Module Guide for Grocery Spending Tracker

Team 1, JARS
Jason Nam
Allan Fang
Ryan Yeh
Sawyer Tang

April 1, 2024

1 Revision History

Date	Version	Notes
17/01/2024	1.0	Initial version of Module Guide

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
SC	Scientific Computing
SRS	Software Requirements Specification
Grocery Spending Tracker	The name of the application being built
UC	Unlikely Change
CRUD	Create, Read, Update, Delete
OCR	Optical Character Recognition

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.

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 receipt data.

AC3: The structure of the data inputted to the server.

AC4: The structure of the data returned from the server.

AC5: The styling and formatting of UI components and user input.

AC6: The table structure of the databases.

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: The user interface will be on a mobile device that has a camera for primary input.

UC2: The server will run on NodeJS.

UC3: The front end mobile app will be built on Flutter.

UC4: The database will be PostgreSQL.

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: Receipt Vision Module (OCR)

M3: Receipt Extraction Module

M4: Location Management Module

M5: User Analytics Module

M6: Users Module

M7: Authentication Module

M8: Recommendation Engine

M9: Classification Engine

M10: Database Driver Module

Note that 1 is a commonly used module and is already implemented by the operating system. It will not be reimplemented.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Receipt Vision Module (OCR) Receipt Extraction Module Location Management Module User Analytics Module Users Module Authentication Module
Software Decision Module	Recommendation Engine Classification Engine Database Driver 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. *Grocery Spending Tracker* means the module will be implemented by the Grocery Spending Tracker 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

7.2.1 Vision Module (M2)

Secrets: The dependencies and algorithms required to facilitate text recognition of receipts.

Services: Provides methods related to usage of the device camera and extraction of the text from captured receipts.

Implemented By: Grocery Spending Tracker

Type of Module: Library

7.2.2 Receipt Extraction Module (M3)

Secrets: The algorithms required to extract relevant information from the scanned receipt text.

Services: Isolates necessary information from scanned receipt text, organizes it, and sends to other modules.

Implemented By: Grocery Spending Tracker

Type of Module: Library

7.2.3 Location Management Module (M4)

Secrets: The dependencies and algorithms for determining location-based relevance.

Services: Determines relevance of information based on location data.

Implemented By: Grocery Spending Tracker

Type of Module: Library

7.2.4 User Analytics Module (M5)

Secrets: The data structures and algorithms for compiling and computing user purchasing analytics.

Services: Provide analytics and feedback on user purchasing behaviour, offering insight on recent and future purchasing.

Implemented By: Grocery Spending Tracker

Type of Module: Library

7.2.5 Users Module (M6)

Secrets: The methods and data structures that support all user operations.

Services: Produce CRUD operations for operations related to users.

Implemented By: Grocery Spending Tracker

Type of Module: Library

7.2.6 Authentication Module (M7)

Secrets: The data structures that hold all account data such as username, password, and email.

Services: Facilitates secure user login and authentication.

Implemented By: Grocery Spending Tracker

Type of Module: Library

7.3 Software Decision Module

7.3.1 Recommendation Engine Module (M8)

Secrets: Language models and query/filtering algorithms required to generate best match output.

Services: Provides methods related to determining recommended items depending on input or provided data.

Implemented By: Grocery Spending Tracker

Type of Module: Library

7.3.2 Classification Engine Module (M9)

Secrets: Configuration models required to classify items from data that has noise.

Services: Provides methods related to interpreting and identifying recognized items from receipt data.

Implemented By: Grocery Spending Tracker

Type of Module: Library

7.3.3 Database Driver Module (M10)

Secrets: The data structures for representing the database schema and providing interface from the other modules.

Services: Facilitates communication between the database and the other modules.

