Module Guide for Room8

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

January 17, 2025

1 Revision History

Date	Version	Members	Notes
2025-01-17	0.0	Team	Initial creation

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	Name of the system
UC	Unlikely Change
UI	User Interface
UX	User Experience

Contents

1 Revision History **i**

2 Reference Material **ii**

2.1 Abbreviations and Acronyms ii

3 Introduction **1**

4 Anticipated and Unlikely Changes **2**

4.1 Anticipated Changes 2

4.2 Unlikely Changes 2

5 Module Hierarchy **3**

6 Connection Between Requirements and Design **3**

7 Module Decomposition **4**

7.1 Hardware Hiding Modules 4

7.1.1 Sensor Reading Module (M1) 5

7.1.2 Image Capture Module (M2) 5

7.1.3 Image Upload Module (M3) 5

7.2 Behaviour-Hiding Module 5

7.2.1 Preprocessing Module (M4) 5

7.2.2 Request Listener Module (M7) 6

7.2.3 Data Uploading Module (M8) 6

7.3 Software Decision Module 6

7.3.1 Object Detection Module (M5) 6

7.3.2 Scoring Module (M6) 6

7.3.3 Chore Schedule Module (M9) 7

7.3.4 Chat Bot Module (M10) 7

7.3.5 Bill Splitting Module (M11) 7

7.3.6 User Authentication Module (M12) 7

7.3.7 Home Management Module (M13) 8

7.3.8 Cleanliness Management Module (M14) 8

8 Traceability Matrix **8**

9 Use Hierarchy Between Modules **9**

10 User Interfaces **10**

11 Timeline **13**

List of Tables

1 Module Hierarchy 4
2 Trace Between Requirements and Modules 9
3 Trace Between Anticipated Changes and Modules 9

List of Figures

1 Use hierarchy among modules 10
2 Activate Chatbot Page 11
3 Chore Schedule Page 11
4 Create bill Page 12
5 Outstanding Bills Page 12
6 Owed Bills Page 13

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.

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

M14: Cleanliness Management 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	M1
	M2
	M3
Behaviour-Hiding Module	M4
	M7
	M8
Software Decision Module	M5
	M6
	M9
	M10
	M11
	M12
	M13
	M14

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. *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

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

7.1.1 Sensor Reading Module (M1)

Secrets: Sensor thresholds for detection.

Services: Uses sensor to detect when user is in shared space and when user is finished using space.

Implemented By: Sensor.

7.1.2 Image Capture Module (M2)

Secrets: Image format.

Services: Captures images on camera system and forwards them to the image upload module.

Implemented By: Camera.

7.1.3 Image Upload Module (M3)

Secrets: Upload location and image format.

Services: Uploads image captured from camera system for system to use in cleanliness detection algorithm.

Implemented By: Raspberry Pi.

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 Preprocessing Module (M4)

Secrets: Input image format/behaviour, transformations applied.

Services: Converts image to tensor and performs filtering/transformations.

Implemented By: Python, PyTorch, Torchvision

7.2.2 Request Listener Module (M7)

Secrets: Sending image to cleanliness detector.

Services: Converts image and information from camera and sensor to be used in system.

Implemented By: cameraToDetectorAPI.

Type of Module: Application programming interface.

7.2.3 Data Uploading Module (M8)

Secrets: Details of how/where python modules are hosted for Room8 backend to access.

Services: Packages Python modules (i.e. Preprocessing, Object Detecion, Scoring) as a RESTful endpoint so that the backend can interface with the Python machine learning functionality.

Implemented By: Python, FastAPI

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 Object Detection Module (M5)

Secrets: Object detector, dataset used for training/testing, map of classes.

Services: Identifies common household objects in a scene.

Implemented By: Python, PyTorch, Torchvision

7.3.2 Scoring Module (M6)

Secrets: Logic to calculate cleanliness score/determine which changes have occurred in the shared space.

Services: Applies algorithm to set of before/after photos to determine a cleanliness score.

