Module Interface Specification for Housemates

Team #9, Housemates
Justin Dang - dangj15
Harris Hamid - hamidh1
Fady Morcos - morocof2
Rizwan Ahsan - ahsanm7
Sheikh Afsar - afsars

April 4, 2024

1 Revision History

Date	Version	Notes
January 17, 2024	1.0	Revision 0
April 03, 2024	2.0	Revision 1

Contents

2 Introduction

The following document details the Module Interface Specifications for Housemates. The Housemates app will allow for its users to better communicate with their housemates. Additionally the app will have a cost management and chore management system to allow for splitting of chores/costs amongst housemates.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/DangJustin/CapstoneProject.

3 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Housemates.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
boolean	$\mathbb{B}$	a boolean value (True or False)

The specification of Housemates uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Housemates uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

4 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Task Management Module Bill Management Module Scheduling Module Account Module Database Model
Software Decision Module	Database Driver Module

Table 1: Module Hierarchy

5 MIS of Task Management Module

5.1 Module

TaskManagement

5.2 Uses

Account, DatabaseDriver

5.3 Syntax

5.3.1 Exported Constants

None

5.3.2 Exported Access Programs

Name	In	Out	Exceptions
addTask	taskName: string, groupID: string, deadlineDate: Date, description: string, usersResponsible: seq of string	seq of string	TaskAddError
getTask	taskID: string	seq of string	NotFoundError
getUserTasks	userID: string	seq of string	NotFoundError
completeTask	taskID: string	-	NotFoundError
reopenTask	taskID: string	-	NotFoundError
editTask	taskData: seq of string	-	NotFoundError

5.4 Semantics

5.4.1 State Variables

None

5.4.2 Environment Variables

None

5.4.3 Assumptions

All taskIDs and userIDs are unique.

5.4.4 Access Routine Semantics

addTask(taskName, groupID, deadlineDate, description, usersResponsible):

- transition: input data == valid ⇒ add task to database
- output: out := new task returned by database
- exception: exc := (invalid input || database error ⇒ TaskAddError)

getTask(taskID):

- output: out := task data from taskID
- exception: exc := (taskID ∉ database ⇒ NotFoundError)

getUserTasks(userID):

- output: out := taskIDs of tasks associated with the current userID
- exception: exc := (userID ∉ database ⇒ NotFoundError)

completeTask(taskID):

- transition: taskID ∈ database ⇒ remove from active tasks
- exception: exc := (taskID ∉ database ⇒ NotFoundError)

reopenTask(taskID):

- transition: taskID ∈ database ⇒ add to active tasks
- exception: exc := (taskID ∉ database ⇒ NotFoundError)

editTask(taskData):

- transition: Edit the details of the task
- exception: exc := (taskID ∉ database ⇒ NotFoundError)

5.4.5 Local Functions

None

6 MIS of Bill Management Module

6.1 Module

BillManagement

6.2 Uses

Account, DatabaseDriver

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
splitExpense	userID: string, amount: $\mathbb{Z}$, billName: string, participants: seq of string, groupID: string, individualAmounts: seq of $\mathbb{Z}$, category: string	-	BillAddError
deleteExpense	billID: string	-	NotFoundError
getExpenses	userID: string	seq of string	NotFoundError
editBill	billID: string, updatedData: seq of string	-	NotFoundError, ValidationError

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

All billIDs and userIDs are unique.

6.4.4 Access Routine Semantics

splitExpense(userID, amount, billName, participants, groupID, individualAmounts, category):

- transition: Once the function validates the input data, bill is added into the database using provided parameters.

