Module Guide for Housemates

Team #9, Housemates
Justin Dang - dangj15
Harris Hamid - hamidh1
Fady Morcos - morcof2
Rizwan Ahsan - ahsanm7
Sheikh Afsar - afsars

January 17, 2024

1 Revision History

Date	**Version**	**Notes**
January 17, 2024	1.0	Revision 0

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SRS	Software Requirements Specification
Housemates	Explanation of program name
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 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 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 9 describes the use relation between modules. Section 10 describes the timeline to complete revision 0 of Housemates. Section 11 contains the reflection for the design stage of Housemates.

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 format of the initial input data.

AC3: The algorithms used for account verification.

AC4: The algorithms used to determine how bills are split between users.

AC5: The algorithms used in the task management system.

AC6: The algorithms used in the scheduling system.

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: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: Changes in the use of the interface design module.

UC3: Changes in the use of the database module.

UC4: Changes in the network libraries used.

5 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: Task Management Module

M3: Bill Management Module

M4: Scheduling Module

M5: Account Module

M6: Interface Design Module

M7: Cryptography Module

M8: Database Interface Module

M9: Network Interface Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Task Management Module Bill Management Module Scheduling Module Account Module Interface Design Module
Software Decision Module	Cryptography Module Database Interface Module Database Module Network Interface Module

Table 1: Module Hierarchy

6 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.

7 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. *Housemates* means the module will be implemented by the Housemates 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.

7.1 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

7.2 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: –

7.2.1 Task Management Module (M2)

Secrets: The format and structure of the input data. The algorithms used for the task management functionality.

Services: Provides the task management functionality of the application. Converts the input data into the data structure used by the database interface module.

Implemented By: Housemates

Type of Module: Library

7.2.2 Bill Management Module (M3)

Secrets: The format and structure of the input data. The algorithms used for the bill splitting functionality.

Services: Provides the bill management and bill splitting functionality of the application. Converts the input data into the data structure used by the database interface module.

Implemented By: Housemates

Type of Module: Library

7.2.3 Scheduling Module (M4)

Secrets: The format and structure of the input data. The algorithms used for the scheduling functionality.

Services: Provides the scheduling functionality of the application. Converts the input data into the data structure used by the database interface module.

Implemented By: Housemates

Type of Module: Library

7.2.4 Account Module (M5)

Secrets: The format and structure of the input data. The algorithms used for the account verification.

Services: Provides the account functionality of the application. Converts the input data into the data structure used by the database interface module.

Implemented By: Housemates

Type of Module: Library

7.2.5 Interface Design Module (M6)

Secrets: The specific algorithms used in the user interface.

Services: The user interface of the application.

Implemented By: React, Housemates

7.3 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: –

7.3.1 Cryptography Module (M7)

Secrets: The specific details of the cryptographic algorithms.

Services: Cryptographic algorithms used in the application.

Implemented By: Crypto (node.js)

7.3.2 Database Interface Module (M8)

Secrets: The specific details of connecting to database.

Services: Object Modelling, Connecting to database.

Implemented By: Mongoose, Housemates

7.3.3 Network Interface Module (M9)

Secrets: The specific details of implementing RESTful API.

Services: Allows for creation of RESTful API to use in application and connecting to that API.

Implemented By: Axios, Express

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements as seen in (Dang et al., 2023) and between the modules and the anticipated changes.

Req.	**Modules**
TM1-5	M2
BM1-7	M3
SS1-3	M4
AS1-5	M5
LF-A1	M2, M3, M4, M5, M6
UH-E1	M2, M3, M4, M5, M6
UH-P1	M2, M3, M4, M5, M6
UH-L1	M2, M3, M4, M5, M6
UH-A1	M2, M3, M4, M5, M6
P-SL1	M2, M3, M4, M5, M6, M7, M8, M9
P-PA1	M3
P-RFT1	M2, M3, M4, M5, M8, M9
P-C1	M2, M3, M4, M5, M8, M9
OE-PE1	M1
S-A1	M2, M3, M4, M5, M7
S-IN1	M2, M3, M4, M5, M8, M7
S-P1	M6

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M2, M3, M4, M5
AC3	M5
AC4	M3
AC5	M2
AC6	M4

Table 3: Trace Between Anticipated Changes and Modules

9 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 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.

