Developer Guide and Manual

The purpose of this document is how to initially set up the bare minimum for the project development environment, how to maintain the project repository, and how to develop for the project. This assumes that nothing is installed on the machine when setup begins.

Project Structure

The project is designed as a monorepo, all code frontend and backend can be found within the budget-helper-full-stack-app repo. To aid new developers and maintainers, we will list the notable files and folders within the repo. From the root of the project the current folder structure is the following:

- **Root folder**
    - **.devcontainer** - files pertaining the docker development container
        - **devcontainer.json** - file defines the runtime arguments for the vscode docker development containers and also defines what extensions should be installed
        - **base.Dockerfile** - DO NOT MODIFY contains the base image information for the typescript/node container
        - **Dockerfile** - contains the additional docker containers dependencies, currently has the dependencies for allowing Cypress to run within the dev container. EDIT WITH CAUTION
    - **.github** - files pertaining to Github specific actions like pull requests, Github actions, and pre commit hooks
        - pull_request_template.md - markdown document describing the pull request template. This allows pull requests to be standardized to make reviewing easier
        - workflows - folder for defining Github actions pipelines
            - github-actions.yml - file contains all the active pipelines that run for the project; currently there is a CI pipeline and integrations testing pipeline
    - **docs** - contains any resources used for markdown READMEs and will contain any documentation pertaining to the project for maintenance
    - **.deepsource.toml** - Configuration for the Deep Source static code analysis tool to allow automatic scanning of the project for bugs, code smells, bad design patterns, security vulnerabilities, and code coverage
    - **docker-compose.yml** - Contains task definitions for services that may be required to run the application, currently has a postgres server for local development
    - **app** - folder that contains all the application code for the application
        - **cypress** - folder contains all of the files relating to the Cypress integration testing suite
            - e2e - contains the specification files for each integration test that needs to be run
            - fixtures - contains json data that can be used to mock response data
            - support - contains files to define custom commands / actions for cypress
        - **prisma** - contains the schema files for Prisma

        - schema.prisma - Contains the primary schema for the application database
- **public** - contains any static files such as images and favicons
- **.eslintrc.js** - setups eslint with recommended linting practices
- **.gitignore** - defines which files should be ignored when you make a new git commit
- **.prettierignore** - defines which files should be ignored from prettier
- **.prettierrc.json** - defines configuration for prettier which performs automatic code formatting
- **cypress.config.ts** - defines any additional configuration for cypress, currently uses default settings
- **jest.config.ts** - defines the settings for Jest unit tests. Allows us to automatically resolve test files and write them in typescript
- **next.config.mjs** - defines custom configuration for next to allow features to be enabled and disabled. Currently enables minification and sets the localization to english.
- **postcss.config.cjs** - defines the plugins and configuration for the postcss compiler which helps to minimize the production css bundle size
- **tailwind.config.cjs** - defines configuration for tailwind, such as custom media breakpoints or thematic elements such as colors or spacing.
- **tsconfig.json** - Defines the options for the typescript compiler. Notable sections includes “compilerOptions.paths” which defines aliases to notable subfolders such as components or providers
- **src** - Primary application folder that holds the entire next js application
    - **components** - contains custom designed UI components
        - Structure for components is a folder with the name of the component which contains an index.tsx file. In the folder you could also include any test specs or scss files if you need to extend the customs styling beyond the capabilities of tailwind
    - **env** - Holds the files that creates the type-safe environment definitions to ensure the correct environment variables are passed in
        - schema.mjs - holds the definitions of environment variables and uses zod to perform input validation
    - **styles** - folder contains any global style sheets that are required for setting up tailwind and any other UI libraries we may add
    - **types** - Holds type references (.d.ts files) that are required for libraries to be type safe
    - **utils** - Holds any utility functions and types that may be required across the system
    - **stores** - hold any files related to global state management
        - HomeStore.tsx - defines the Zustand store for storing home information such as the homes a user belongs to

and their currently selected home. Also defines a context and provider to allow information to be pre populated on login

