TECHNICAL REPORT: ARUPPI PROJECT

Universidad Distrital Fracisco Jose de Caldas
Software Modeling
Xiomara Salome Arias-Carlos Andres Celis
June 11th, 2024

USER STORIES

- As an administrator, I want to add anime, so what to have updated content.
- As an administrator, I want to add news, so what to update the content
- As an administrator, I want to remove news, so what to have updated information.
- As a user, I want to pause at a station, so what to stop listening to music.
- As a user, I want to play on a station, so what to listen to music.
- As a user, I want to watch anime, so what I hang out.
- As a user, I want to listen to Japanese stations, so what I learn Japanese music and known facts of their country.
- As a user, I want to know the news about Japanese culture, so what I keep me informed.
- As a user, I want to add my favorite content, so I categorize them according to my preferences.
- As a user, I want to see my history of episodes, so what I don't lose the thread of it.As a user, I want add content to the list queue, so what I can see it after.

ENTITIES

- Anime
- News
- Radio
- User

CRC CARDS

Anime_factory	
Responsibility	Contributors
• **Create diferents types of anime**	• main

AnimeDAO	
Responsibility	Contributors
• Add anime • Get anime • Delete anime	• AnimeFacade • Anime

User	
Responsibility	Contributors
• Log in • Sign in	• main

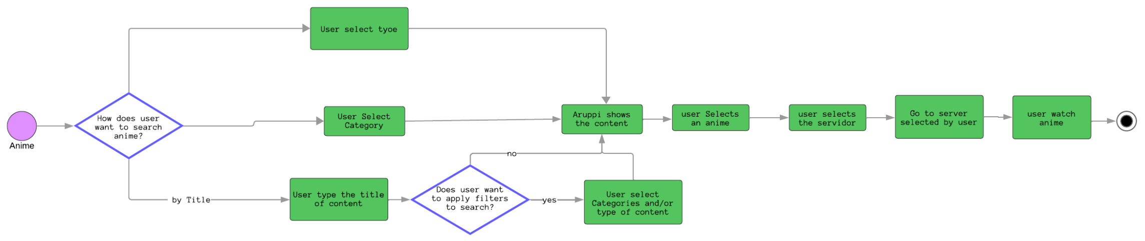
UserFacade	
Responsibility	Contributors
• Add to favorites. • Add to recommended. • Add to queue. • Show favorites. • Show recommended. • Show queue.	• main • Collection

NewsDAO	
Responsibility	Contributors
• Show news. • Get new's information. • Add news. • Remove news	• NewsFacade • News

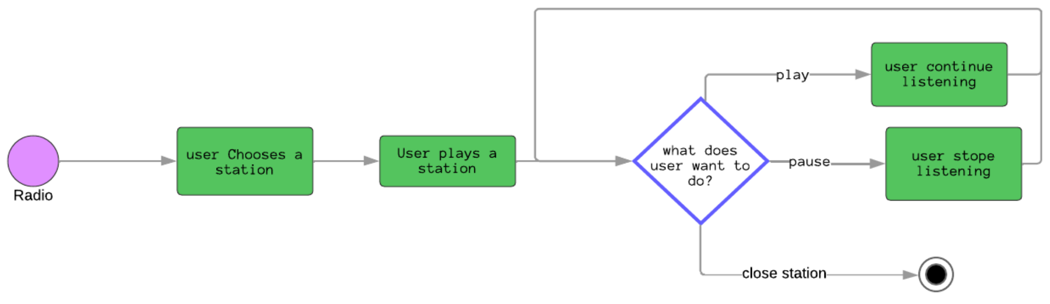
Radio	
Responsibility	Contributors
• Pause and play the station	• RadioFacade • State

DIAGRAMS

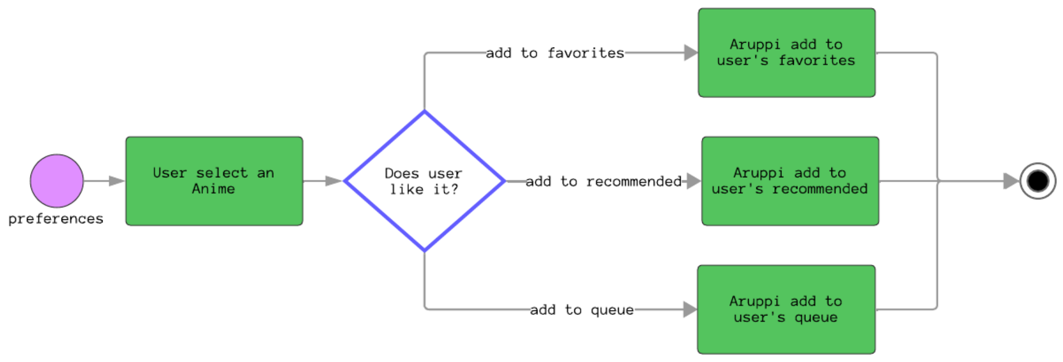
- Anime activitty diagram.


