



Group Assignment Design Document

[Link to the prototype](#)

GROUP JMSK
Jani Ilkka, jani.ilkka@tuni.fi Markus Siitonen, markus.siitonen@tuni.fi Kian Moloney, kian.moloney@tuni.fi

Document version history	
Document created	14.9.2024 (Jani Ilkka)
Added movie functionality-related documentation and a diagram	19.9.2024 (Jani Ilkka)
Updated Movies-table and the diagram	29.9.2024 (Markus Siitonen)
Numerous updates for mid-term	23.-24.10.2024 (Kian Moloney, Markus Siitonen, Jani Ilkka)
Final document	25.10 – 1.12.2024 (Kian Moloney, Markus Siitonen, Jani Ilkka)

Table of Contents

Introduction.....	3
Application Idea	3
Schedule and Division of Work.....	3

Central Features.....	4
Views	4
User Guide	6
Search for a Movie.....	8
Rating a movie	9
Saving preferences.....	10
Check statistics	11
APIs Utilized	13
Classes, Interfaces, and Their Responsibilities	13
Movies.....	13
Subtitles.....	15
Other & Utilities	16
BONUS: Snacks (TBD)	18
Architecture	18
Architecture Patterns	18
Design Patterns.....	18
SOLID.....	19
Navigation	19
Interfaces	20
Implementation Technologies	20
Views	20
ControlsFX	20
HTTP Client.....	21
GSON.....	21
AI Disclosure	21
Idea.....	21
Code	22
Graphic Design Resources	22
Javadoc	22
Usefulness	22
External Libraries.....	23
Division of Work	23
Responsibilities	23
Self-evaluation	24
Design	25
Technical aspects	25

Known issues	25
References.....	28

Introduction

Application Idea

Our application is called Movie Night Planner.

Movie Night Planner allows users to select a movie to watch based on search criteria. The application then tells which streaming service(s) has/have the movie in their catalog.

BONUS: As a potential addition to the aforementioned functionality, we have an idea about using a third-party API to get recommendations as to which snacks go with the theme of the movie.

NOTE!

During the latter part of this project, the API key usage count logic suddenly changed. Instead of having low daily limits, the service deprecated the original plan and now has only one monthly option. This may affect testing. Please contact us, if you get the HTTP error message 429.

Also, the APIs are occasionally under a lot of strain. Be patient. The application fetches data using several asynchronous HTTP operations. They should all eventually return.

Schedule and Division of Work

Course week	Tasks	Deadline	Responsible
1	Post a group formation message.	27.8.	Jani Ilkka, Markus Siitonen
2	Choose application idea.	10.9.	Whole team
3	Pivot the application idea.	18.9.	Whole team
4	First Prototype Views	20.9.	Markus Siitonen, Sviatoslav Vasev, Kian Moloney
5	JavaFX application creation, TMDB-related Java code, including relevant documentation.	20.9.	Jani Ilkka

6	Additional TMDb java code added. Documentation updated.	25.9.	Jani Ilkka
7	TA Meeting	1.10.	Whole team

Due to the hectic nature of the project implementation, after the first TA-meeting, this table is no longer being updated. You can find the final “Division of Work” table near the end of this document.

Central Features

Movie Night Planner recommends movies to watch.

We have included 10 streaming services, which can be used to stream or buy a movie, or simply to watch it based on a subscription. Movies can be filtered according to individual services.

In addition to streaming services, movie suggestions can be narrowed down according to genres, spoken languages, and/or subtitle language.

Preferences can be used to store the aforementioned filters.

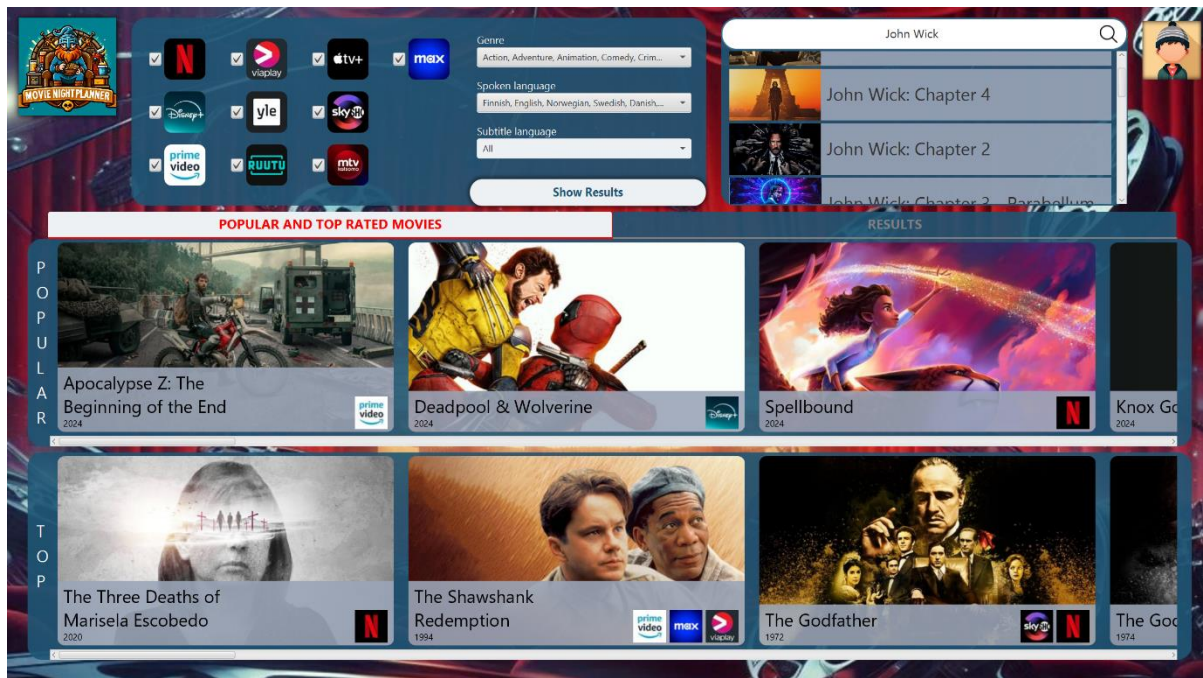
Graphs and diagrams provide the user with information about, e.g., favourite genres and movie watch history.

Views

Movie Night Planner has three main views: Search view, Movie details view, and Profile view.