- exception: exc := (invalid input || database error ⇒ BillAddError)

deleteExpense(billID):

- transition: Remove appropriate amount from the user's total expense and add/remove amount from associated user's.
- exception: exc := (billID ∉ database ⇒ NotFoundError)

getExpenses(userID):

- output: out := billIDs of bills associated with current userID from account module.
- exception: exc := (billID ∉ database ⇒ NotFoundError)

editBill(billID, updatedData):

- transition: Edit the details of the bill
- exception: exc := (billID ∉ database ⇒ NotFoundError || updatedData == invalid ⇒ ValidationError)

7 MIS of Scheduling Module

7.1 Module

Scheduling

7.2 Uses

Account, DatabaseDriver

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
addEvent	eventName: string, dateTime: Date, endDateTime: Date, groupID: string	seq of string	EventAddError
editEvent	eventID: string, updateData: seq of string	seq of string	NotFoundError, ValidationError
deleteEvent	eventID: string	-	NotFoundError
getGroupEvents	groupID: string	seq of string	NotFoundError
getUserEvents	userID: string	seq of string	NotFoundError

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

All eventIDs, groupIDs and userIDs are unique.

7.4.4 Access Routine Semantics

addEvent(eventName, dateTime, endDateTime, groupID):

- transition: create event in database using parameter.
- output: out := eventID returned by database.
- exception: exc := (invalid input || database error ⇒ EventAddError)

editEvent(eventID, updateData):

- transition: eventID ∈ database && updateData == valid ⇒ update event
- output: out := event data returned by database.
- exception: exc := (eventID ∉ database ⇒ NotFoundError || updateData == invalid ⇒ ValidationError)

deleteEvent(eventID):

- transition: eventID $\in$ database $\Rightarrow$ delete event
- exception: exc := (eventID $\notin$ database $\Rightarrow$ NotFoundError)

getGroupEvents(groupID):

- output: out := details of Events associated with groupID in the database.
- exception: exc := (groupID $\notin$ database $\Rightarrow$ NotFoundError)

getUserEvents(userID):

- output: out := details of Events associated with userID in the database.
- exception: exc := (userID $\notin$ database $\Rightarrow$ NotFoundError)

7.4.5 Local Functions

None

8 MIS of Account Module

8.1 Module

Account

8.2 Uses

DatabaseDriver

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
createAccount	userData: Tuple of (userName: string, password: string, firstName: string, lastName: string, phone: string, email: string)	-	UserCreationError
login	email: string, password: string	string	ValidationError
getUserGroups	userID: string	seq of string	NotFoundError
getUser	userID: string	seq of string	NotFoundError
getGroup	groupID: string	seq of string	NotFoundError
logout	-	-	-
deleteAccount	email: string, password: string	-	ValidationError

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

None

8.4.3 Assumptions

All UserIDs and GroupIDs are unique

8.4.4 Access Routine Semantics

createAccount(userData):

- transition: create user in database using userData parameter
- exception: exc := (invalid input || database error $\Rightarrow$ UserCreationError)

login(email, password):

- transition: email $\in$ database && correct password $\Rightarrow$ login user
- ouput: out := userID
- exception: exc := (email $\notin$ database || wrong password $\Rightarrow$ ValidationError)

getUserGroups(userID):

- output: out := (userID $\in$ database $\Rightarrow$ group data)
- exception: exc := (userID $\notin$ database $\Rightarrow$ NotFoundError)

getUser(userID):

- output: out := (userID $\in$ database $\Rightarrow$ user data)
- exception: exc := (userID $\notin$ database $\Rightarrow$ NotFoundError)

getGroup(groupID):

- output: out := (groupID $\in$ database $\Rightarrow$ group data)
- exception: exc := (groupID $\notin$ database $\Rightarrow$ NotFoundError)

logout():

- transition: log out user in database
- exception: none

deleteAccount(email, password):

- transition: email $\in$ database && correct password $\Rightarrow$ delete user
- exception: exc := (email $\notin$ database || wrong password $\Rightarrow$ ValidationError)

8.4.5 Local Functions

None

9 MIS of Database Driver Module

9.1 Module

Database

9.2 Uses

None

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
insert	tableName: string, values: string	-	-
update	tableName: string, values: string, conditions: string	-	-
delete	tableName: string, conditions: string	-	-
select	fields: string, tableName: string, conditions: string	string	-

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

Database db

9.4.3 Assumptions

- The schema is set up as described in Section 10.1
- The connection to the database is established with necessary permissions.

