Module Guide for MacONE

April 4, 2024

1 Revision History

Date	Version	Notes
2024-01-17	1.0	First version of module guide
2024-04-03	2.0	Update after final demo

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
SRS	Software Requirements Specification
UC	Unlikely Change
etc.	...

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 [?]. We advocate a decomposition based on the principle of information hiding [?]. 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 [?], 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 [?]. 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 user interface design could be changed, especially the layout, colour, and fonts.

AC2: The algorithms might be changed, such as the algorithm for PDF extraction and task priority prediction.

AC3: The database storage system could be modified, such as the database schema, tables, constraints or other elements.

AC4: The API usage in the back-end system could be upgraded or changed.

AC5: The web load or data scale could be changed in the development of the application.

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 decisions 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 programming languages and technical stack won't be changed.

UC2: The major functionality and purpose of this application won't be changed from educational use to other business use.

UC3: The deployment of the database on AWS cloud service won't be changed because of high cost.

UC4: The application's overall architecture won't be changed dramatically.

UC5: The interface device would always be the computer keyboard and monitor.

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: Interface Module

M2: Back End Web Service Module

M3: User Authentication Module

M4: App Grid Module

M5: Task Module

M6: Course Module

M7: User Module

M8: Pomodoro Module

M9: Forum Module

M10: Feedback Module

M11: PDF Extraction Module

M12: cGPA Calculation Module

M13: Database Module

Level 1	Level 2
Hardware-Hiding Module	Interface Module
Behaviour-Hiding Module	Back End Web Service Module
	User Authentication Module
	App Grid Module Module
	Task Module
	Course Module
	User Module
	Pomodoro Module
	Forum Module
	Feedback Module
Software Decision Module	PDF Extraction Module
	cGPA Calculation Module
	Database 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 **?**. 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

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

7.1.1 Interface Module (M1)

Secrets: Hide the detailed structure and logic of front-end technical implementation, which involves the graphic design and user interface.

Services: Serves as the medium for users to interact with the application to implement diverse functionalities effectively. The client side of the application sends user data to the backend, and then in turn processes the received information and displays it in the interface in a user-friendly and easily understandable form.

Implemented By: Developed using web technologies like HTML, CSS

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 Back End Web Service Module (M2)

Secrets: Internal Logic and data processing methods

Services: Offers web services for front-end modules, handling requests, responses and exceptions

Implemented By: Server-side Languages and Principles

7.2.2 User Authentication Module (M3)

Secrets: The confidential authentication mechanism, involves the encryption of the password, the location of data stored, and the process of verification of data.

Services: Provides verification service for checking user identities during the login procedure. This module ensures only the authenticated user can access certain features of this application.

Implemented By: Integrates with User Module and Database module, utilizes data mapping and hashing technique for verification.

7.2.3 App Grid Module (M4)

Secrets: The categorization and manipulation of apps.

Services: Provides functionalities for users to change the position of icons in the app grid.

Implemented By: Integrates with the database module for consistent data storage of app grid information.

7.2.4 Task Module (M5)

Secrets: The categorization and manipulation of tasks.

Services: Provides functionalities for users to manipulate tasks, including creating, reading, updating and deleting tasks. Also, the service involves additional features: setting deadlines, tasks, set difficulty levels.

Implemented By: Integrates with the database module for consistent data storage of task information. Also, utilize request transmission technique API for sending data to the Task prioritization module, and receive requests from the back-end server-side to execute instructions.

7.2.5 Course Module (M6)

Secrets: Course Data structure and management algorithms

Services: Manages course information, including the creation, modification, and deletion.

Implemented By: Integrates with the database for storing the course data

7.2.6 User Module (M7)

Secrets: User info data handling method.

Services: Manages User information, including the creation, modification, and deletion.

Implemented By: Integrates with the database module and authentication module.

7.2.7 Pomodoro Module (M8)

Secrets: Specific study time interval and design technique

Services: Provides a Pomodoro timer for users productively conduct study tasks.

Implemented By: Integrate with the user interface and provide a real-time interactive feature by scripting language and UI design.

7.2.8 Forum Module (M9)

Secrets: Forum info data handling method.

Services: Manages forum information, including the creation, modification, and deletion.

Implemented By: Integrate with the user interface and provide a real-time interactive feature by scripting language and UI design.

