DMH

ShowTracker
Software Architecture Document

Version 1.3

Revision History

Date	Version	Description	Author
09/04/2024	1.0	Starting Section 1	Carlos Mbendera
10/04/2024	1.1	Section 5	Blake Ebner
11/04/2024	1.2	Section 3	Cole DuBois
11/04/2024	1.2	Section 2	Kenji Craig
11/04/2024	1.2	Section 4	John Tran
14/04/2024	1.3	Final Edits	Carlos Mbendera

Table of Contents

Software Architecture Document

1. Introduction

This Software Architecture Document (SAD) outlines the architectural framework and detailed design for ShowTracker, an innovative application aimed at enhancing how users monitor and manage their TV show interests across multiple streaming platforms. Key features detailed in this document include integration with external APIs, data management through Firebase, and user interactions via iOS and web interfaces. This blueprint helps ensure that the development aligns with the Software Requirements Specifications (SRS), guiding the team towards successful implementation.

1.1 Purpose

This document delivers a comprehensive architectural overview of ShowTracker, capturing key decisions and presenting them via various architectural views. It is designed for the project team, to guide implementation, and for stakeholders seeking detailed insights into the software's capabilities.

1.2 Scope

The scope of this document encompasses the complete architecture of the Show Tracker system, including error management, API interactions for TV show retrieval, data uploading to Firebase, and user data manipulation across iOS and web interfaces.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **iOS**: iPhone Operating System
- **API**: Application Programming Interface
- **UI**: User Interface
- **Firebase**: A cloud-based, NoSQL database platform designed for app development, offering real-time data storage and synchronization.

1.4 References

- EECS 348 Software Engineering Course Syllabus, Professor Hossein Saiedian, University of Kansas, Spring Semester 2024.
- Show Tracker Software Requirements Specifications, DMH Team Members, University of Kansas, Spring 2024

1.5 Overview

This document covers our software design architecture. It will touch on various items such as size, interface, and structure. It covers:

- **Architectural Representation:** Outlines views and models representing the system.
- **Architectural Goals and Constraints:** Details architectural requirements and objectives.
- **Logical View:** Describes system decomposition into packages and classes.
- **Interface Description:** Provides high-level details of key interfaces.
- **Quality:** Explains how the architecture supports quality attributes such as extensibility and reliability.

2. Architectural Representation

The software architecture for the Show Tracker system encompasses multiple views that collectively provide a comprehensive understanding of the system's structure and behavior. Each view focuses on specific aspects of the architecture, facilitating communication among stakeholders and guiding the development and evolution of the system.

1. Overview:

Description: Provides a high-level overview of the system architecture, highlighting its key components and their interactions.

Model Elements:

- Main components of the system (e.g., user interface, backend services, external integrations).
- High-level communication pathways between components.
- Deployment environment (e.g., iOS devices, backend servers, external APIs).

2. Logical View:

Description: Describes the logical organization of the system, focusing on the decomposition of functionality into modules or layers.

Model Elements:

- Modules representing major functional areas (e.g., user management, show tracking).
- Relationships and dependencies between modules.
- Interfaces exposed by each module for interaction with other parts of the system.

3. Process View:

Description: Shows the dynamic behavior of the system, illustrating how processes and threads interact to fulfill user requests.

Model Elements:

- Processes representing user interactions (e.g., browsing shows).
- Communication pathways between processes (e.g., API calls, database queries).

4. Physical View:

Description: Depicts the physical deployment of system components across hardware infrastructure.

Model Elements:

- Hardware nodes (e.g., iOS devices, servers hosting backend services).
- Network topology showing how components communicate over the network.

5. Development View:

Description: Focuses on the organization of the development effort, including the structure of source code, version control, and development environments.

Model Elements:

- Source code repositories and their organization (e.g., repositories for frontend, backend, testing).
- Development environments (e.g., local development setups, continuous integration servers)

3. Architectural Goals and Constraints

- **Developmental tools**: We will utilize Visual Studio Code for its versatility, Git and GitHub for version control, and Xcode for iOS specific development, alongside PyCharm, ensuring a robust development environment.
- **Design:** Object Oriented Programming - to facilitate clear modularity and extensibility of the software components.
- **Usability**: Develop an intuitive user interface to enhance user experience and facilitate easy navigation within the app.
- **Reuse**: Architect the codebase to promote code reuse and modularity for easier maintenance and scalability.
- **Off-the-shelf product:** Integrate third-party APIs or libraries for features such as content recommendation
- **Design and implementation strategy:** We plan to follow a waterfall development methodology, to ensure phases and requirements are met in a structured progression.

