Module Guide for Room8

Team 19
Mohammed Abed
Maged Armanios
Jinal Kasturiarachchi
Jane Klavir
Harshil Patel

January 14, 2025

1 Revision History

Date	**Version**	**Notes**
Date 1	1.0	Notes
Date 2	1.1	Notes

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
Room8	Explanation of program name
UC	Unlikely Change
UI	User Interface
UX	User Experience
[etc. —SS]	[... —SS]

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 implementation details of the detection algorithm. Including data preprocessing techniques and machine learning techniques.

AC2: The specific hardware in which the camera and sensor system is running.

AC3: The kind of sensor used to detect activity.

AC4: The criteria for signalling the camera to capture.

AC5: The output of the cleanliness detection algorithm whether it be an image outlining changes or a number quantifying the change in a room.

AC6: The frontend technologies used to create the user interface for the web platform (ex: React, NextJs).

AC7: Specific flows of frontend features such as the bill splitting feature and chore scheduling. Changes are dependant on usability test results.

AC8: Allowing users to see the cleanliness score of another user and detecting which user made a mess.

[Anticipated changes relate to changes that would be made in requirements, design or implementation choices. They are not related to changes that are made at run-time, like the values of parameters. —SS]

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 use of Python for the implementation of the cleanliness detection algorithm. This is due to the plethora of machine learning libraries such as PyTorch available.

UC2: The user facing application being implemented as a web application.

UC3: The data persistence for the application being implemented with a relational database, specially PostgreSQL.

UC4: Utilizing a third-party service to implement OAuth for user login.

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: Sensor Reading Module

M2: Image Capture Module

M3: Image Upload Module

M4: Image Preprocessing Module

M5: Object Detection Module

M6: Scoring Module

M7: Request Listener Module (Cleanliness Detection System)

M8: Data Uploading Module (Cleanliness Detection System)

M9: Chore Schedule Module

M10: Chat Bot Module

M11: Bill Splitting Module

M12: User Authentication Module

M13: Home Management Module

- Sensor reading - Harsh
- Image capture module - Harshi
- Image upload module - Harshy

- Preprocessing module - Janet
- Object detection module - Janel
- Scoring module - Janille
- Bill splitting - Maged
- (Thing that exposes cleanliness detector to camera (makes it an API) - Patel
- (cleanliness detector Communicates with backend) module - enaJ
- Chore Scheduling Module - Maged
- Group chat module - Maged
- (FRONTEND Communicates with backend) module - Maged

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	?
	?
	?
	?
	?
	?
	?
	?
Software Decision 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.

[The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be made. Rather than make these decisions implicit, they are explicitly recorded here. For instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password. —SS]

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. *Room8* means the module will be implemented by the Room8 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 (M??)

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 Input Format Module (M??)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.2 Etc.

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 Etc.

8 Traceability Matrix

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

Module	**Reqs.**
M1	NFR237
M2	NFR231-232, NFR234
M3	NFR233
M4	NFR235
M5	FR233, NFR235-236
M6	FR231
M7	NFR235
M8	NFR233
M9	FR241-245, NFR241-244
M10	FR221-224, NFR221-222
M11	FR251-255, NFR251-252
M12	FR211-213, NFR211-212, NFR214
M13	FR214-218, FR234-235, NFR213

Table 2: Trace Between Requirements and Modules

AC	**Modules**
M1	M4, M5, M6
M2	M1, M2, M3
M3	M1
M4	M2
M5	M6, M3, M13
M6	M9, M10, M11, M12, M13
M7	M9, M10, M11, M12, M13
M8	M5, M6, M8, M13

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.

[The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without an import, but the arrows in the diagram typically correspond to the presence of import statement. —SS]

The diagram denotes uses relationships with a solid arrow. If a component communicates with another component via some remote protocol like HTTP/HTTPS it is denoted with a dotted arrow. Finally, modules that are not implemented by the team but are utilized in the system like web frameworks or database systems are denoted with a blue square.

