latter defines two routes in the web application that didn't really logically fit with any of the other controllers.
- **dk.kea.stud.biotrio.administration –** Holds the Employee class, its repository class that handles the CRUD operations of employee data to the database, and the controller that defines the routes for employee data management throughout the application.

  See Appendix_A_Administration.png
- **dk.kea.stud.biotrio.cinema –** This package contains the Java classes that are related to the cinema's management, namely the Movie, Screening, Technology, Theater classes, their respective repository classes, and their related controller classes that handle the routing for their management. MovieForm is an extra class that we have used for a more seamless data transfer between the add and edit movie views and the controller, to avoid certain data binding errors.

  See Appendix_B_Cinema.png
- **dk.kea.stud.biotrio.security –** Holds the user class, which represents a user account that employees use to authenticate within the system, its repository, and the controller class that defines the routes for user management. In addition it also contains the two files needed to configure Spring Security, SecurityConfig.java, that defines the authentication method to be

used, and sets up the privileges needed to access the different routes throughout the system, and CustomAuthentication.java that implements the method by which to authenticate a user.

See Appendix_C_Security.png

- **dk.kea.stud.biotrio.ticketing –** Contains the classes related to the customer aspects of the business, specifically the Booking, Seat, and Ticket classes, their respective repository classes, and finally the controller classes that define the routes related to them. This package also includes a class called SeatData that is used in a few places in the application to load data into, and extract from, the views where the user has to selects cinema seats.

  See Appendix_D_Ticketing.png

We have also organized the other application resource files into folders for easier management. In the static folder one can find the following subfolders: css (contains multicarousel.css, and style.css that are used for styling the application), img (that holds the graphic icons that are used on the management dashboard page), and js (which contains multicarousel.js, that is responsible for the behaviour of the scrollable movies lists on the index view)

The Thymeleaf .html templates can be found in subfolders inside the templates folder. These are organized in a similar fashion to the packages, but not exactly, as their number was fairly high and we felt the need to better separate them, compared to the Java classes.

f. Cost and pricing (Chris)

The main cost is the development of the system itself, and eventual maintenance fees that might occur in the future, as described in the contract. Otherwise the system is free for use both internally within BioTrio, but also externally for the customers. The system's return on investment will stem from the added value to the company's daily operations, and the easing of customer interactions.

g. Licensing and installation (Chris)

The system is released to BioTrio under a proprietary license, as agreed upon in the contract. Installation will be performed by the development team, but the configuration of the running environment must be performed by the system administrator, who is expected to coordinate with the development team in regards to setting it up.

6. Product Features

a. Use case descriptions (Team)

UC01 – View movie list

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: None

Main success scenario: The user navigates to the movie list view and the system shows the complete list of movies.

Extensions: None

UC02 – View movie details

Scope: The system

Level: User-goal

Primary actors: Customer, Employee, Projections Manager, and Administrator

Preconditions: None

Success guarantee: None

Main success scenario: The user selects the movie’s title or poster and the system displays all the available information about that particular movie.

Extensions: None

UC03 – Add new movie

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: The input movie data is stored in the database.

Main success scenario: The user navigates to the “Add new movie” view, where the system displays the add movie form. They type in all the required information and submit it, and then the system saves the movie in the database and redirects them to the movies list view.

Extensions: None

Technology and data variations list:

- All data is input from the keyboard.
- Runtime can be any positive integer that represents the movie’s length in minutes, or 0 if the movie’s length is unknown.
- Trailer link is the YouTube string id of a clip. For example, a full YouTube URL looks like this: https://www.youtube.com/watch?v=rGbe5qy5274 . The video’s id is the string value passed to the “v” parameter, namely “rGbe5qy5274” in this case.
- Poster is the full URL to an image, that represents the movie’s associated poster, for example: https://i.jeded.com/i/monty-pythons-life-of-brian-life-of-brian.13318.jpg.

UC04 – Update movie

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: The data of an existing movie record is updated in the database.

Main success scenario: The user selects “Edit” from the movies list view. The system shows all the movie’s attributes the user can modify. The user changes the data accordingly and submits the form. The system updates the existing movie in the database and redirects the user to the movies list view.

Extensions: None

Technology and data variations list:

- All data is input from the keyboard.

- Runtime can be any positive integer that represents the movie's length in minutes, or 0 if the movie's length is unknown.
- Trailer link is the YouTube string id of a clip. For example, a full YouTube URL looks like this: https://www.youtube.com/watch?v=dQw4w9WgXcQ . The video's id is the string value passed to the "v" parameter, namely "dQw4w9WgXc" in this case.
- Poster is the full URL to an image, that represents the movie's associated poster, for example: https://cdn.sinemia.com/posters/1486456396_5899864c351c2.jpg.

UC05 – Delete movie

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: The desired movie record is deleted from the database.

Main success scenario: The user selects "Delete" from the movies list view and confirms the action. The system deletes the movie from the database.

Extensions: If the movie has any associated screenings, the system displays an error and the movie is not deleted.

UC06 – View upcoming movies list

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: None

Main success scenario: The user navigates to the upcoming movies list view, the system shows the complete list of upcoming movies.

Extensions: None

UC07 – Add movie to the upcoming movies list

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: The selected movie is saved in the upcoming movies list in the database.

Main success scenario: The user selects "Add movie to list" from the upcoming movies list view, and the system displays a list of all the movies that aren't yet in the upcoming movies list. The user selects the desired movie, inputs an estimated screening date, and submits the form. The system saves the entry in the database and redirects the user to the upcoming movies list view.

Extensions: None

Technology and data variations list:

- All data is input from the keyboard.
- The estimated screening date should be of the format YYYY-MM-DD.

UC08 – Remove movie from the upcoming movies list

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: The desired movie is removed from the upcoming movies list in the database.

Main success scenario: The user selects "Delete" from the upcoming movies list view and confirms the action. The system deletes the upcoming movie entry from the database.

Extensions: None

UC09 – View theaters list

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: None

Main success scenario: The user navigates to the theater list view and the system shows the complete list of theaters.

Extensions: None

UC10 – Add new theater

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: A new theater entry is saved in the database.

Main success scenario: The user navigates to the “Add new theater” view, and the system displays the form. The user types all the required information and submits the form. The system saves the theater in the database and redirects the user to the theater list view.

Extensions: None

Technology and data variations list: All data is input from the keyboard.

UC11 – Update theater

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions:

- The user is authenticated.
- The theater must have no screenings associated.

Success guarantee: An existing theater record’s data is updated in the database.

Main success scenario: The user selects “Edit” from the theater list view, and the system displays all the theater’s attributes that the user can modify. The user changes the data as desired and submits the form. The system updates the existing theater in the database and redirects the user to the theater list view.

Extensions: None

Technology and data variations list: All data is input from the keyboard.

UC12 – Delete theater

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: The desired theater record is deleted from the database.

Main success scenario: The user selects “Delete” from the theater list view and confirms the action. The system deletes the theater record from the database.

Extensions: If the theater has one or more screenings scheduled, the system displays an error message and does not delete the theater record from the database.

UC13 – View upcoming screenings

Scope: The system

Level: User-goal

Primary actors: Employee, Projections Manager, Administrator and Customer

Preconditions: None

Success guarantee: None

Main success scenario: The user navigates to the upcoming screenings list view and the system shows the complete list of upcoming screenings, grouped by date.

Extensions: None

UC14 – View screenings list

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: None

Main success scenario: The user navigates to the screening management list view and the system shows the complete list of screenings.

Extensions: None

UC15 – Add new screening

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: A new screening record is saved in the database.

Main success scenario: The user navigates to the "Add new screening" view, and the system displays the associated form. The user then inputs the required data and submits it. The system saves the new record in the database and redirects the user to the screening management list view.