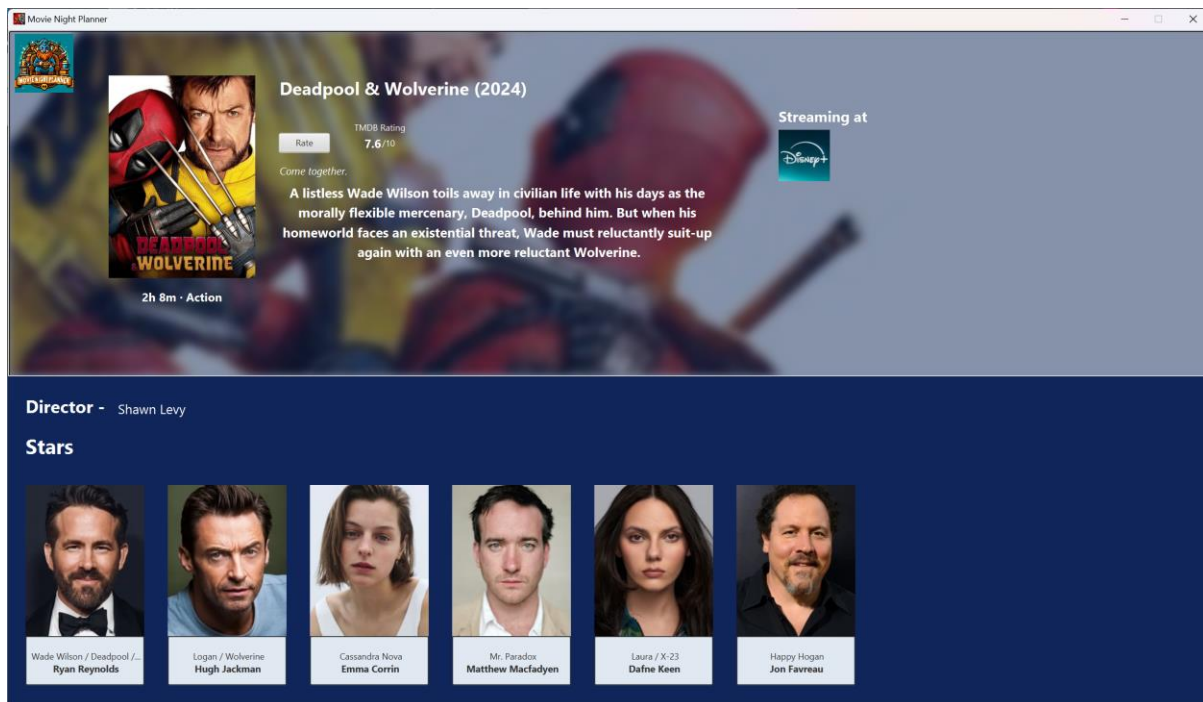
Search View

The search view is the default starting view. The search view has three main sections. The one on the top left is the filter area. On the top right you can see a search bar. The main area is taken by a tab view. The first tab displays two horizontally scrollable lists, containing the currently most popular and top-rated movies. The other tab shows the search results based on the selection in the filter area



Movie Detail View

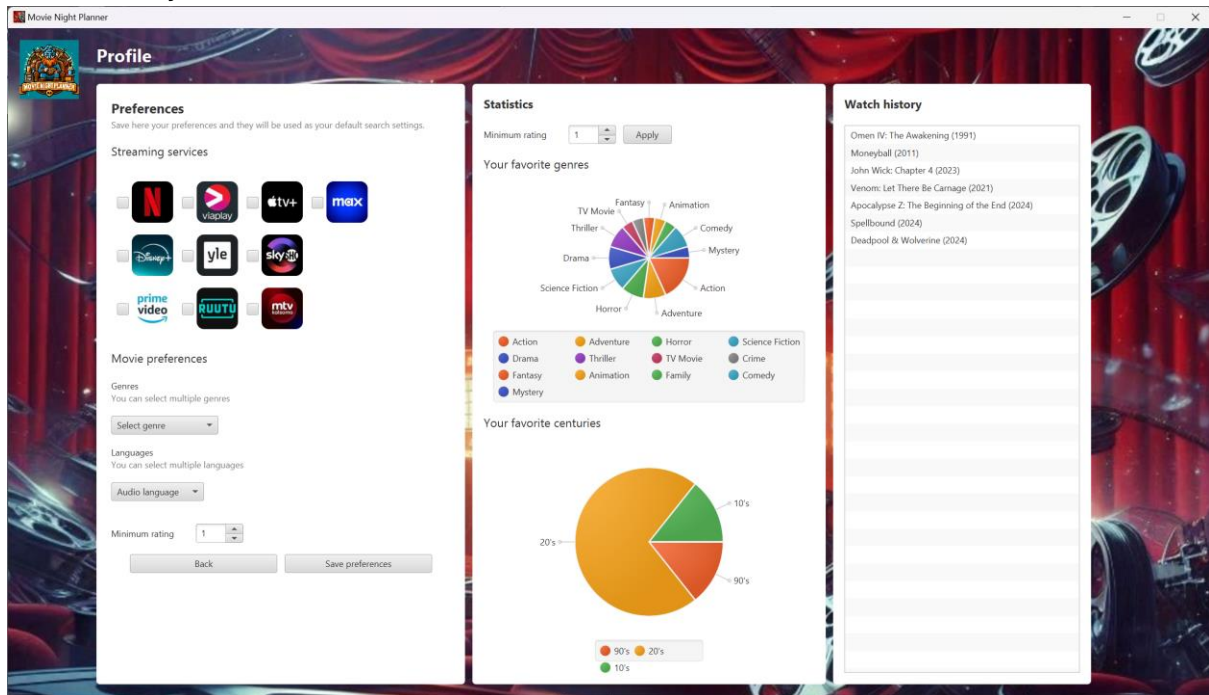
The movie detail view shows, e.g., a short description text and a few of the most important cast members.



Profile view

The profile view contains three subsections. Starting from the right, the Watch history lists all movies that have been marked as having been watched. The Statistics area displays information about favorite movie genres and movie production centuries, based on the watch history. The Preferences area enables users to define and persist a set of preferences, that are

automatically used as a basis for the filter area on the Search view.



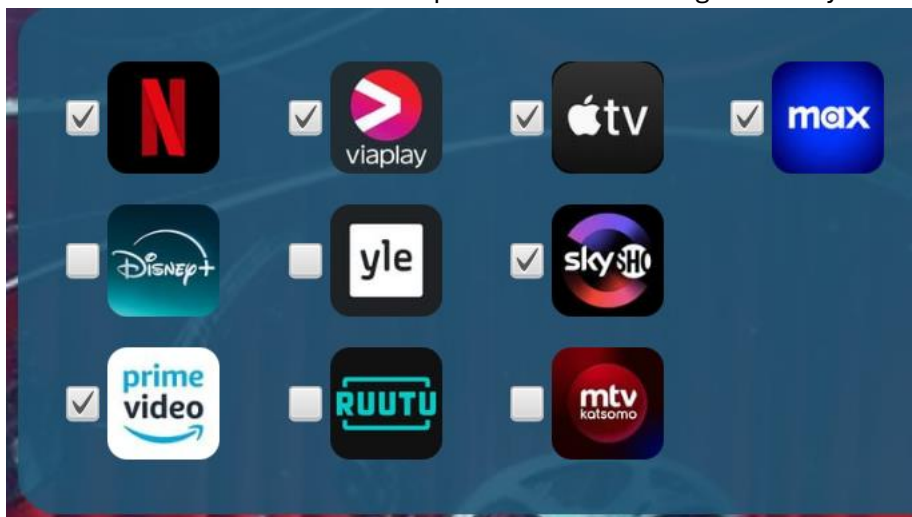
User Guide

After the application has started, the user is taken first to the search view.

