RoomEase Software Requirements Specification

Alex Vrhel (avrhel) **Cheryl Wang (cwang7)** **Jakob Sunde (jsunde)** **Matt Mans (mans1626)**

Omar AlSughayer (oasugher) **Sid Gorti (sgorti3)** **Weijia Dai (weijid)**

Project Description

What is it:

RoomEase is a roommate coordination app which provides a better experience of living together. Its main goal is to help roommates manage shared resources and monitor responsibilities.

Problems it solves:

· One's routine could collide with that of their roommates. E.g. watching different TV shows at the same time or taking morning showers around the same time.
· Food in common storage units would often rot before anyone have removed it.
· Meeting with all roommates to create a shared document, such as a roommate agreement or a shopping list, and then handing everyone a copy tends to be hard. Making sure that every roommate has a copy of said document is equally tough as well.
· Dividing chores could be a mess, as well as remembering whose job was to do what and if it was completed, or even to do your own.

Features:

Reservation *Main Goal*
Allows users to reserve any of the common resources (e.g. TV, kitchen, etc.) for a specified period of time.

Food Management *Main Goal*
List all food items in the fridge or storage, who is it owned by, and the expiration date on every item. This feature also notifies users when an item approaches its expiration date regardless of owner.

Shared List *Main Goal*
Gives the option of creating mutable or immutable lists that can be viewed by all or some roommates.

Reminders Feed *Main Goal*
Personal, regularly updated feed of chores, reservations, nearly-expired food, and to-do's.

Payment *Stretch Goal*
A direct implementation of *Venmo* which processes money transactions that could occur between roommates for any given reason. E.g. buying groceries or the paying the rent.

Gamification *Stretch Goal*
Tracks missed chores while granting points to completed ones. Punishment and award system is left for the users to decide.

Who's Home *Stretch Goal*
An opt-in option for users to know who is currently at home. If not, it does not provide any additional information to a user's whereabouts.

Chores Preferences *Stretch Goal*

Each user has the option to arrange chores from most to least preferred. Users' preferences are to be taken into consideration when dividing chores.

Alternatives:

Roommates could choose to coordinate amongst each other verbally, by using any texting service, or a standard scheduling app. What sets RoomEase aside from all of this is that integrates all of these into one and provides a record of past interactions as well.

Product v. RoomEase

Home Slice

Home Slice can be used in only two areas when put in contrast to RoomEase. It functions either as a shared list, or to keep track of roommates' debts to each other. It does not provide any of the extra functionality that RoomEase provides.

Roommates

Although Roommates provides most of the functionality RoomEase intends to, it falls short with its user-unfriendly UI that can be hard to understand. It also does not attempt to address the food management problem that RoomEase solves.

My Roommates

Like Roommates, My Roommates does not provide any food management functionality either. It is also only available on Android which makes it less ideal when roommates do not use the same operating system.

Non-functional requirements:

Usability

We recognize that having to type into your phone the expiry date of every item you buy is annoying, and so is typing before using the TV. Therefore, every task accomplished by RoomEase should be achievable in no more than a few buttons presses as not to bore the user away.

Availability

To ease communication between roommates with different operating systems, RoomEase has to be runnable cross-platforms. Mainly on iOS and Android.

Performance

Under optimal network conditions, no screen should load for more than two seconds. Elsewise, users will be less willing to use our product.

Accessibility

Despite using server-stored databases, our app should still be accessible offline. Other than Who's Home, the latest-viewed version of the contents of the fridge, shared lists, reservations, and reminders feed should be available.

Documentation:

RoomEase's UI is going to be enough to explain the flow of the program and how tasks are completed. However, for the case when a user cannot navigate their way solely, we are going to include a HELP section in-app. Users also will be able to report any bug they encountered to the developers through a Google form linked to within the app.