4. Logical View

4.1 Overview

The overall design of the program can be split into three subprograms/subsystems to perform in union to make up the program's required functionalities. These subprograms would be the User Interface, Database, and Data Fetcher.

The User Interface provides the user functionality to access and use the application. Movies and Shows are retrieved from the Database and displayed dynamically for the user to interact with. The Database stores the movies/shows that are fetched by the Data Fetcher which sends movies/shows via an API in the format of: title, genre, actors, and rating.

4.2 Architecturally Significant Design Modules or Packages

User Interface

retrieveShows()
displayShows()
buttonFunctions()

1. **retrieveShows()** - Allows the interface to retrieve shows from the database in a format in respect to the programming language.
2. **displayShows() -** Cisplays the shows in an user-friendly manner and allows users to interact with them.
3. **buttonFunctions()** - Consists of different functions for elements, such as buttons, for the user to track their shows or rate them.

Database

sendShows()
retrieveShows()
storeShows()

1. **sendShows**() - Sends the shows that are stored in storeShows().
2. **retrieveShows**() - Retrieves shows from the Data Fetcher.
3. **storeShows**() - Stores the shows that are from retrieveShows().

Data Fetcher

fetchShows()
sendShows()

1. **fetchShows**() - Fetches shows from an API and formats them in title, genre, actors, and rating.
2. **sendShows**() - Sends the shows over to the Database to be retrieved.

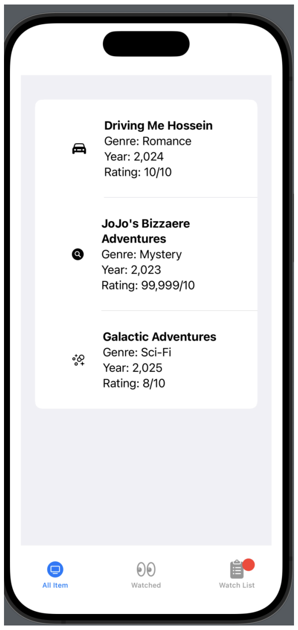
5. Interface Description

The interface for ShowTracker will be designed to be compact, neat, and visually appealing, ensuring a smooth user experience that encourages repeat usage. The interface will handle various inputs and outputs:

- **Valid Inputs:** User inputs will include TV show names, movie names, options to create new lists, delete existing ones, and reorder items within the lists.
- **Outputs**: The system will dynamically update the user's lists and display them following any input modifications.
- **Data Storage:** To enhance privacy and security, user-curated lists will be stored locally on devices rather than in the central database. This approach minimizes the storage of personal data on external servers, aligning with privacy best practices.

5.1 iOS User Interface MockUp

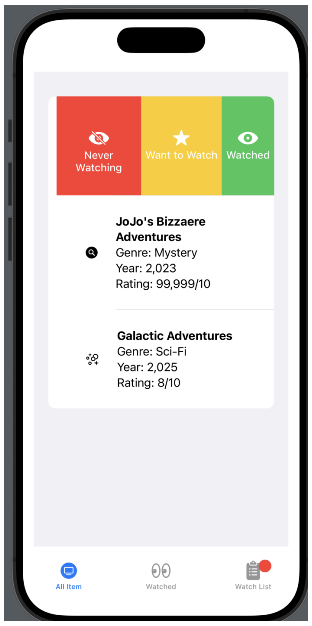
The following mockups have been created based on the detailed interface specifications outlined above.



Driving Me Hossein
Genre: Romance
Year: 2,024
Rating: 10/10

JoJo's Bizzaere Adventures
Genre: Mystery
Year: 2,023
Rating: 99,999/10

Galactic Adventures
Genre: Sci-Fi
Year: 2,025
Rating: 8/10



JoJo's Bizzaere Adventures
Genre: Mystery
Year: 2,023
Rating: 99,999/10

Galactic Adventures
Genre: Sci-Fi
Year: 2,025
Rating: 8/10

6. Quality

ShowTracker will be widely accessible and free to use, designed to operate reliably without crashes, thereby ensuring a positive user experience. The architecture supports high availability and robust performance:

- **Portability**: The application will be developed to run on multiple platforms, including iOS and web browsers, ensuring broad accessibility.
- **Security**: While user-specific data like curated lists will be stored locally to minimize privacy risks, essential security measures will be implemented to protect the application and its data integrity, particularly in interactions with external APIs and during data transmission.
- **Reliability**: System design includes error handling and redundancy to prevent crashes and ensure continuous operation.

Additional Considerations:

- **Cache and Local Storage Usage**: On web browsers, user session data will be temporarily stored in cache to enhance responsiveness and reduce data retrieval times. Similarly, iOS applications will utilize local storage to improve performance and user experience.