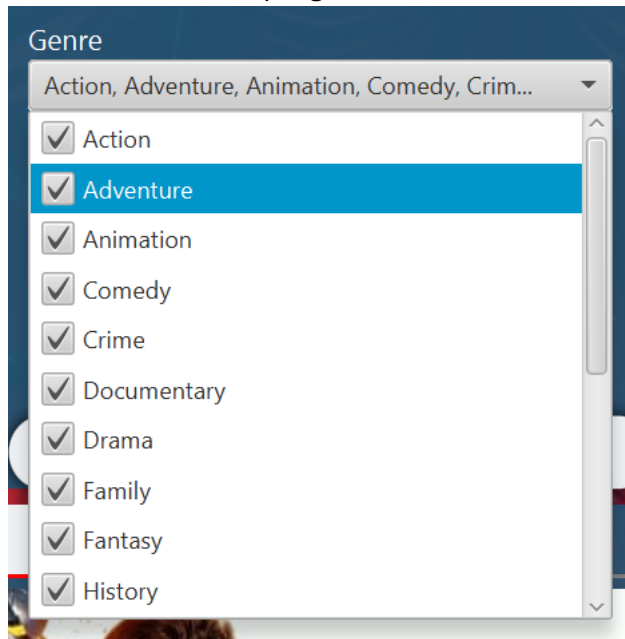
If the default lists already show an interesting movie, click it.

If you wish to use the filters:

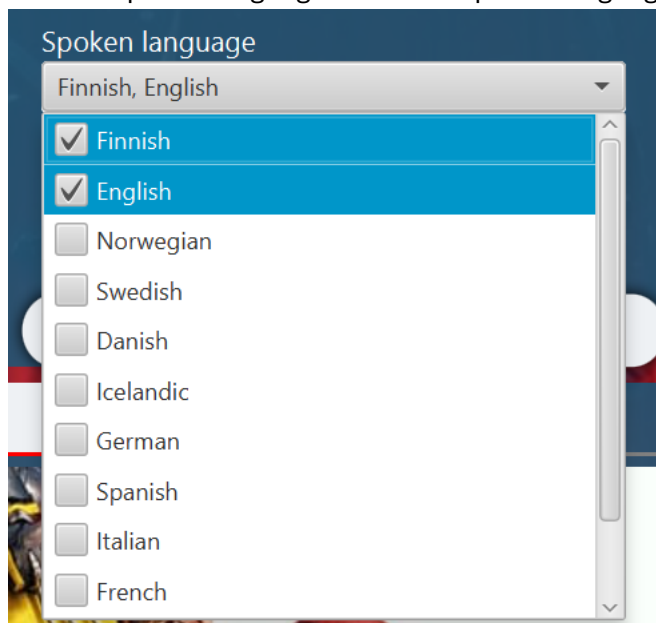
1. Select the checkboxes that correspond to the streaming services you wish to use.



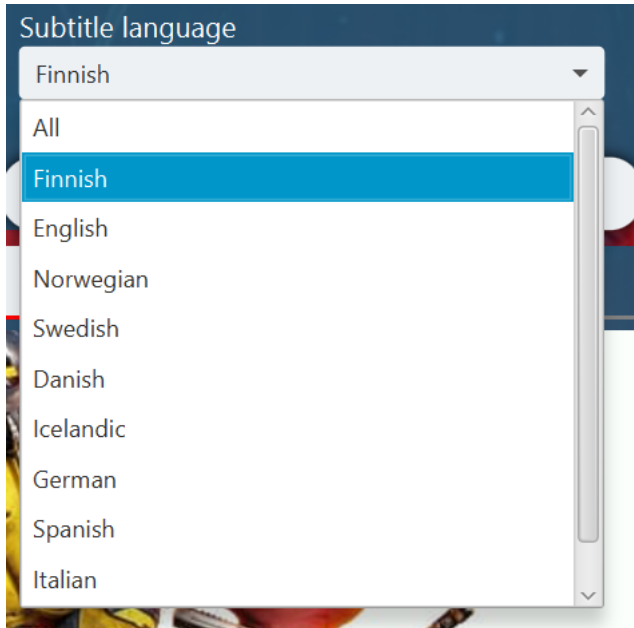
2. Choose one or multiple genres from the Genre menu.



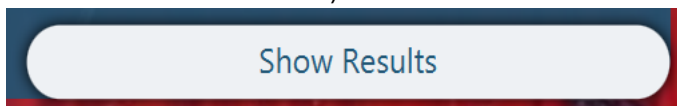
3. Choose spoken languages from the Spoken language menu.



4. Choose a single subtitle language from the Subtitle language menu.



5. Click Show results. (Please note that the Show Results button will stay inactive until all results have been loaded.)



6. Navigate to the movie detail view by clicking a movie picture/name.

Search for a Movie

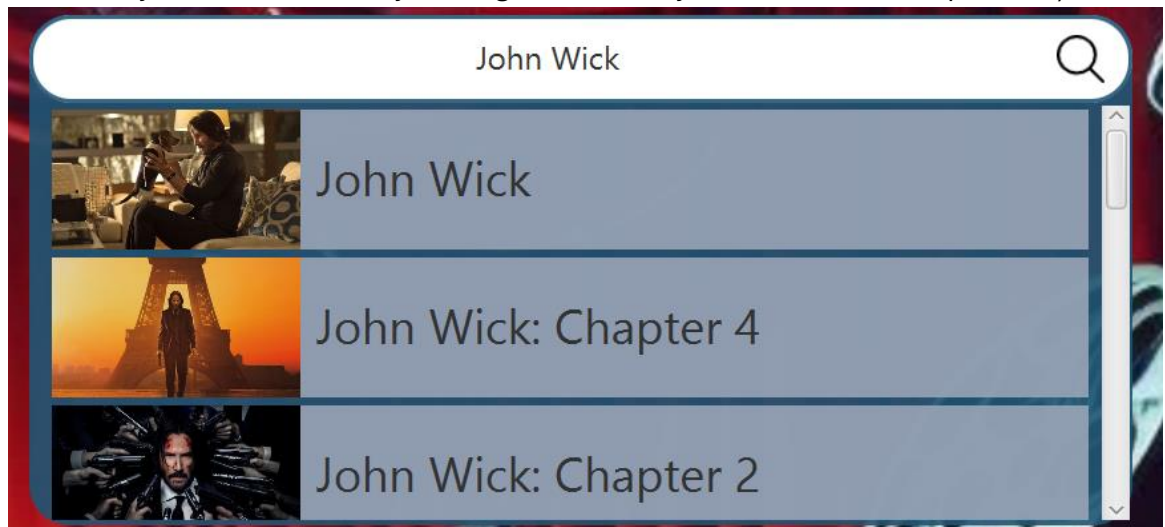
The search for a movie functionality is a completely generic feature. It does not use any of the search filters. Therefore, it can be used to find any movie from the database, including even those that are currently not available in any of the ten streaming services.

