Module Guide for SyncMaster

Team 15, SyncMaster
Kyle D'Souza
Mitchell Hynes
Richard Fan
Akshit Gulia
Rafeed Iqbal

April 4, 2025

1 Revision History

Date	Version	Notes
January 17, 2024	1.0	Initial version of Module Guide
April 2, 2025	1.1	Remove and added modules to reflect current software architecture

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
API	Application Programming Interface
AWS	Amazon Web Services
DAG	Directed Acyclic Graph
DB	Database
GPD	Global Positioning System
JSON	Javascript Object Notation
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
REST	Representational State Transfer
RPC	Remote Procedure Call
SC	Scientific Computing
SOAP	Simple Object Access Protocol
SRS	Software Requirements Specification
SyncMaster	Syncing to a single source of truth
UC	Unlikely Change
UI	User Interface

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: Updates to the GPS authentication mechanism.

AC2: Changes in the geofencing radius or conditions for location-based authentication.

AC3: Changes to user authentication methods such as the addition of one time passwords.

AC4: The different types of supported document formats to be stored in the system.

AC5: Incorporating APIs from other city management systems (e.g. Infor EAM)

AC6: Analytic information requested by administrators, such as located-based usage patterns.

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: Removing location authentication features completely.

UC2: AWS as cloud infrastructure provider.

UC3: Removing document uploading, sharing, or storage functionalities.

UC4: Application type from web to desktop or any other type.

UC5: Web API type from REST to anything else (e.g. RPC, SOAP, GraphQL).

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: ~~Audit and Compliance Module~~

M2: User Authentication Module

M3: Location Verification Module

M4: Logging Module (Site Visits)

M5: Analytics and Reporting Module

M6: User Management Module

M7: Document Management Module

M8: Site Management Module

M9: ~~Job Management Module~~

M10: API Integration Module

M11: Database Interaction Module

M12: File Storage Interation Module

M13: Routing Module

M14: Function Compute Module

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.

Level 1	**Level 2**
Hardware-Hiding Module	N/A
Software Decision Modules	~~Audit and Compliance Module~~
	User Authentication Module
	Location Verification Module
	Logging Module
	Analytics and Reporting Module
	~~Job Management Module~~
	User Management Module
	Document Management Module
	Site Management Module
Behaviour-Hiding Modules	API Integration Module
	Database Interaction Module
	File Storage Interaction Module
	Request Routing Module
	Function Compute Module

Table 1: Module Hierarchy

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. *SyncMaster* means the module will be implemented by the SyncMaster 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

N/A

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 API Integration Module (M10)

Secrets: Raw user inputs.

Services: Module responsible for communication between the user interface and the backend API's.

Implemented By: SyncMaster

Type of Module: Abstract Object

7.2.2 Database Interaction Module (M11)

Secrets: Records to add, remove, or update in a database.

Services: Module responsible allowing read and write access to a database.

Implemented By: boto3, AWS DynamoDB, SyncMaster

Type of Module: Library

7.2.3 File Storage Interaction Module (M12)

Secrets: File to add, remove, or update in the file management system.

Services: Module responsible allowing read and write access to the file management system.

Implemented By: boto3, AWS S3, SyncMaster

Type of Module: Library

7.2.4 Routing Module (M13)

Secrets: Requests sent to backend API.

Services: Module responsible routing requests to the appropriate functions in the backend.

Implemented By: AWS API Gateway, AWS Lambda Powertools

Type of Module: Library

7.2.5 Function Compute Module (M14)

Secrets: Requests sent to backend API.

Services: Module responsible for provisioning compute in response to requests recieved.

Implemented By: AWS Lambda

Type of Module: Library

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 Location Verification Module (M3)

Secrets: user location/proximity to a known station.

Services: Uses the haversine distance formula to calculate distance from a station based on coordinates and rejects the user if they are too far from a site.

Implemented By: SyncMaster

7.3.2 User Authentication Module (M2)

Secrets: Uses the User's email, password, and location verification state.

Services: Issues authorization token to the user to allow access to backend API's if the email, and password match a known user, and the user location has been verified to be on a known site.

Implemented By: SyncMaster

7.3.3 Logging Module (M4)

Secrets: Users entry/exit time, and current site.

Services: Saves users entry/exit time for the current site into the database.

Implemented By: SyncMaster

7.3.4 ~~Audit and Compliance Module (M1)~~

Secrets: ~~Data obtained from the Logging Module.~~

Services: ~~Allows admin users users to retrieve data regarding contractors in the frontend~~

Implemented By: ~~SyncMaster~~ No longer implementing this as it is part of the Logging Module.

7.3.5 Analytics and Reporting Module (M5)

Secrets: Data obtained from the Server Log.

Services: Retrieves server analytics and reports

Implemented By: SyncMaster

7.3.6 User Management Module (M6)

Secrets: Requests for creating, updating, and deleting users.

Services: Allows the management of users, by an admin user.

Implemented By: AWS Cognito, SyncMaster

7.3.7 Document Management Module (M7)

Secrets: A file.