[If module A uses module B, the arrow is directed from A to B. —SS]

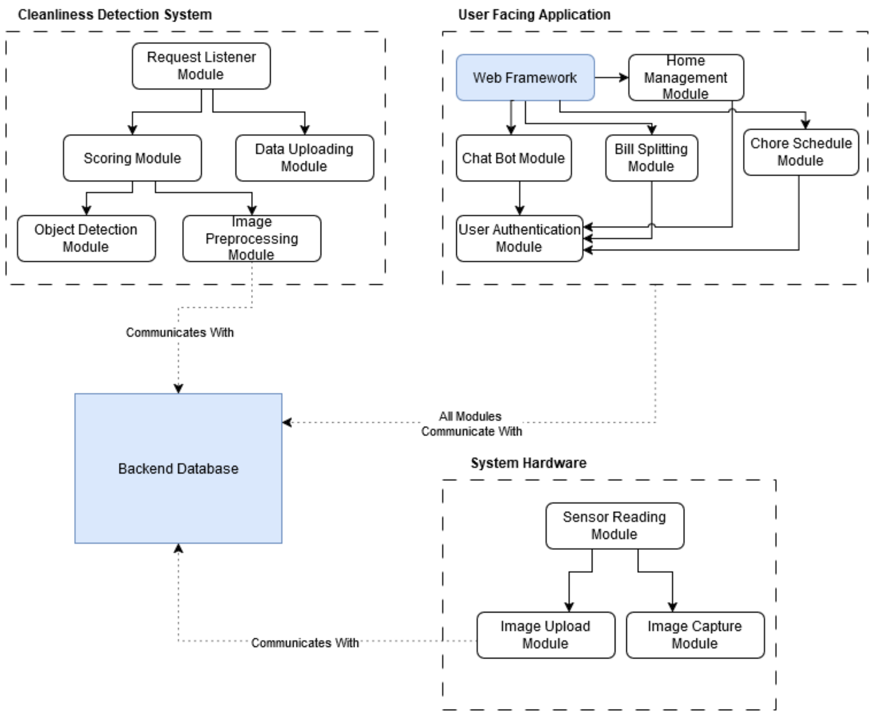
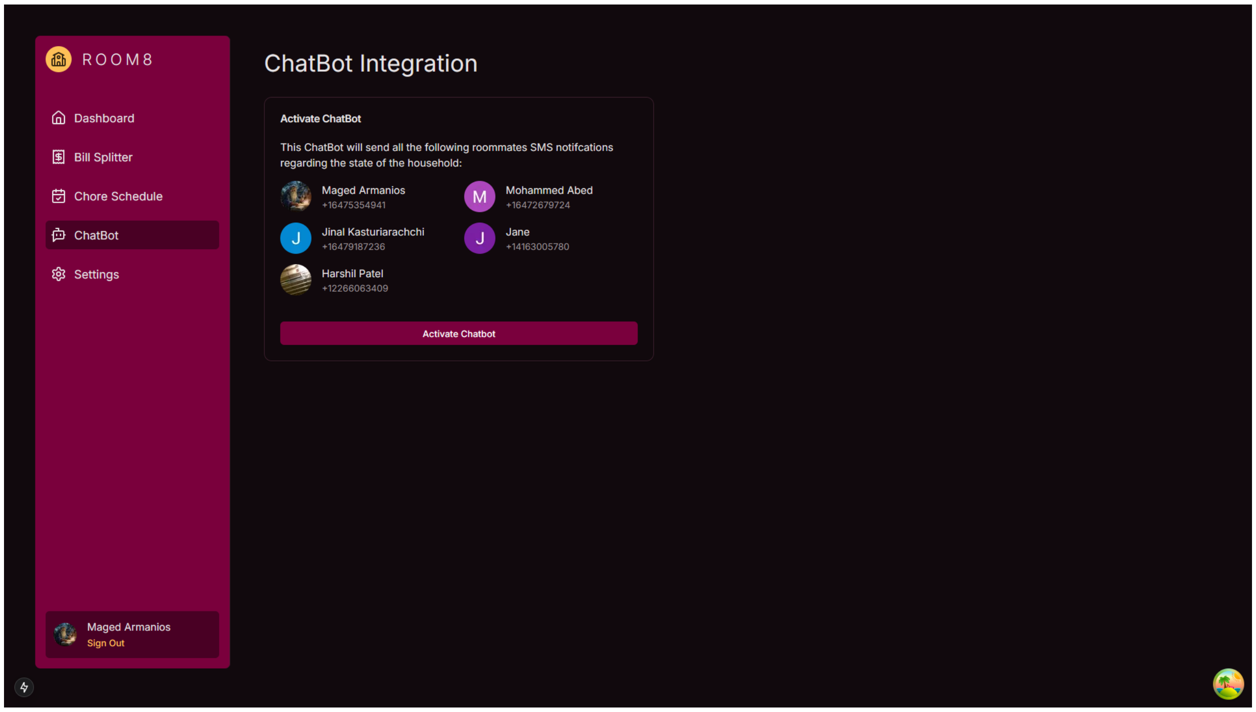