(Please note that there is a known issue with the JavaFX ListView control regarding empty, unselectable cells.)

1. Enter the name of the movie into the search field.



2. The list shows the results. (Please note that the list shows absolutely everything returned by the API, without any filtering. Occasionally, the results are a bit peculiar.)



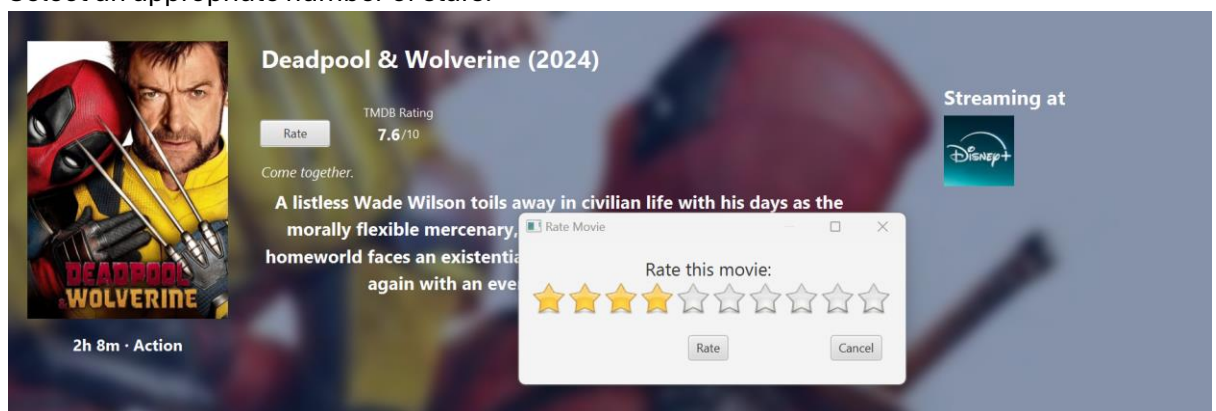
3. See the movie details by clicking the corresponding row. Only the movies that have been clicked are stored in the search history.

Rating a movie

In addition to seeing the TMDb rating, you can give the movies your own ratings.

Rating also means that the movie has been watched. This means that it will appear in the history list in the profile view.

1. Navigate to the Movie Details view by selecting a movie.
2. Choose Rate.
3. Select an appropriate number of stars.



4. Choose Rate. Or choose Cancel, if you do not wish to rate the movie.

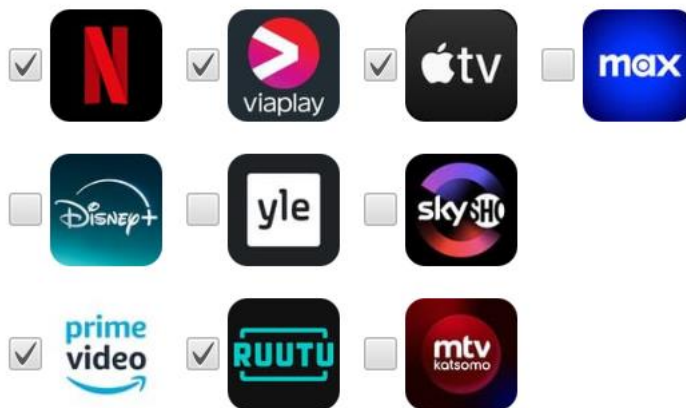
Saving preferences

1. Click the profile picture.



2. Choose the streaming services you want to be selected by default.

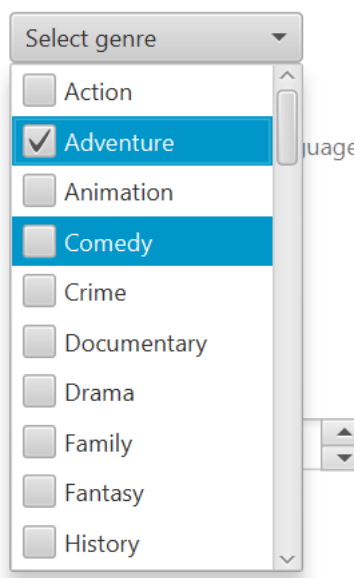
Streaming services



3. Select the genres you want to be selected by default.

Genres

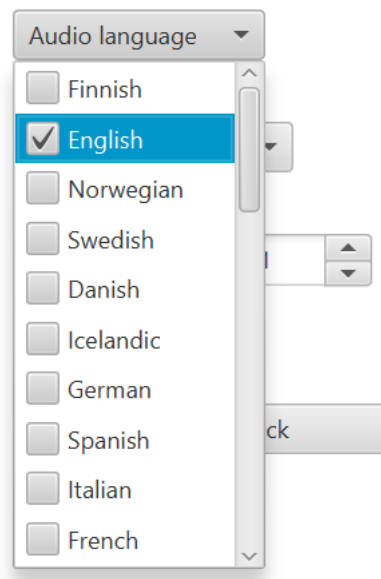
You can select multiple genres



4. Select the audio languages you want to use by default.

Languages

You can select multiple languages



Audio language

- ☐ Finnish
- ☒ English
- ☐ Norwegian
- ☐ Swedish
- ☐ Danish
- ☐ Icelandic
- ☐ German
- ☐ Spanish
- ☐ Italian
- ☐ French