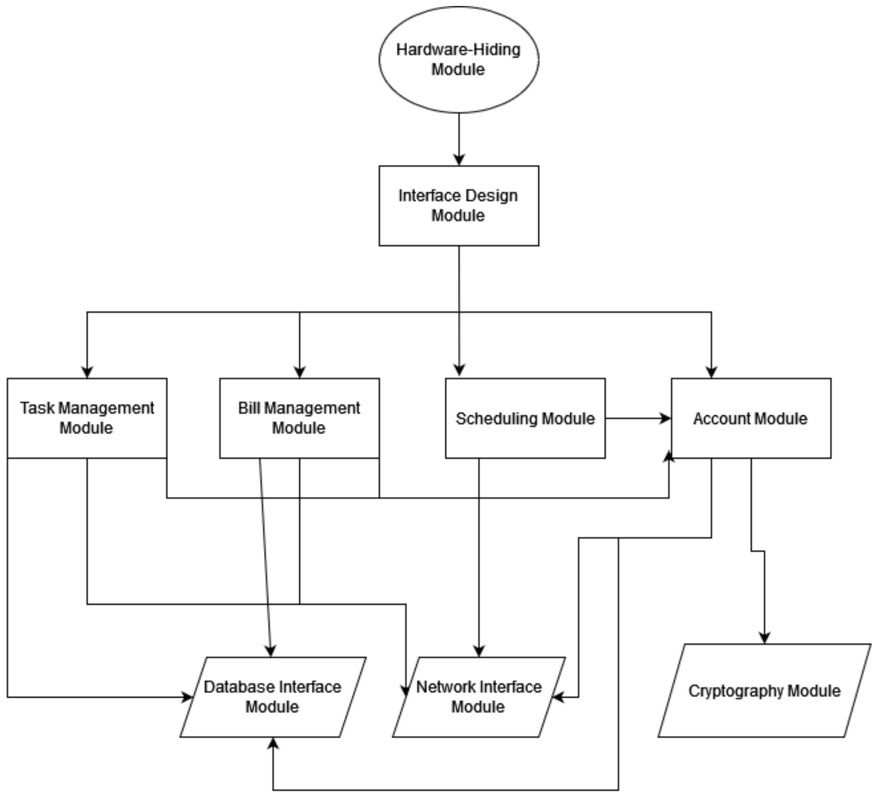

Figure 1: Use hierarchy among modules

10 Timeline for Revision 0

Modules	Rev 0 Implementation Tasks	Deadline	Team Members Responsible
Hardware-Hiding Module	Provided by OS	N/A	N/A
Task Management Module	Implement all the functions included in the module	February 4, 2024	Justin, Harris
Bill Management Module	Implement all the functions included in the module	February 4, 2024	Harris, Fardeen
Scheduling Module	Implement all the functions included in the module	February 4, 2024	Rizwan, Fady
Account Module	Implement all the functions included in the module	February 1, 2024	Fardeen, Rizwan
Interface Design Module	Implement all the functions included in the module	February 4, 2024	Fady, Justin
Database Interface Module	Set up MongoDB Schema and Mongoose objects	February 1, 2024	Fardeen, Rizwan
Testing and Verification	Unit testing of functions in modules	February 6, 2024	Everyone
Network Interface Module	Set up API Routes as required	February 4, 2024	Everyone
Cryptography Module	Provided by external libraries	N/A	N/A

Table 4: Timeline

Though team members are designated to specific modules and tasks, it's crucial to understand that this doesn't create rigid boundaries. The assignments highlight primary responsibilities and leadership roles in developing particular components. While certain individuals lead the charge for specific modules, all team members actively contribute, albeit with varying degrees of impact. The designated leads play a central role in shaping the direction of development for their respective areas, encouraging a collaborative approach within the team.

11 Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design. Please answer the following questions:

1. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

   The main limitations of the Housemates application right now is the lack of integration that Housemates has with other software at the current moment. For example, the scheduling feature of the Housemates application could be integrated with popular calendar services like Google calendar. With unlimited resources the Housemates application would have these types of integrations with other software, which would make it more useful to our stakeholders.

2. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select documented design? (LO_Explores)

   The first design solution that we considered for this Housemates application was creating a native mobile version of the application using Kotlin. This would have the advantage of having better performance for users on mobile devices as well as having a more adaptable UI for different mobile devices. The main reason that we went with a web app using React is that it allows stakeholders to access the Housemates application on a variety of different devices (desktop and mobile). Additionally, we already have familiarity with creating webapps, which means that we can focus more on specifics of Housemates rather than learning Kotlin.

References

Justin Dang, Harris Hamid, Fady Morcos, Rizwan Ahsan, and Sheikh Afsar. Software requirements specification for software engineering: Housemates, 2023. URL https://github.com/DangJustin/CapstoneProject/blob/main/docs/SRS-Volere/SRS.pdf.

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.