9.4.4 Access Routine Semantics

insert(tableName, values):

- transition: the target table in *db* specified by *tableName* adds *values*.
- output: none
- exception: none

update(tableName, values, conditions):

- transition: the target items in *db* specified by *tableName* and *conditions* is updated with *values*.
- output: none
- exception: none

delete(tableName, conditions):

- transition: the target items in *db* specified by *tableName* and *conditions* are deleted.
- output: none
- exception: none

select(fields, tableName, conditions):

- transition: none
- output: *out* := *data*, where *data* describes the fields specified by *fields* of the target items specified by *tableName* and *conditions* in db.
- exception: none

9.4.5 Local Functions

None

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

10 Appendix

10.1 Database Specification

In this section, the description of the database schema of Housemates will be provided. The database for Housemates will be relatively simple with only a few entities (account, user, task, group, events, bills) which cover the main functionalities of Housemates. The relationships between these entities are described in Figure 1.

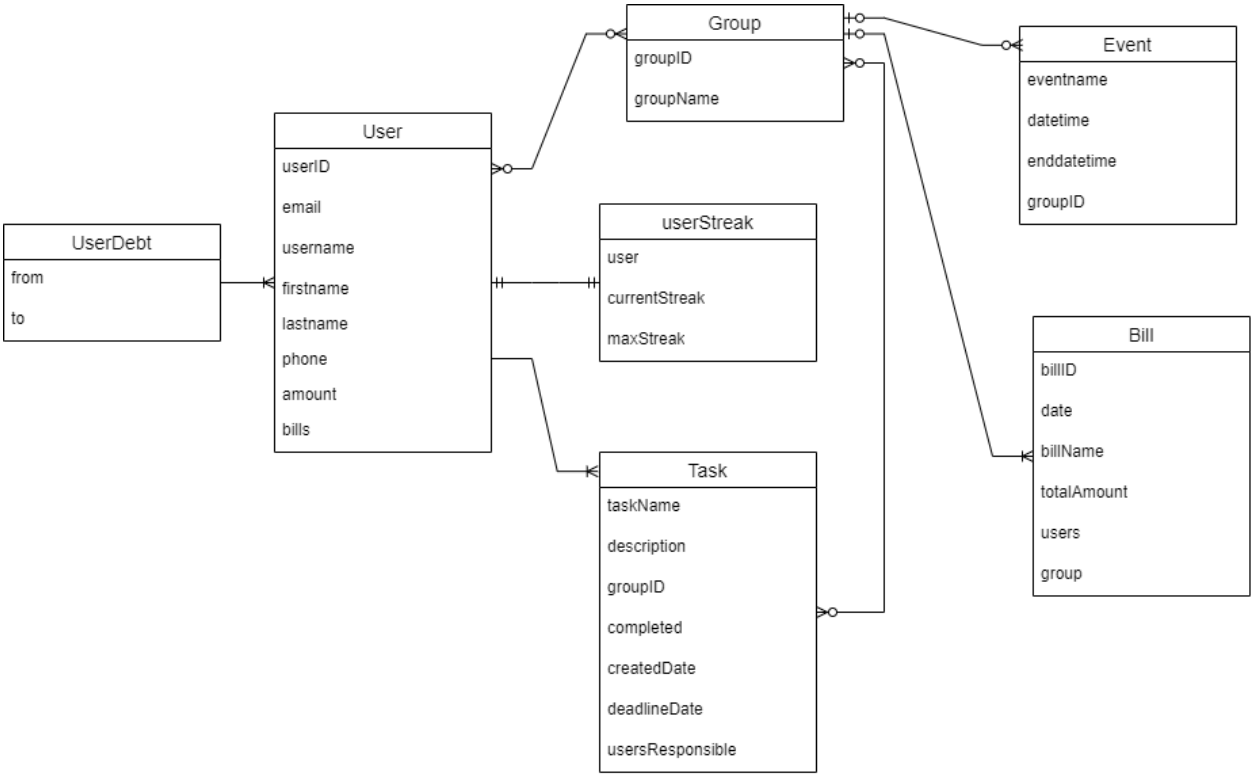

Figure 1: Entity-Relationship Diagram of the Housemates Database

10.2 Interface Specification

In this section, the description of the user interface of Housemates will be provided. The user interface of Housemates is designed to be minimalist and simple to use. This will allow the users of Housemates to quickly access the main functions of Housemates. Some examples of the interface are described in the figures below and at https://www.figma.com/file/lMZxonql0nhowgpIslIwns/Housemates-Interface-Design?type=design&node-id=0%3A1&mode=design&t=iuU1JQzgxRP93dCL-1. The interface for Housemates may change in the final implementation.