- **server** - holds backend functions such as services and routers
  - **common** - hold utility functions that can be used throughout the backend application
  - **db** - holds backend services that modify the database and defines the prisma client
  - **router** - holds the TRPC routers that allow the client to call backend functions (similar to a controller in MVC)
    - index.ts - defines the primary app router. New routers must be registered in this file using .merge on the appRouter
  - **pages** - holds the primary pages for the frontend of the application
    - **api** - defines backend functions that are specific to client interaction such as setting up trpc and NextAuth
    - _app.tsx - defines the entry point for the NextJs frontend. Contains any context providers such as Session and HomeState, and it also adds the routing guard for preventing unauthorized users from accessing a page. It also defines the connection to TRPC
    - All other pages in the application will follow the following convention <name-of-url>.tsx for example “createhome.tsx” will be accessible by going to “/createhome”, if a file is in a subfolder, the folder name is appended to the path by the NextJS router. Dynamic paths can be created by naming the file [slug-name].tsx. For example a file in the home folder called [id].tsx could be accessible by “/home/a-cool-home-id” and the id is passed into the page via props.

Dev Environment Installation Instructions

Windows Dev Environment Setup

1. Install docker desktop for windows https://docs.docker.com/desktop/install/windows-install/
2. Install visual studio code for windows https://code.visualstudio.com/Download

3. Open VS Code and follow instructions to install docker extension. https://code.visualstudio.com/docs/containers/overview
4. Continue to repository setup section

Linux Dev Environment Setup

1. Install docker desktop for linux, make sure to select option for respective distribution https://docs.docker.com/desktop/install/linux-install/
2. Install visual studio code for linux, choose installation method best for distribution https://code.visualstudio.com/Download
3. Open VS Code and follow instructions to install docker extension. https://code.visualstudio.com/docs/containers/overview
4. Continue to repository setup section

Mac Dev Environment Setup

1. Install docker desktop for linux, make sure to select option for respective distribution https://docs.docker.com/desktop/install/mac-install/
2. Install visual studio code for linux, choose installation method best for distribution https://code.visualstudio.com/Download
3. Open VS Code and follow instructions to install docker extension. https://code.visualstudio.com/docs/containers/overview
4. Continue to repository setup section

Repository Setup

1. Make sure that you have followed the dev environment instructions above on dev environment installation.
2. Clone the repository using the following command
   git clone git@github.com:Roommate-Budget-Helper/budget-helper-full-stack-app.git
   a. if you do not have SSH key setup for GitHub, follow this guide before running the above command: https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account
3. Open VSCode
4. Make sure you have the following extensions installed by going to the extensions tab:
   a. Docker
   b. Remote Development
5. Open the command palette (⌘+shift+p or ctrl+shift+p)
6. Type `Remote Containers: Open Folder in Container` and select the option
7. Select the option
8. Open the repository you cloned with the file selection window
9. It should now start opening the container and creating the dev container

NOTE: Make sure in the docker container that your user has permissions to edit files. If given a warning when creating or editing files in the docker container use sudo chown -R $USER <directory-name>

Creating the .env File

1. Create a file in the app folder called `.env` with the following values:

```
# Note that not all variables here might be in use for your selected configuration
# When adding additional env variables, the schema in /env/schema.mjs should be updated accordingly

# Prisma
DATABASE_URL=postgresql://postgres:postgres@host.docker.internal

# Next Auth
NEXTAUTH_SECRET=<super-secret-key-here>
NEXTAUTH_URL=http://localhost:3000

# Next Auth Cognito
COGNITO_CLIENT_ID=<cognito-id-here>
COGNITO_USER_POOL=<user-pool-id-here>

# Google Credentials
GOOGLE_CLIENT_ID=<google-client-id-here>
GOOGLE_CLIENT_SECRET=<google-client-secret-here>

S3_BUCKET_NAME=roommate-budget-helper
```

2. While in the app folder run `npm install`

Commands

Make sure the terminal is in the app/ folder before running a command.

