Module Interface Specification for Housemates

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

January 17, 2024

1 Revision History

Date	Version	Notes
January 17, 2024	1.0	Revision 0

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 Interface Design Module
Software Decision Module	Cryptography Module Database Interface Module Network Interface Module

Table 1: Module Hierarchy

5 MIS of Task Management Module

5.1 Module

taskManagementModule

5.2 Uses

accountModule, interfaceDesignModule, databaseInterfaceModule

5.3 Syntax

5.3.1 Exported Constants

None

5.3.2 Exported Access Programs

Name	In	Out	Exceptions
addTask	taskData: Tuple(users: seq of $\mathbb{Z}$ uID, details: seq of char)	tID: $\mathbb{Z}$	TaskConflictError, ValidationError
updateTask	tID: $\mathbb{Z}$, details: seq of string	-	-
findTasks	-	tIDs: seq of $\mathbb{Z}$	-
getTaskDetails	tID: $\mathbb{Z}$	details: seq of string	DocumentNotFoundError
completeTask	tID: $\mathbb{Z}$	-	-

5.4 Semantics

5.4.1 State Variables

None

5.4.2 Environment Variables

None

5.4.3 Assumptions

All tIDs are unique.

5.4.4 Access Routine Semantics

addTask(taskData):

- transition: Once the function validates the input data, task is added into the database using taskData.
- out: tID := tID returned by database
- exception: Task is not added due to conflict, or due to invalid input

updateTask(tID, details):

- transition: update task in the database associated with tID in database with details parameter.
- exception: none

findTasks():

- output: tIDs := tIDs of tasks associated with current uID from account module.
- exception: none

getTaskDetails(tID):

- transition: Function will find task with the associated tID.
- output: details := details of task associated with tID stored in database.
- exception: Task does not exist

completeTask(tID):

- transition: Function will remove task from list of current tasks.
- output: details := details of task associated with tID given in database.
- exception: none

5.4.5 Local Functions

None

6 MIS of Bill Management Module

6.1 Module

billManagementModule

6.2 Uses

accountModule, interfaceDesignModule, databaseInterfaceModule

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
addBill	billData: Tuple of (billName: string, billDescription: string, gID: $\mathbb{Z}$, amount: $\mathbb{R}$)	bID: $\mathbb{Z}$	ValidationError
updateBill	-	bID: $\mathbb{Z}$	ValidationError
findBills	-	bIDs: seq of $\mathbb{Z}$	DocumentNotFoundError
getBillDetails	bID: $\mathbb{Z}$	details: seq of string	DocumentNotFoundError
completeBill	bID: $\mathbb{Z}$	-	-

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

All bIDs are unique.

6.4.4 Access Routine Semantics

addBill(billData):

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

- output: bID := bID returned by database
- exception: ValidationError

updateBill(billData):

- transition: Once the function validates the input data which can be null, bill is updated into the database using billData.
- output: bID := bID returned by database
- exception: ValidationError

findBills():

- output: bID := bID tasks associated with current uID from account module.
- exception: none

getBillDetails(bID):

- transition: Function will find bill with associated bID.
- output: details := details of bill associated with bID stored in database
- exception: bill does not exist

completeBill(bID):

- transition: Function will remove bill from list of current bills.
- output: details := details of bill associated with bID given in database.
- exception: none

6.4.5 Local Functions

Name	**In**	**Out**	**Exceptions**
addAttachment	img: 2D seq of pixels	-	ValidationError

7 MIS of Scheduling Module

7.1 Module

schedulingModule

7.2 Uses

databaseInterfaceModule, interfaceDesignModule, accountModule

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
addEvent	eventData: Tuple of (eventName: string, gID: $\mathbb{Z}$, date: string, time: string)	eID: $\mathbb{Z}$	ValidationError
updateEvent	eID: $\mathbb{Z}$, details: seq of string	-	ValidationError, DocumentNotFoundError
findEvents	-	eIDs: seq of $\mathbb{Z}$	ValidationError, DocumentNotFoundError
getEventDetails	eID: $\mathbb{Z}$	details: seq of string	DocumentNotFoundError

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

All EventIDs (eID) are unique.

7.4.4 Access Routine Semantics

addEvent(eventData):

- transition: create event in database using eventData parameter.
- output: eID := EventID returned by database.
- exception: ValidationError

updateEvent(eID, details):

- transition: update event associated to eID in database with details parameter.