Anime
How does user want to search anime?
User select tyoe
User Select Category
by Title
User type the title of content
Does user want to apply filters to search?
no
yes
User select Categories and/or type of content
Aruppi shows the content
user Selects an anime
user selects the servidor
Go to server selected by user
user watch anime

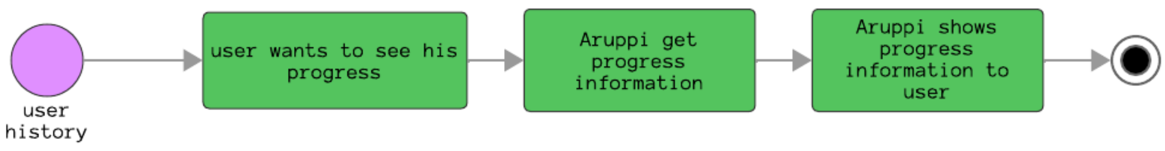
- Radio activity diagram



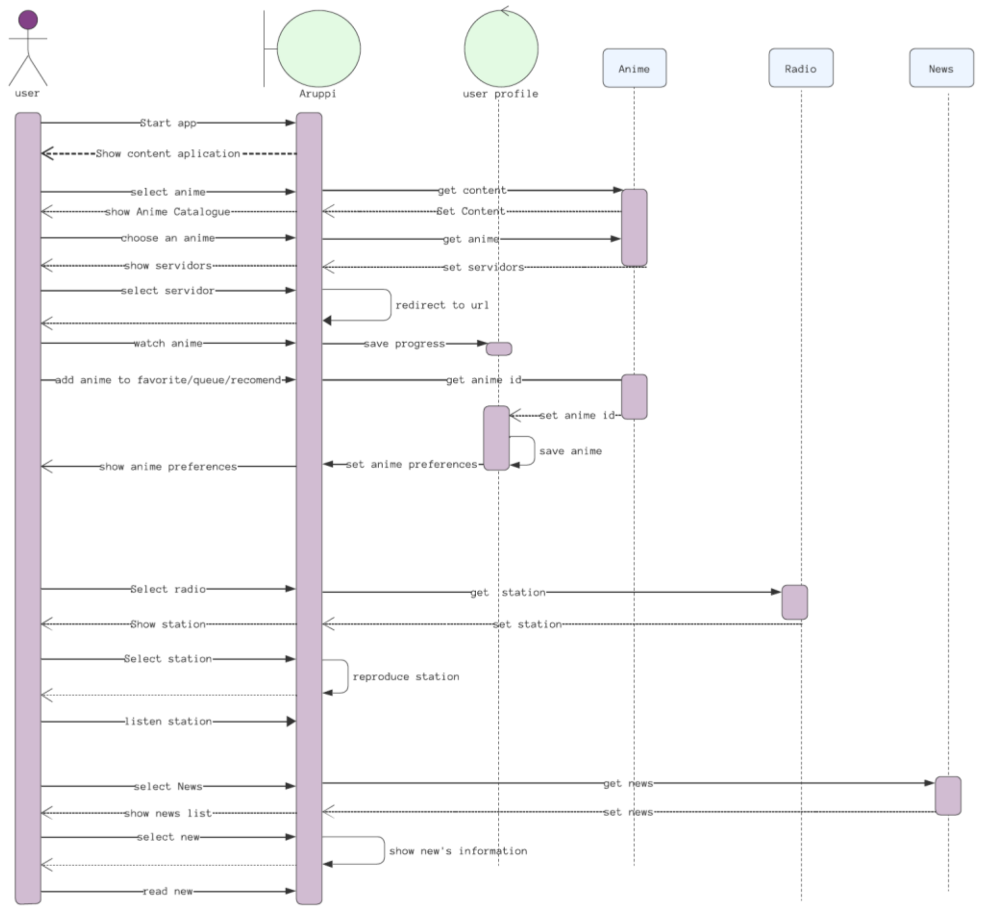
- Preferences activity diagram



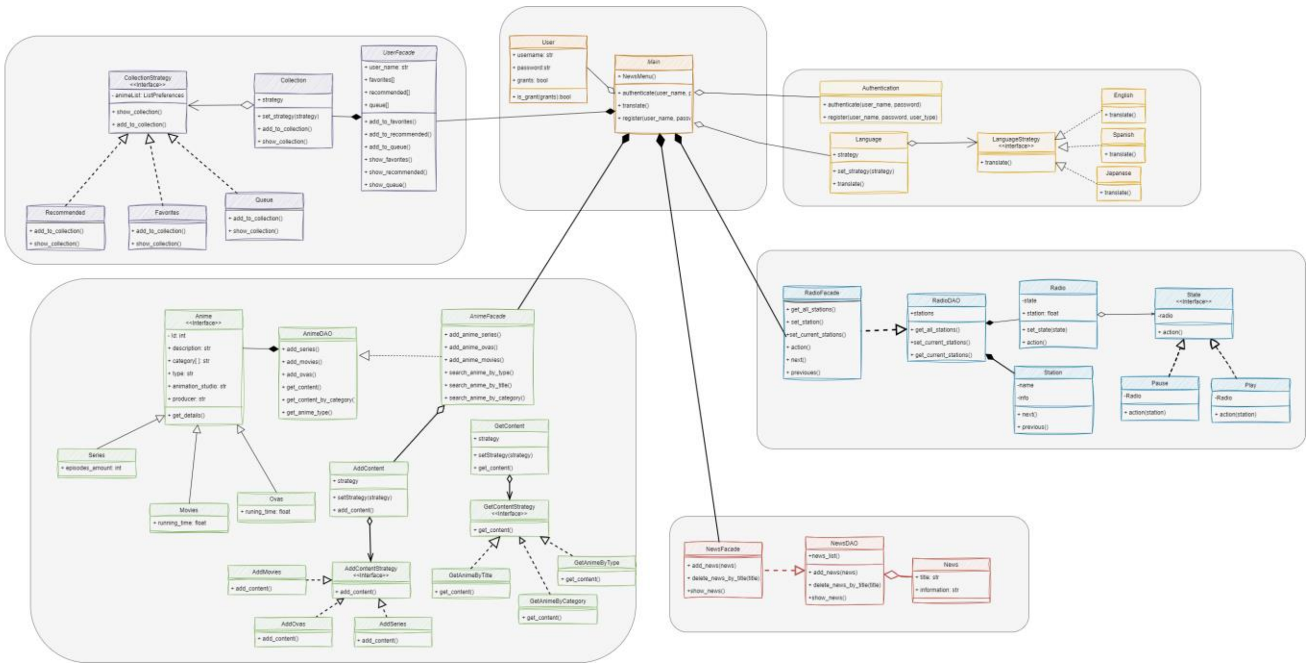
- User history activity diagram


user history
user wants to see his progress
Aruppi get progress information
Aruppi shows progress information to user

Sequence Diagrams



- Class diagram



For the class diagram, significant changes were made with respect to the initial one, since it sought to comply with the SOLID principle, and to implement design patterns that would fit the problems that Aruppi was trying to solve.

TECHNICAL DECISIONS

- **Strategy Pattern:**The Strategy pattern was used in the Anime subsystem to handle different content addition strategies (series, movies, OVAs). Each type of content (series, movies, OVAs) had its own aggregation strategy implemented in separate classes (AddSeriesAnime, AddMoviesAnime, AddOvasAnime), allowing for easy addition of new content types without modifying existing code. Also in Collection, get content and languages.
- **State Pattern:** In the Radio subsystem, the State pattern was employed to control the state of the radio player (play or pause). The Play and Pause classes represented the player states, and the current state was dynamically changed based on user interactions. This allowed for more flexible and extensible management of the radio player behavior.
- **Facade Pattern:** The facade was used in various subsystems, such as Anime and News. The AnimeFacade class provided a simplified interface for interacting with the Anime subsystem, hiding the underlying implementation details. Similarly, the NewsFacade class offered an easy-to-use interface for working with the News subsystem.
- **Factory Pattern:** Factory pattern to create object instances, for example, when adding anime content, to create instances of different types of anime (series, movies, OVAs) based on user-provided data. Therefore, for the creation of these objects this pattern is very helpful because they are created in a similar way and so the client will only interact with the factory.