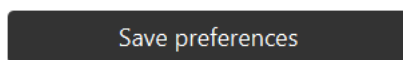
Back

5. Set a minimum rating value (uses movie TMDb rating).



Minimum rating

6. Choose Save preferences.



Save preferences

7. If you wish to cancel the editing of preferences, choose Back or click the logo in the upper left corner.

Check statistics

1. Click the profile picture.

2. See statistical information about your watch history in a graphical form.

Statistics

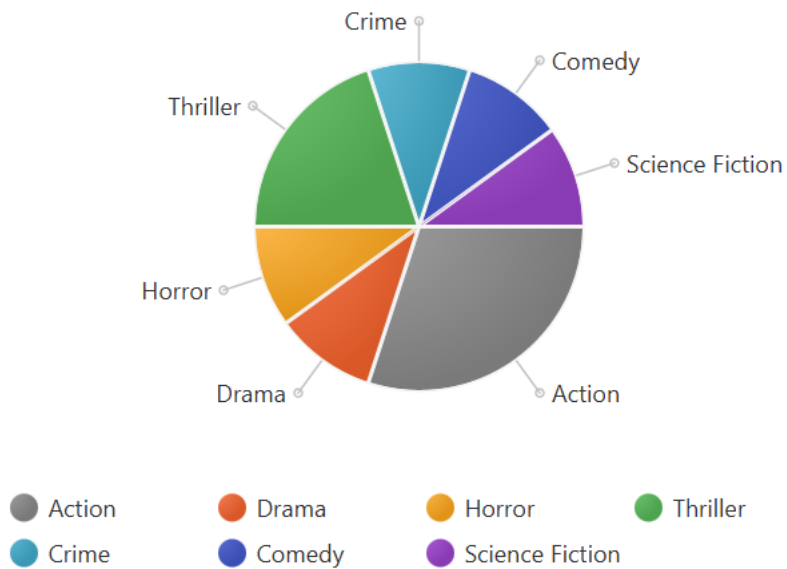
Minimum rating

1

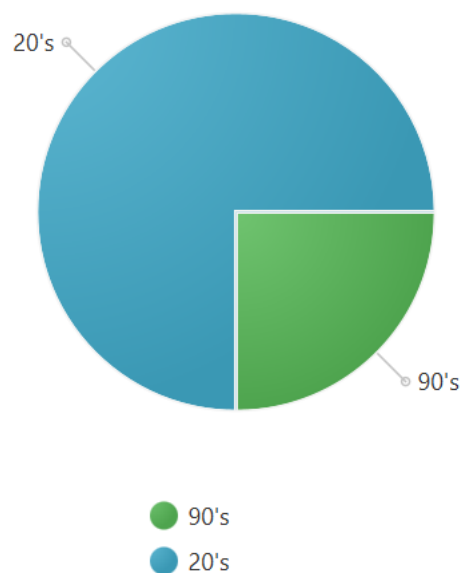


Apply

Your favorite genres



Your favorite centuries



3. You can filter the statistics by changing the movie minimum rating value.

Minimum rating

5



Apply

APIs Utilized

API	Functionality
The Movie Database (Getting Started (themoviedb.org))	Provides movie and tv-show data.
Movie of the Night (Shows - Streaming Availability API Reference (movieofthenight.com))	Lists streaming services that can be used to watch the movie/tv-show selected in the first phase. We use this to get language-area-specific streaming service and subtitle-information.
BONUS: OpenAI API	Provides snack and recipe information.

Classes, Interfaces, and Their Responsibilities

Below you'll find a description of the technical implementation, on a general level. At the time of this writing, this document is almost 30 pages long. Therefore, we left more detailed documentation for the project Javadoc files. Nonetheless, the central features are shown below.

Movies

The majority of the application implementation relates to movies.

Movies (TMDB API)		
Views	SearchView	Start view, initially showing a list of most popular and top-rated movies in the database. Enables searching for movies based on search criteria.
	MovieDetailsView	Displays detailed data about a selected movie.
	MovieLabel	A reusable component, showing a movie image, name, year and logos for all relevant streaming services. Used on the search view.
	MovieSearchResult	A reusable component, showing movie image and title. Used below the search field.
	RateView	Enables star-based movie rating.
Controllers	MovieDataController	Houses application logic for all movie-related functionalities.
	MovieDetailsController	Handles all things related to the movie details page.
	ImageController	Handles all image-related features.
	RateViewController	Initializes rate view and handles rate button press with appState.

	FilterViewController	handles setting filter options for genres, languages, subtitles, and streaming providers, as well as managing the logic for user interactions with filter-related UI components.
	MovieLabelController	Application logic for MovieLabel view.
	MovieResultsController	Application logic for movie search results.
	SearchViewController	Application logic for the SearchView.
Models	Genre	Represents movie genre.
	GenresResponse	Response data from the “get all genres” API. (https://api.themoviedb.org/3/genre/movie/list)
	Movie	Non-detailed data about a single movie. Used by PopularMoviesResponse and MovieSearchByTitleResponse.
	MovieDetails	Detailed data about a single movie.
	MoviesResponse	Response data for any data request about a movie/movies. ({All movie related APIs})
	Cast	Detailed data about a member of the cast.
	Crew	Detailed data about a crew member of the movie (writers etc.)
	Credits	Has both the crew and credits of the movie.
	SpokenLanguage	Represents spoken language information. (Inside MovieDetails.)
	StreamingProvider	Holds the information of a streaming provider
	StreamingResponse	Response data for a List of Streaming Providers (https://api.themoviedb.org/3/watch/providers/movie?language=en-US&watch_region=FI)

MovieDataController

Method	Functionality
getPopularMovies()	Returns a list of currently most popular movies.
getMovieDetails(movieID)	Returns detailed movie data.
searchMoviesByTitle(title)	Returns a movie list based on a title search string.
getAllGenres()	Returns a list of all available genres.

MovieDetailsController

Method	Functionality
setMovie()	Loads all of the movie details into the view.
clearMovieDetails()	Empties labels, images and elements.
setProviderImages(movie)	Sets the two provider images. Will be changed to use TilePane later.

setCast(movie)	Sets the relevant cast info including profile images.
handleRateButtonAction()	Passes the appState to the rate view controller and sets the scene.

ImageController

Method	Functionality
loadImageIntoView(path, fxId, width, scene)	Loads images from the TMDB API in to view based on fx:id and crops the image if necessary.
loadLogosIntoMovieLabel(path, tilepane)	Loads provider logos into TilePane in the movie label.
loadPosterIntoMovieLabel(path, imageview)	Loads movie poster into the movie label.

NOTE! As the movie-related functionality is in practice almost the whole application, there is no separate diagram. Please see the “The Big Picture” diagram at the end of this document.

Subtitles

Subtitles (Movie of The Night API)		
Controllers	SubtitleDataController	Application logic for all subtitle-related features.
Models	SubtitleService	Represents a json object that contains the language and subtitle information, in addition to the rental/purchasing/subscription services information.

SubtitleDataController

Method	Functionality
areFinnishSubtitlesAvailableAtAll (int movieId)	Returns true, if Finnish subtitles are available for any streaming service for the movie.
areFinnishSubtitlesAvailableForMovieInService(String streamingServiceId, int movieId)	Returns true, if Finnish subtitles are available for the chosen streaming service for the movie.
getServicesWithFinnishSubtitles(int movieId)	Returns a list of streaming services that offer Finnish subtitles for the selected movie.
IsFinnishAreaSupported(int movieId)	Returns true, if the movie is available on the “fi”-area.
CheckRequiredSubtitlesForMovieInService(int movieId, String streamingServiceName, List<String> languageList)	Checks if the selected subtitles are provided by the selected service for a movie.