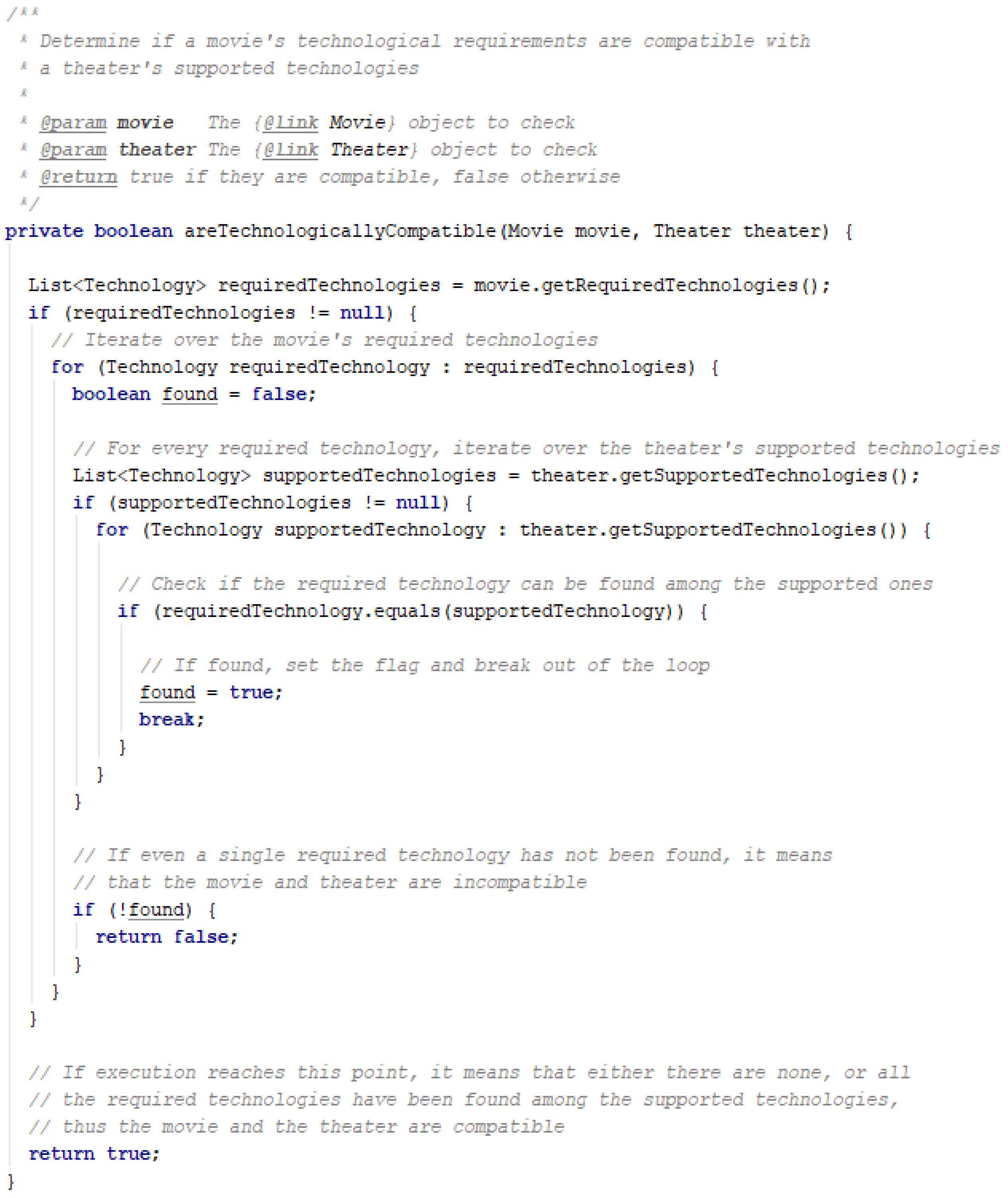
Extensions:

- If the selected movie's required technologies are not the same or a subset of the selected theater's supported technologies, the system displays an error and does not save the record in the database.
- If the screening would conflict with another scheduled screening, meaning that they are both taking place in the same theater, and their start times are too close to one another to be able to play both movies until the end, also accounting for a time buffer for people to leave and the employees to clean the theater, the system displays an error and does not save the record in the database.

Technology and data variations list:

- The movie and theater are selected from respective lists
- All other data is input from the keyboard.
- The screening's start time should be of the format YYYY-MM-DD HH:MM

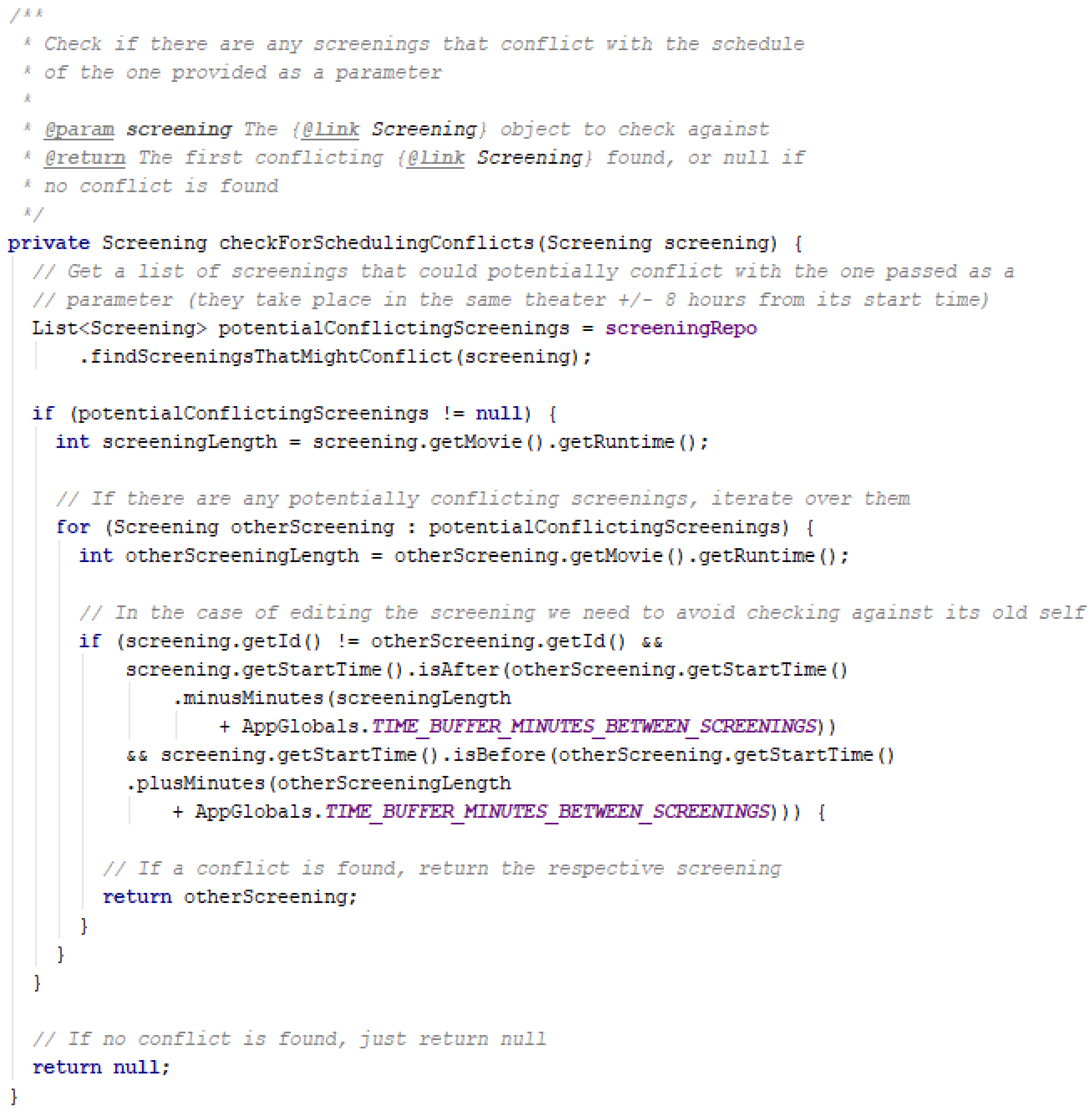
Other details: A detail worthwhile mentioning is that, before we save a screening to the database (and this same pattern applies to "UC16 – Update screening" as well), we have to run two checks. First we make sure that the chosen movie’s technological requirements can be fulfilled by the chosen theater. This happens in the following block of code:



```java
/**
 * Determine if a movie's technological requirements are compatible with
 * a theater's supported technologies
 *
 * @param movie   The {@link Movie} object to check
 * @param theater The {@link Theater} object to check
 * @return true if they are compatible, false otherwise
 */
private boolean areTechnologicallyCompatible(Movie movie, Theater theater) {

  List<Technology> requiredTechnologies = movie.getRequiredTechnologies();
  if (requiredTechnologies != null) {
    // Iterate over the movie's required technologies
    for (Technology requiredTechnology : requiredTechnologies) {
      boolean found = false;

      // For every required technology, iterate over the theater's supported technologies
      List<Technology> supportedTechnologies = theater.getSupportedTechnologies();
      if (supportedTechnologies != null) {
        for (Technology supportedTechnology : theater.getSupportedTechnologies()) {

          // Check if the required technology can be found among the supported ones
          if (requiredTechnology.equals(supportedTechnology)) {

            // If found, set the flag and break out of the loop
            found = true;
            break;
          }
        }
      }

      // If even a single required technology has not been found, it means
      // that the movie and theater are incompatible
      if (!found) {
        return false;
      }
    }
  }

  // If execution reaches this point, it means that either there are none, or all
  // the required technologies have been found among the supported technologies,
  // thus the movie and the theater are compatible
  return true;
}
```

