Module Guide for Flick Picker

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

April 6, 2023

1 Revision History

Date	Version	Notes
Jan 14	0.1	Adding modules and objects to store in databases - Talha
Jan 15	0.1	Adding descriptions to modules - Talha
April 5	1.0	Updated Custom Data Types, minor changes - Madhi
April 5	1.1	Added links to relevant documents - Ali

2 Reference Material

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

See Module Interface Specification at https://github.com/Flick-Picker/full-stack/blob/develop/docs/Design/SoftDetailedDes/MIS.pdf. See System Requirement Specifications at https://github.com/Flick-Picker/full-stack/blob/develop/docs/SRS/SRS.pdf.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Flick Picker	Application
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New Project Members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 6 summarizes the module decomposition that was constructed according to the likely changes. Section 7 specifies the connections between the software requirements and the modules. Section 5 provides detailed descriptions of native data types. Section 8 gives a detailed description of the modules. Section 9 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 10 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The `preferences` could be modified, depending on what the general response to the initial show preferences exist

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: Mouse, Output: File, Memory, and/or Screen)

UC2: The `user` and `group` data types will retain the information describe in Section 5

UC3: The API for movies and tv shows will stay the same

UC4: The API for animes will stay the same

5 Custom Data Types

Flick Picker uses standard data types, but a handful custom ones have to be created, defined below. All the data types are a key-value pair.

5.1 Preferences

Key Name	Value Type	Description
`anime`	bool	Allow animes as recommendations
`movie`	bool	Allow movies as recommendations
`tv`	bool	Allow TV shows as recommendations
`series`	bool	Allow a show with multiple seasons
`genres`	List⟨enum⟩	Which type of media is allowed
`disliked genres`	List⟨enum⟩	Which type of media is not allowed
`runtime`	enum	Length of show allowed
`rating`	enum	Minimum medias' rating allowed
`popularity`	enum	User weighting for medias' popularity
`recent release`	enum	User weighting for medias' recent release date
`release range`	String	Release date range of media allowed

5.2 User

Key Name	Value Type	Description
`id`	int	Unique user identifier
`name`	String	Name of the user
`email`	String	The email user created their account with
`friends`	List⟨int⟩	Friends the user has
`preference`	Preferences	Show constraints user has selected
`groupsJoined`	List⟨String⟩	Groups that user has joined
`groupsOwned`	List⟨String⟩	Groups that user owns
`currentSession`	Session	Current Voting Session for user

5.3 Group

Key Name	Value Type	Description
`id`	int	Unique group identifier
`owner`	int	Group owner identifier
`users`	List⟨int⟩	List of users in the group
`preferences`	List⟨Preferences⟩	All preferences of users in the group
`currentSession`	Session	Current Voting Session for group
`doneSessions`	List⟨Session⟩	All completed Voting Sessions for group

5.4 Session

Key Name	Value Type	Description
`id`	int	Unique voting session identifier
`isGroup`	bool	Identifies whether session is for Individual or Group
`user`	User	User reference, if it is an Individual session
`group`	Group ⟨int⟩	Group reference, if it is an Individual session
`recommendations`	List⟨Recommendation⟩	Media recommendations for session

5.5 FriendInvite

Key Name	Value Type	Description
`id`	int	Unique invite identifier
`senderUser`	User	User reference for sender of invite
`requestedUser`	User	User reference for invite receiver
`isAccepted`	bool ⟨int⟩	Identifies if invite has been accepted

5.6 GroupInvite

Key Name	Value Type	Description
`id`	int	Unique invite identifier
`senderUser`	User	User reference for sender of invite
`requestedUser`	User	User reference for invite receiver
`group`	Group	Group reference
`isAccepted`	bool	Identifies if invite has been accepted

5.7 Recommendation

Key Name	Value Type	Description
`name`	String	Name of media
`algorithmRating`	int	Rating determined by Recommendation Algorithm
`userVotes`	List⟨Vote⟩	User Votes submitted
`voteRating`	int	Total Rating from votes submitted

5.8 Vote

Key Name	Value Type	Description
`user`	User	User that submitted the vote
`vote`	enum	Vote given by user
`time`	DateTime	Time the vote was submitted

5.9 Authentication

Key Name	Value Type	Description
`id`	int	Unique user identifier, linked to `User`
`email`	int	Email identifier, linked to `User`
`password`	Hash	Hashed password stored