`npm run dev` => Runs the development server
`npm run test` => Runs the jest test suite
`npm run e2e` => Runs the cypress test suite
`npm run lint` => Runs the linter
`npm run build` => Runs the build command
`npx prisma generate` => Rebuilds the prisma client for accessing the database from typescript

`npx prisma db push` => Synchronize the database with the prisma schema
`npx prisma studio` => Opens prisma studio for debugging

Troubleshooting Table

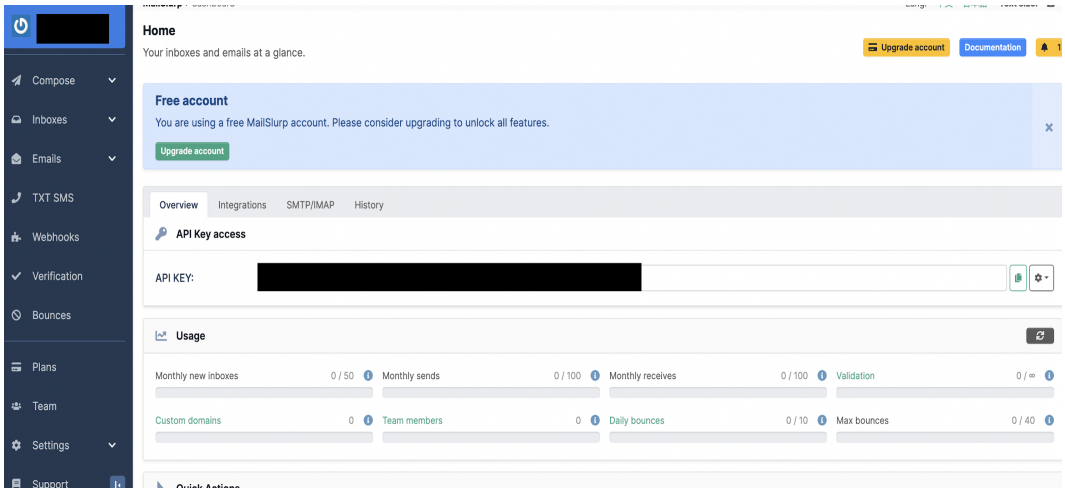
What the error looks like	Problem	Solution
Given a warning when creating or editing files in the docker container.	Docker user does not have permissions to edit files	Into the docker terminal, type: sudo chown -R $USER <directory-name>
When running an npm command, it returns: ENOENT: no such file or directory, open '/workspaces/budget-helper-full-stack-app/package.json' This is related to npm not being able to find a file.	Npm cannot find the package.json it wants.	You are likely outside of the app/ folder. cd to the app/ folder, then rerun the npm command. Otherwise, the package.json folder is missing. You can retrieve it from the github repository.
git complains about package-lock.json	Merge conflict with package-lock.json	1. Delete the package-lock.json file 2. Pull from the git 3. Run ‘npm i’

Continued on the next page.

What the error looks like	Problem	Solution
Login and Registration tests begin to fail without code changes.	Mailslurp's free tier email limit of 50 emails per month was reached.	Note: You must have permissions to edit the github actions pipeline to perform this solution. Create a new mailslurp account, and place the new api key in the github actions parameters, and the local cypress.env.json file. Detailed Instructions: 1. Use temp-mail.org to create a disposable email 2. Create a new mailslurp account with the email 3. Open the verification email, and click 'Access Mailslurp' 4. Navigate to settings 5. Copy the api key 6. If you are performing tests on the github actions pipeline, go to the github actions tab, and insert the mailslurp api key in the mailslurp api key variable. Wth the following format: '{ "API_KEY": "<replace-this-w-api-key>" }' 7. If you are performing tests locally, replace the api key in your cypress.env.json. Also, consider using a separate account's api key for local tests than the github actions pipeline.

Setting up testing

1. Create a file called `cypress.env.json` in the app folder
2. Enter the following values:
   `{ "API_KEY": "<replace-this-with-your-actual-api-key>" }`
3. Go to [MailSlurp](https://www.mailslurp.com/) and create an account
4. After logging in confirm you see a dashboard similar to this:



5. Copy the API Access key and place it in the `cypress.env.json`
   You should now be able run the project
6. Run `npx cypress install` in the /app folder to prepare cypress for the first time
7. Setup an X-Server on your host machine and allow connections to localhost:

- Mac: `brew install --cask Xquartz`. Run `xhost + localhost`
- Windows: [Xming](https://sourceforge.net/projects/xming/)
- Linux: Install `xorg-xhost` package if not already on your system. In docker desktop, enable file sharing for `/tmp/.X11-unix` Run `xhost + local:docker`

Notes for Windows:

```
# Change your DISPLAY location for xming
        export DISPLAY=<your ip address>:0
# Run the xming application in your local computer
        <xming executable location in explorer> :0 -ac
# Run the e2e test in dev container
        npm run e2e
```