ScreeningController.java

As this is the less expensive operation (because there are no repository calls within it), it runs first. Secondly, we check whether the screening in cause has any scheduling conflicts with any of the other existing screenings with the following method:



```java
/**
 * Check if there are any screenings that conflict with the schedule
 * of the one provided as a parameter
 *
 * @param screening The {@link Screening} object to check against
 * @return The first conflicting {@link Screening} found, or null if
 * no conflict is found
 */
private Screening checkForSchedulingConflicts(Screening screening) {
  // Get a list of screenings that could potentially conflict with the one passed as a
  // parameter (they take place in the same theater +/- 8 hours from its start time)
  List<Screening> potentialConflictingScreenings = screeningRepo
      .findScreeningsThatMightConflict(screening);

  if (potentialConflictingScreenings != null) {
    int screeningLength = screening.getMovie().getRuntime();

    // If there are any potentially conflicting screenings, iterate over them
    for (Screening otherScreening : potentialConflictingScreenings) {
      int otherScreeningLength = otherScreening.getMovie().getRuntime();

      // In the case of editing the screening we need to avoid checking against its old self
      if (screening.getId() != otherScreening.getId() &&
          screening.getStartTime().isAfter(otherScreening.getStartTime()
              .minusMinutes(screeningLength
                  + AppGlobals.TIME_BUFFER_MINUTES_BETWEEN_SCREENINGS))
          && screening.getStartTime().isBefore(otherScreening.getStartTime()
          .plusMinutes(otherScreeningLength
              + AppGlobals.TIME_BUFFER_MINUTES_BETWEEN_SCREENINGS))) {

        // If a conflict is found, return the respective screening
        return otherScreening;
      }
    }
  }

  // If no conflict is found, just return null
  return null;
}
```

ScreeningController.java

UC16 – Update screening

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: The data of an existing screening record is updated in the database.

Main success scenario: The user selects “Edit” from the screening management list view. The system shows all the screening’s attributes the user can modify. The user changes the data as desired and submits the form. The system updates the existing screening record in the database and redirects the user to the screening management list view.

Extensions:

- If the selected movie’s required technologies are not the same or a subset of the selected theater’s supported technologies, the system displays an error and does not save the record in the database.
- If the screening’s modified start time would conflict with another scheduled screening, meaning that they are both taking place in the same theater, and their start times are too close to one another to be able to play both movies until the end, also accounting for a time buffer for people to leave and the employees to clean the theater, the system displays an error and does not update the record in the database.

Technology and data variations list:

- The movie and theater are selected from respective lists
- All other data is input from the keyboard.
- The screening’s start time should be of the format YYYY-MM-DD HH:MM

Sequence Diagram: See Appendix_E_UC16_SD.png

UC17 – Delete screening

Scope: The system

Level: User-goal

Primary actors: Projections Manager and Administrator

Preconditions: The user is authenticated.

Success guarantee: The desired screening is deleted from the database.

Main success scenario: The user selects “Delete” from the screening management list view and confirms the action. The system deletes the theater from the database.

Extensions: If the screening has any associated tickets or bookings, the system displays an error message and does not delete the record from the database.

UC18 – View employee list

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: None

Main success scenario: The user navigates to the employee list view and the system displays the complete list of employees.

Extensions: None

UC19 – Add new employee

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: A new employee record is saved in the database.

Main success scenario: The user navigates to the "Add new employee" view, and the system shows the associated form. The user types all the required information and submits the it. The system saves the employee in the database and redirects the user to the employee list view.

Extensions: None

Technology and data variations list:

- The data is input from the keyboard.

UC20 – Update employee

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: The data of an existing employee record is updated in the database.

Main success scenario: The user selects “Edit” from the employee list view, and the system shows all the employee’s attributes the user can modify. The user changes the data as desired and submits the form. The system updates the existing employee record in the database and redirects the user to the employee list view.

Extensions: None

Technology and data variations list:

- The data is input from the keyboard.

UC21 – Delete employee

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: The desired employee record is deleted from the database.

Main success scenario: The user navigates to the employee list view and the system displays the complete list of employees.

Extensions: None

UC22 – View user list

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: None

Main success scenario: The user navigates to the user list view and the system displays the complete list of user accounts.

Extensions: None

UC23 – Add new user

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: A new user account record is saved in the database.

Main success scenario: The user navigates to the "Add new user" view. The system shows all the data labels the user needs to fill. The user types all the required information and submits the form. The system saves the user in the database and redirects the user to the user list view.

Extensions:

- If the input username is already taken, the system displays the "Add new user" view again with an error message stating this.
- If the input password and repeat password don't match, the system displays the "Add new user" view again with an error message stating this.

Technology and data variations list:

- The data is input from the keyboard.
- An associated employee record can be selected from a list.

UC24 – Update user

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: The data of a user account record is updated in the database.

Main success scenario: The user navigates to the "Add new user" view. The system shows all the data labels the user needs to fill. The user types all the required information and submits the form. The system saves the user in the database and redirects the user to the user list view.

Extensions:

- If the username has been changed, and the desired username is already taken, the system displays the “Update user” view again with an error message stating this.
- If the old password is wrong, and / or the new password and repeat new password don’t match, the system displays the “Update user” view again with an error message stating this.

Technology and data variations list:

- The data is input from the keyboard.
- An associated employee record can be selected from a list.

UC25 – Delete user

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: The desired user account record is deleted from the database.

Main success scenario: The user selects “Delete” from the user list view and confirms the action. The system deletes the user account record from the database and redirects the user to the employee list view.

Extensions: None

UC26 – Sell ticket

Scope: The system

Level: User-goal

Primary actors: Employee, Projections Manager, and Administrator

Stakeholders and Interests:

- Customer: wants to buy tickets for a specific screening.
- BioTrio: wants to sell tickets more efficiently by using an automated system.

Preconditions: The user is authenticated.

Success guarantee: The tickets are successfully printed and saved in the database.