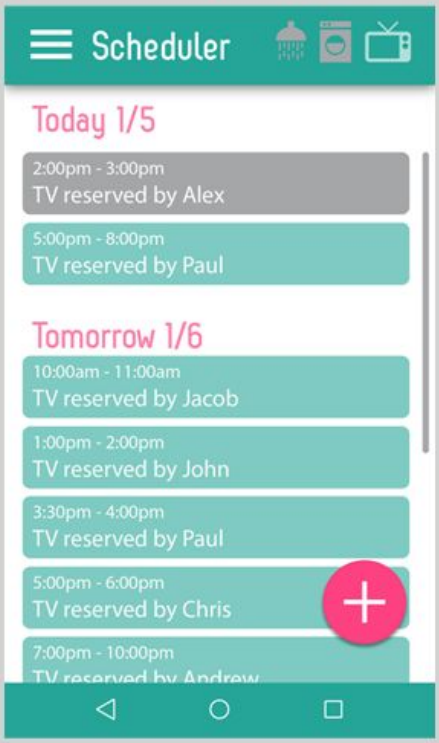
UI Diagrams


II. Adding Item to Fridge/Pantry
1
Fridge
Mine
Shared
Milk Expired
Eggs Expire in 3 days
Shared
Cheddar Cheese
Expire in a week
2a
Fridge
Mine
Shared
Milk Expired
Eggs Expire in 3 days
Shared
Cheddar Cheese
Expire in a week
Add Item
Name
Expires in days
Not Shared
Shared
Done
Next Item
3
Fridge
Mine
Shared
Milk Expired
Eggs Expire in 3 days
Shared
Cheddar Cheese
Expire in a week
Add Item
Name Ice Cream
Expires in 14 days
Not Shared
Shared
Done
Next Item
4
Fridge
Mine
Shared
Milk Expired
Eggs Expire in 3 days
Shared
Cheddar Cheese
Expire in a week
Ice Cream
Expire two weeks

III. Adding Item to shared list (Scheduler)

1

2-3



In this special case the content of the item being added is decided by which resource (either shower, laundry, or TV) is currently selected.

4

4-5

In this case the calendar and clock are used to enter the times in which they would like reserve the current resource.

Use Cases

I. Signing up with roommates

Precondition: User has downloaded the app
Trigger: User opens the app for the first time
Success condition: User has created a RoomEase group with their roommates
End failure condition: User has not created a RoomEase group, or their roommates could not all join it
Steps to success condition:

1. User authenticates their identity via facebook login
2. User indicates they want to create a new group
3. User is prompted to invite their roommates via email address
4. User's roommates download the application and perform step 1
5. User's roommates indicate they want to join the group
6. User's roommates use the contents of the email invitation to indicate they will join the user's group
7. All roommates are logged in and in a group together

Extensions (error scenarios):

1a. User does not provide correct Facebook login info
    1a.1 App returns user to login screen
    1a.2 User tries again or backs out
3a. One or more of the entered email addresses is not valid
    3a.1 User will be informed that an email did not send
    3a.2 User will attempt to enter a different email and re-send the invitation
6a. Roommate enters incorrect information when attempting to join user's group
    6a.1 App displays a message informing them the group was not found
    6a.2 Roommate tries to enter the information again or backs out

Variations:

3a. Upon creation of group, user invites their roommates via Facebook
    3a.1 Invitations to join group are sent to roommates
    3a.2 Roommates can join group by following a link in the invitation, bypassing steps 5 and 6

II. Adding an item to the fridge/pantry

Precondition: User is logged into the app
Trigger: User goes to the food management screen while putting away food
Success condition: User's food item and its relevant info is visible to their group
End failure condition: User's food item is not entered into the system
Steps to success condition:

1. User presses a button indicating they want to add a food item
2. Application prompts user to enter information about the item
    a. User is prompted for name, expiration date, and sharing settings
3. User enters the relevant information and puts the food item away
4. Food item and its information is now visible to all roommates in the group

III. Adding an item to a shared list

Precondition: User is logged into the app
Trigger: User has the app open and wants to add an item to a list for their group to see
Success condition: The item has been added to the list, all group members can see it
End failure condition: The item was not added, group does not see the new item
Steps to success condition:

1. User navigates to one of the shared lists screen on the application
2. User selects the list that they would like to add to, from a selection of all existing lists for their group
3. User presses a + button to add a new item to the list
4. User enters the content of the item they would like to add in a pop-up
5. User indicates they are done, and the item is added to the specified list, now visible to all group members

Variations:
1. “A shared list” may refer to many of the modules in the application (which follow this use case with only slight differences in the details of the UI)
    1a. User navigates to the chore schedule screen to add a new chore
    1b. User navigates to the shared list screen to add something to a list
    1c. User navigates to the scheduler screen to reserve a resource
2a. User may add a new list rather than selecting an existing list
    2a.1 User presses a + button before selecting any existing list
    2a.2 User enters a name for the new list

IV. Editing an item on a shared list

Precondition: User is logged into the app, at least one list has an item on it
Trigger: User wishes to change an item (either notification settings or content) of an item on a list
Success condition: The item has been modified on the list, all group members can see the change
End failure condition: The item was not added, group does not see the changes for the item
Steps to success condition:

1. User selects the “shared lists” screen from the menu at the landing screen
2. User clicks the list that they would like to edit
3. User clicks the item that they want to change
4. System prompts user whether they will be deleting or changing the item
5. User indicates they want to change the item
6. User edits the content or notification settings of the item as desired
7. User confirms their changes to the item

Extensions (error scenarios):
7a. User attempts to delete/edit an item while their roommate also edits/deletes the same item
    7a.1 System will reflect the changes that were made and confirmed most recently
    7a.2 User whose changes were overwritten backs out, accepting their roommate’s edit, or tries again