GitHub Version Control

General Rules

The “main” branch is the protected branch that will be used for “deployable” code. It is viewed as the production branch and therefore requires quality working code. We have set up protection rules to prevent any code from being pushed to the main branch without a heavily reviewed pull request. The pull request has a template that requires the following information:

- Jira Ticket number and link
- Description of work completed
- Acceptance Criteria
- Steps of how to test
- List of changes
- Screenshots(for frontend code)

We also require that at least one quality comment is made per code reviewer and that all discussions have been relevantly followed up. The main branch has a CI pipeline that will lint, test, and build the code when the pull request is opened and will rebuild after each new commit while the pull request is open. We plan on adding a CD pipeline to deploy the code to AWS after being merged into the main branch.

Branch Naming Convention

Since we are using Jira for issue tracking, Jira can integrate with GitHub to automatically assign issues to the branch by using the following naming convention: RBH-[Jira ticket number]-ticket-name. For example, ticket 18 named “Configuration Management and Preliminary Metrics” would have a branch named:

”RBH-18-configuration-management-and-preliminary-metrics”. Naming branches in this way allows Jira to have better tracking of the work being completed.

Pipeline Integration

Since we are using GitHub Actions for CI/CD, we are able to prevent code that fails one or both of these pipelines from being merged. The current CI pipeline will run the following actions:

- Lint (Checking the code style / quality)
- Build (Make sure code can be built in order to deploy)
- Test (Make sure all unit tests pass)
    - Run all Jest unit tests
    - Run end to end and frontend cypress tests

As the project evolves, we will add an additional stage to the CI pipeline. We are planning to add Deepsource as a vulnerability and code quality analyzer, and we will add a stage in the pipeline to send data to Deepsource. We are also in the process of creating CD integration to automatically deploy the code to AWS. This will be determined as we are first determining our architecture requirements.

-

Test Plan

Overview

We will be performing end-to-end and acceptance testing for this system. We test to ensure product reliability and requirements satisfaction. Initially, we planned on performing unit testing as well, but when trying to use our planned unit testing framework, Jest, we found making a single test took very long to write and compile, due to the mocking process. However, with Cypress, we still can perform smaller scale tests that target individual components. So, we are placing more focus on integration and end-to-end and testing through cypress which allows us to better test errors and things that a user would encounter through manual testing. Cypress enables testing through automated interaction with the web application while it runs in a

browser. We write tests according to our Test Cases, documented here, which link to Use Cases to ensure requirements coverage.

Test Cases

User Maintenance

Name of Test	Use Case	Input	Expected Output	Error Handling
Successful leaving of Home	Ability to Leave a Home	Home ID: "56" Signed In User: "davis@gmail.com" Database of Homes	Home object mutated to remove the new User to the ""Occupants"" section: Home object mutated to change the target User's permissions: Home{"Name": "4th Street Apartment", "Home_ID": "56", "Occupants":{"adminGuest@gmail.com"}, "adminIDs":{"adminGuest@gmail.com"}}	No Error
Only User Leaves Home	Ability to Leave a Home	Home ID: "56" Signed In User: "davis@gmail.com" Database of Homes	Home object with ID "56" deleted from database	No Error
Successful Addition of a Non-User Roomate	Ability to invite over Users to the Home	Home ID: "56" User email: "tester@gmail.com"	Home object mutated to add the new User to the "Occupants" section: "User{"Email":"tester@gmail.com", "Username": "tester", "ID": "15"}	No Error
Successful Addition of a User Roomate	Ability to invite over Users to the Home	Home ID: "56" User email: "existing@gmail.com"	Home object mutated to add the existing User to the "Occupants" section: "User{"Email":"existing@gmail.com", "Username": "Curtiss", "ID": "3"}	No Error
Successful Change of User's permissions	Ability to allow another User to create payments / give permissions	Home ID: "56" User email: "adminDavis@gmail.com" Target Email: "adminGuest@gmail.com"	Home object mutated to change the target User's permissions: Home{"Name": "4th Street Apartment", "Home_ID": "56", "Occupants":{"adminDavis@gmail.com","adminGuest@gmail.com"}, "adminIDs":{"adminDavis@gmail.com", "adminGuest@gmail.com"}}	No Error

