# Smart Movie Recommendation Project

## Introduction - Motivation

As media enthusiasts, we found it in our best interest and passion to create a movie recommendation system. The main issue with being a lover of movies is that you never know what to watch next, let alone where you can watch the next movie you're interested in. We seek to eliminate this impending problem by curating limitless options for users to choose from based on personal preference and popularity, while assuring users of where they can watch their next film depending on the streaming service. The inclusion of specialized features such as personalized filtering, smart suggestions, dynamic watchlist and history display, and serendipitous discovery gives users the option to experience the recommendation system however they please. Moreover, the smart movie recommendation system is something we as its programmers wished we had for ourselves. Therefore, every minuscule detail implemented has the user's experience at the forefront of its' development. The programming language chosen for such a demanding and personable project is Ada.

## Language Specifications:

The programming language used for this project, Ada, is a statically-typed, structured and object-oriented language that follows the multi-paradigm approach. It is particularly known for its emphasis on safety, reliability, and maintainability, especially in high-integrity and real-time systems. Ada emerged in the late 1970s as part of the U.S. Department of Defense initiative to develop a standardized language suitable for embedded and mission-critical applications. Its roots lie in several influential languages, including Pascal and ALGOL, from which it inherits a strong type system and structured control flow mechanisms. The first official version, Ada 83, was later enhanced to Ada 95, which introduced object-oriented features, and all subsequent versions of Ada have continued to evolve the language to include better support for concurrency and contract-based programming.

Ada supports multiple paradigms including imperative, structured, modular and object-oriented programming. The language emphasizes strong typing, compile-time checking, and built-in support for concurrent programming, making it ideal for real-time and embedded systems. Ada is commonly used in aviation, military, and transportation industries, where system failure could have catastrophic consequences.

Ada's language elements include a rich set of reserved words such as procedure, function, begin, end, if, then, loop and exit, which contribute to its strong readability. The primitive data types in Ada include Integer, Float, Boolean and Character, while structured types as Record, Array and Access types allow for more complex data organization. In this project, for example, the Movie_Record type is a user-defined record that holds information about each movie, demonstrating Ada's support for structured data.

The syntax of Ada is designed to be highly readable and self-documenting. It uses explicit declarations, strong typing and block structures with clearly marked beginning and end points. For instance, a procedure begins with the procedure keyword and ends with "end", and conditionals follow an if..then..else..end if format, which helps maintain clarity even in large codebases. The control structures include standard constructs like for and while loops, if-else conditionals, and case statements, all of which enforce strict type compatibility and scope definition.

Ada supports abstraction through procedures, functions, packages (modules), and objects. In this project, procedures, such as Load_Movies and Display_Movie are used to modularize functionality, demonstrating procedural abstraction. Ada's strong support for modularity via packages makes it well-suited for large-scale software systems. Although object-oriented programming was not extensively used in this implementation, Ada does support it through tagged types and inheritance, introduced in Ada 95.

Ada is especially strong in readability because of its english-like syntax and clearly defined structure, which makes code easier to understand and maintain. Writability, on the

other hand, is more mixed. While Ada encourages good programming habits and helps prevent mistakes, its strict rules can make writing code feel slower compared to more modern, flexible languages. With features such as strong typing, thorough compile-time checks and support for defensive programming, it is well suited for building software where safety and correctness are essential.

The Movie Recommendation project highlighted Ada's strengths in managing structured data and supporting modular program design. The Movie_Record type and the vectors from the Ada containers library made it easier to manage lists of movies efficiently. We used procedures like Load_Movies and Display_Movie to break the program into clear, logical parts, showing Ada's support for procedural abstraction. Ada's strong typing and modular design helped keep things organized, even if we had to be more intentional with our code because of its lack of dynamic typing and more limited standard libraries.

Problem definition(Include use cases)

The implementation of a console-based movie recommendation system in Ada. Our implemented system in Ada allowas a single actor, the user, to interact with a predefined catalog of movies/films. At its core, the catalog is stored as an 'Ada.Containers.Vectors.Vector' of Movie_Record, with each record holding an inbounded title and genre, an integer duration, service string , and rating. All user interactions happen through a main 'Get_User_Input' procedure that presents a looping menu. Each menu option responds to a nested procedure, ensuring the code stays modular and each piece of functionality resides in its own denoted subprogram.