- exception: ValidationError, DocumentNotFoundError

findEvents():

- output: eIDs:= EventIDs of events associated with uID from accountModule, sorted by date and time.
- exception: ValidationError, DocumentNotFoundError

getEventDetails(eID):

- output: details := details of Event associated with eID in the database.
- exception: DocumentNotFoundError

7.4.5 Local Functions

None

8 MIS of Account Module

8.1 Module

accountModule

8.2 Uses

databaseInterfaceModule, interfaceDesignModule

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)	uID: $\mathbb{Z}$	ValidationError
login	username: string, password: string	uID: $\mathbb{Z}$	ValidationError, DocumentNotFoundError
getUserID	-	uID: $\mathbb{Z}$	-
getUserDetails	-	details: seq of string	-
updateDetails	details: seq of string	-	ValidationError
createGroup	groupData: Tuple of (uIDs: seq of $\mathbb{Z}$, groupName: string)	gID: $\mathbb{Z}$	ValidationError
getGroups	-	gIDs: seq of $\mathbb{Z}$	-
getGroupDetails	gID: $\mathbb{Z}$	details: seq of string	DocumentNotFoundError
logout	-	-	-
deleteAccount	username: string, password: string	-	ValidationError

8.4 Semantics

8.4.1 State Variables

uID: $\mathbb{Z}$

8.4.2 Environment Variables

None

8.4.3 Assumptions

All UserIds (uID) and GroupIds (gID) are unique

8.4.4 Access Routine Semantics

createAccount(userData):

- transition: create user in database using userData parameter; self.uID := uID returned by database
- ouput: uID := self.uID

- exception: ValidationError

login(username, password):

- transition: login user in database; self.uID := uID associated with username and password in database.
- ouput: uID := self.uID
- exception: ValidationError, DocumentNotFoundError

getUserID():

- output: uID := self.uID
- exception: none

getUserDetails():

- output: details := details of user in database associated with self.uID in database.
- exception: none

updateDetails(details):

- transition: update details of user in database associated with self.uID in database with the parameter userDetails.
- exception: ValidationError

createGroup(groupData):

- transition: create group in database using self.uID and groupData parameter;
- output: gID := gID returned by database.
- exception: ValidationError

getGroupDetails(gID):

- output: details := details of user in database associated with gID in database.
- exception: ValidationError, DocumentNotFoundError

logout(username, password):

- transition: log out user in database; self.uID := none;
- exception: none

deleteAccount(username, password):

- transition: delete user associated with username, password parameters in database; self.uID := none.
- exception: ValidationError

8.4.5 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.

9 Appendix

9.1 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.

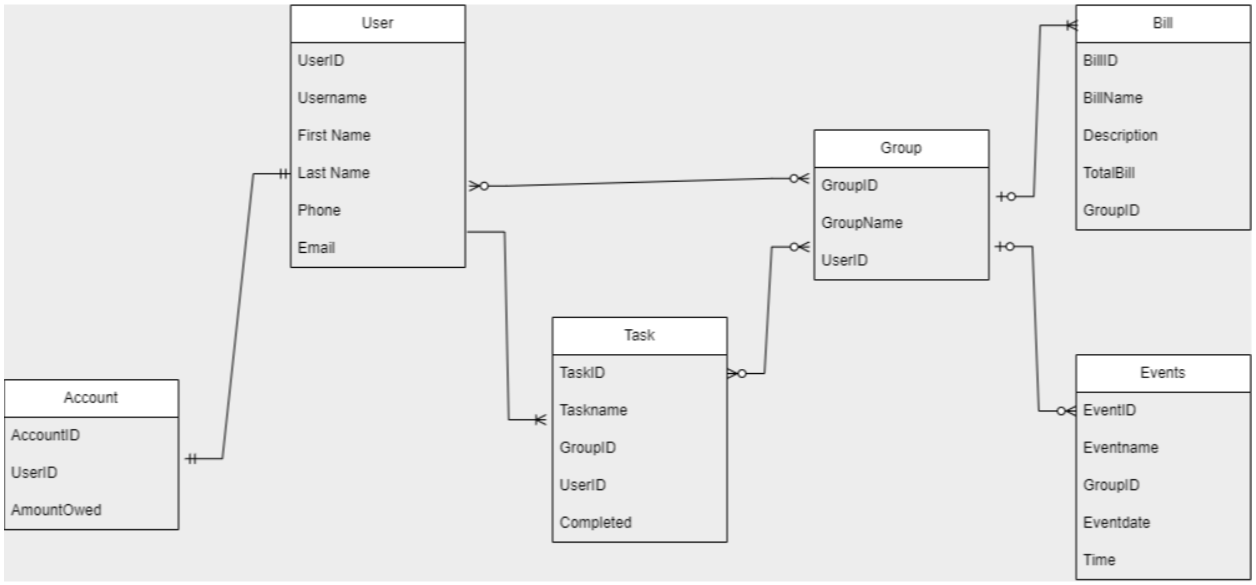


Figure 1: Entity-Relationship Diagram of the Housemates Database

9.2 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.