Variations:
    3a. User may change the list as a whole rather than a single item
        3a.1 User interacts with the title of the list

3a.2 Use case continues as above, but for the list rather than a single item

5a. User indicates they want to delete the item

5a.1 System prompts user to ensure deletion is intentional

5a.2 User confirms and the item is deleted from the list, bypassing steps 6,7

For the use cases above, the main failure condition would be caused by an error in the system (the changes made on a user's app aren't properly transmitted to the data that our system stores), which will ideally be ironed out as we design our front end and back end, or by an issue in the user interface (such as confusion in how to proceed with their desired action). Since most of the use cases are simple sequences of operations within the UI, we will make it a main goal of our project that the UI flows logically and is very responsive, so that users don't fail these use cases due to confusion or frustration.

We feel that the use cases listed above cover the important scenarios, because adding items to lists and editing/removing those items are the core action behind many of the different modules of the application (including the fridge management, shared lists, chore schedules, and reservation portions of the app). Although the cases may vary slightly, such as what options need to be entered in or the details of the UI at these different screens, the core process is very similar. Thus, outlining how to sign up and how to interact with these list modules covers the functionality of the app at a high level.

Process Description

Software Toolset

Client Application:

The client application will be developed using Phonegap. Phonegap uses CSS, HTML and Javascript for designing the client application. We decided to use Phonegap, since it allows us to develop one application and compile for multiple platforms. This is important for development, as we want to deploy for Android and IOS to allow nearly all roommates to use the application. Development for IOS and Android has a steep learning curve, and given the tight timeline, it is easier to learn HTML/CSS/JS from scratch than development for IOS or Android.

We will use Materialize for stylizing the application. Materialize provides CSS/HTML/JS implementations for common UI animations and layouts, and therefore we will not have to spend time developing this portion of the UI.

Server Application:

The application will be accessing resources from the internet and therefore a server application will be required. The server application will be developed using Node.JS. Node.JS is relatively simple for new users to learn, and since PHP is relatively difficult to develop with, Node.JS is the best option.

The server application will be hosted using AWS for the reliability and ease of use of the service.

Database:

The application needs to access data from a database and push requests offline and therefore a database is needed. PouchDB will be used for queueing requests offline then pushing them into CouchDB.

Source Control:

We will use git and host our project on github, since all team member know git and hosting on github allows us to make our source code public.

Task Tracking and Bug Reporting:

We will use Google Docs to keep a list of what tasks are currently being worked on, what tasks are completed, what needs to be completed and what bugs are currently present in the application. A formal task tracking tool is unnecessary for a project of this scope.

Group Dynamics

Alex is serving as the team's Project Manager, and all other group members will share in development, with some specialization. Specifically, Cheryl, Sidd, and Jakob will be working primarily on front end development and UI design, while Matt, Omar, and Weijia will be working more on the back end of the application. These roles were chosen according to our group member's preferences and strengths, although they are flexible and may change as we progress and help is needed in various areas. Disagreements will be handled by taking the issue to the group and deciding democratically, which has proved easy so far due to our open communication channel through Slack. As the PM, Alex is willing to step in to resolve conflicts and mediate issues between group members if necessary. We feel that deciding things as a group when possible is the best method, since everyone has a chance to make their voice heard and be a part of the decision.

Schedule/Timeline

Set** / **Flexible

Jan 27 23:59**:**

Everyone: Polished draft of software design spec (for review by TA)

Back-end: AWS & DynamoDB set up

Feb 01: Software design specification

Feb 06 23:59**:**

Back-end: finish shared list feature (does not need to be linked up to 0-feature release)

- *Some communication with the app & DB set up*

Front-end: main pages & ½ of module pages need to be set up (for each main feature)

Feb 08: Zero-feature release

Feb 12 23:59:

Back-end: notifications feed works & reservations

Front-end: UI is linked up with shared list & prepared to integrate with notifications feed

- *Facebook user login*

Feb 16 23:59

Back-end: Reservation feature works

Front-end: full interactivity & integration with backend

Feb 17-18

Testing

Feb 19: Beta Release

Feb 23 23:59

Front end and back end are fully connected, all core functionality complete

Feb 24 - 25

App has been thoroughly tested for Android & iOS devices

Feb 26: Feature-complete release

Feb 26 - March 03

More testing/Early user testing; stretch goals?

March 04: Release candidate & user testing

March 04 - 08

More testing/User testing; stretch goals?

March 08: Final release

JUSTIFICATIONS:

We want half of our app's functionality by the zero feature release because of the short amount of time between ZFR and beta release. Much of our app is UI based and it is critical that we leave enough time for testing, and get features working early enough to get user feedback. For our back end, we want to make sure that our app can communicate with our database by ZFR, since we need to have our back end working and communicating with both our app and our database by the beta release. We want to do at least some early user testing so we can identify early on what some of the potential issues may be.