The main use cases utilized fall under the following functions: filtering movies by genre, filtering movies by duration, filtering movies by streaming service accessibility, and filtering movies by popularity or rating. Each actor for the use cases is the user, the accompanying trigger is "Filter by" with the corresponding attribute that the user wishes. The flow is very simple, the system will display a number list of results based on the user's selected attribute. The result is a clean numbered list of movie titles, genres, durations,

services, and ratings presented instantly without any extra clutter. This allows for a more intuitive and minimalistic design for our users to ensure that when they are using our program there is no confusion on how to navigate nor read the results from our system.

The other use case functions include; adding movies to a user's watchlist, adding movies to the user's history, suggesting movies based on user's history, displaying individual watchlists, searching for random movie recommendations, customizing user movie selection, and of course exiting the program. The actor for these cases are the same as the filters, however, the property of the trigger is more specific to each case. The flow is just as simple for each case, depending on the user's needs the system will allow the user to add, remove, generate, or search movie titles. Most use cases will provide the user with a menu or generated list to manipulate movies they are interested in, while other cases prompt the user to denote the movie they are interested in to search and find the results they are looking for.

Furthermore regarding intelligent recommendations, the system will allow the user to select "Get Suggestions Based on History," which triggers a two-step course of action. The Suggest_Based_On_History routine first tallies the most frequently watched genre from the User_History vector, then delegates back to Genre_Filter to display all movies within that genre. Alternatively, "Get Random Movie" leverages Ada's Discrete_Random package: it constructs a random index into the Movies vector and hands that single film record to Display_Movie, offering an unpredictable discovery experience. Finally, the "Custom Movie" option scaffolds a more advanced multi-criteria filter by asking the user, for the genre, service, rating, and duration consecutively. Thus, it echoes those choices via the User_Movie procedure, laying the groundwork for future improvements and implementation.

Proposed method

When a user chooses an option from our menu such as, a random movie, finding movies based on genre, duration, rating, or streaming service, our program will use a filtering system for each choice. The system will filter through our database of movies and based on the users input such as if a user chooses an option from the menu that finds a random movie

based on a specified genre such as horror, our program will search through our movie database for movies that fall under the category of horror. When movies are found, based on the user's input value it will display the full movie details, including the genre, rating, duration, streaming service and title of the movie. The filtering algorithm was chosen as it is the simplest and most logical implementation for our project's goal and for our programs system.

Compared to other movie recommendations our system offers a user-friendly interface that is simple and minimalistic. Our program is uncomplicated and short, when run a menu is shown in the terminal or console depending on the system the user is using. The menu shown isn't too cluttered unlike other movie recommendations systems, we prioritized an easy-use system thus allowing anyone of all ages to enjoy our program and understand how it works. When a user chooses an option, on their end, they can efficiently enter their choice without it being too confusing or inefficient. The most a user will have to submit, based on their choice from the menu, is entering a genre and searching for a movie, everything else is straightforward input.

Another feature we have implemented that some movie recommendation systems do not have is a watchlist. Users will be able to store any movie they have found based on their choice, they can save random movies, movies they decided to search, movies they found based on rating, etc. This is done easily, as when a movie is found for the user they will have an option to go back to the menu and save that specific movie to their watchlist, and have access to these movies when they return to the main menu of our program. Although we do not have a fancy GUI interface for our users to use, we ensured that our system was an effortless and manageable program that our users can utilize to store, search, filter and find movies to expand their knowledge of media today.

The main algorithm this system used as stated before is a filtering algorithm. This algorithm is used to filter through our movies database and find a movie for the user based on the input they entered in our program. One of the filters that can happen based on the user's

choices is a content-based filter, which is done when a user interacts with our program, chooses a choice and enters their input in the console. This is a simple algorithm that searches through our database based on what the user enters and displays the movie depending on the user's input. Our program applies a manageable execution of our backend and an intuitive frontend, with this our program stands out in terms of simplicity and implementation.