Figure 1: Use hierarchy among modules

10 User Interfaces

User interface yet to be designed.

[Design of user interface for software and hardware. Attach an appendix if needed. Drawings, Sketches, Figma —SS]



ChatBot Integration

Activate ChatBot

This ChatBot will send all the following roommates SMS notifcations regarding the state of the household:

Maged Armanios +16475354941

Mohammed Abed +16472679724

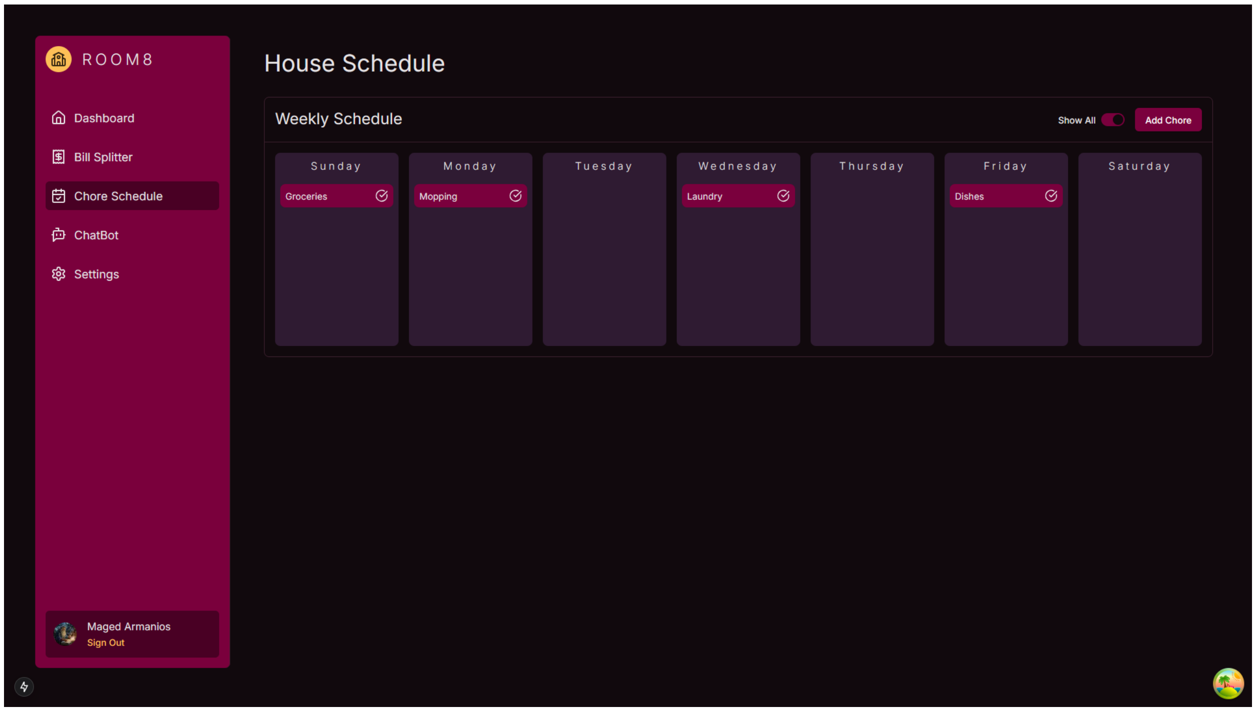
Jinal Kasturiarachchi +16479187236

Jane +14163005780

Harshil Patel +12266063409

Activate Chatbot

Figure 2: Activate Chatbot Page

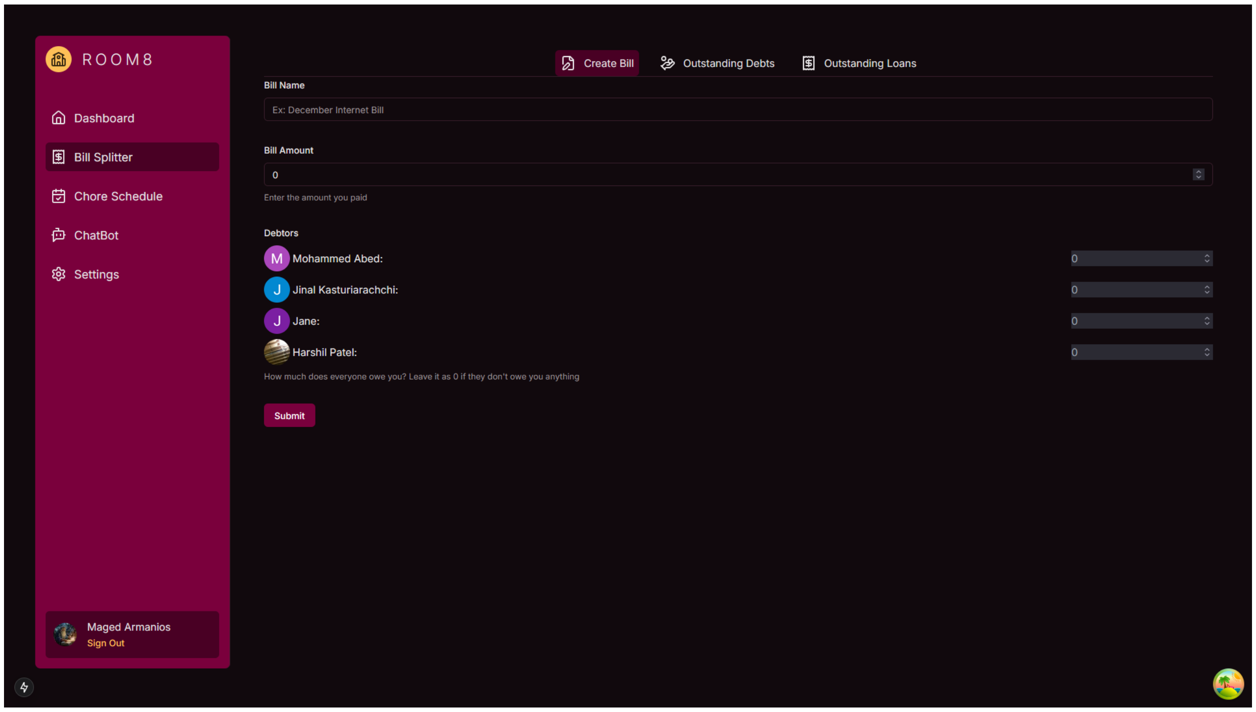


House Schedule

Weekly Schedule

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Groceries	Mopping		Laundry		Dishes	