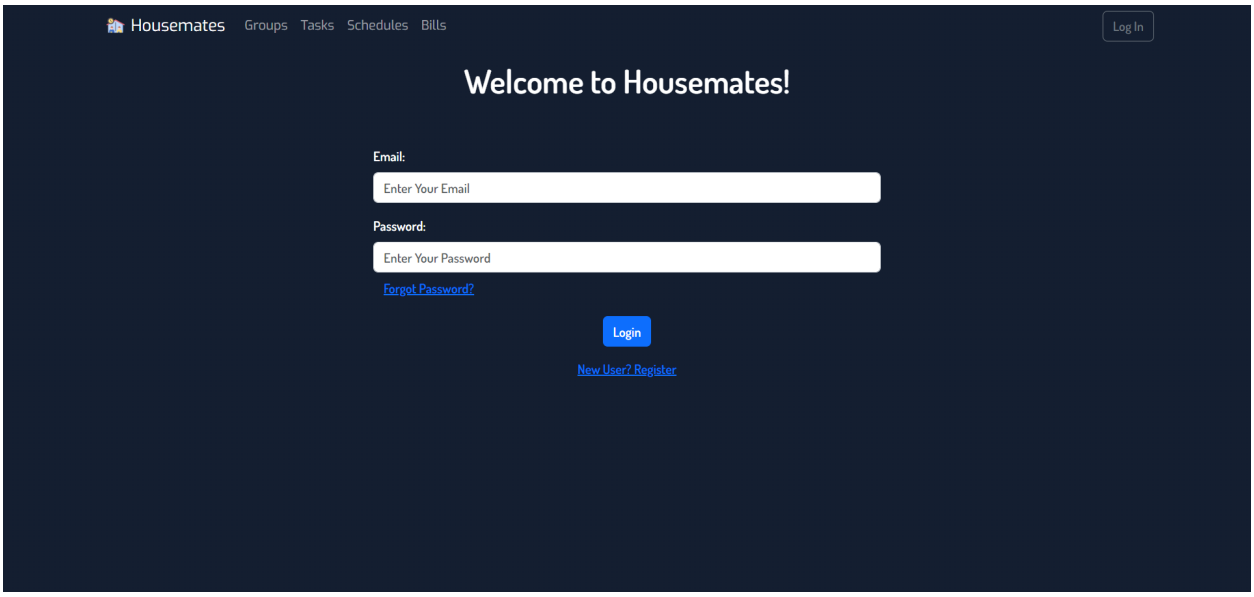

Figure 2: Login screen of Housemates

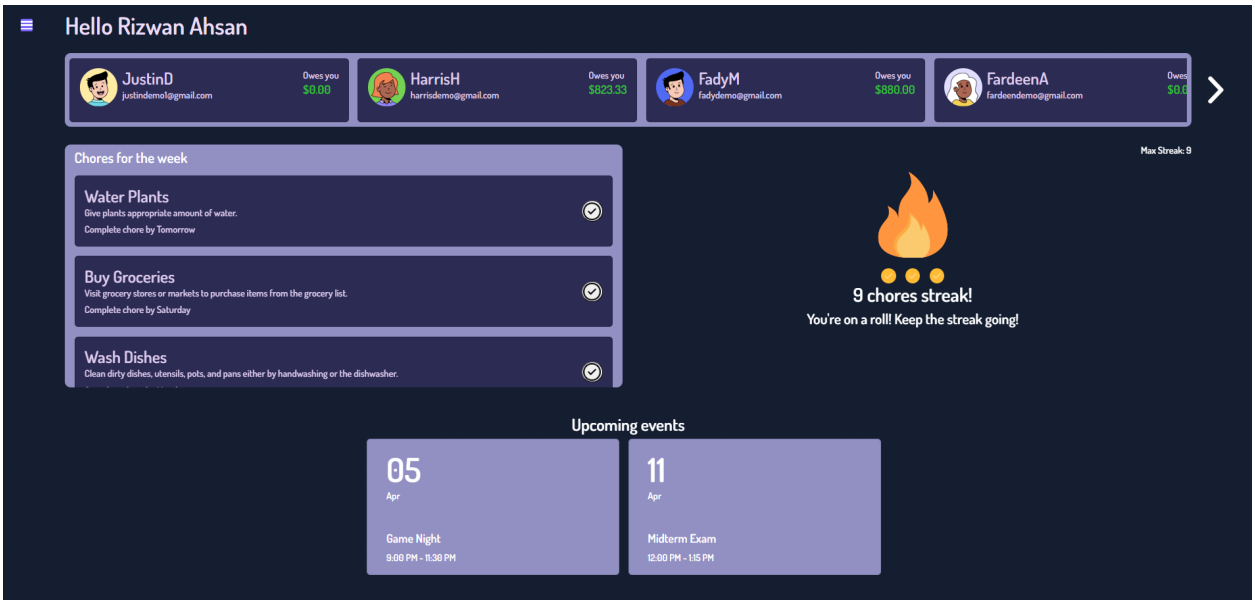