Figure 1 Subtitle functionality information flow

Other & Utilities

Application State and User Data		
Views	ProfileView	Displays user-specific data, including view history and genres of movies rated by the user. Previous viewing data and ratings are used to display a collection of charts and figures.
Controllers	AppState	Application logic for user information, search history, movie ratings and application state management.
	ProfileViewController	Application logic for the ProfileView.
	TMDbUtility	Provides access to constant-based information, such as language codes.
	FileDataController	Handles reading data from the disk and writing data to the disk.
	GSONTools	Utility class for JSON-operations.
	HTTPTools	Utility class for HTTP-operations.
	LanguageCodes	Utility class to translate between two language code types.
	SceneController	Manages the creation of individual views and navigation between them.
	MovieGenres	Utility class to facilitate movie genre handling.
Models	ErrorModel	Utility class to handle TMDb errors.

AppState

Method	Functionality
getSearchHistory	Returns true, if Finnish subtitles are available for any streaming service for the movie.
addSearchedMovie(Movie movieSearch)	Returns the user's movie search history.
saveMovieRating(int movieId, int rating)	Adds a movie to the user's search history.
getMovieRating(int movieId)	Gets the rating for a movie.
getMovieRatings()	Gets all movie ratings.

ProfileViewController

Method	Functionality
--------	---------------

setStreamingProviders()	Sets the streaming providers by iterating over the streaming provider nodes and assigning unique IDs from the provider list.
setStreamingProviderLogos()	Sets the streaming provider logos in the search view by fetching and displaying them in the UI.

TMDbUtility

Method	Functionality
getProviderIds()	Retrieves a list of streaming provider IDs by using reflection to access the static fields in the <code>{@code PROVIDERS}</code> class.
getProvidersString()	Encodes the provider IDs into a URL-friendly string, where IDs are concatenated and separated by the ' ' character.
getProvidersString(List<Integer> list)	Encodes a given list of provider IDs into a URL-friendly string, where IDs are concatenated and separated by the ' ' character.
getLanguages()	Retrieves the list of supported languages.
getPopularMoviesUrl()	Generates the URL for retrieving popular movies from TMDb, with the list of providers encoded as a URL parameter.
getTopRatedMoviesUrl()	Generates the URL for retrieving top-rated movies from TMDb, sorted by vote average and including only providers supported by the app.
getGenresUrl()	Retrieves the URL for fetching movie genres from TMDb.
getFilteredUrl(List<Integer> list)	Generates a filtered URL for discovering movies from TMDb based on a specific list of providers.

FileDataController

Method	Functionality
readStateFromFile()	Reads app state data from a JSON file.
writeStateToFile(AppState appState)	Writes state data to a JSON file.

GSONTools

Method	Functionality
convertJSONToObjects(String jsonString, Class<?> targetClass)	Converts a JSON string to an object of the specified class.
convertJSONToObjects(String jsonString, Type type)	Converts a JSON string to an object of the specified type.

HTTPTools

Method	Functionality
makeGenericHttpRequest(String url)	Handles HTTP requests and responses asynchronously.

makeTMOTNHttpRequest(String url)	Handles HTTP requests and responses asynchronously. Only works with The Movie of the Night due to the mandatory headers.
----------------------------------	--

LanguageCodes

Method	Functionality
getLanguageCode(String key)	Gets the language code for the specified key.
Various getters for the language name in English, the Country code or the ISO code. Comprehensive listing in JavaDocs	

BONUS: Snacks (TBD)

Snacks (ChatGPT API)		
Views	RecipeView	Displays a recipe for a movie-theme-appropriate snack.
Controllers	RecipeDataController	Application logic for all recipe-related features.
Models	GenAIPrompt	Message to the generative AI service.
	GenAIResponse	Response from the generative AI service (the recipe).

Architecture

Architecture Patterns

The selected architecture pattern is MVC.

We spent considerable time researching^{iiiiiiiiv} and discussing this choice. As JavaFX does not dictate using a single pattern, it boils down to what suits the application best and what are the preferences of the development team.

The application itself is rather simple. A classic, clear division between the view, application logic, and the model was deemed sufficient. We opted for a version where the models are as lightweight as possible. In fact, they only represent data with appropriate getters and setters.

The length of previous experience and amount of knowledge of using architecture patterns varied greatly within the development team – starting with no experience and ending with professional work experience with multiple patterns. This impacted the selection process in favor of a simple pattern.

Design Patterns

Singleton

We had originally thought that we would implement all helper features, such as HTTP requests and GSON-operations adhering to the singleton design pattern. In principle, we do not need multiple HTTP-clients. In practice, as we are running multiple asynchronous HTTP-requests in

parallel, we apparently came across the exact same problems that Mr. Rantanen warned about during the eighth lecture. Thus, HTTP clients are not singleton.

Application state uses the singleton approach. The state object is a property of the Application (more about this below). In order to persist the application state, the state object (AppState) is saved on the disk as a JSON file each time the application is shut down. The state information is automatically loaded each time the application starts. This logic is based on the Application *start* and *stop* lifecycle methods.

All Other Design Patterns

As our application is a simple, data-driven app, where the whole approach is dictated by the use and structure of the API JSON data structure, there are no viable opportunities to use any other design patterns.

SOLID

Single Responsibility Principle

All the classes adhere to the single responsibility principle, perhaps even to a ridiculous extent. I.e., each class relates to a single task. As our app is based on a large, diverse set of data, this results in a similarly large and diverse collection of classes. The number of said classes is quite high.

There would have been opportunities to combine some of the classes by making them generic (for example the object-specific HTTP-response classes), but this would lose semantic information, consequently making it harder to understand the architecture of the application.

Open/Closed Principle

All of our classes adhere to the open/closed Principle. In practice, there is no opportunity to extend or need to modify any of the classes.

Liskov Substitution Principle

Movie Night Planner has only one inherited class (MovieDetails inherits from Movie). MovieDetails adheres to LSP – where applicable, it could be used as a Movie.

Interface Segregation Principle

We only have three interfaces that are used by appropriate classes. This principle is fully in effect in Movie Night Planner.

Dependency Inversion Principle

Movie Night Planner is fully compliant with the dependency inversion principle. In practice, we do not have high-level modules that could be affected by changes in low level modules.

Navigation

When we were thinking about how to solve the flow of information from one view to another and how to store application state data, we decided to implement a top-level navigation structure to tackle several of these questions on one go.