Suceessful Splitting of payments	Ability to split the payments between each home member	Home ID: "56" Signed In User: "davis@gmail.com" Target email: "guest@gmail.com"	Charge object created in Users "davis@gmail.com" & "guest@gmail.com": Charges{"ID":"65", "HomeID": "56", "Date Created": 11/8/2022, "Amount": "15.50", "Comment": "Dinner", "Breakdown": "50-50", "Date Due": "N/A", "Charged": "guest@gmail.com", "Issues": "davis@gmail.com"}	No Error
Successful Due Date on Charge	Ability to set a "due date" on bills within a home	Charge ID: "65" Signed In User: "davis@gmail.com"	Charge objects with ID "65" mutated in Users "davis@gmail.com": Charges{"ID":"65", "HomeID": "56", "Date Created": 11/8/2022, "Amount": "15.50", "Comment": "Dinner", "Breakdown": "50-50", "Date Due": "11/25/2022", "Charged": "guest@gmail.com", "Issues": "davis@gmail.com"}	No Error

Sending and Receiving Payments

Name of Test	Use Case	Input	Expected Output	Error Handling

Verify Charge Creation	Send a charge	Issuer User ID: "cla1fr4o80000qd23gukqcovf"
		Receiver User ID: "cla1enkau00002a6dwqtbofm5"
		Amount: 5.32
		Description: "Big Mac"
		Breakdown: "Total"
		Date Due: getCurrentDate() + 30
			Charge
			{
			ChargeID: "cla1fr5rl0000qd2h6sjt16zb",
			Issuer CUID: "cla1fr4o80000qd23gukqcovf",
			Receiver CUID: cla1enkau00002a6dwqtbofm5,
			Amount: 5.32,
			Description: "Big Mac"
			Breakdown: "Total"
			Date: getCurrentDate()
			Date Due: getCurrentDate() + 30
			Status: "Unpaid"
			}
				No Error

Verify Transaction History Retrieval	View Transaction History	User CUID: "cla1enkau00002a6dwqtbofm5"	[{ ChargeID: "cla1fr5rl0000qd2h6sjt16zb", Issuer CUID: "cla1fr4o80000qd23gukqcovf", Receiver CUID:	No Error

			"cla1enkau00002a6dwqtbofm5" , Amount: 5.32, Description: "Big Mac" Breakdown: "Total" Date: getCurrentDate() Date Due: getCurrentDate() + 30 Status: "Unpaid" }, { ChargeID: "cla1fvnan0000qd3maflv6610", Issuer CUID: "cla1fr4o80000qd23gukqcovf", Receiver CUID: "cla1enkau00002a6dwqtbofm5" , Amount: 1.48, Description: "Fries" Breakdown: "50%" Date: getCurrentDate() Date Due: getCurrentDate() + 30 Status: "Pending" }]	

View Unpaid Charges on a bill	View a Bill	User CUID: "cla1enkau00002a6dwqtbofm5"	[{
			ChargeID: "cla1fr5rl0000qd2h6sjt16zb",
			Issuer CUID: "cla1fr4o80000qd23gukqcovf",
			Receiver CUID: "cla1enkau00002a6dwqtbofm5"
			,
			Amount: 5.32,
			Description: "Big Mac"
			Breakdown: "Total"
			Date: getCurrentDate()
			Date Due: getCurrentDate() + 30
			Status: "Unpaid"
			}
			]
				No Error

Sent Charges by user	See sent charges by user	Receiver CUID: "cla1enkau00002a6dwqtbofm5", Issuer CUID: "cla1fr4o80000qd23gukqcovf"	[{ ChargeID: "cla1fr5rl0000qd2h6sjt16zb", Issuer CUID:	No Error

			"cla1fr4o80000qd23gukqcovf", Receiver CUID: "cla1enkau00002a6dwqtbofm5" , Amount: 5.32, Description: "Big Mac" Breakdown: "Total" Date: getCurrentDate() Date Due: getCurrentDate() + 30 Status: "Unpaid" }]	

Verify Pay Charges	User Pays Charges	User CUID:
		"cla1enkau00002a6dwqtbofm5",
		Charge IDs: [
		"cla1fr5rl0000qd2h6sjt16zb"]
			[{
			ChargeID:
			"cla1fr5rl0000qd2h6sjt16zb",
			Issuer CUID:
			"cla1fr4o80000qd23gukqcovf",
			Receiver CUID:
			"cla1enkau00002a6dwqtbofm5"
			,
			Amount: 5.32,
			Description: "Big Mac"
			Breakdown: "Total"
			Date: getCurrentDate()
			Date Due: getCurrentDate() + 30
			Status: "Pending"
			}
			]
				No Error