Main success scenario:

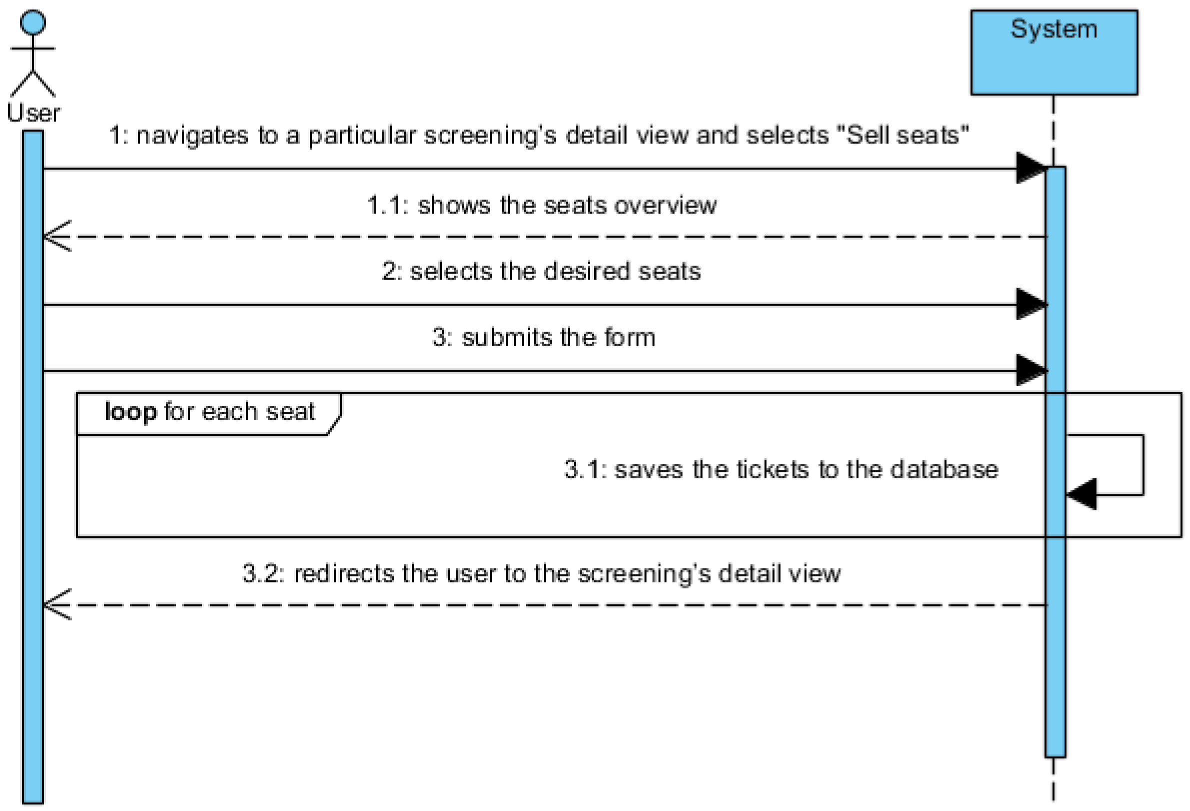
1. The user navigates to a particular screening’s detail view from the manage screenings list view
2. User selects “Sell seats”
3. System shows the seats overview
4. User selects the desired seats
5. User submits the form
6. System saves the ticket(s) to the database
7. System redirects the user to the screening’s detail view

Extensions: None

Technology and data variations list: Seats are selected with a pointing device (mouse, touchscreen, etc)

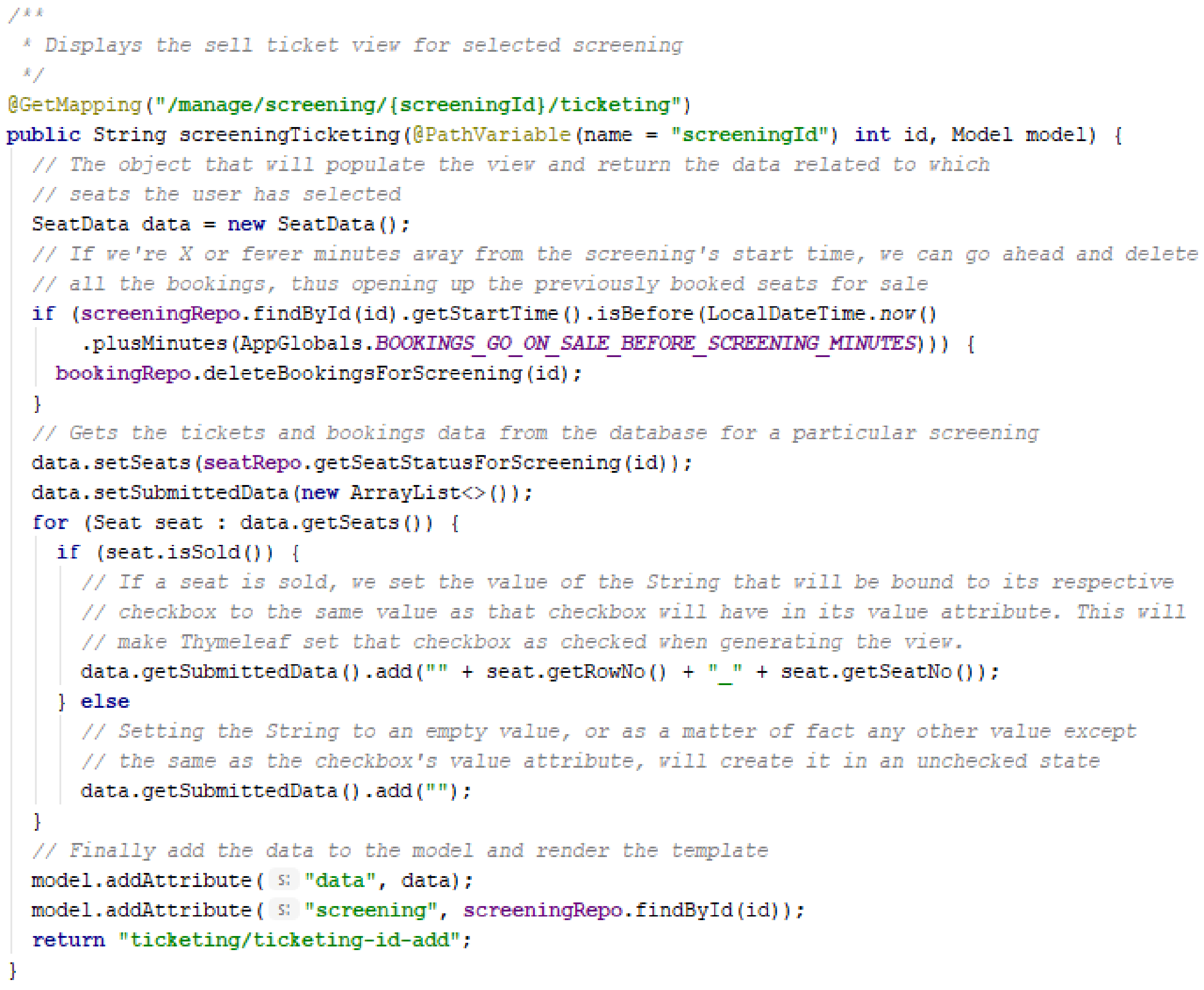
Frequency of occurrence: Very frequent

System sequence diagram:


User
System
1: navigates to a particular screening's detail view and selects "Sell seats"
1.1: shows the seats overview
2: selects the desired seats
3: submits the form
loop for each seat
3.1: saves the tickets to the database
3.2: redirects the user to the screening's detail view

Other details: While building this part of the project (this pattern also repeats in “UC27 – Void ticket” and “UC29 – Create booking”), we ran into some quirks of the Spring framework, in relation to programatically creating a checkbox for each theater seat, and binding them to a data structure so as to successfully receive the data of the seats that the user has selected. The way we managed to implement this was to send an array of our custom `Seat` objects containing the data needed to generate the view, along with an `ArrayList` of `String` which is bound to the checkbox `<input>` tags by the `th:field` attribute. When the user submits the form, Spring populates this list with `Strings` of only the checked checkboxes' value attribute, which we programatically set with the help of Thyemleaf: `th:value="${seat.getRowNo()} + '_' + ${seat.getSeatNo()}"`.

