# SE 3XA3: Module Guide
# MovieGuide Application

Team 35, PGH Software Solutions
Pratyush Bhandari, bhandarp
Gazenfar Syed, syedg1
Hamid Ghasemi, ghasemih

November 9, 2018

# Contents

# List of Tables



# List of Figures



Table 1: **Revision History**

| **Date** | **Version** | **Notes** |
|---|---|---|
| Nov 8 | 1.0 | Parts 1 and 2 |
| Date 2 | 1.1 | Notes |

# 1 Introduction

## 1.1 Overview

The MovieGuide project is a re-implementation of an open-source movie reviews Android application which allows users to view ratings, description, and trailers for any movie within the API database.

## 1.2 Context

This is the Module Guide Design Documentation for the MovieGuide re-implementation. Readers of this document include, but are not limited to:

- New project members: This document gives an insight into the design structure of the project. It contains information about the module hierarchy along with the modular decomposition, so that new members can quickly learn how each module is integrated into the program.
- Maintainers: This document also outlines areas where changes are anticipated which will be helpful to the maintainers as they will be able to quickly identify areas of the application that need to be improved. Moreover, maintainers can utilize the breakdown of the modules to better understand the project structure so that they know which modules to update.
- Designers/Developers: This module guide describes the design choices that were made when developing the program. It contains an overview of the relationships between all modules which will aid the designers and developers in determining whether the product design is satisfiable for the software requirements.

## 1.3 Design Principles

The design principles utilized for our design choices include information hiding and encapsulation. Information hiding keeps certain aspects of the code a secret from the other modules which reduces the complexity of the program and protects it from extensive modifications that could potentially harm the implementation. Encapsulation combines data and methods allowing internal data to be accessed by public methods. Furthermore, modules were implemented with the principles of high cohesion and low coupling to ensure that the modules are related but not heavily dependent on each other.

## 1.4 Document Structure

This document is organized as specified below:

- Section 2: anticipated changes and unlikely changes for the current implementation

- Section 3: outlines the module hierarchy
- Section 4: summarizes the relationships between modules and software requirements
- Section 5: explains the secret and service/responsibility of each module
- Section 6: relates modules to requirements and anticipated changes
- Section 7: describes the uses relations between modules

# 2 Anticipated and Unlikely Changes

## 2.1 Anticipated Changes

- AC1: The API used to retrieve information about movies
- AC2: The format of the input data
- AC3: The format of the output data
- AC4: The User Interface utilized to navigate through the application
- AC5: How movie data is stored locally on the application (ie. improving performance)

## 2.2 Unlikely Changes

- UC1: The input/output devices (system is designed to run on Android devices)
- UC2: The purpose of the software to provide ratings and descriptions about movies
- UC3: The storage method of output data (system assumes there is sufficient storage space on the device, so output data will be stored locally)

# 3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

**M1:** Hardware-Hiding Module

**M2:** Input Module

**M3:** Search Movies Module

**M4:** Sort Movies Module

**M5:** Output Module

**M6:** Searching Algorithm Module

**M7:** List Object Module

**M8:** retrofitURL Module

| **Level 1** | **Level 2** |
|---|---|
| Hardware-Hiding Module | N/A |
| Behaviour-Hiding Module | Input Module<br>Search Movies Module<br>Sort Movies Module<br>Output Module |
| Software Decision Module | Searching Algorithm Module<br>List Object Module<br>retrofitURL Module |

Table 2: Module Hierarchy

# 4 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 3. With regards to the main functional requirement of providing a synopsis of every movie, this requirement is fulfilled through the output module, which will provide the service of displaying a specific movie's information when it is clicked on by the user. The searching module, in tandem with the searching algorithm module, will fulfill the functional requirement of having the program list movies based on a specific user query. Lastly, the SortMovies module along with the retrofitURL module will fulfill the functional requirement of allowing the user to sort movies by name, rating or genre.

# 5 Module Decomposition

## 5.1 Hardware Hiding Module

**Secret**: 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**: Android OS

## 5.2 Behaviour Hiding Module

**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**: MovieGuide

### 5.2.1 Input Module

**Secret**: Inputs
**Service**: Allows user to scroll, search and click on the list of movies being displayed on the screen.
**Implemented By**: MovieGuide

### 5.2.2 Search Movies Module

**Secret**: Search Query
**Service**: Allows user to enter a search query in the search bar which is then passed as an endpoint to the retrofitURL Module.
**Implemented By**: MovieGuide

### 5.2.3 Sort Movies Module

**Secret**: Movies
**Service**: Allows user to select different options for sorting the list of movies in the database.
**Implemented By**: MovieGuide

### 5.2.4 Output Module

**Secret**: MovieInfoPage
**Service**: Displays movie information about a specific movie that the user clicked on from the displayed list of movies.
**Implemented By**: MovieGuide

## 5.3 Software Decision Module

**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**: -

### 5.3.1 Searching Algorithm Module

**Secret**: Search Algorithm
**Service**: Uses an algorithm to efficiently find movie matching the search query.
**Implemented By**: -

### 5.3.2 List Object Module

**Secret**: List Object
**Service**: Stores the list of movies in a ListView object.
**Implemented By**: -

### 5.3.3 retrofitURL Module

**Secret**: API URL
**Service**: Stores the URL of the API that GET and POST requests will be sent to.
**Implemented By**: -

# 6 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 | M2, M??, M??, M?? |
| R2 | M??, M?? |
| R3 | M?? |
| R4 | M??, M?? |
| R5 | M??, M??, M??, M??, M??, M?? |
| R6 | M??, M??, M??, M??, M??, M?? |
| R7 | M??, M??, M??, M??, M?? |
| R8 | M??, M??, M??, M??, M?? |
| R9 | M?? |
| R10 | M??, M??, M?? |
| R11 | M??, M??, M??, M?? |

Table 3: Trace Between Requirements and Modules

| AC | Modules |
|---|---|
| AC?? | M2 |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |

Table 4: Trace Between Anticipated Changes and Modules

# 7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** 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 1 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 1: Use hierarchy among modules