Risk Summary

1. **Risks to the project caused by inadequate or untestable requirements.**
   Coming up with good and complete requirements is difficult. The functionality of RoomEase are hard to convey precisely and in detail. Moreover, the requirements are hard to communicate effectively for a group of size 7. To reduce this risk, we will try to make our project requirements as detailed as possible during the early stages of development. This also forces us to actively receive feedback from TAs early so we reduce the risk of needing to change our requirements in the future. If we still encounter any problems caused by inadequate requirements, we may adjust our requirements and change our approach so we can deliver our product on time.

2. **Risks of changing software toolkit**
   It is possible that some of the software tools we listed do not work as we expected, or the learning curve associated with the tool is too high. In that case, we will need to find alternative tools. To reduce the risk of changing tools, early on we will experiment with the tools we have chosen so we can see if we need to change the tools we need to use. Even though it is unlikely to happen, if we realise we need to change a tool in the middle of development, we will have to adjust our group dynamics, having one person search for alternatives and others making sure all the other parts of the project are no longer dependent on the old tool. Considering the ramp-up time of getting familiar with a new tool, we will have to change our schedule to accommodate.

3. **Risk of not achieving deadlines**
   Even though we have set milestones for our project, things can go wrong and this makes estimating progress difficult. In order to meet the deadlines, our group will have team meetings frequently to assess our progress. We will also aim to finish things ahead of

the specified deadline so we have extra time in case things do not go as planned. When things do go wrong, we will shift the duties of different teammates to help address the issue at hand. If we encounter a major issue that pushes our deadlines extremely far back, we may have to cut features.

4. **Risks of non user friendly UI**
   We are trying our best to make our UI clear and user friendly. To ensure usability, we have had multiple team meetings exclusively for the UI design, and we will continue to do so as the project progresses. Also, we will make sure to continuously show the UI design to our TAs and friends to continuously improve it based on feedback. If we have to modify our UI after it is linked with the backend, we will have to make sure that our changes to the front end do not require heavy changes to the backend. Also we will adjust our group dynamics as needed.
5. **Risks of failing to deliver the app of both Android and iOS**
   One of the main things that sets aside from our competitors is delivering our app cross-platform which allows nearly all roommates to use the app. Failing to do so could render our app useless as limiting the usage to half a house's population does not solve the problems the app is set to solve. Although PhoneGap works greatly to minimize this risk, plugins and platform-specific pugs still raise the same risk.

Generally speaking, the feedback from users will the most useful at the beginning of project development. At initial stage, things are not settled yet and the cost of making even drastic changes is comparatively low. Therefore, we will seize our time and ask for as much feedback as possible before our zero-feature release.

Design Changes and Rationale, Beta Release 02/19/2016:

Project Description

- Features.Shared Lists: This feature will *not* come with a built-in Roommate Agreement list that the user is prompted add.
    - We removed this built-in list because we think that the majority of the users will not use it; and it is distracting from the main purpose behind shared lists.
- Documentation: We are going to add the ability to report bugs to the developers.
    - The in-app HELP section and the UI should be more than enough to guide the user through the app. HELP, however, won't be beneficial to the user when they encounter a bug in the system. Therefore we had to add a bug report.

Use Cases

- Use Case II.2.a: Expiration date of an item is to be inputted as a calendar date instead of number of days from present.
    - We chose to not prompt the user to enter expiration dates in number of days from present because we think it is a burden for the user to have to calculate this number.

Process Description

- Databases Backend is now implemented using PouchDB and CouchDB instead of DynamoDB and AWS.
    - Through development we decided that we want the user to be able to push requests to the database while offline. PouchDB allows the functionality of queueing requests offline then pushing them into a CouchDB database once an Internet connection have been established, while DynamoDB does not. For this reason we decided to switch away from DynamoDB.
- Schedule.Feb 16: Fridge management feature is not to be completed by this date anymore. Instead, Facebook login, shared lists, and reservations are fully functional.
    - We originally decided to first finish implementing shared lists completely, by then we recognize that reservations share a much closer resemblance to shared lists. So, it was more logical to build reservations on top of shared lists before implementing fridge managment.
- Risk Summary: Add 5. Risk of failing to deliver our product on both Android and iOS.
    - Having our app run cross-platform is essential to set it apart from our competitors and allow for full communication among roommates. Initially, we did not consider this a risk because PhoneGap advertises itself as being fully functional on Andriod and iOS. We later found out though, while implementing Facebook login, that PhoneGap's plugins and Facebook's developers restrictions could restrict the compatibility of our code cross-platform. We also encountered platform-specific bugs that required many hours of debugging. All of these issues have been solved so far but a new risk have been realized.