Experiments/Description of experiments

Numerous experiments were conducted during the creation of this system program. Ensuring that the values our users input during our system was one of the main experiments that was carried out and implemented. If a user enters an incorrect genre name, the program will tell the user there are no movies found and they will have to enter a genre again. For entering a rating if a user enters a string value they will be prompted to enter a different input that is an integer. When asked to enter a duration, similar to how the rating filter worked a user must enter an integer value, if they do not they will be prompted to enter an integer value. When it comes to streaming service it is similar to what happens when a user enters an incorrect genre, the user will be prompted to enter a correct streaming service name. There were experiments done on the menu that first shows when the user opens and runs on our program. When the menu is shown choices will display and the user will have to enter the corresponding number to the menu choice they would like to continue with. If a user enters a number is not an option within the menu they will be prompted to enter a number that is displayed within their console. When a user enters the correct information into the console, the result they enter must be accurate to their input. This was done by ensuring that what the user inputs is first correct and secondly that the program understands what it is filtering.

Print logs were done to ensure that the program understood what the user input was for example when an user entered data such as a streaming service such as HBO Max, it will be displayed after they have entered it to not only confirm to the user their input but also for us as developers to ensure that the correct data is being filtered within our database and that the user will have accurate movie information based on their data. To ensure that the value

the user was entering into the program was correct, exceptions were done. For example when a user runs our program they will be shown the main menu and to navigate to a displayed option the user must input the corresponding number to that option, if a user input something such as a word or particularly a string, instead of an integer the user will be prompted to re-enter a valid choice. If a user enters an integer that was out of bounds of the choice listed on the menu, the user will be asked to input an integer that falls within the range of the our menu options. This same logic is applied in other choices that require integer or number value input from the user.

Based on these experiments, we were able to observe that the programming language Ada relies heavily on making certain that programmers are using exception handling and debugging to make sure that their code is correct and the input that may be given to a specific program by a user is accurate. Ada is designed to catch errors during compilation time instead of runtime, forming a programming language that is reliable, being the reason why it is used primarily in critical systems such as aviation, space technology, and the military. The language assists software developers in creating robust and safe software systems without having too much to worry about optimization, leading to developing safe and reliable software systems.

Test bed

The test bed developed for the Movie Recommendation project is a command-line based interface that interacts with a static dataset of movies. Each movie entry is structured as a Movie_Record, which holds key attributes such as title, genre, duration, streaming service availability and user rating. Depending on the entry and attribute the user must enter a value that is a string or integer in the command line. For instance, if the user chooses our duration or rating option from the menu they must enter an integer value to find and access the data within our movies database, if not the program will prompt the user to enter again until they input a value that is an integer. This structured database serves as the foundation for testing the system's core functionalities and its ability to provide accurate recommendations based on specific user inputs.

The experiments conducted within this test bed are designed to address several critical questions. First, we want to determine whether the system can accurately filter and recommend movies that match user-defined criteria such as preferred genre, maximum runtime, movie rating and desired streaming platform. Second, we aim to evaluate how Ada's language features—particularly its strong static typing, modular design, and procedural abstraction—contribute to the implementation and reliability of the recommendation logic. Third, we aim to evaluate the extent to which the current structure supports extensibility, such as incorporating user history, dynamic input parsing, or more sophisticated search and sort features. Lastly, we also assess how the lack of dynamic typing and some of the more limited standard libraries in Ada affect the development process and runtime flexibility. By systematically varying user input and observing program responses, the test bed provides valuable insight into both the strengths and limitations of Ada in developing scalable, dependable software solutions.

## Conclusions

In conclusion, the Ada-based movie recommendation system effectively demonstrates the language's ability to create reliable and maintainable software. By organizing the movie database with a structured Movie_Record and utilizing procedures for modular functionality, the system allows users to easily filter and find movies based on their preferences. Ada's strong typing and exception handling ensured that user inputs were validated and that the program remained stable throughout the process. Although the system uses a simple command-line interface, it is intuitive and user-friendly, providing an easy way for users to manage their movie preferences and discover new movies to add to their watchlist or expand their knowledge of media. Moving forward, the system could be expanded with more advanced algorithms and live streaming data to enhance its capabilities. A GUI may also be implemented to level up the users overall experience with our system and their interactions. Overall, this project highlights how Ada's features can be applied to build effective applications while maintaining safety and reliability.