Implemented By: Grocery Spending Tracker, PostgreSQL

Type of Module: Library

8 Traceability Matrix

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

Req.	Modules
FR1	M7
FR2	M7, M10
FR3	M2

FR4	M4, M6, M10
FR5	M6, M10
FR6	M6, M10
FR7	M9, M10
FR8	M8, M10
FR9	M8, M10
FR10	M5, M10
FR11	M3
FR12	M9, M10
AR1	M2, M3, M4, M5, M6, M7, M8
AR2	M2, M3, M4, M5, M6, M7, M8
AR3	M2, M3, M4, M5, M6, M7, M8
SR1	M2, M3, M4, M5, M6, M7, M8
SR2	M2, M3, M4, M5, M6, M7, M8
SR3	M2, M3, M4, M5, M6, M7, M8
EUR1	M2, M3, M4, M5, M6, M7, M8
EUR2	M2, M3, M4, M5, M6, M7, M8
EUR3	M2, M3, M4, M5, M6, M7, M8
PIR1	M2, M3, M4, M5, M6, M7, M8
PIR2	M2, M3, M4, M5, M6, M7, M8
PIR3	M6
LR1	M2, M3, M4, M5, M6, M7, M8
LR2	M2, M3, M4, M5, M6, M7, M8
UPR1	M2, M3, M4, M5, M6, M7, M8
UPR2	M2, M3, M4, M5, M6, M7, M8
ACR1	M2, M3, M4, M5, M6, M7, M8
ACR2	M2, M3, M4, M5, M6, M7, M8
SLR1	M2, M3, M4, M5, M6, M7, M8, M9, M10
SLR2	M2, M3, M4, M5, M6, M7, M8
PAR1	M2, M3, M4,M5, M6, M7
PAR2	M5
RFR1	M2, M3, M4,M5, M6, M7
CR1	M2, M3, M4, M5, M6, M7, M8, M9
EPER1	M2, M3, M4, M5, M6, M7, M8, M9
EPER2	M2, M3, M4, M5, M6, M7, M8, M9

EPER3	M2, M3, M4, M5, M6, M7, M8, M9
WER1	M2
WER2	M3, M4, M5, M6, M7, M8, M9
IASR1	M2, M3, M4, M5, M6, M7, M8, M9
IASR2	M2, M3, M4, M5, M6, M7, M8, M9
SPR1	M2, M3, M4, M5, M6, M7, M8
SPR2	M2, M3, M4, M5, M6, M7, M8, M9
APR1	M2, M3, M4, M5, M6, M7, M8, M9
APR2	M2, M3, M4, M5, M6, M7, M8, M9, M10
ACR1	M2, M3, M4, M5, M6, M7, M8, M9, M10
INR1	M2, M3, M4, M5, M6, M7, M8, M9, M10
INR2	M2, M3, M4, M5, M6, M7, M8, M9, M10
PRR1	M2, M3, M4, M5, M6, M7, M8, M9, M10
PRR2	M2, M3, M4, M5, M6, M7, M8, M9, M10
AUR1	M2, M3, M4, M5, M6, M7, M8, M9, M10
CUR1	M2, M3, M4, M5, M6, M7, M8, M9, M10
CUR2	M2, M3, M4, M5, M6, M7, M8, M9, M10
CUR3	M2, M3, M4, M5, M6, M7, M8, M9, M10
LR1	M2, M3, M4, M5, M6, M7, M8, M9, M10
LR2	M2, M3, M4, M5, M6, M7, M8, M9, M10

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M2, M3
AC3	M3, M4, M6, M7, M8, M9
AC4	M5, M4, M6, M7, M8, M9
AC5	M2, M3, M4, M5, M6, M7, M8
AC6	M10

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.