Services: Allows an admin user to upload a document relevant to a specific site and/or job for contractors to see as the come on site.

Implemented By: SyncMaster

7.3.8 ~~Job Management Module~~ (~~M~~9)

Secrets: ~~A job identifier, job details, and status.~~

Services: ~~Allows contractor to fill in information relevant to the job they are completing.~~

Implemented By: ~~SyncMaster~~

There is no longer a concept of jobs in the system as per the discussions had with the city.

7.3.9 Site Management Module (M8)

Secrets: The site identifier and location.

Services: Allows an admin user to add/remove/edit site locations in the system.

Implemented By: SyncMaster

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, M12, M10
FR3	M6, M7, M10, M12, M2
FR4	M6, M10, M4, M4, M2, M11
FR5	M3, M2
FR6	M7
FR7	M7, M11
FR8	M7
FR9	M7

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M3
AC2	M3
AC3	M2
AC4	M7
AC5	M10
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.

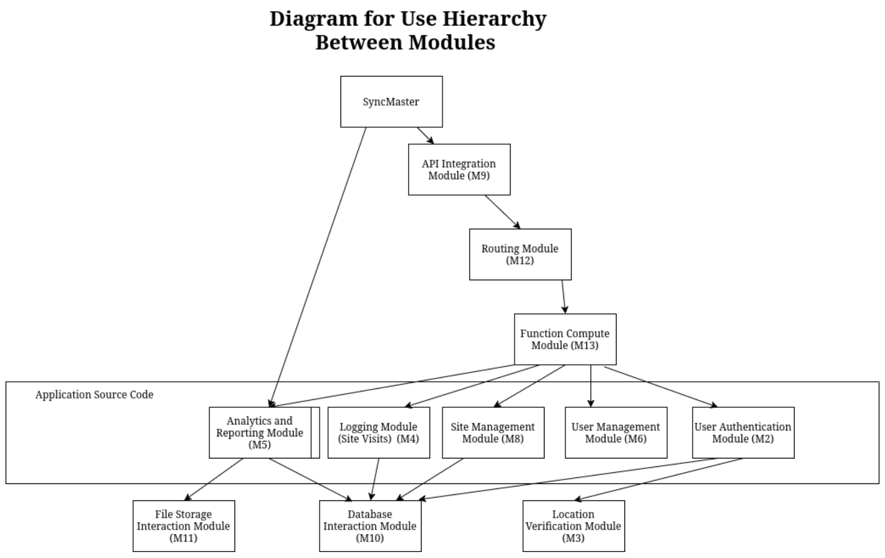
Diagram for Use Hierarchy Between Modules
SyncMaster
API Integration Module (M9)
Routing Module (M12)
Function Compute Module (M13)
Application Source Code
Analytics and Reporting Module (M5)
Logging Module (Site Visits) (M4)
Site Management Module (M8)
User Management Module (M6)
User Authentication Module (M2)
File Storage Interaction Module (M11)
Database Interaction Module (M10)
Location Verification Module (M3)

Figure 1: Use hierarchy among modules

10 User Interfaces

In this section, we have included our Figma design and prototype links for the user interfaces of the contractor and admin portals. The following are the links:

Contractor Portal Figma Design Link: Click here to view

Contractor Portal Figma Prototype Link: Click here to view

Admin Portal Figma Design Link: Click here to view

Admin Portal Figma Prototype Link: Click here to view

11 Design of Communication Protocols

N/A

12 Timeline

Below is an timeline for the implementation of modules in this project. The foundational modules User Authentication, User Management, Database Interaction, Logging, Routing, and API Integration will be developed first, as they form the core infrastructure on which the remaining modules will depend. Once these core modules are in place the other modules will be implemented. Task responsibilities and deadlines are tracked on GitHub using GitHub Issues here.

Module	**Responsible**	**Deadline**
~~M1: Audit and Compliance Module~~	~~Akshit Gulia~~	~~January 31st, 2025~~
M2: User Authentication Module	Rafeed Iqbal	January 24th, 2025
M3: Location Verification Module	Rafeed Iqbal	January 31st, 2025
M4: Logging Module	Kyle D'Souza	January 24th, 2025
M5: Analytics and Reporting Module	Akshit Gulia	January 31st, 2025
M6: User Management Module	Rafeed Iqbal	January 24th, 2025
M7: Document Management Module	Richard Fan	January 31st, 2025
~~M8: Job Management Module~~	~~Mitchell Hynes~~	~~January 31st, 2025~~
M8: Site Management Module	Mitchell Hynes	January 31st, 2025
M9: API Integration Module	Richard Fan	January 24th, 2025
M10: Database Interaction Module	Kyle D'Souza	January 24th, 2025
M11: File Storage Interaction Module	Kyle D'Souza	January 31st, 2025
M12: Routing Module	Kyle D'Souza	January 24th, 2025
M13: Function Compute Module	Kyle D'Souza	January 31st, 2025

Table 4: Schedule for module implementation

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.