In addition to aforementioned issues, we analyzed what happens from the JavaFX application perspective, when a user navigates from one view to another. In practice, we thought about optimizing the application functioning under runtime based on either processing power or

memory usage. We decided to save processing power and take the penalty in the form of reserved memory.

SceneController creates a navigation concept that keeps already created views alive in the memory. Therefore, they are not recreated every time a user navigates to a view. This also enables us to keep state information alive in the views, saving HTTP requests.

Interfaces

As Movie Night Planner is a strongly-API-data-driven application, the only appropriate places to actually use interfaces are related to our own custom classes. Therefore, the number of interfaces is limited to only three.

Interface	Used by	Functionality
iFileOperations	FileDataController	Defines file read/write methods.
iGSONOperations	GSONTools	Defines JSON-related methods.
iHTTPOperations	HTTPTools	Defines HTTP-data communications methods.



Figure 2 - Interfaces and their usages.

Implementation Technologies

Below you'll find explanations for key technological selections. If we opted for using a third-party library, the library is listed below in a table with a source link.

Views

The user interface is implemented using JavaFX. More specifically, we opted for the XML-based approach whereby a baseline implementation of each view is created using Gluon Scene Builder. This will be later enhanced using the declarative Java approach, as needed due to view dynamics.

This makes view design quick and as effortless as possible.

ControlsFX

ControlsFX is a library that has several neat looking UI controls for JavaFX.

The rationale for this is simply to save development time. Also, the controls look really nice visually.

We use a rating control and a multi-select drop-down menu from this library.

HTTP Client

We use Apache HttpClient (version 5). ([Apache HttpComponents – HttpClient Quick Start](#))-

There are two separate clients for different services. The reason is that each client requires different headers, etc. We thought about creating a single general client, that would be configured per request, but we came to the conclusion that it would be absolutely ludicrous. Thus, we have a single client per service, preconfigured with appropriate settings.

The rationale for choosing this client is based on multiple criteria. The popularity of a library is always a good thing when doing technology selection. Additionally, it is flexible and comes with several advanced features. We based our selection rationale also on this comparison: [Which Java HTTP client should I use in 2024?](#)

GSON

Our whole app is based heavily on automated JSON data serialization and deserialization. Our technology of choice is GSON, reasons it being a really versatile library. Additionally, we were already familiar with it.

AI Disclosure

Various AI services were used with several tasks during the project, as detailed below.

Idea

We used several prompts to create dozens of application ideas. The idea that was ultimately chosen was from ChatGPT. The chosen prompt-idea combination is:

“Suggest ideas for a programming group project in an university software design class. The application should use and combine data from at least two open REST application programming interfaces.”

Answer from ChatGPT:

Movie Night Planner

- **Description:** A collaborative movie recommendation platform where users can select a movie and receive recommendations for streaming services and snacks.
- **APIs to Use:**
 - **The Movie Database (TMDb) API:** For fetching movie information and ratings.
 - **Utelly API:** For determining which streaming platforms have the selected movie.
 - **Spoonacular API:** For suggesting themed snacks and drinks based on the movie genre.

It was later revealed that we do not need Utelly API at all, but we actually do need to get subtitle information from somewhere. Thus, we dropped Utelly and started to use the Movie of the Night API.

Later, due to the changes in the group's headcount, we decided to drop also Spoonacular.

Code

All initial Java model class files were created by Copilot. For each API, we requested Copilot to create a Java class that corresponds to the JSON data that each individual API returns. The Author-tag in such Java classes include "Copilot" or "ChatGPT", in addition to at least one human name.

Example prompt:

Please give me a Java class that corresponds to the following JSON: <insert JSON from API response here>

The draft level Java code was refined by the team members. This means adjusting property visibility settings and, e.g., refactoring the Movie-class out of multiple API responses into a class of its own.

Graphic Design Resources

All app logos, background images etc. Were originally created using DALL-E.

Javadoc

The majority of all comments in the code were created with the help of AI (Copilot and/or ChatGPT).

Example prompt:

Please create Javadoc comments for the following Java class: <insert Java class to be documented here>

Usefulness

Code Generation

AI can be used to quickly create a foundation for the application. This means that the programmer must be a skilled application developer in order to be able to prompt just the required parts and exactly in the right way. A "Please create me an app" approach does not work.

The code generated contains absolutely everything. Also, GenAI is not deterministic - results may vary. For example, one day you get toString(), other days you do not. Thus, the human programmer must review everything carefully.

Graphical Assets

This results in generic, run-of-the-mill stuff. Better than nothing, but a professional human graphic artist creates better visuals.

Documentation (Commenting)

Completely usable text. ChatGPT surprised us this time around by creating comments that reference other classes and the overall context. Copilot did fine, but all comments relate only to the local context.

External Libraries

This is just a list for clarity and links. The rationale for selection is described above in the technology section.

Library	Source
Apache HttpClient	Apache HttpComponents – HttpClient Overview
ControlsFX	ControlsFX ControlsFX
GSON	Google - GSON

Division of Work

The division of work can be seen on the GitLab issue board on a detailed level. Work has also been discussed on Teams to a smaller extent, but the actual work items have been made visible on the board.

The table below shows our design/implementation responsibilities on a high level.

Responsibilities

Feature	Task/Responsible
Application functional design	All three of us
Application testing	All three of us
Documentation	All three of us
Code review	All three of us
Design and implementation of the initial application skeleton (architecture, first views, movie controllers, and movie models)	Jani Ilkka
Design and implementation of models and logic for top rated movies.	Kian Moloney
Design and implementation of models and logic for streaming providers.	Markus Siitonen
Design and implementation of the Search view with appropriate application logic and models	Markus Siitonen
Design and implementation of the Movie Details view with appropriate application logic and models	Kian Moloney
Initial visual wireframe of the Profile view	n.n.

Design and implementation of the preferences functionality on profile and search view	Markus Siitonen
Design and implementation of the statistics functionality on the profile view	Kian Moloney
Design and implementation of the data persistence functionality (for ratings, preferences, etc.) for the application	Jani Ilkka
Design and implementation of the generic HTTP functionality for the application	Jani Ilkka
Design and implementation of the navigation/view persistence functionality for the application	Markus Siitonen
Design and implementation of the subtitle service-related controller and models.	Jani Ilkka
Design and implementation of the generic JSON-functionality (GSON) for the application.	Jani Ilkka
Design and implementation of all interfaces in the application	Jani Ilkka
Design and implementation of automated unit tests	Jani Ilkka
Create Javadoc comments for all classes	All three of us.
Design and implementation of the rate view	Kian Moloney
Refactoring code to adhere to OO principles	Kian Moloney
Refactoring code to ensure that the application structure adheres to the MVC design selection	Markus Siitonen
Design and implementation of data statistics and movie rating data persistence.	Kian Moloney

Self-evaluation

This section is divided into two separate subsections, one pertaining to the design and another to the technical aspects.