Figure 3: Homescreen of Housemates

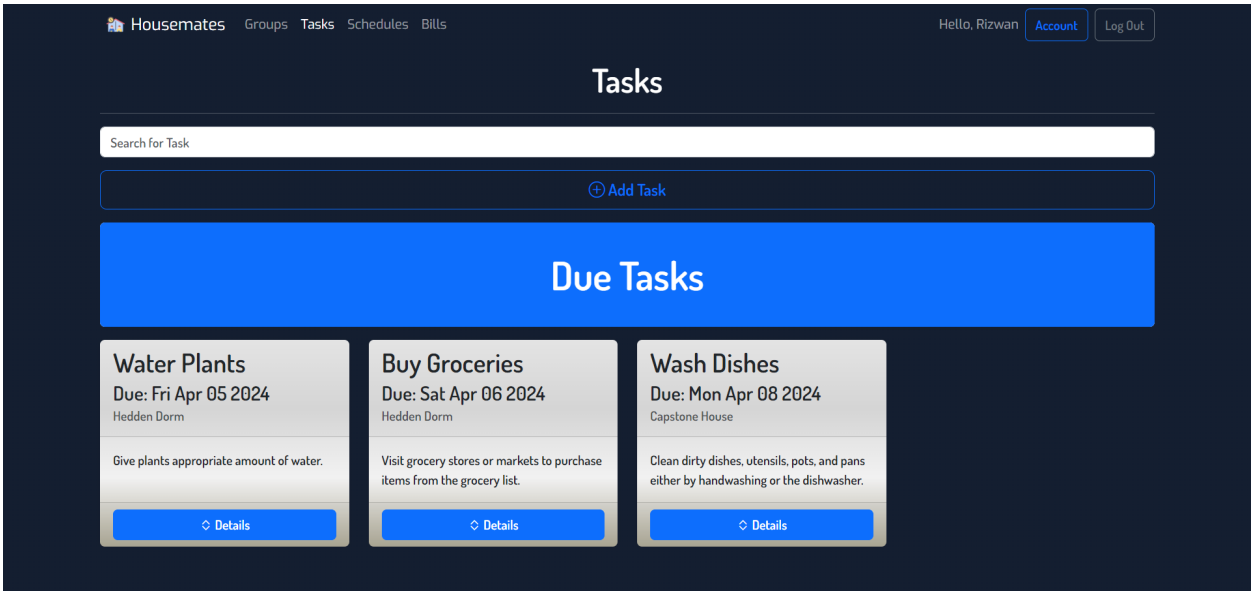

Figure 4: Task screen of Housemates

