Verification and Validation Report: Room8

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

March 10, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

Acronym	description
SRS	Software Requirements Specification
VnV	Verification and Validation
CI/CD	Continuous Integration and Continuous deployment
API	Application Programming Interface
Exp.	Expected
Act.	Actual

[symbols, abbreviations or acronyms – you can reference the SRS tables if needed —SS]

Contents

List of Tables

List of Figures

This document ...

3 Functional Requirements Evaluation

3.1 FR211

The system shall allow users to create an account using their Google account.

FST-UAHM-1	**Profile Creation**
Description	Tests if the system is able to handle creating user profiles in the system from frontend inputs
Input	Valid Google account sign in
Exp. Output	Room8 account created and Frontend redirects user to the dashboard
Act. Output	Room8 account created and Frontend redirects user to the dashboard
Result	Pass

3.2 FR212

The system shall allows users to log in to their account.

FST-UAHM-2	**System Login**
Description	Tests if the system is able to recognize a user profile and to authenticate them to the system
Input	Valid Google account matching the profile in the system
Exp. Output	User profile is authenticated with a new session entry. Frontend redirects user to the dashboard
Act. Output	User profile is authenticated with a new session entry. Frontend redirects user to the dashboard
Result	Pass

3.3 FR213

The system shall allows users to log out of their account.

FST-UAHM-3	**System Logout**
Description	Tests if the system logs the user out, invalidating their session and returning them to the homepage
Input	User clicks logout button
Exp. Output	The system invalidates the session and redirects the user to the homepage
Act. Output	The system invalidates the session and redirects the user to the homepage
Result	Pass

3.4 FR214

The system shall allow users to create a home group using the home name, address, and number of roommates.

FST-UAHM-4	**Create Home Instance**
Description	Tests if the system allows users to create a new home group with specified details
Input	Home name, address, and number of roommates to their respective fields
Exp. Output	A new home group instance is created with the specified details, and the user is added as a member
Act. Output	A new home group instance is created with the specified details, and the user is added as a member
Result	Pass

3.5 FR215

The system shall allow users to invite other users to join their home group.

3.6 FR216

The system shall allow users to view the list of users in their home group.

3.7 FR217

The system shall allow users to remove users from their home group.

3.8 FR218

The system shall allow users to leave their home group.

3.9 FR221

The system shall allow users to configure the ChatBot settings to include or exclude messages corresponding to chore schedule, cleanliness manager, and bill splitter.

3.10 FR222

The ChatBot shall send reminders to the group chat about upcoming chores and events in the schedule 2 days in advance.

3.11 FR223

The ChatBot shall send notifications to the group chat about new shared living space cleanliness scores immediately after an event is added to the cleanliness manager page.

3.12 FR224

The ChatBot shall send notifications to the group chat about new shared expenses added to the bill splitter page immediately after its addition.

3.13 FR231

The system shall evaluate the cleanliness of the shared living space before and after a user enters and exits the space.

3.14 FR232

The system shall display the current cleanliness score of the shared living space.

3.15 FR233

The system shall display the detected messes in the shared living space.

3.16 FR234

The system shall allow users to view the cleanliness score of other users.

3.17 FR235

The system shall allow users to view the history of cleanliness scores and detected messes.

3.18 FR241

The system shall allow users to add a new chore to the schedule.

3.19 FR242

The system shall allow users to add a new event to the schedule.

3.20 FR243

The system shall allow users to input chore and event details (name, description, time, frequency, assigned users, etc.).

3.21 FR244

The system shall allow users to edit and delete chores and events.

3.22 FR245

The system shall allow users to view the schedule and mark chores as complete.

3.23 FR251

The system shall allow users to add a new expense to the bill splitter and notify the involved users.

3.24 FR252

The system shall allow users to view what they owe other housemates.

3.25 FR253

The system shall allow users to view what they owe others.

3.26 FR254

The system shall allow users to mark expenses as paid.

3.27 FR255

The system shall calculate debts in order to minimize the amount of transactions required between housemates.

4 Nonfunctional Requirements Evaluation

4.1 Usability

4.1.1 NFR237

The system shall declare an instances of someone altering a room finished one there has been no activity in the room for a designated period of time.

4.1.2 NFR242

The calendar system shall display all calendar events to users in their time zone.

4.2 Performance

4.2.1 NFR214

The system should be able to authenticate a user with a median response time of under 1 second.

4.2.2 NFR234

Photos captured with the system will be in a quality high enough to differentiate objects within frame.