This four-person project suddenly turned into a three-person project. One group member decided to leave the project half-way without doing pretty much anything. In addition, the three of us had paced our work effort to do most of the work during the first period, due to the second period being more demanding for all of us. Unfortunately, this turn of events in practice meant

that the three of us did simply not have the resources to implement all the features we would have liked to have implemented. This also meant that some of the technical solutions are not as refined as we had originally planned. Not everything has been as thoroughly tested as we had hoped (there are no known show-stopping bugs in our code).

Design

We were initially a bit worried about the potential limiting factors of JavaFX when compared with what we had done with Figma. As is evident from the final UI, our fears did not come true. With enough patience and tinkering, JavaFX can be used to create visually impressive user interfaces with, e.g., transparency and blur.

Nonetheless, there are small “features” that may present themselves during the execution of the program. Some dropdown may occasionally display null. The movie list may blink, if new data keeps being added asynchronously in the background.

Technical aspects

After the Programming 3 course, our team consists of seasoned JavaFX veterans. We had a pretty good idea as to what is possible and how things should be implemented.

Nonetheless, after a couple of weeks, we pivoted the whole application idea from a map-based app to the Movie Night Planner. Originally, we had thought that we would create a Tampere area event app. As it turned out, LeafletJS-support has been broken for a while on JavaFX. (It still works, but only with old libraries and we did not wish to go down that route.) We tried to find a workaround or a comparable library. We found Gluon maps and quickly created a couple of working prototypes but felt a little unsure about that choice. A really low number of examples and tutorials also steered us away from that.

Regarding the movie app, the largest changes were related to the APIs we thought we would be using. For streaming services and subtitles we dropped Utelly and went with Movie of the Night. Regarding snacks or food, we noticed that Spoonacular (obviously) has U.S.-specific products. A generative AI-based approach was deemed better. Due to changes in the group project personnel, we were unable to finish this.

Reuse of components

We were originally planning to divide the UI into separate components and reuse those in multiple places. This was deemed unfeasible. What we noticed in practice was that while the reuse of FXML was effortless, when connecting controllers and trying to reference UI controls in an imported view, we ran across fatal problems. Obviously, on a view-specific level, there is a lot of reuse. For example, both search view movie lists repeat a single movie-specific FXML-design.

Known issues

There are a few minor issues with the application that a user may come across. These are either bugs in the Oracle or third-party components or inconsistencies in service data. Within the timetable and with the constrained resources that we had, we were unfortunately not able to fix these.

Issue	Description
Genre and Spoken languages dropdown lists sometimes show null or goes otherwise visually haywire.	Known bugs. The data should still be OK. For example, in the cases where the individual checkboxes refuse to activate, you can still see the correct data string on the top.
The data is not perfect – not on TMDB or Movie of the Night.	Not all movie data is error-free in any way.
Null search result from the Movie of the Night	Even though this service uses the TMDB API, not all movies are found on the MOTN-database. For example, with Finnish speaking movies, this may lead to null being returned
Subtitle API key limit logic changed	During the latter part of this project, the API key usage count logic suddenly changed. Instead of having low, daily limits, the service deprecated the original option and now has only one monthly option. This may affect testing. Please contact us, if you get the HTTP error message 429.
Search ListView shows empty rows, if the search history is smaller than three, four items.	This is a known issue in the JavaFX ListView. ListView - Don't display empty unselectable rows. - Oracle Forums Unfortunately, we simply did not have time to fix this.

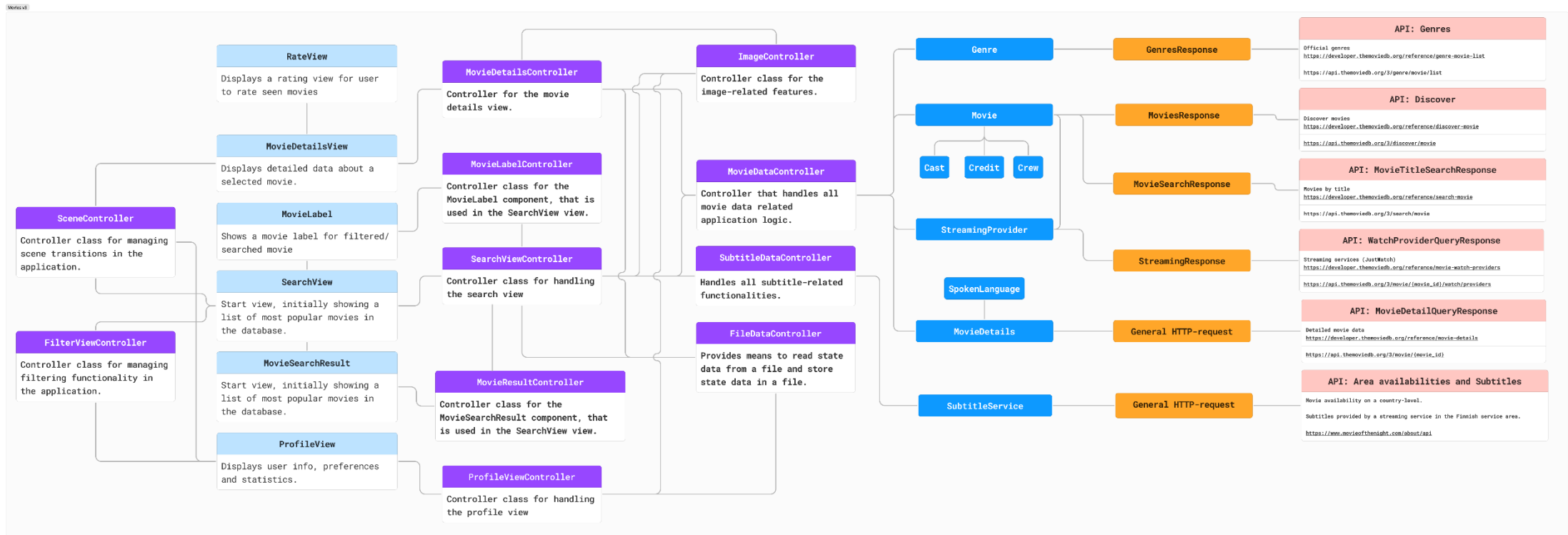


Figure 3 The Big Picture.

References

- ⁱ [Design pattern to use : r/JavaFX \(reddit.com\)](#)
- ⁱⁱ [Implementing JavaFX Best Practices | JavaFX 2 Tutorials and Documentation \(oracle.com\)](#)
- ⁱⁱⁱ [Best practices for JavaFX architecture and patterns? - Oracle Forums](#)
- ^{iv} [Implementing MVC in JavaFX - PragmaticCoding](#)