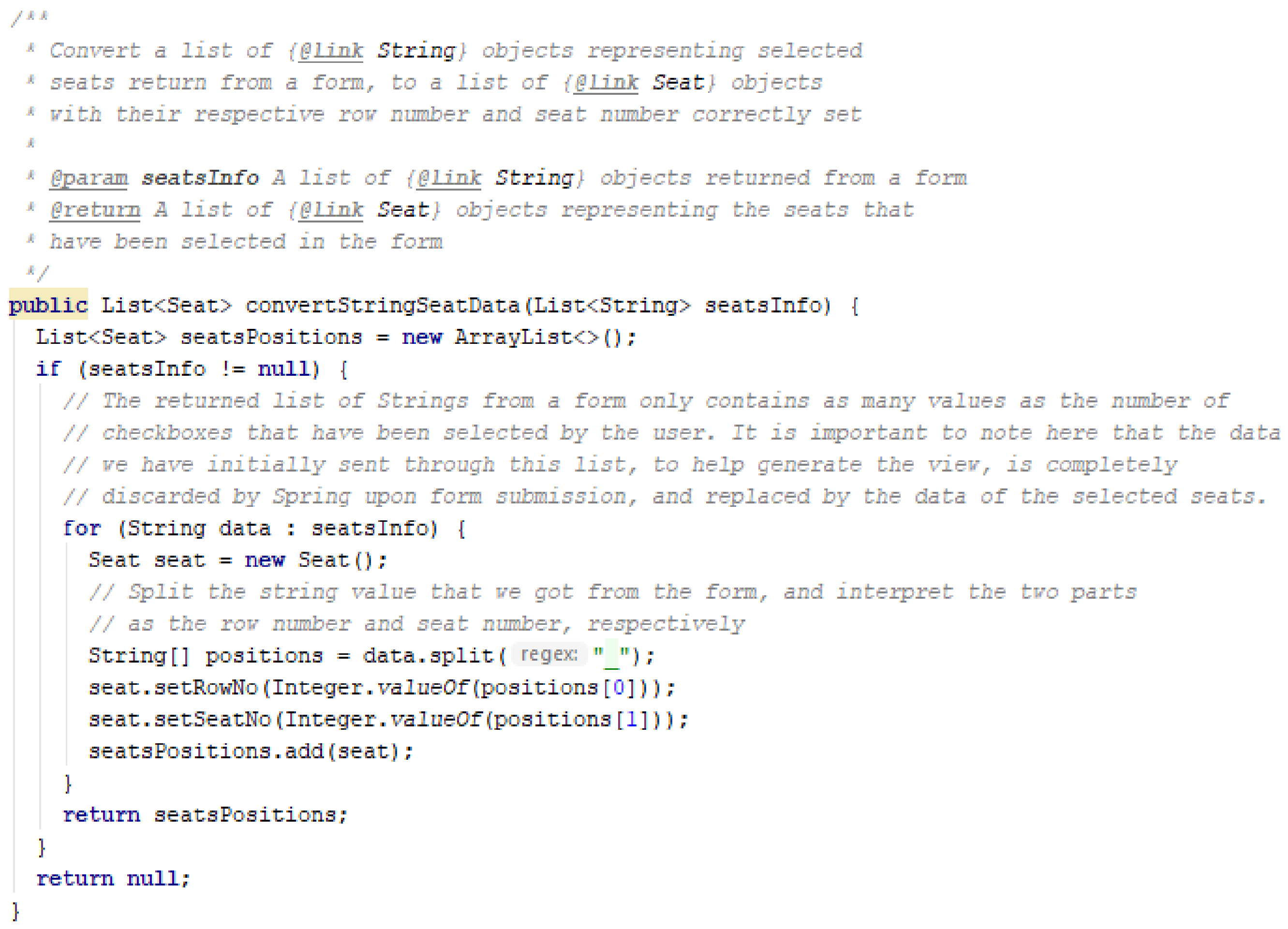
We send the data to the view with the following code:



```java
/**
 * Displays the sell ticket view for selected screening
 */
@GetMapping("/manage/screening/{screeningId}/ticketing")
public String screeningTicketing(@PathVariable(name = "screeningId") int id, Model model) {
    // The object that will populate the view and return the data related to which
    // seats the user has selected
    SeatData data = new SeatData();
    // If we're X or fewer minutes away from the screening's start time, we can go ahead and delete
    // all the bookings, thus opening up the previously booked seats for sale
    if (screeningRepo.findById(id).getStartTime().isBefore(LocalDateTime.now()
            .plusMinutes(AppGlobals.BOOKINGS_GO_ON_SALE_BEFORE_SCREENING_MINUTES))) {
        bookingRepo.deleteBookingsForScreening(id);
    }
    // Gets the tickets and bookings data from the database for a particular screening
    data.setSeats(seatRepo.getSeatStatusForScreening(id));
    data.setSubmittedData(new ArrayList<>());
    for (Seat seat : data.getSeats()) {
        if (seat.isSold()) {
            // If a seat is sold, we set the value of the String that will be bound to its respective
            // checkbox to the same value as that checkbox will have in its value attribute. This will
            // make Thymeleaf set that checkbox as checked when generating the view.
            data.getSubmittedData().add("" + seat.getRowNo() + "_" + seat.getSeatNo());
        } else
            // Setting the String to an empty value, or as a matter of fact any other value except
            // the same as the checkbox's value attribute, will create it in an unchecked state
            data.getSubmittedData().add("");
    }
    // Finally add the data to the model and render the template
    model.addAttribute( s: "data", data);
    model.addAttribute( s: "screening", screeningRepo.findById(id));
    return "ticketing/ticketing-id-add";
}
```

TicketController.java

Then, in the post mapping that receives the form data, we call the following method to interpret it and convert the `List` of `String` into a `List` of `Seat` to be used in creating the `Ticket` objects, in this case:



```java
/**
 * Convert a list of {@link String} objects representing selected
 * seats return from a form, to a list of {@link Seat} objects
 * with their respective row number and seat number correctly set
 *
 * @param seatsInfo A list of {@link String} objects returned from a form
 * @return A list of {@link Seat} objects representing the seats that
 * have been selected in the form
 */
public List<Seat> convertStringSeatData(List<String> seatsInfo) {
  List<Seat> seatsPositions = new ArrayList<>();
  if (seatsInfo != null) {
    // The returned list of Strings from a form only contains as many values as the number of
    // checkboxes that have been selected by the user. It is important to note here that the data
    // we have initially sent through this list, to help generate the view, is completely
    // discarded by Spring upon form submission, and replaced by the data of the selected seats.
    for (String data : seatsInfo) {
      Seat seat = new Seat();
      // Split the string value that we got from the form, and interpret the two parts
      // as the row number and seat number, respectively
      String[] positions = data.split( regex: "_");
      seat.setRowNo(Integer.valueOf(positions[0]));
      seat.setSeatNo(Integer.valueOf(positions[1]));
      seatsPositions.add(seat);
    }
    return seatsPositions;
  }
  return null;
}
```

SeatRepository.java

UC27 – Void ticket

Scope: The system

Level: User-goal

Primary actors: Employee, Projections Manager, and Administrator

Preconditions: The user is authenticated.

Success guarantee: The desired ticket(s) are voided and deleted from the database.

Main success scenario: The user navigates to a particular screening’s detail view from the manage screenings list view and selects “Void tickets”. The system shows the seats overview with only the sold seats selectable. The user selects the seats as desired, submits the form and the system deletes the associated tickets from the database and redirects the user to the screening’s detail view.

Extensions: None

UC28 – Redeem booking

Scope: The system

Level: User-goal

Primary actors: Employee, Projections Manager, and Administrator

Stakeholders and Interests:

- Customer: wants to redeem the tickets for a specific screening they booked.
- BioTrio: wants to allow redeeming of online booked seats to increase the efficiency of the employees' workflow.

Preconditions: The user is authenticated.

Success guarantee: The tickets are successfully printed and saved in the database.