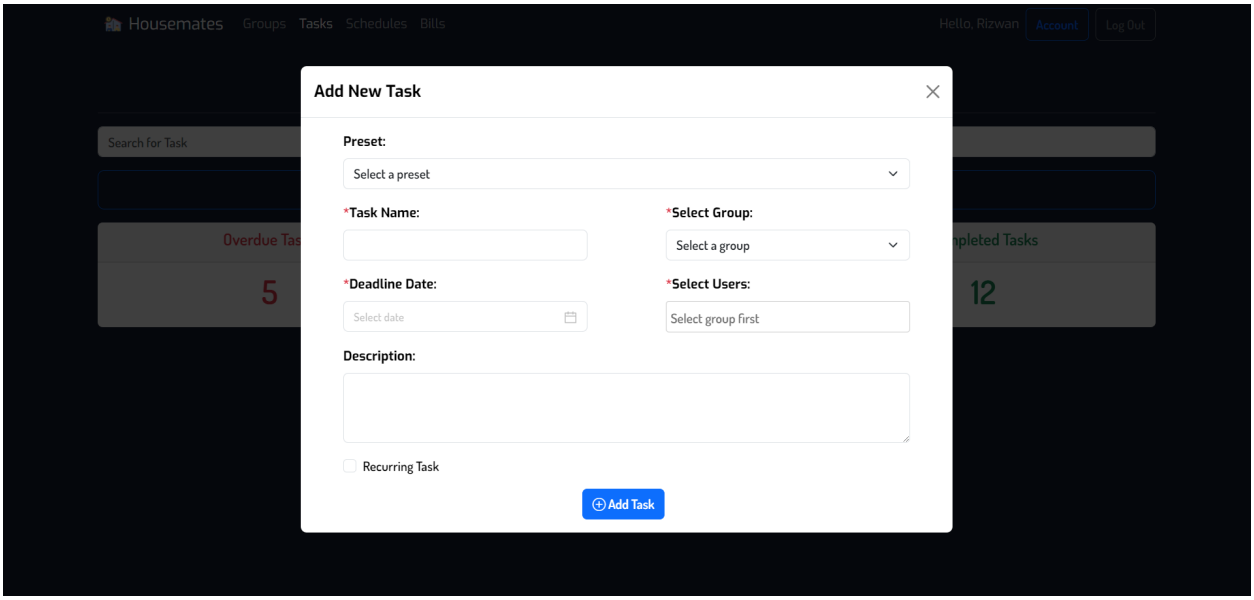

Figure 5: Add tasks screen of Housemates

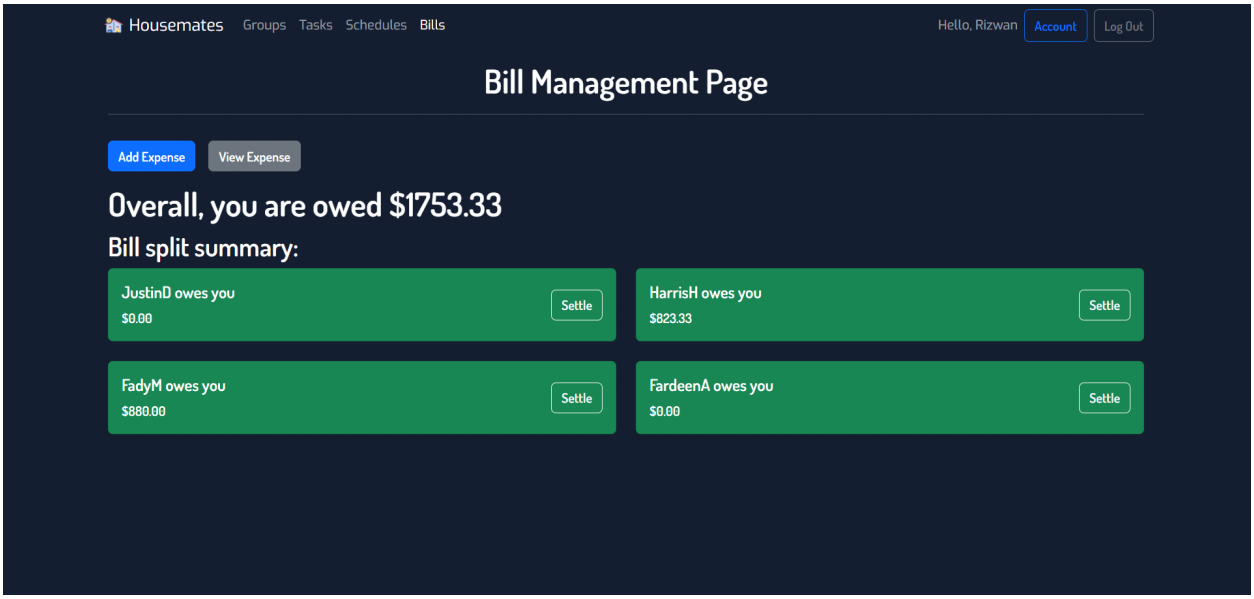