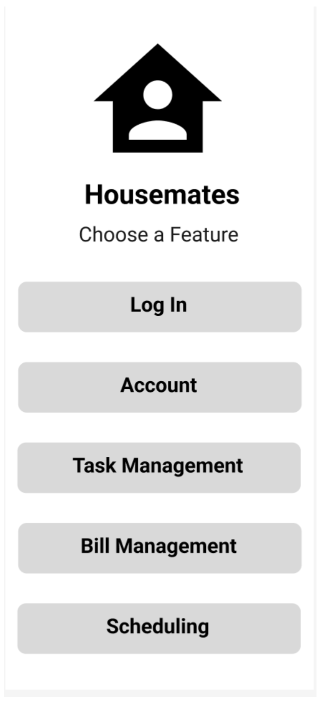


Figure 2: Homescreen of Housemates

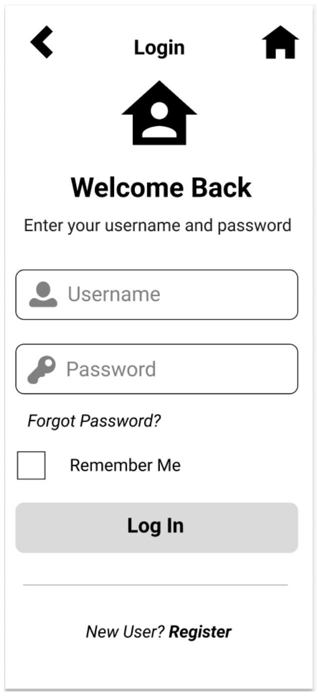
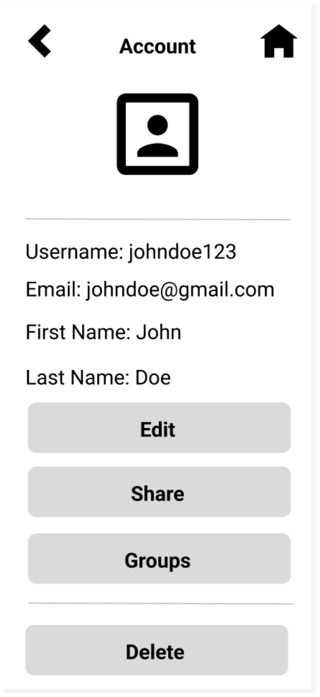


Figure 3: Login screen of Housemates



Account

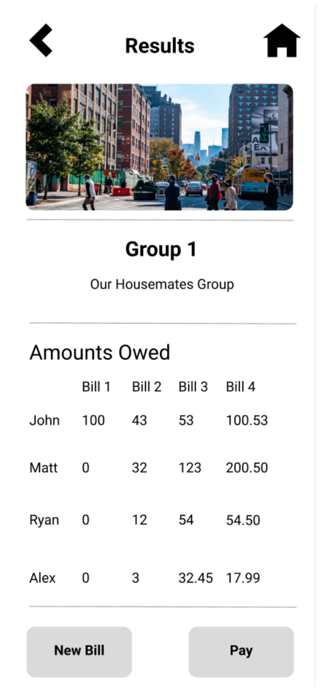
Username: johndoe123

Email: johndoe@gmail.com

First Name: John

Last Name: Doe

Figure 4: Account screen of Housemates



Results

Group 1

Our Housemates Group

Amounts Owed

	Bill 1	Bill 2	Bill 3	Bill 4
John	100	43	53	100.53
Matt	0	32	123	200.50
Ryan	0	12	54	54.50
Alex	0	3	32.45	17.99

Figure 5: Bills screen of Housemates

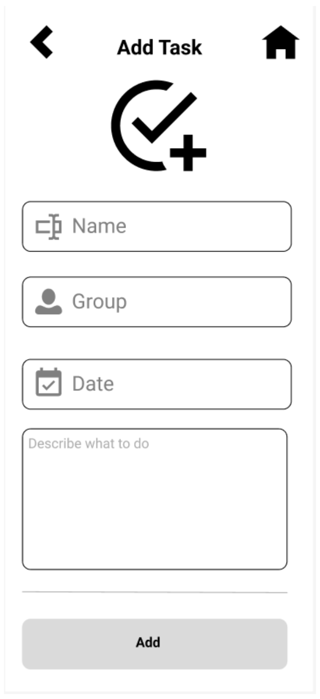


Figure 6: Add tasks screen of Housemates

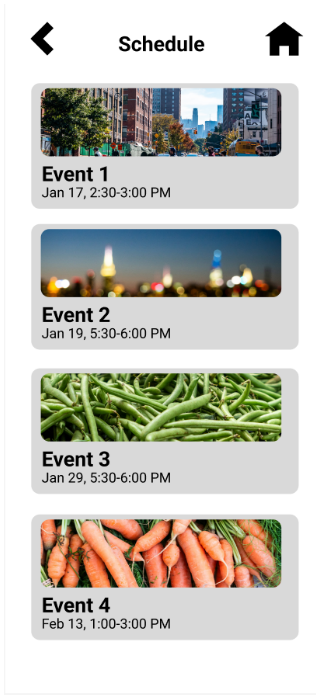


Figure 7: Events screen of Housemates