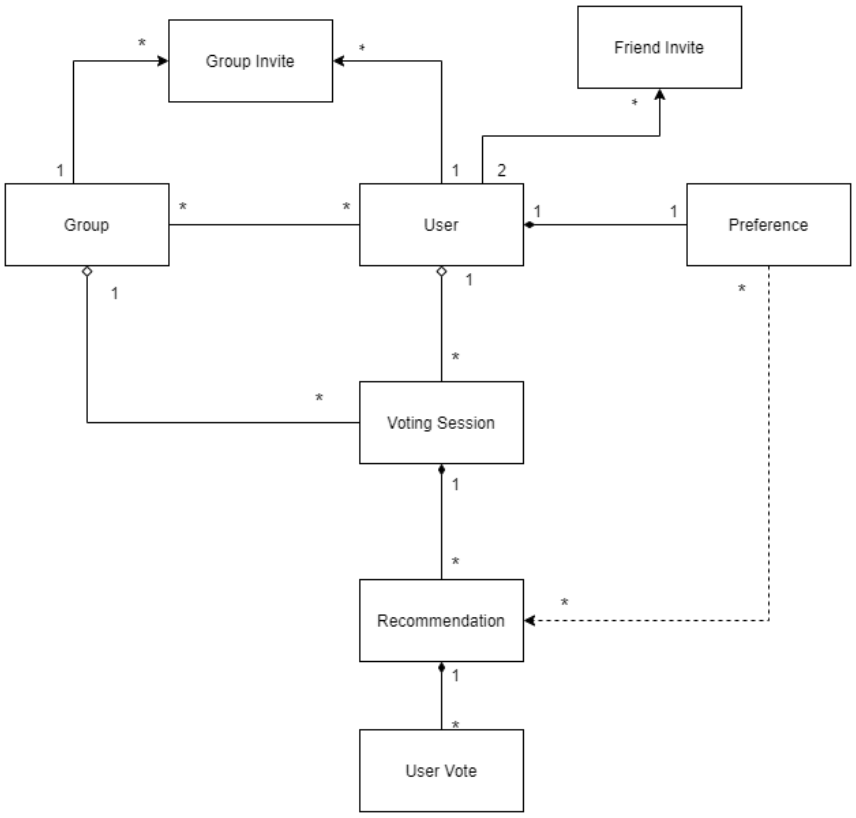

Figure 1: UML Diagram across data models

6 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Behaviour-Hiding Module

M3: Native Login Module

M4: Friends Module

M5: Groups Module

M6: Profile Module

M7: Software Decision Module

M8: Matching Algorithm Module

M9: OAuth Login Module

M10: API Module

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

7 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

8 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Flick Picker* means the module will be implemented by the Flick Picker software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

8.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

8.2 Behaviour-Hiding Module (M2)

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

8.2.1 Native Login Module (M3)

Secrets: Hides authentication data from the rest of the software, isolating the authentication as the rest of the software has no use for a user's email or password

Services: Native log in to the application

Implemented By: Flick Picker

Type of Module: Record

8.2.2 Friends Module (M4)

Secrets: The methodology and data related to adding/deleting friends

Services: Allows a user to add and delete friend's but sharing their nickname or email address with one another

Implemented By: Flick Picker

Type of Module: Record

8.2.3 Groups Module (M5)

Secrets: The methodology and data used to create a group with other friends

Services: Allows the user to create, join, leave, and delete individual groups, invite friends, and receive recommendations on a show to watch based on the groups' preferences

Implemented By: Flick Picker

Type of Module: Abstract Object

8.2.4 Profile Module (M6)

Secrets: The methodology and data for an individual user

Services: Allows the user to modify their preferences and general profile

Implemented By: Flick Picker

Type of Module: Abstract Object

8.3 Software Decision Module (M7)

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

8.3.1 Matching Algorithm Module (M8)

Secrets: The methodology and data on how recommendations are created

Services: Converts the input data to a list of recommendations for a `group`

Implemented By: Flick Picker

Type of Module: Abstract Object

8.3.2 OAuth Login Module (M9)

Secrets: OAuthentication login details, through Google

Services: Allows the user to sign up and login through an OAuth provider

Implemented By: Google OAuth services paired with Flick Picker

Type of Module: Library

8.3.3 API Module (M10)

Secrets: Data behind fetching all the shows and organizing them

Services: Allows Flick Picker to return a set of shows that match user preferences