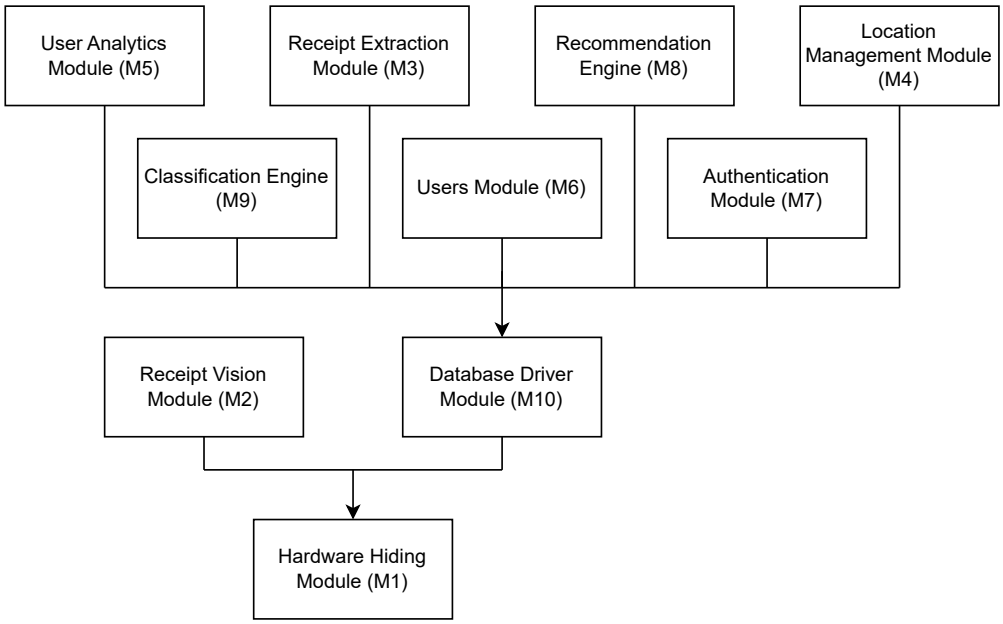


Figure 1: Use hierarchy among modules

10 Timeline

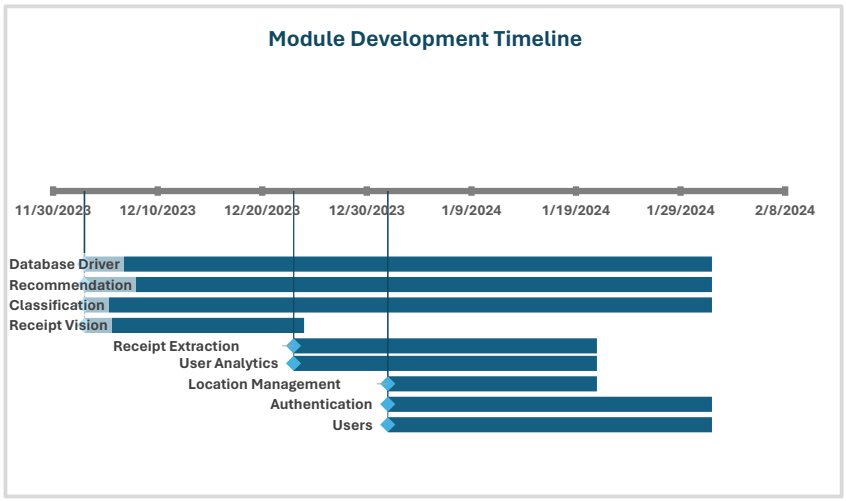
10.1 Module Development Timeline

Development of modules started in December 2023 and will continue into January 2024. Table 4 shows the different modules being developed as well as the developer(s) who will be taking the lead on it. This means they will be responsible for the primary planning and coding of the module. Developers not listed on a module will still take part in development but on a more adhoc basis or as assistance is needed. The timeline is also shown graphically in Figure 2.

Table 4: Module Development Timeline

Module Name	**Development Start/End Date**	**Developer(s)**
Database Driver Module	Dec. 3 2023 — Jan. 31, 2024	Jason Nam, Sawyer Tang, Ryan Yeh, Allan Fang
Recommendation Engine	Dec. 3 2023 — Jan. 31, 2024	Jason Nam
Classification Engine	Dec. 3, 2023 — Jan 31, 2024	Jason Nam
Receipt Vision Module	Dec. 3 2023 — Dec. 23, 2023	Ryan Yeh
Receipt Extraction Module	Dec. 23, 2023 — Jan. 20, 2024	Ryan Yeh
User Analytics Module	Dec. 23, 2023 — Jan. 20, 2024	Allan Fang
Location Management Module	Jan. 1, 2024 — Jan. 20, 2024	Allan Fang
Authentication Module	Jan. 1, 2024 — Jan. 31, 2024	Sawyer Tang
Users Module	Jan. 1, 2024 — Jan. 31, 2024	Sawyer Tang

