Module Interface Specification for Flick Picker

Team 7, 7eam
Talha Asif - asift
Jarrod Colwell - colwellj
Madhi Nagarajan - nagarajm
Andrew Carvalino - carvalia
Ali Tabar - sahraeia

January 18, 2023

1 Revision History

Date	Version	Notes
January 18	1.0	Added title, Module Decomposition table from Module Guide
January 18	2.0	Updated section 2, 3, 4

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/Flick-Picker/full-stack/blob/develop/docs/SRS/SRS.pdf

Term	Abbreviation
True	T
False	F

Contents

3 Introduction

The following document details the Module Interface Specifications for Flick Picker. Flick Picker is a web application that lets people select their preferences for watchable media and find recommendations of things to watch, being able to take multiple sets of preferences into account to find new media. Users can find new things to watch by themselves using only their preferences, or join user groups that find media based on what everyone likes in the group.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/Flick-Picker/full-stack.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Flick Picker.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
boolean	bool	either T or F

The specification of Flick Picker uses some common derived data types: sequences and strings. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Flick Picker also uses enumerated types, which are data types that hold a static set of constant values. In addition, Flick Picker uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

Flick Picker also uses its own custom data types, summarized in the following table.

Data Type	Notation	Description
Preferences	Preferences	a class that stores multiple boolean and enumerated values to keep track of user preferences such as genre or runtime
User	User	a class that stores user information such as an ID, name, and Preferences
Group	Group	a class that stores information about a user group, including an ID, owner, and user list.
Authentication	Authentication	a class that stores information about a user's login, connecting a user ID to the appropriate password

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Native Login Module Friends Module Groups Module Profile Module
Software Decision Module	Matching Algorithm Module OAuth Login Module API Module

Table 1: Module Hierarchy

6 MIS of [Module Name —SS]

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

6.1 Module

[Short name for the module —SS]

6.2 Uses

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

6.4 Semantics

6.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

6.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

6.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

6.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

6.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

7 Appendix

[Extra information if required —SS]