Issuer Verify Charge Paid Success	Issuer Confirms Charge Paid	Issuer CUID:
		"cla1fr4o80000qd23gukqcovf"
			[{
			ChargeID:
			"cla1fr5rl0000qd2h6sjt16zb",
			Issuer CUID:
			"cla1fr4o80000qd23gukqcovf",
			Receiver CUID:
			"cla1enkau00002a6dwqtbofm5"
			,
			Amount: 5.32,
			Description: "Big Mac"
			Breakdown: "Total"
			Date: getCurrentDate()
			Date Due: getCurrentDate() + 30
			Status: "Paid"
			}
			]
				No Error

Issuer Verify Charge Paid Fail	Issuer Confirms Charge Paid	Issuer CUID: "cla1fr4o80000qd23gukqcovf"	[{
			ChargeID: "cla1fr5rl0000qd2h6sjt16zb",
			Issuer CUID: "cla1fr4o80000qd23gukqcovf",
			Receiver CUID: "cla1enkau00002a6dwqtbofm5",
			Amount: 5.32,
			Description: "Big Mac"
			Breakdown: "Total"
			Date: getCurrentDate()
			Date Due: getCurrentDate() + 30
			Status: "Pending"
			}
			]
				No Error

Security and Login

Name of Test	Use Case	Input	Expected Output	Error Handling

Successful Log In	User Sign In	Username: "daviscl"
		
		Password: "tester"
			model Account {
			id String @id @default(cuid())
			userId String
			type String
			provider String
			providerAccountId String
			refresh_token String? @db.Text
			access_token String? @db.Text
			expires_at Int?
			token_type String?
			scope String?
			id_token String? @db.Text
			session_state String?
			user User @relation(fields: [userId], references: [id], onDelete: Cascade)
			}
				No Error

Wrong Password on Log In	User Sign In	Username: "daviscl" Password: "testr"	Error with message: Incorrect Username or Password	Improper Credentials

Forgot Password provide Username	User Sign In	Username: "daviscl"	Messaging Service is used to send recovery steps	No Error
Forgot Password provide Invalid Username	User Sign In	Username: "daviscl2"	Error with Message: Improper Username or Email provided	Improper Email/User name provided
Forgot Password provide Email	User Sign In	Email: "userdavid@gmail.com"	Messaging Service is used to send recovery steps	No Error
Forgot Password provide Invalid Email	User Sign In	Email: "userdavis@gmail.com"	Error with Message: Improper Username or Email provided	Improper Email/ Username provided

Forgot Password Update Password	User Sign In	Password: "new_password677"	model Session {
			id String @id @default(cuid())
			sessionToken String @unique
			userId String
			expires DateTime
			user User @relation(fields: [userId], references: [id], onDelete: Cascade)
			}
				No Error

Successful Sign Up Email/Password	User Sign up	Email: "userdavid@gmail.com"
		Username: "user_david_1"
		Password: "test_password778"
			model Session {
			id String @id @default(cuid())
			sessionToken String @unique
			userId String
			expires DateTime
			user User @relation(fields: [userId], references: [id], onDelete: Cascade)
			}
				No Error

Duplicate Email on Signup	User Sign up	Email: "userdavid@gmail.com" Username: "user_david_1" Password: "test_password778"	Error with message: Account with Email already in use	Improper Email Provided

Note: We aren't going to store passwords and information regarding that in a database or in an object, we are going to use AWS Cognito tokens to handle verification