Figure 2: Graphical Module Development Timeline



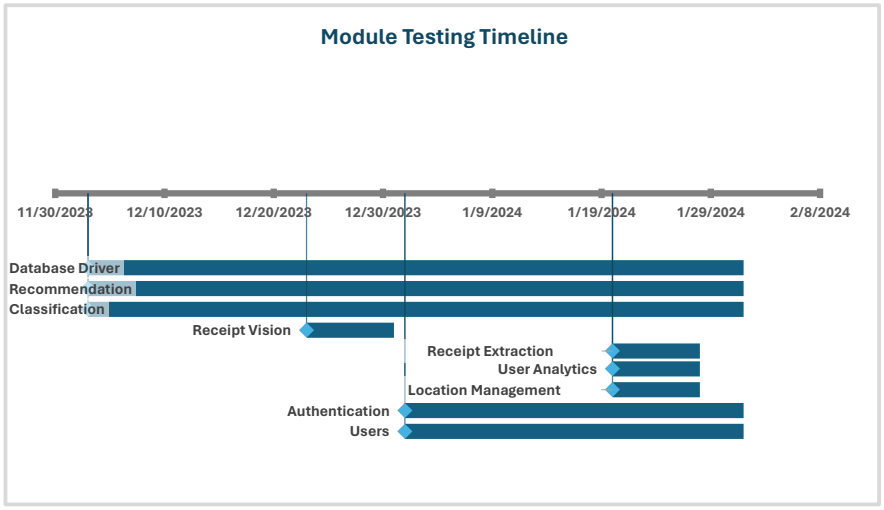
10.2 Module Testing Timeline

Testing of modules also takes place during the development window set in the previous section. In general, testing will involve basic manual testing of the application features to build confidence for the Revision 0 Demo. For modules whose development end date is prior to January 31, testing will take place during the following week. For the other modules, testing will largely take place concurrently with development and small unit tests will be performed as the modules are created. The developers in charge of testing will generally involve the main developer as written in Table 4 in addition to another developer to help test the module with a fresher perspective. The details can be found in Table 5 and graphically shown in Figure 3.

Table 5: Module Testing Timeline

Module Name	**Testing Start/End Date**	**Developer(s)**
Database Driver Module	Dec. 3 2023 — Jan. 31, 2024	Jason Nam, Sawyer Tang, Ryan Yeh, Allan Fang
Recommendation Engine	Dec. 3 2023 — Jan. 31, 2024	Jason Nam, Sawyer Tang
Classification Engine	Dec. 3, 2023 — Jan 31, 2024	Jason Nam, Ryan Yeh
Receipt Vision Module	Dec. 23, 2023 — Dec 30, 2023	Ryan Yeh, Allan Fang
Receipt Extraction Module	Jan. 20, 2023 — Jan. 27, 2024	Ryan Yeh, Jason Nam
User Analytics Module	Jan. 20, 2024 — Jan. 27, 2024	Allan Fang, Sawyer Tang
Location Management Module	Jan. 20, 2024 — Jan. 27, 2024	Allan Fang, Ryan Yeh
Authentication Module	Jan. 1, 2024 — Jan. 31, 2024	Sawyer Tang, Allan Fang
Users Module	Jan. 1, 2024 — Jan. 31, 2024	Sawyer Tang, Jason Nam

Figure 3: Graphical Module Testing Timeline



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)**

   Currently, there are two primary limitations with our project: time and money. In terms of time, it generally is not feasible to implement everything we would have liked to implement. Many features had to be scoped down to meet the time constraint we had and many other features had to be cut altogether. Some of the features and ideas we had discussed are:

   - Providing support for all grocery stores in Canada
   - Implementing a way to update our database with flyer data and deals from various grocery stores in Canada on a regular basis
   - Putting more emphasis in application security and general security practices
   - Further optimizations in terms of performance
   - More emphasis and support for accessibility settings
   - Adding receipt barcode support to potentially fetch more accurate data and give users an alternative input method

   We believe these features would be nice to include in our application and would greatly improve its quality, however many of them are not feasible given the limited development time frame we have.

   Additionally, monetary limitations have led to further constraints on the development of our application. Given unlimited money, improvements could be made to our database and server. In particular, a greater capacity database and higher bandwidth server would have been nice in terms of scalability of our project. With the additional funds, we would be able to create an application that better aligns with our goal by allowing more users to use the application, thus helping more people save money on groceries.

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)**

   One design solution we considered early on in the project was implementing our own OCR. We believed that having our own OCR pipeline to leverage might lead to better results over a general-purpose OCR. Since we only needed to read receipt data, we

thought we could optimize it for this use case at the cost of using up our development time. We also thought it would be nice to have more transparency and control over the OCR when compared to off-the-shelf solutions. Despite our efforts however, our implementation ended up being far slower in comparison to existing solutions and more resource-intensive. Additionally, the OCR implementation ended up taking far longer than we anticipated so due to the time constraint among the other mentioned issues, it was not feasible to keep proceeding as we were. As a result, we ended up looking to use an existing solution in the form of *Google MLKit* which has solved all of the previously mentioned problems at the cost of having less control compared to a custom solution.

Another design solution we considered involved where we would be processing the OCR data. We were unable to use the information picked up by the OCR on its own due to noise and unnecessary information being present so we had to extract only what was necessary. We had considered doing this on the backend in order to reduce the amount of resources the application would consume on the user's device. This would have come at the cost of being more intensive on the server instead. Considering our monetary limitations and seeing the server as a more limited resource however, we made the decision to move the processing to the frontend. This also allowed us to send relatively clean data to the backend allowing for easier to handle and more consistent data.

Overall, most of the design decisions that we selected involved identifying constraints in our project and choosing the best solution that met our requirements. As a result, we ended up with our current design which we believe offers the best balance between meeting our requirements as well as being feasible with our current resources.

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.