Main success scenario:

1. User selects a booking either by phone number search, or from a screening's bookings list
2. System shows the list of booked seats
3. User selects the desired seats for which tickets are to be sold
4. User submits the form
5. System shows a confirmation window stating that any seats not selected will become available for other people to buy.
6. User confirms the action
7. System prints the tickets and saves them to the database
8. System deletes the booking from the database
9. System redirects the user to screening's detail view

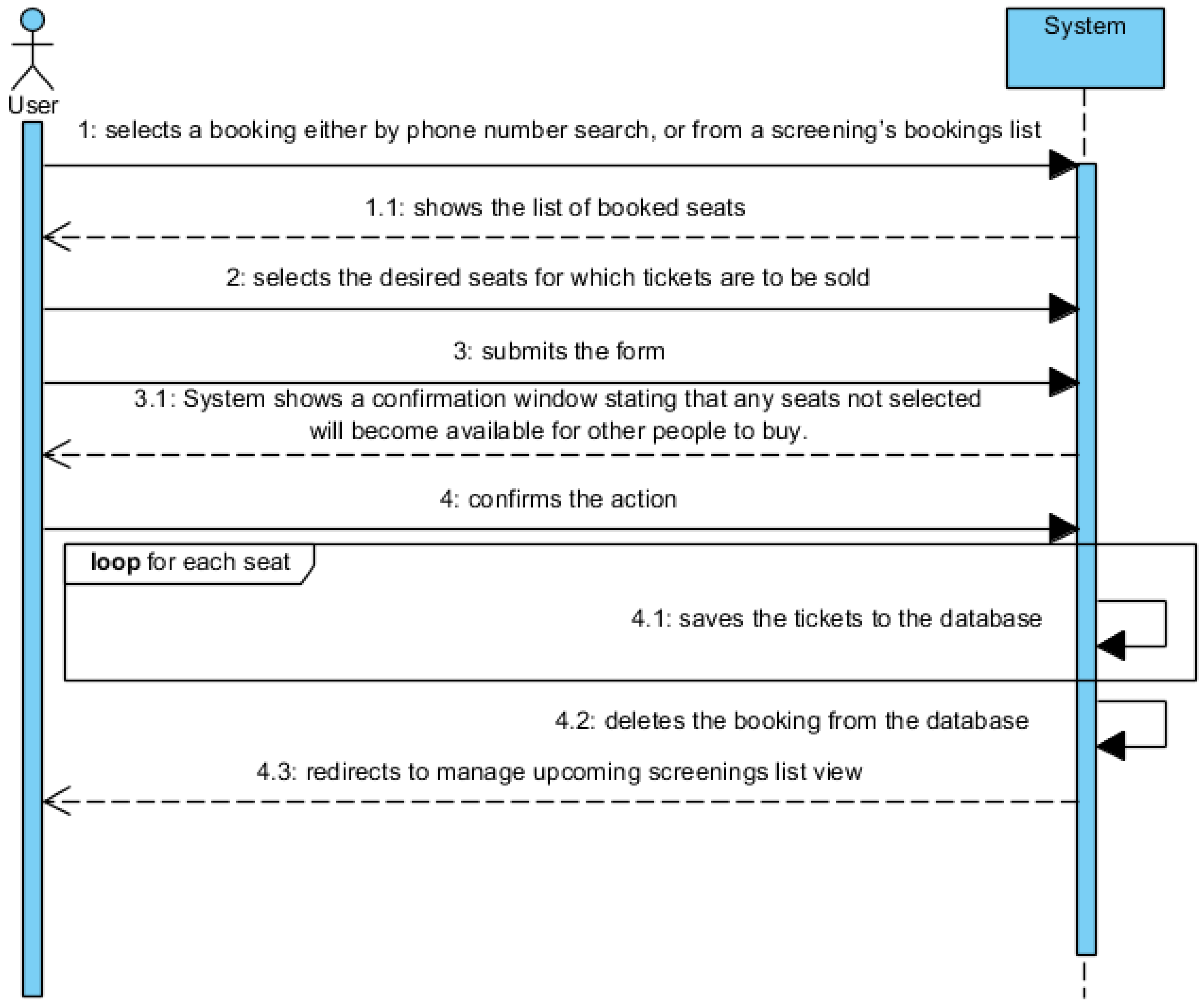
Extensions: None

Technology and data variations list: Seats are selected with a pointing device (mouse, touchscreen, etc)

Frequency of Occurrence: Very frequent

Sequence diagram: See Appendix_F_UC28_SD.png (describes how the system finds the list of bookings for a given screening id)

System sequence diagram:


User
System
1: selects a booking either by phone number search, or from a screening's bookings list
1.1: shows the list of booked seats
2: selects the desired seats for which tickets are to be sold
3: submits the form
3.1: System shows a confirmation window stating that any seats not selected will become available for other people to buy.
4: confirms the action
loop for each seat
4.1: saves the tickets to the database
4.2: deletes the booking from the database
4.3: redirects to manage upcoming screenings list view

UC29 – Create booking

Scope: The system

Level: User-goal

Primary actors: Customer

Stakeholders and Interests:

- Customer: wants to book seats for a specific screening.
- BioTrio: wants to allow the online booking of seats for more efficiency, thereby reducing the employees' workload.

Preconditions: None

Success guarantee: The booking is saved to the database and an SMS sent to the customer.

Main success scenario:

1. User selects “Book tickets” from the upcoming screenings list view
2. System shows the “Create booking” view
3. User inputs their mobile phone number
4. User selects between one and four seats to be booked
5. User submits the form
6. System saves the booking to the database
7. System shows the booking confirmation view

Extensions:

3a. User fails to input a phone number

1. The system notifies the user that the phone number is required

4a. User does not select any seats

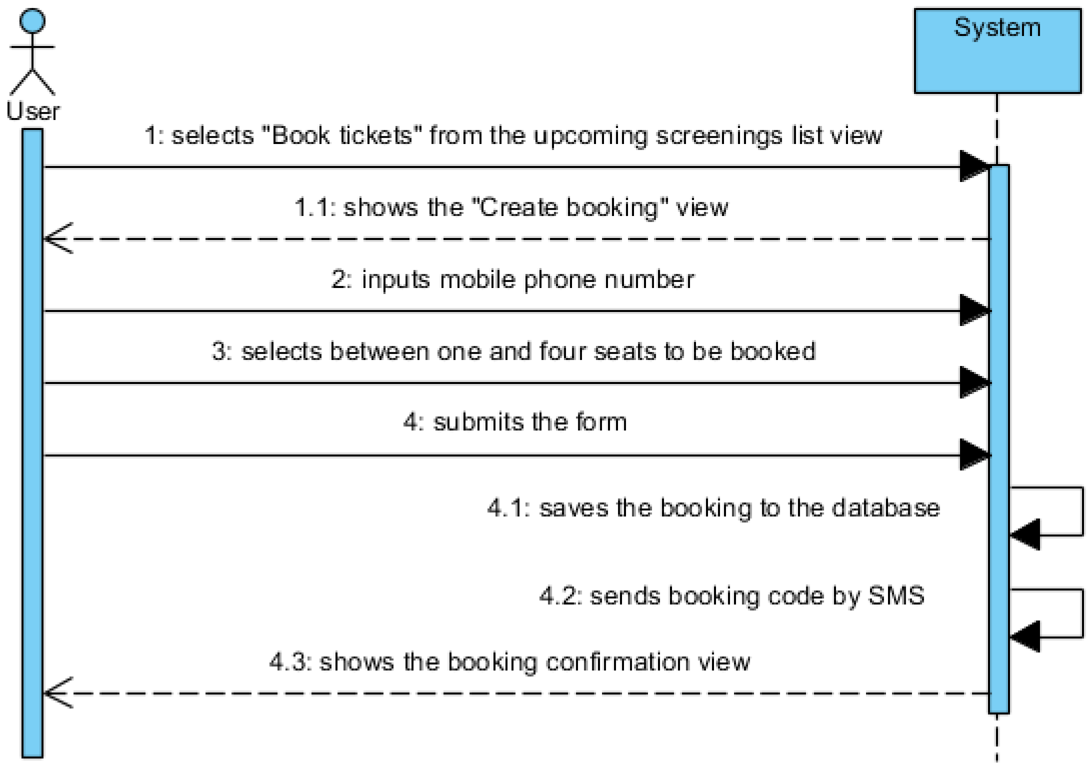
1. The system displays an error screen

Technology and data variations list:

- The data is input from the keyboard
- Seats are selected with a pointing device (mouse, touchscreen, etc)

Frequency of Occurrence: Very frequent

System sequence diagram:


User
System
1: selects "Book tickets" from the upcoming screenings list view
1.1: shows the "Create booking" view
2: inputs mobile phone number
3: selects between one and four seats to be booked
4: submits the form
4.1: saves the booking to the database
4.2: sends booking code by SMS
4.3: shows the booking confirmation view

UC30 – Cancel booking

Scope: The system

Level: User-goal

Primary actors: Customer

Stakeholders and Interests:

- Customer: wants to cancel one of his/her booking for a specific screening.
- BioTrio: wants to allow the online canceling of bookings to lessen the employees' workload.