Implemented By: Python

7.3.3 Chore Schedule Module (M9)

Secrets: Chore management logic and database implementation.

Services: Allows the user to create, modify, and delete existing chore data from the client-side.

Implemented By: Room8.

Type of Module: Abstract object.

7.3.4 Chat Bot Module (M10)

Secrets: Interface with SMS API and logic which supports sending of relevant notifications.

Services: Displays the current group SMS status for the house, and allows users to create a group/send a message.

Implemented By: Room8.

Type of Module: Abstract object.

7.3.5 Bill Splitting Module (M11)

Secrets: Calculation of splitting bills and assigning outstanding debts/credits to users.

Services: Displays and allows the user to create, modify, and delete existing bills data from the client-side.

Implemented By: Room8.

Type of Module: Abstract object.

7.3.6 User Authentication Module (M12)

Secrets: Authentication process for creation of users on sign up and determining whether or not a user is logging in with valid credentials on sign in.

Services: Authenticates user to the client-side application, provides them with a JSON Web Token (JWT) and allows them to perform authorized actions relevant to themselves/their house.

Implemented By: Room8.

Type of Module: Abstract object.

7.3.7 Home Management Module (M13)

Secrets: Logic behind managing a user's profile/house.

Services: Provides the user with the ability to view and modify details of their house. This includes, but is not limited to: updating personal information, adding roommates, removing roommates, leaving a house, and joining a house if they are not already a part of one.

Implemented By: Room8.

Type of Module: Abstract object.

7.3.8 Cleanliness Management Module (M14)

Secrets: Conversion of data received from the camera system and scoring module to the interface that users are able to view.

Services: Provides the user with the ability to view the details of messes that exist within their shared space. The view will allow them to see an image of where the mess exists, along with which user it is assigned to, and a toggle for whether or not the mess has been cleaned.

Implemented By: Room8.

Type of Module: Abstract object.

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, NFR213
M14	FR234

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M4, M5, M6
AC2	M1, M2, M3
AC3	M1
AC4	M2
AC5	M6, M3, M13, M14
AC6	M9, M10, M11, M12, M13, M14
AC7	M9, M10, M11, M12, M13, M14

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 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.

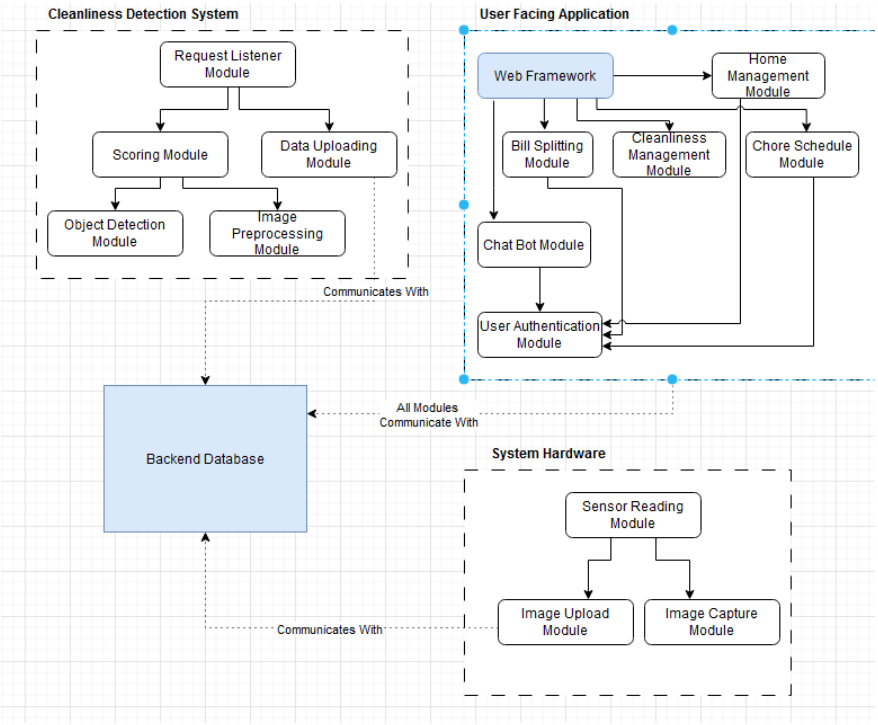

Figure 1: Use hierarchy among modules

10 User Interfaces

User interface of the home management, authentication, splash, and dashboard page yet to be designed. They will be included in future revisions of this document in addition to the UI of the hardware system.