Figure 3: Chore Schedule Page



Bill Name

Ex: December Internet Bill

Bill Amount

0

Enter the amount you paid

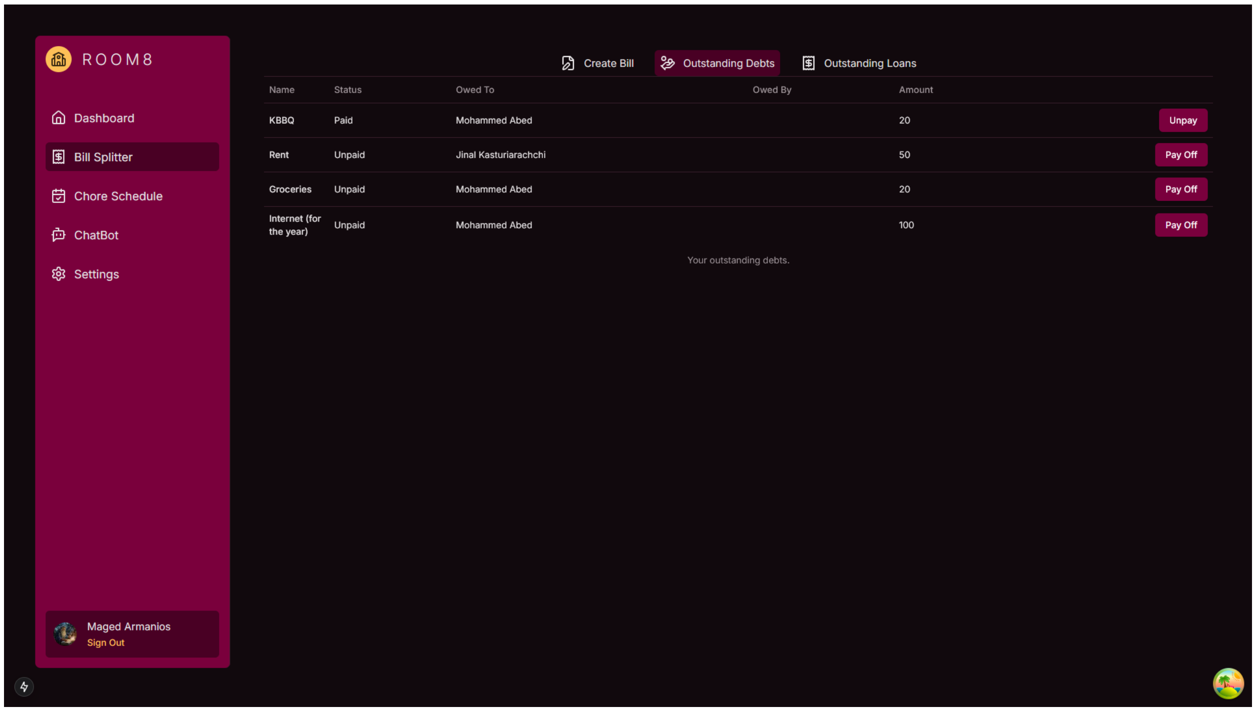
Debtors

Mohammed Abed:	0
Jinal Kasturiarachchi:	0
Jane:	0
Harshil Patel:	0

How much does everyone owe you? Leave it as 0 if they don't owe you anything

Submit

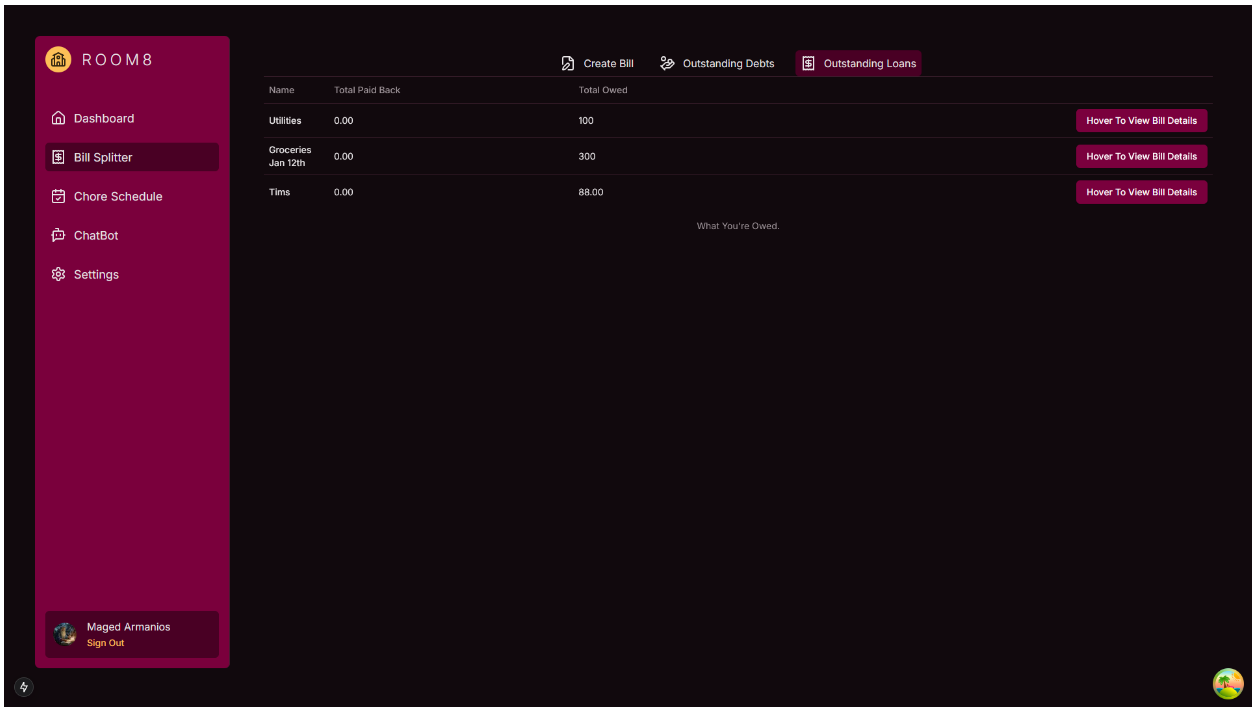
Figure 4: Create bill Page



Name	Status	Owed To	Owed By	Amount	
KBBQ	Paid	Mohammed Abed		20	Unpay
Rent	Unpaid	Jinal Kasturiarachchi		50	Pay Off
Groceries	Unpaid	Mohammed Abed		20	Pay Off
Internet (for the year)	Unpaid	Mohammed Abed		100	Pay Off

Your outstanding debts.

Figure 5: Outstanding Bills Page



Name	Total Paid Back	Total Owed	
Utilities	0.00	100	Hover To View Bill Details
Groceries Jan 12th	0.00	300	Hover To View Bill Details
Tims	0.00	88.00	Hover To View Bill Details

What You're Owed.

Figure 6: Owed Bills Page

11 Timeline

- **Week 1 - 4:** Jan 15 - Feb 11, 2025
  - Complete modules M9, M11, M12, M13
  - Begin working on the cleanliness detection system's algorithm (M4, M5, M6)
- **Week 5 - 8:** Feb 12 - Mar 10, 2025
  - Begin working on frontend for cleanliness management system (M8, M13)
  - Designing and create system hardware (M1, M2, M3)
  - Create CI/CD and automated testing using GitHub Actions and the appropriate testing libraries for each component. Generate unit and integration tests.
  - Perform usability testing
- **Week 9 - 10:** Mar 11 - Mar 24, 2025
  - Apply feedback from usability testing to finalize UI/UX.
  - Implement M7 so the cleanliness management system can receive input from the hardware system.

  - All 3 systems (user application, cleanliness management system, and hardware system), should be completed by the end of this period.

- **Week 11 - 12:** Mar 25 - Apr 7, 2025
  - Perform acceptance tests on the system.
  - Prepare demo examples and showcase.
  - Patch bugs detected from tests and do refinements based on user and stakeholder feedback.

[Schedule of tasks and who is responsible —SS]
[You can point to GitHub if this information is included there —SS]

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.