Preconditions: None

Success guarantee: The booking is deleted from the database.

Main success scenario:

1. User selects "Cancel a booking" from the upcoming screenings list view
2. System shows the "Canceling booking" view
3. User inputs the booking code
4. User submits the form
5. System deletes the booking from the database

6. System redirects the user to booking canceled confirmation view

Extensions:

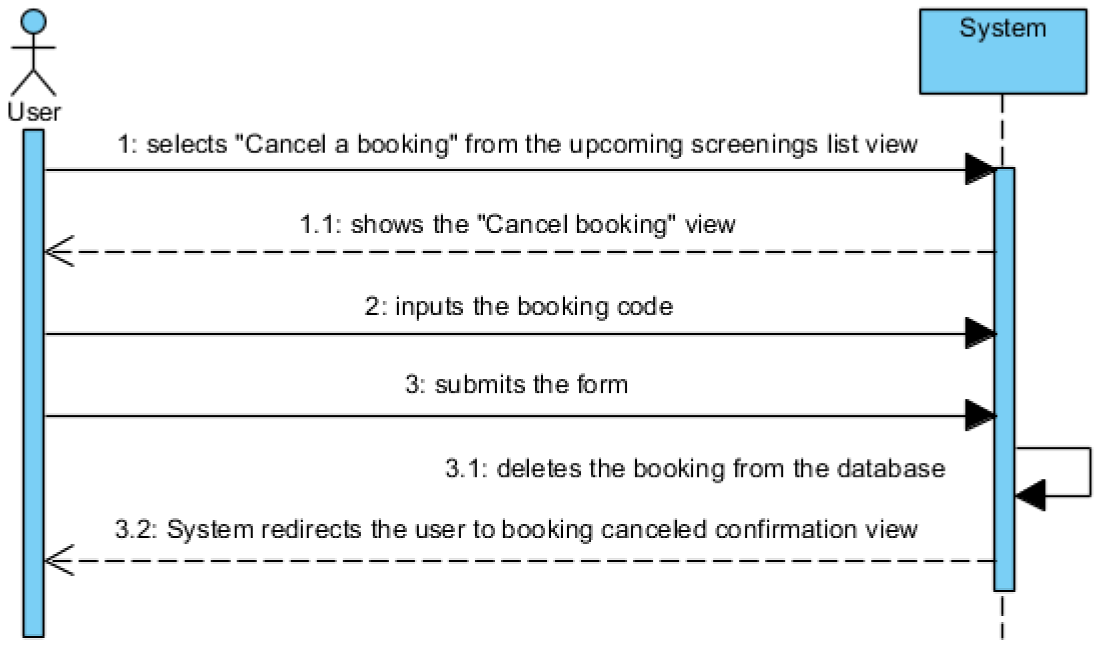
3.a. If the user leaves the field blank or inputs an invalid code

1. The system displays an error message stating that the booking could not be found

Technology and data variations list: The data is input from the keyboard

Frequency of Occurrence: Occasional

System sequence diagram:


User
System
1: selects "Cancel a booking" from the upcoming screenings list view
1.1: shows the "Cancel booking" view
2: inputs the booking code
3: submits the form
3.1: deletes the booking from the database
3.2: System redirects the user to booking canceled confirmation view

UC31 – View technologies list

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: None

Main success scenario: The user navigates to the technology list view and the system shows the complete list of technologies.

Extensions: None

UC32 – Add new technology

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: The new technology record is saved in the database.

Main success scenario: The user navigates to the technology list view, types in the desired name under the “Add new technology” form, and submits it. The system saves this as a new record in the database.

Extensions: If the input name is already taken within the database, the system refreshes the technology list view, displaying and error that states this fact, and no changes are made to the database.

Technology and data variations list: The data is input from the keyboard

UC33 – Delete technology

Scope: The system

Level: User-goal

Primary actors: Administrator

Preconditions: The user is authenticated.

Success guarantee: The desired technology is deleted from the database.

Main success scenario: The user selects “Delete” from the technology list view and confirms the action. The system deletes the technology from the database.

Extensions: None

UC34 – Delete past screenings

Scope: The system

Level: User-goal

Primary actors: Projections manager, and Administrator

Stakeholders and Interests: BioTrio: wants to optimize the storage space taken up by the database.

Preconditions: The user is authenticated.

Success guarantee: The past screenings and all the related data are successfully deleted from the database.

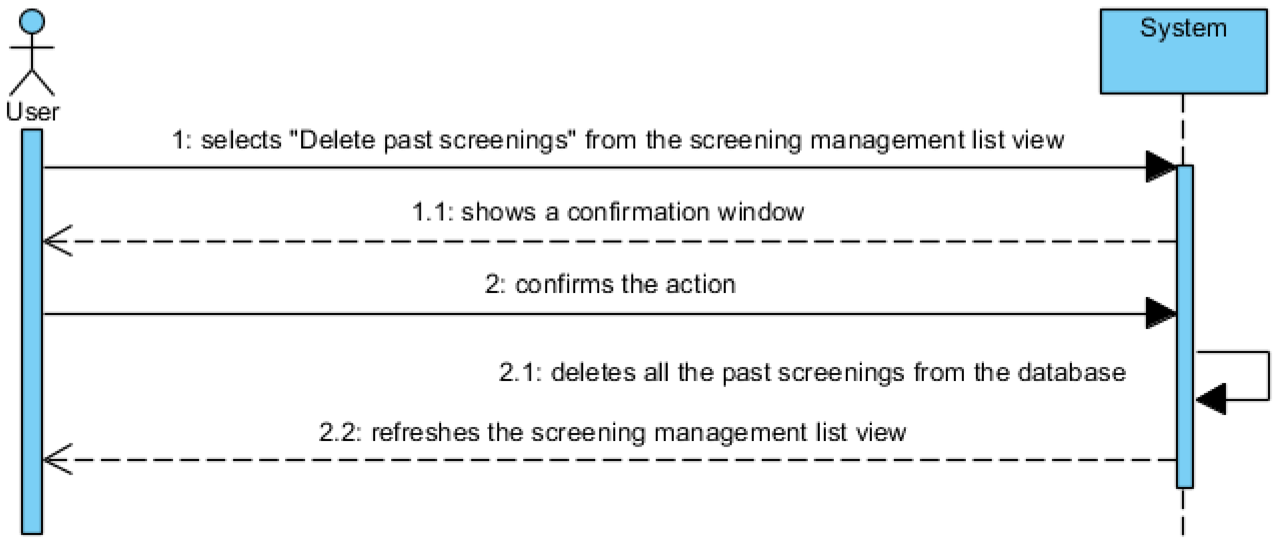
Main success scenario:

1. User selects "Delete past screenings" from manage screening list view
2. System shows a confirmation window
3. User confirms the action
4. System deletes all the past screenings, and any associated bookings and / or tickets from the database.
5. System refreshes the manage screening list view.