- **Facade pattern:**The facade was used in various subsystems, such as Anime and News. The AnimeFacade class provided a simplified interface for interacting with the Anime subsystem, hiding the underlying implementation details. Similarly, the NewsFacade class offered an easy-to-use interface for working with the News subsystem. In order to divide responsibilities and look for a better understanding of the project, it was divided into a series of subsystems that control all the logic of the project. With the implementation of this pattern these sub-systems show what is necessary for the users without the need to see the logic of the project.
- **Modularity implementation.** Modularity is a fundamental part of Aruppi's modeling, so it was divided into 6 subsystems, to divide responsibilities and have a better management over the code. That is to say, a subsystem for each type of content of the Japanese culture (Anime, Radio and News), where the logic of each of these is implemented; on the other hand, the core-subsystem, implements the central and initial logic of the application, such as authentication and translation; finally, user profile subsystem, is the one that manages information of the user's profile, regarding their preferences.
- **Divergent change.** In Aruppi there are two classes that due to the need and purpose of this app were going to have a larger size than the others, the “user” and “anime”, therefore, from the beginning they were built dividing responsibilities. For example: for the functions that anime was going to provide we opted for two patterns that would help to complete them and avoid the unmeasured growth of this.

- **Switch Statements.** Aruppi was going to handle the user options which could have many different options which would pose a problem to have many switches, if or else to handle these user requests, so we opted for the use of design patterns such as “strategy” or “state” to avoid this problem.
- **Web services** :This main file of the Aruppi project serves as the entry point where various web services are defined to interact with external users, including front-end clients. The services encompass functionalities ranging from user authentication to managing anime content and news. Each service is carefully structured and adheres to the FastAPI framework, ensuring efficient request handling and response generation. The implementation includes Pydantic model definitions for data validation and serialization, ensuring data integrity and type safety. Through this organized and well-designed approach, the Aruppi project facilitates seamless communication between users and the underlying system, enhancing flexibility and scalability for future developments and collaborative efforts.