Implemented By: OMDb API and Jikan Anime API

Type of Module: Library

9 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M3
R2	M9
R3	M3, M9
R4	M6
R5	M6
R6	M5
R7	M4, M5
R8	M4
R9	M5, M8, M10
R10	M5, M6
R11	M5, M6

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M6

Table 3: Trace Between Anticipated Changes and Modules

10 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 2 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

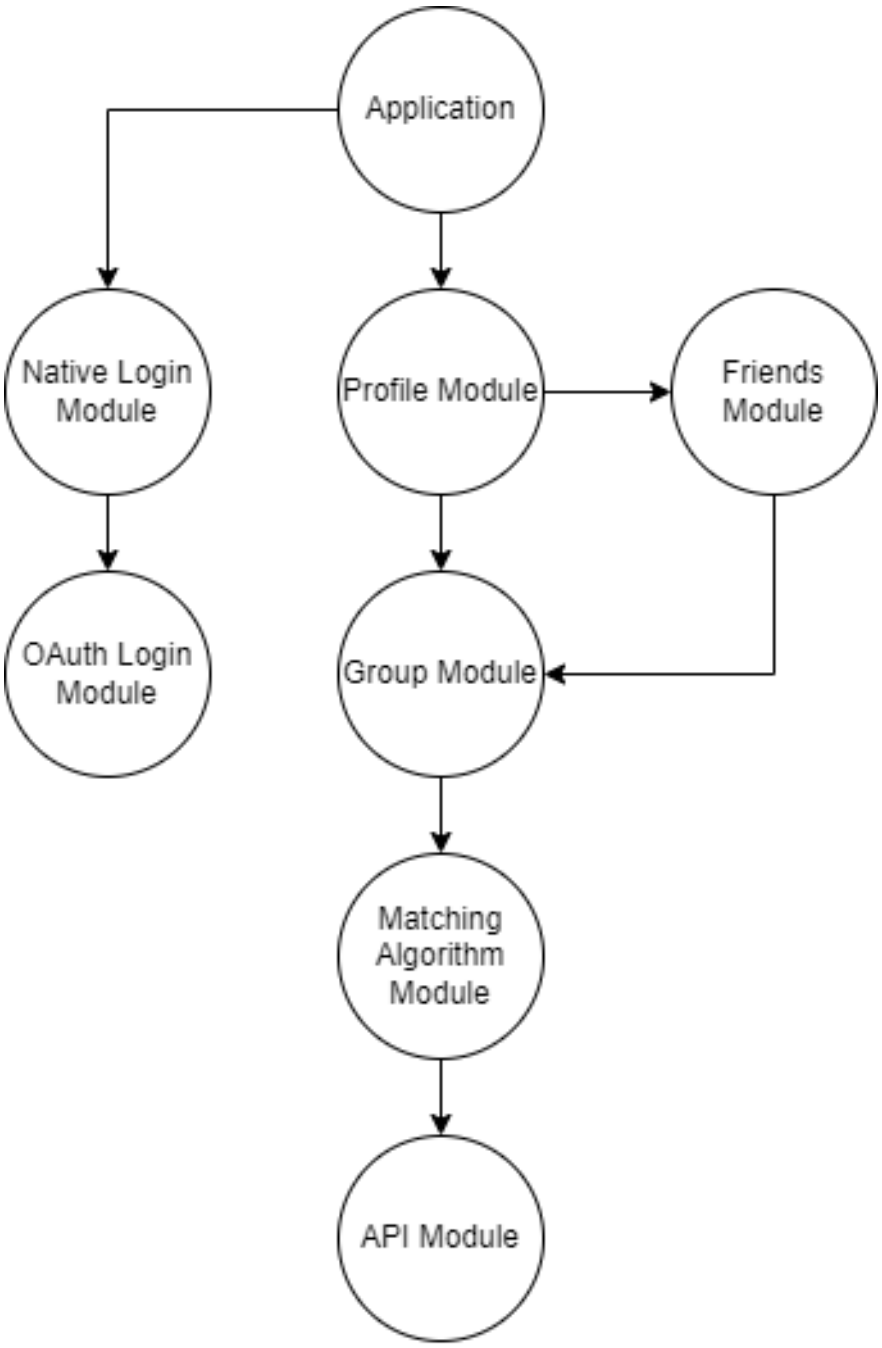

Figure 2: Use hierarchy among modules

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.