7.2.9 Feedback Module (M10)

Secrets: Feedback info data handling method.

Services: Manages feedback information, including the creation and modification.

Implemented By: Integrates with the database module and UI design.

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 PDF Extraction Module (M11)

Secrets: Techniques and specific algorithms or logics for extracting data from PDF files, and the standardized format of data while to be stored in the database.

Services: Extracts and processes information from the user-uploaded PDF documents.

Implemented By: Utilizes specific libraries for pdf processing

7.3.2 cGPA Calculation Module (M12)

Secrets: Techniques and specific algorithms or logics for calculating cGpa from PDF files.

Services: Extracts and processes information from the user-uploaded PDF documents.

Implemented By: Utilizes specific libraries for pdf processing

7.3.3 Database Module (M13)

Secrets: Database schema and query optimization techniques

Services: Provides CRUD (create, read, update, and delete) manipulation of all the data used in the application efficiently.

Implemented By: Deployed on the AWS cloud service platform and utilized a relational database system.

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	M6, M2, M11 M13
FR2	M6, M5, M8, M2, M13
FR3	M7, M6, M2, M13, M5
FR4	M7, M6, M2, M13, M5
FR5	M7, M2, M13, M8
FR6	M7, M2, M13, M8
FR7	M7, M13, M8
FR8	M7, M13, M8
FR9	M7, M13, M5, M2
FR10	M1, M13, M5
FR11	M1, M13, M5 M2
FR12	M1, M13, M5
FR13	M1, M5
FR14	M7, M1, M2, M9, M13
FR15	M9, M13
FR16	M7, M1, M2, M9, M13
FR17	M7, M1, M2, M9, M13
FR18	M1, M7, M4, M13
FR19	M1, M7, M4, M13 M2
FR20	M1, M10, M2, M13
FR21	M1, M10, M13
FR22	M1, M10, M13
FR23	M1, M3, M7, M2, M13
FR24	M1, M2, M12

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M12, M11
AC3	M13
AC4	M2
AC5	M13

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

The use hierarchy between these modules is represented using a DAG diagram, which visualizes the interconnections between different modules inside this software application system. Each node represents a module stated in the previous module decomposition section. The edges indicate the dependencies between modules. An arrow from module A to module B implies that module A depends on module B, both the functionality support and data transmission. We can see that the Back End Web Service Module is located as a central node on this diagram, which serves as a communication hub for the breach of sub-system modules.

This diagram reflects several software architecture concepts. Modularity can be easily understood, the information-hiding principle is also an additional effect. Each module hides its internal details from other modules, and the design idea of loose coupling and high cohesion are proven. Such layer architecture makes this application can be high maintainable and reusable because the changes in one individual module would have little effect on others.

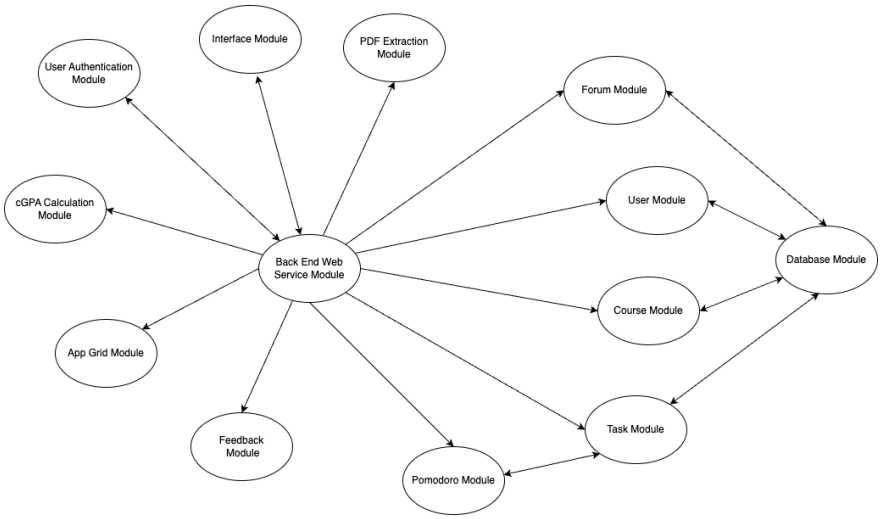

Figure 1: Use hierarchy among modules

10 Timeline

This section provides a detailed timeline for the implementation, testing, and verification of the system modules, split across two pages for clarity. Each task requires the collaboration of two or more team members to ensure thoroughness and accuracy. The timeline is structured to ensure functional implementation and basic testing by February 5, 2024, and complete full testing and verification by February 20, 2024.