Figure 6: Bills screen of Housemates

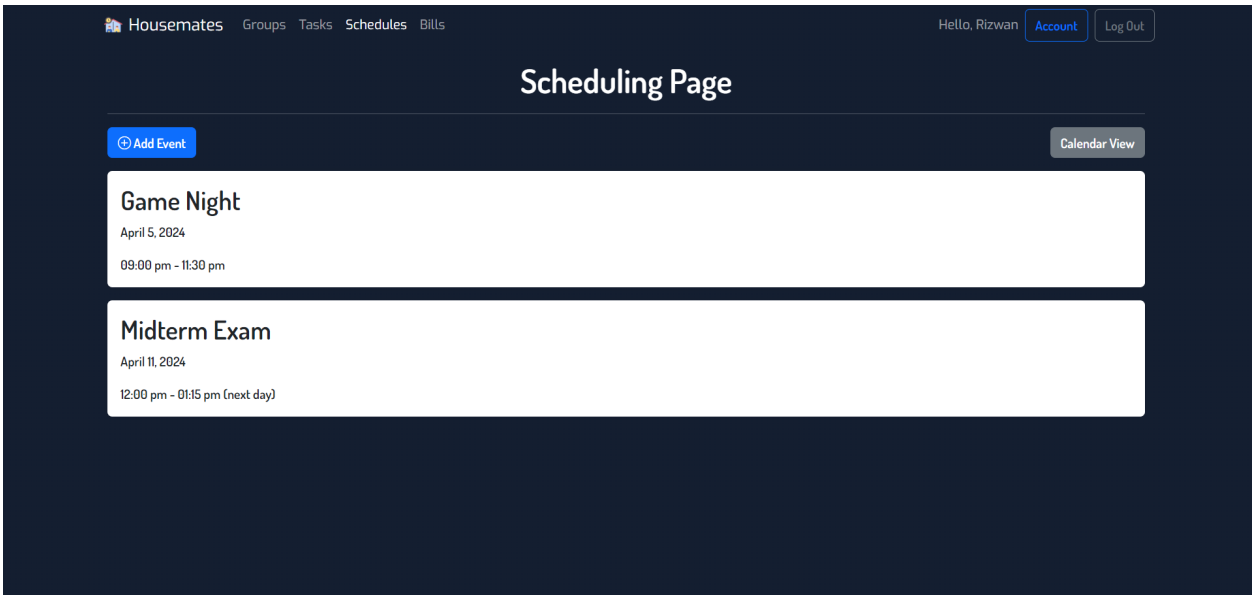

Figure 7: Scheduling screen of Housemates