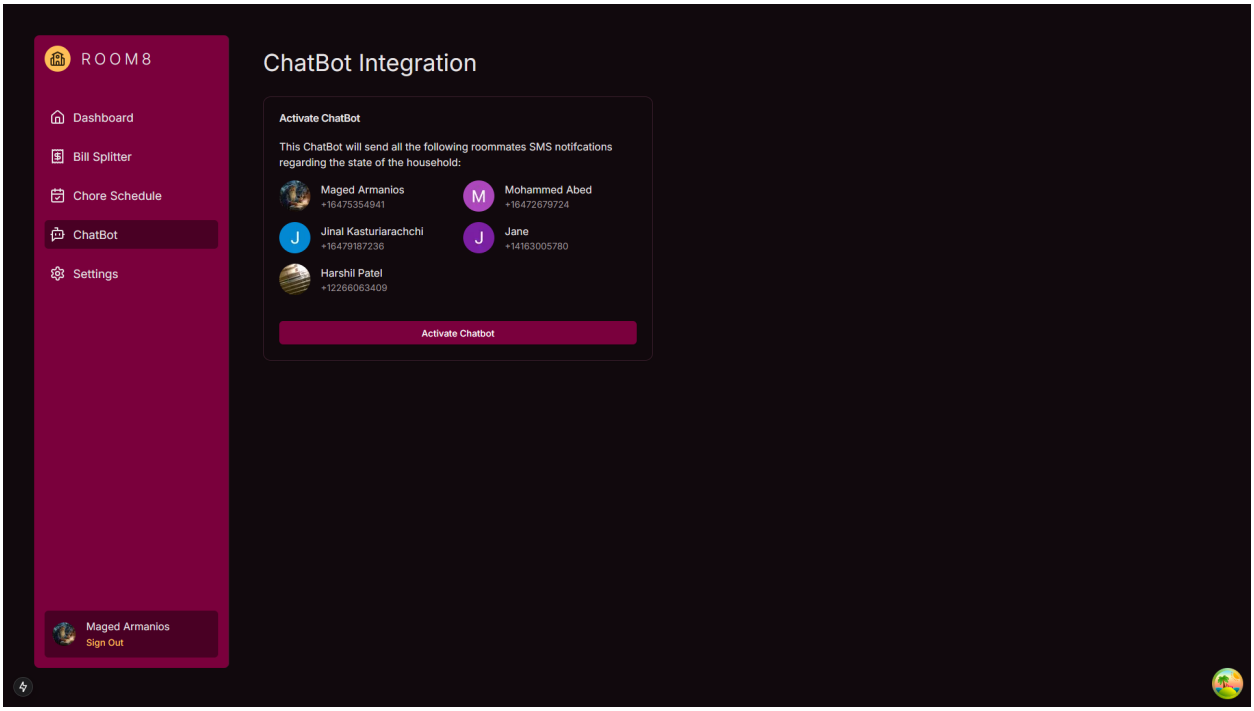

Figure 2: Activate Chatbot Page

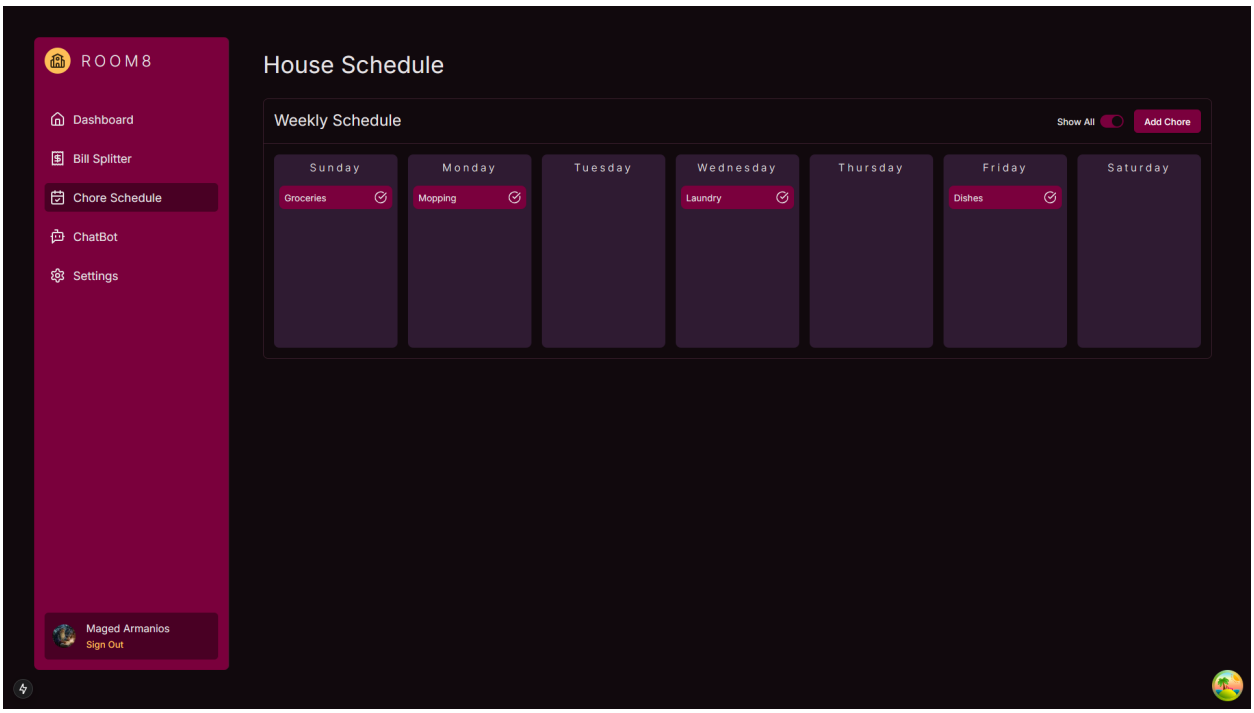

Figure 3: Chore Schedule Page

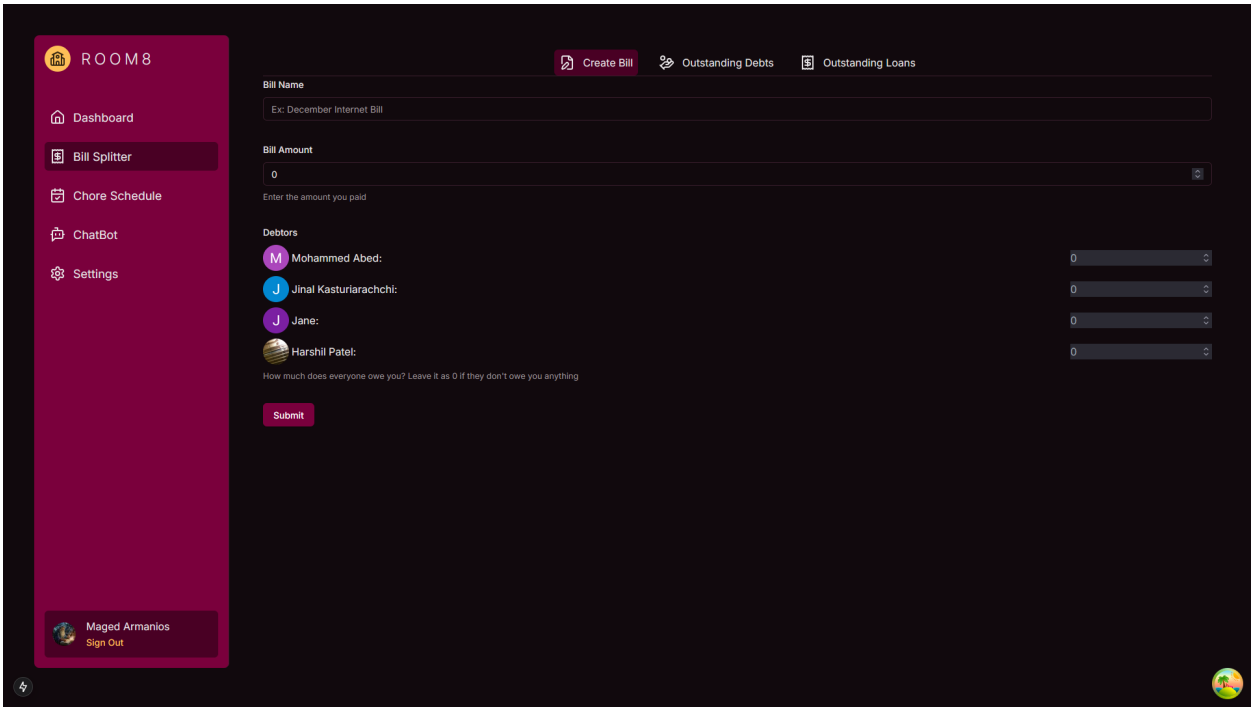

Figure 4: Create bill Page