4.2.3 NFR235

The system shall process image data in under 30 minutes.

4.3 Privacy and Security

4.3.1 NFR211

All data related to authentication must be encrypted in both transit and in storage.

4.3.2 NFR212

Error messages related to authentication should not disclose sensitive details such as "Incorrect Password".

4.3.3 NFR213

All data related to houses such as addresses and residents should be encrypted in transit and in rest.

4.3.4 NFR221

The chatbot shall not disclose any sensitive information in its messages such as addresses or full names.

4.3.5 NFR231

The system shall not record users.

4.3.6 NFR232

The system shall not capture images of users.

4.3.7 NFR233

The system shall encrypt and securely store all images of homes.

4.3.8 NFR244

The calendar system shall encrypt all events stored.

4.3.9 NFR251

The Bill Splitter system shall encrypt all events stored.

4.4 etc.(MAYBE RENAME TO "OTHER")

4.4.1 NFR222

The chatbot shall not send users too frequently to prevent annoying users.

4.4.2 NFR236

The system shall not report false events which accuse someone of reducing the cleanliness score of an environment.

4.4.3 NFR241

The calendar system shall store all calendar events in UTC.

4.4.4 NFR243

The calendar system shall have a granularity of 5 minutes.

4.4.5 NFR252

The Bill Splitter shall allow users to record numerical values of prices with a granularity of two decimal places.

5 Comparison to Existing Implementation

This section will not be appropriate for every project.

6 Unit Testing

Unit testing for this project is defined as a way of testing the smallest piece of code that can be logically isolated. Logical isolation means to isolate a piece of code based on a particular function. In this project, unit testing was performed on the user-facing application of the project while the other codebases of the project will be tested with other methods.

For context on the this section of the report, the client-facing application was built using the following tools:

- **Next.js** as the JavaScript framework for both front-end and back-end
- **ReactQuery** for data synchronization and caching

and is broken up into different pages based on application features such as:

- Cleanliness management system
- Chore scheduling system
- Bill splitting system
- ...

While it is recommended to test as much of your code as possible. Due to time constraints and generally better use of our time, the team developed unit tests only for the critical components (Referring to React components) of our application, while other components that exist for purposes such as reusable styling or wrappers were ignoring. In general, unit tests were written with the objective of:

- Ensure key UI components were rendering
- Ensuring the UI displayed information fetched from the back-end
- Ensuring inputs and buttons on the UI are able to be interacted with

In the front-end, 105 different tests were written for over 20+ components resulting in the following code-coverage report.