Extensions: None

Frequency of Occurrence: Frequent

System sequence diagram:


User
System
1: selects "Delete past screenings" from the screening management list view
1.1: shows a confirmation window
2: confirms the action
2.1: deletes all the past screenings from the database
2.2: refreshes the screening management list view

b. Database structure (Team)

One of the project’s requirements is to use a database management system (DBMS) as a means to achieve data persistence. As such, we have opted to use MySQL (or its closely compatible, and generally interchangeable, open source alternative, MariaDB) which is a relational DBMS (RDBMS). An RDBMS can be defined as a software system used to maintain data organized into one or more tables of columns and rows. Each row (also called record, or tuple) must have a unique key (called a primary key) for the purpose of being able to reference it. The columns (also called attributes) represent the values associated with the rows. What defines a database as relational is the existence of columns that link to other tables’ primary key (called foreign keys), thus creating relations between records in different tables.

After having explored the problem domain, we identified the data that our system needs to keep track of, and the relations between them:

1. Movies
2. Theaters
3. Screenings – Many-to-one relation to movies, and many-to-one relation with theaters
4. Bookings – Many-to-one relation to screenings
5. Booked Seats – Many-to-one relation to bookings
6. Tickets – Many-to-one relation to screenings
7. Upcoming Movies – One-to-one relation to movies
8. Technologies – Many-to-many relation to movies, and many-to-many relation to theaters
9. Employees
10. Roles
11. Users – Many-to-one relation to roles, and one-to-one relation to employees

While one-to-one relations could be simply included in the same table, we decided to separate these data into different tables for the purpose of achieving logical separation of data, improving flexibility, and to ease future extensibility.

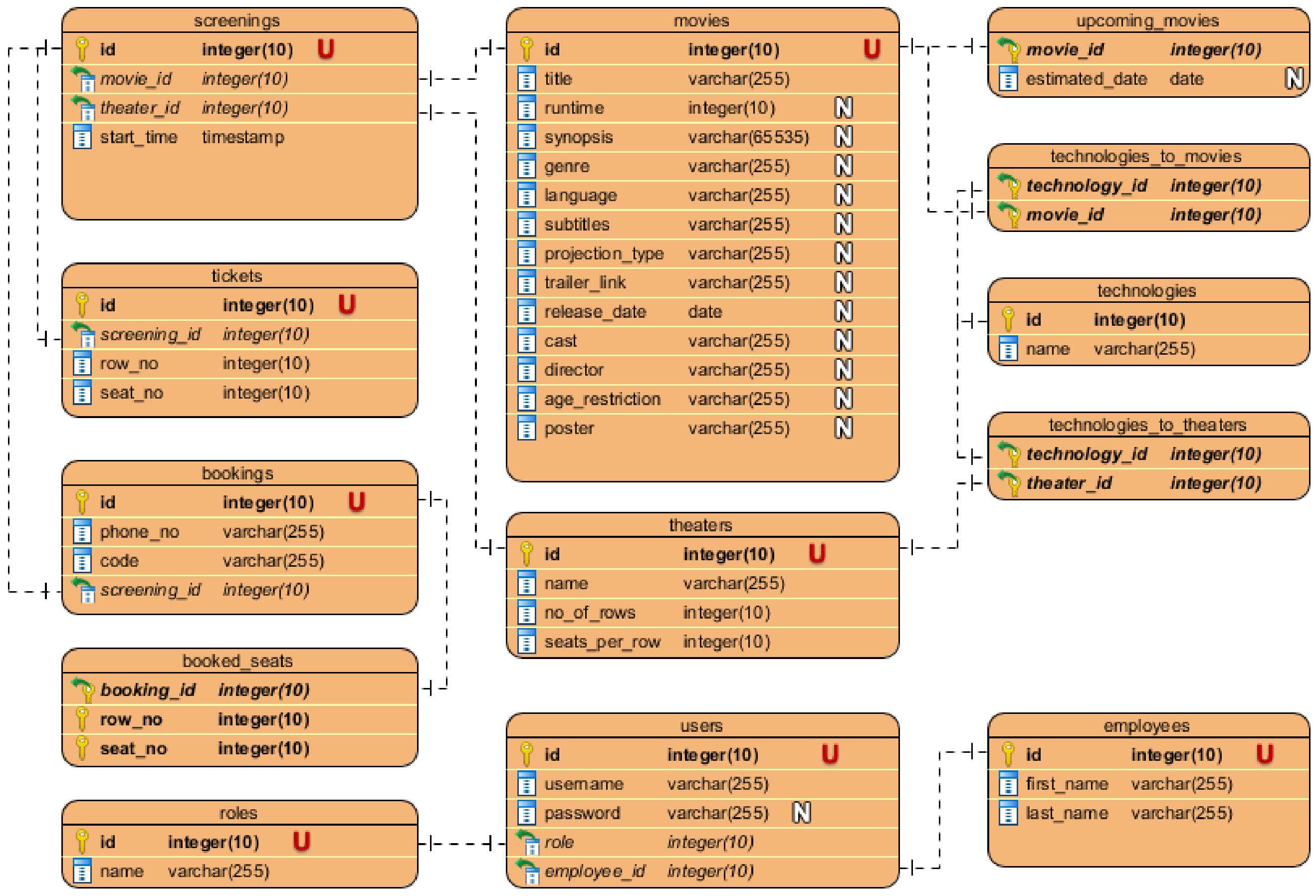
Many-to-one relations are achieved by marking a table’s column as a foreign key, and having it reference the primary key of another table. For example, several entries in the screenings table can have the same value in their movie_id column (which is a foreign key referencing the movies table’s primary key, the id column), thus pointing to the same entry in the movies table, and thereby creating this relation.

Many-to-many relations, on the other hand, cannot be created in such a direct fashion in most RDBMSs, and are modeled by creating an additional intermediary table, which has to the least two columns. Together they are defined as being a primary composite key. One of these two columns is also defined as being a foreign key referencing the first table, while the other column is defined as a foreign key referencing the second table. Each record in this intermediary table creates a link between two entries in the original tables, and this allows linking an arbitrary number of entries in the two tables to one another, in both directions. It’s worth noting that creating the same identical link twice is not allowed, as the combination of these two values must be unique (which is a condition of being a primary composite key). In our tables, we identified two such many-to-many relations, between technologies and movies, and technologies and theaters, respectively. Therefore, we needed to create the two intermediary tables technologies_to_movies and technologies_to_theaters.

After having considered the data relationships that needed to be modeled, and having settled on our table structure, we analyzed the result and realized that it already fulfills the conditions for third normalized form. Therefore no more work needed to be done from the perspective of database normalization.

As for deletion anomalies, we have set “on delete cascade” for most relations. If we do permit the deletion of a certain record (on the Java side there are additional checks), we want to delete the dependent records as well, for example in the case of tickets and bookings. It makes no sense to keep these data around if the associated screening record is deleted. The only notable exception is the relation between user accounts and employee records, where we want to keep them independent, so we have set “on delete set null” for this relation.

Below you can see the entity relation diagram, which gives a better overview of the tables’ structure and the relations between them:



screenings
id integer(10) U
movie_id integer(10)
theater_id integer(10)
start_time timestamp
tickets
id integer(10) U
screening_id integer(10)
row_no integer(10)
seat_no integer(10)
bookings
id integer(10) U
phone_no varchar(255)
code varchar(255)
screening_id integer(10)
booked_seats
booking_id integer(10)
row_no integer(10)
seat_no integer(10)
roles
id integer(10) U
name varchar(255)
movies
id integer(10) U
title varchar(255)
runtime integer(10) N
synopsis varchar(65535) N
genre varchar(255) N
language varchar(255) N
subtitles varchar(255) N
projection_type varchar(255) N
trailer_link varchar(255) N
release_date date N
cast varchar(255) N
director varchar(255) N
age_restriction varchar(255) N
poster varchar(255) N
theaters
id integer(10) U
name varchar(255)
no_of_rows integer(10)
seats_per_row integer(10)
users
id integer(10) U
username varchar(255)
password varchar(255) N
role integer(10)
employee_id integer(10)
upcoming_movies
movie_id integer(10)
estimated_date date N
technologies_to_movies
technology_id integer(10)
movie_id integer(10)
technologies
id integer(10)
name varchar(255)
technologies_to_theaters
technology_id integer(10)
theater_id integer(10)
employees
id integer(10) U
first_name varchar(255)
last_name varchar(255)