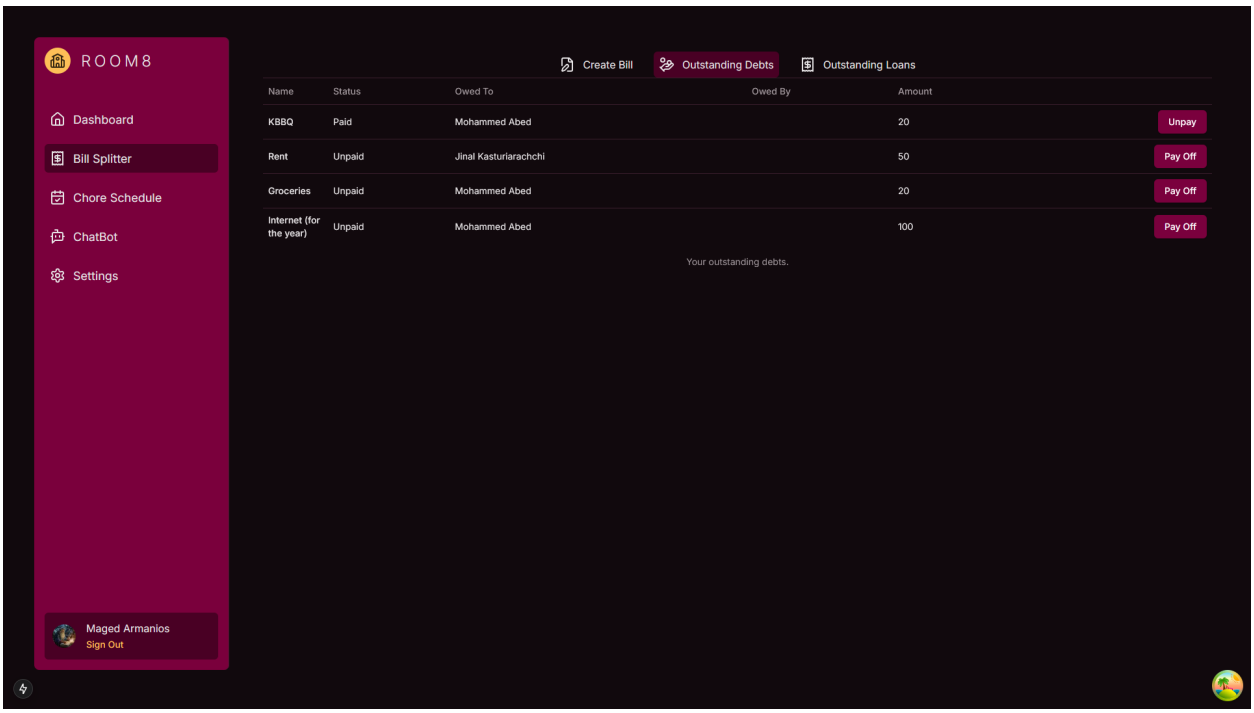

Figure 5: Outstanding Bills Page

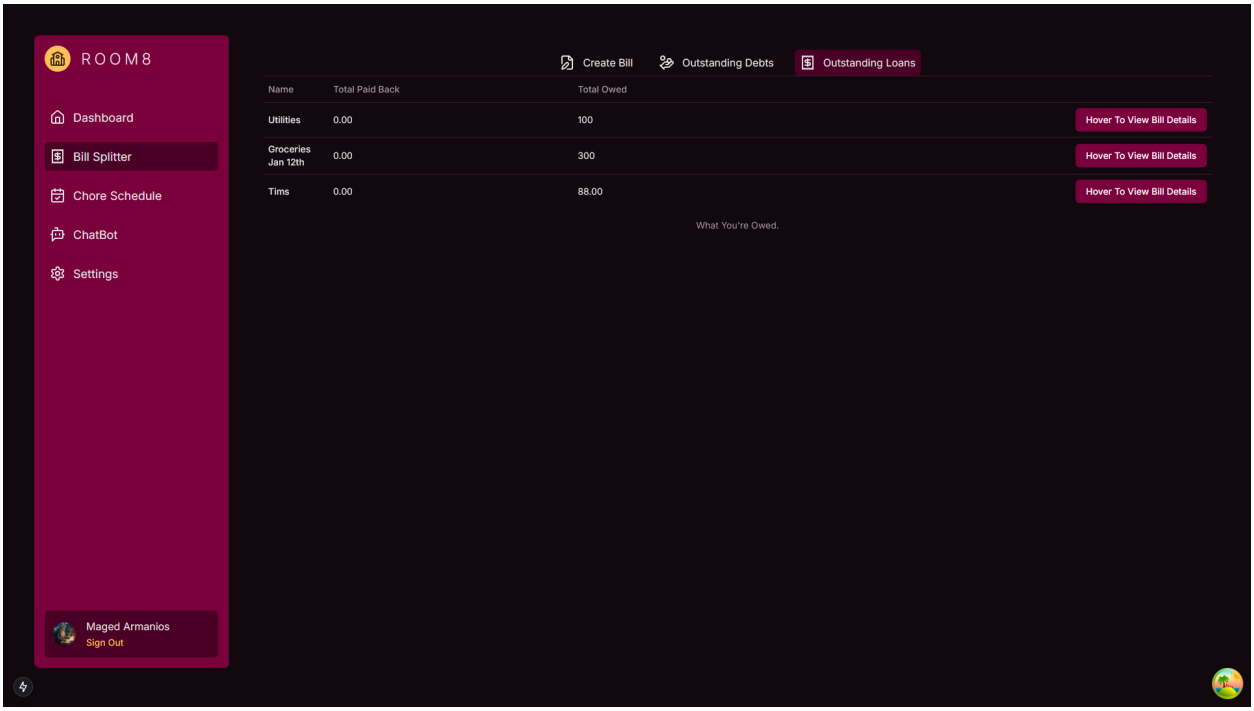

Figure 6: Owed Bills Page

11 Timeline

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

  - Implement M7 so the cleanliness management system can receive input from the hardware system. - [Maged, Mohammed, Jinal]
  - 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. - [Team]
  - Prepare demo examples and showcase. - [Team]
  - Patch bugs detected from tests and do refinements based on user and stakeholder feedback. - [Team]

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.