Module	**Task**	**Responsible Members**	**Completion Date**
Interface Module	UI Design and Initial Development	Jingyao Qin, Chenwei Song	2024-01-24
	UI Module Testing	Qianni Wang, Shuting Shi	2024-01-28
Back End Web Service	Server Setup and API Development	Chenwei Song, Qiang Gao	2024-01-26
	Back End Module Testing	Jingyao Qin, Qianni Wang	2024-01-30
User Authentication	Authentication Mechanism Implementation	Qianni Wang, Shuting Shi	2024-01-25
	Authentication Module Testing	Chenwei Song, Qiang Gao	2024-01-29
Task Module	Task Management Functions Implementation	Shuting Shi, Qiang Gao	2024-01-27
	Task Module Testing	Jingyao Qin, Chenwei Song	2024-01-31
Database Module	Database Design and Integration	Qiang Gao, Jingyao Qin	2024-01-26
	Database Module Testing	Qianni Wang, Shuting Shi	2024-01-30
Pomodoro Module	Timer Functionality Implementation	Jingyao Qin, Qianni Wang	2024-01-23
	Pomodoro Module Testing	Chenwei Song, Qiang Gao	2024-01-27
PDF Extraction Module	PDF Processing and Extraction Logic	Chenwei Song, Shuting Shi	2024-01-28
	PDF Extraction Module Testing	Qianni Wang, Qiang Gao	2024-02-01
Feedback Module	App order Functions Implementation	Qianni Wang	2024-01-27
	Feedback Module Testing	Jingyao Qin, Chenwei Song	2024-01-31
cGpa calculation Module	App order Functions Implementation	Qianni Wang	2024-01-27
	cGpa calculation Module Testing	Jingyao Qin, Chenwei Song	2024-01-31

Table 4: Timeline (Part 1)

Module	Task	Responsible Members	Completion Date
App grid Module	App order Functions Implementation	Qianni Wang	2024-01-27
	App grid Module Testing	Jingyao Qin, Chenwei Song	2024-01-31
Forum Module	App order Functions Implementation	Chenwei Song	2024-01-27
	Forum Module Testing	Chenwei Song	2024-01-31
Basic Testing	Integration Testing for Each Module	All Members	2024-02-05
Refinement	Bug Fixing and Code Refinement	All Members	2024-02-10
Full Testing and Verification	Comprehensive System Testing	All Members	2024-02-15
Final Verification	Final Review and Verification	All Members	2024-02-20

Table 5: Timeline (Part 2)

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 current scope of our application is very limited because of the cost consideration. If the user amount grows significantly, our backend module may face the condition of overload, and traffic burden. Also, The central module plays a very important role, if it fails, then most functionalities could be affected. Therefore, the current application is not fault-tolerant and error-free. The current module architecture design appears that low flexibility of inner connections between each other, so new features and module additions could be a heavy workload if the new one has close relationships with many modules which need a lot of data transmission and interaction.

   Therefore, if given unlimited resources, a failure-tolerant mechanism system should be deployed for a better user experience. Also, investing in more optimized algorithms with unlimited data resources to do analysis could make our prediction more accurate and increase computing efficiency. Also, to solve the problem of scope limitation,

conduct a distributed system to relieve the stress of the server when customer usage is too high.

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 a documented design?(LO_Explores)

   Instead of having a centralized back-end service center, the micro-service architecture design could be a valuable approach for a software application, which is composed of small independent services that could communicate over individual APIs. There are many benefits of this design. This design is very flexible, expanding each service to satisfy the specific need, and maintaining good usability. Continuous integration and deployment are also supported. rollback can also be triggered when the error occurs. Because the cost of maintainability for this kind of architecture is low, experiments and new changes are encouraged, then the product release cycle can be highly reduced.

   The reason why we chose our current design is because of the balance between complexity and maintainability. We conduct modularization but still centralize the back-end service, which eases the build complexity and effort at the initial stage of development while maintaining flexibility. Continuous integration and deployment are supported in our current design system, the decentralized optimization can also easily be brought in the future if necessary.

References