Table 1: **Front-end Code Coverage Report**

File	**% Stmts**	**% Branch**	**% Funcs**	**% Lines**	**Uncovered Line #s**
All files	82.9	93.24	58.87	82.9	
app	100	100	100	100	
page.tsx	100	100	100	100	
app/(main)/bill-splitter/components	83.37	100	60	83.37	
debtsTable.tsx	83.72	100	25	83.72	26–32, 34–38, 76–77
historyTable.tsx	100	100	100	100	
loansTable.tsx	63.82	100	57.14	63.82	22–67, 91–93, 95–96
summaryCard.tsx	100	100	100	100	
summaryCardStub.tsx	100	100	100	100	
app/(main)/bill-splitter/hooks	33.33	100	12.5	33.33	
patchOwe.ts	55.26	100	50	55.26	5–21
useBillHistory.ts	19.44	100	0	19.44	6–29, 32–36
useBills.ts	26.08	100	0	26.08	5–16, 19–23
useOwes.ts	26.08	100	0	26.08	5–16, 19–23
app/(main)/chatbot	95.62	80	50	95.62	
page.tsx	95.62	80	50	95.62	81–85, 101
app/(main)/chatbot/components	95.74	33.33	50	95.74	
chatbot-setting-stub.tsx	95.74	33.33	50	95.74	21–22
app/(main)/chatbot/hooks	24.39	100	0	24.39	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
useActivateChatbot.ts	24.39	100	0	24.39	9–24, 27–41
app/(main)/dashboard/components	100	100	100	100	
dashboard-cards.tsx	100	100	100	100	
app/(main)/house-settings/components	100	100	100	100	
create-note-modal.tsx	100	100	100	100	
edit-house-modal.tsx	100	100	100	100	
house-invites.tsx	100	100	100	100	
house-notes.tsx	100	100	100	100	
invite-user-modal.tsx	100	100	100	100	
app/(main)/house-settings/hooks	35	100	20	35	
useRemoveRoomate.ts	35	100	20	35	5–15, 22–23, 25–32, 34–38
app/(main)/profile	87.22	80	60	87.22	
page.tsx	87.22	80	60	87.22	25–31, 39–43, 94–101, 159–161
app/(main)/schedule	23.07	100	0	23.07	
adapters.ts	23.07	100	0	23.07	4–13
app/(main)/schedule/components	98.76	95.74	100	98.76	
chore-history.tsx	100	100	100	100	
create-chore-modal.tsx	99.41	85.71	100	99.41	143
pending-chores.tsx	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
pending-item.tsx	100	100	100	100	
schedule-item.tsx	95.58	92	100	95.58	64–69, 76–77
schedule.tsx	100	100	100	100	
app/(main)/schedule/hooks	33.48	100	0	33.48	
useCreateChore.ts	14.28	100	0	14.28	5–17, 20–42
useDeleteChore.ts	14.63	100	0	14.63	5–16, 19–41
useGetAllActivities.ts	36.36	100	0	36.36	7–15, 18–22
useGetAllCompletedChores.ts	69.76	100	0	69.76	29–36, 39–43
useGetCompletedChores.ts	50	100	0	50	12–19, 22–26
useUpdateCompletedChore.ts	24	100	0	24	11–23, 26–50
app/auth/hooks	47.36	100	0	47.36	
useUser.tsx	47.36	100	0	47.36	19–38
components	100	100	85.71	100	
loading.tsx	100	100	100	100	
modal.tsx	100	100	100	100	
mutate-loading.tsx	100	100	100	100	
query-provider.tsx	100	100	100	100	
roommates-table.tsx	100	100	50	100	
components/ui	97.69	88.23	100	97.69	
button.tsx	100	50	100	100	42
card.tsx	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
checkbox.tsx	100	100	100	100	
dialog.tsx	100	100	100	100	
form.tsx	92.26	92.85	100	92.26	50–51, 123–133
input.tsx	100	100	100	100	
label.tsx	100	100	100	100	
table.tsx	100	100	100	100	
hooks	45.16	75	33.33	45.16	
useGetHouse.ts	24.13	100	0	24.13	6–22, 25–29
useRoommates.ts	63.33	60	100	63.33	9–11, 13–20
useToast.ts	45.5	100	14.28	45.5	27–30, 59–72, 75–125, 131–136, 140–167
lib	61.53	100	50	61.53	
utils.ts	61.53	100	50	61.53	9–13
lib/constants	100	100	100	100	
index.ts	100	100	100	100	
lib/supabase	44.44	100	0	44.44	
browser.ts	44.44	100	0	44.44	5–9

7 Changes Due to Testing

[This section should highlight how feedback from the users and from the supervisor (when one exists) shaped the final product. In particular the feedback from the Rev 0 demo to the supervisor (or to potential users) should be highlighted. —SS]
FR211 - Change description to use Google account instead of using name, email and password.
FST-UAHM-1 - See above description.
FST-UAHM-2 - Change input field of testcase to using Google account instead of email and password.

7.1 Changes Due to Unit Testing

Unit testing discovered several bugs and anti-patterns in the front-end codebase that were resolved. Some examples of the discovered flaws include:

- Invalid references for form inputs and form labels. Leading to input labels not focusing their corresponding form input.
- Unneccessary props on React components.
- Missing semantic HTML attributes such as "role" on various items. Making the app less accessible and more difficult for the testing library to find components.
- Checking for lists to be empty with ¡LIST VARIABLE NAME¿. This resulted in empty lists still being evaluated to "True" because in JS empty lists are truthy and resulted in code logic that was supposed to trigger when the list was empty to not work as expected.
- Displaying times without the timezone. This was detected because the testing suite failed on various environments due to the timezone printing differently on machines with different timezones.

The code coverage reports shows some trends, mainly that all most if not all of our hooks have poor code coverage rates. This is expected and acceptable because the purpose of the hooks is to fetch data from the back-end and display it on the front-end and since we were mocking the back-end data, most of the hooks' functionality were not used.

8 Automated Testing

Automated tests include the unit tests and the cleanliness detection system's test samples. The front-end tests were additionally automated to run on push and on merge requests to main and dev. These automated tests on GitHub in conjunction with branch rules that prevent merging branches which fail tests ensure that no changes the fail tests will make it to the production environment.

9 Trace to Requirements

10 Trace to Modules

11 Code Coverage Metrics

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)