Home CRUD

Name of Test	Use Case	Input	Expected Output	Error Handling
Successful Home Creation	Create a Home	Home name: "4th Street Apartment" Signed In User: "user@gmail.com" Database of Homes	Home object created: Home{"Name": "4th Street Apartment", "Home_ID": "56", "Occupants":{"user@gmail.com"}, "AdminIDs":{"user@gmail.com"}} Home object added to Database	No Error
Successful View of Home	View a Home	Home ID: "56" Signed In User: "davis@gmail.com" Database of Homes belonging to User:	Home object in User's Homes: Home{"Name": "4th Street Apartment", "Home_ID":"56", "Occupants":{"davis@gmail.com"}}	No Error
Sucessful Deletion of Home	Delete a Home	Home ID: "56" Database of Homes belonging to User	Removes the Home object with Home ID: "56" from the Database of Homes for all Users	No Error
Failed Deletion of Home	Delete a Home	User with "Admin" recently removed Home ID: "56" Database of Homes beloning to User	No Outcome	Cannot Delete a Home without "Admin" Error

Engineering Standards and Rationale

IEEE 26515 -

The team chose to conform to this standard to improve organization in documentation when utilizing an agile development strategy. It also chose this standard to improve the development process and hopefully deliver an accurate product in a faster and more organized fashion. Our assumption when using this standard is that the only documentation created is the documentation for end users fulfilling scenarios described in use cases. Our observations are that user documentation is created with a description of what the documentation does, we plan and assign the development of end-user documentation in our agile environment, and documentation is verified based on the success of system tests from end users following the documentation. Revisions to documentation are evaluated and assigned based on the failure of systems tests and on changes to requirements as development is performed. As a team, we also decided to utilize the agile development process for this project. We have observed that using sprints has helped us set goals and maintain a timeline. It has helped the team distribute tasks by assigning them during a sprint. It has helped monitor progress by having expectations for when a task should be finished by and it has a QA stage to review documentation, code, and other work produced during a sprint. We chose to use this because the agile process has been used and succeeded on many projects in the past.

Trade Offs: The methods in this standard require extra documentation and process steps to achieve the desired system, when compared to less structured and direct approaches.

Although it takes a bit more time, hopefully it will help us avoid the costs of reworking design decisions later in the project.

ISO 12207 -

The team chose to conform to this standard to help develop, maintain, and supply software products. It is used as a guideline for both system context processes and software processes. This means that we assume to use them when acquiring requirements for the software using a stakeholder requirement gathering process, evaluating for risks and identifying blockers, creating a configuration management plan, and creating a testing process. We observe that this has helped the software design process so far because we have gathered all our requirements from the stakeholders using this process and have used it for the software development process as well.

Trade Offs: Tradeoffs to using this versus not using this standard are that it can help identify possible risks, help identify important or unwanted features in software through a requirement gathering process, and with creating strategies for how the software should work, be tested, and managed despite maybe taking longer than not using it.

OWASP Top 10 -

The OWASP Top 10 standard is an awareness standard for preventing common web application vulnerabilities. OWASP Top 10 compliance requires the web application to be resilient to the following types of vulnerabilities:

- A01:2021-Broken Access Control
- A02:2021-Cryptographic Failures
- A03:2021-Injection
- A04:2021-Insecure Design
- A05:2021-Security Misconfiguration
- A06:2021-Vulnerable and Outdated Components
- A07:2021-Identification and Authentication Failures
- A08:2021-Software and Data Integrity Failures
- A09:2021-Security Logging and Monitoring Failures
- A10:2021-Server-Side Request Forgery

Our software will need to keep these vulnerabilities in mind during the development process to ensure the safety of user data.

Trade Offs: OWASP Top 10 vulnerabilities may take additional LoC(lines of code) since the system may need to add additional data validation to conform to these standards. It can create a potential loss in system performance and increase development time. Conforming to these standards, however, is a crucial part of ensuring data integrity and confidentiality. For financial data, it is a must that OWASP's Top 10 vulnerabilities are mitigated as this data is extremely sensitive to users.

WCAG 2.1 -

The Web Content Accessibility Guidelines (WCAG) 2.1 standard describes the processes for making web applications accessible for users with disabilities. This ensures that our web application meets accommodations for those with conditions such as blindness and low vision, deafness and hearing loss, limited mobility/movement, and others. This document specifies recommendations to meet minimum text contrast, screen reader/aria compliance, and general requirements for conforming to accessibility. Our software could be used by anyone who has roommates, therefore, we must ensure that our application meets the minimum amount of accessibility as recommended by W3C.

Trade Offs: Web accessibility requires more LoC(lines of code) since appropriate labels and aria information must be created. This also makes maintainability harder since content requires its associated labels to be updated as well. Conforming to this standard, however, will allow a larger user base to be able to use our application.

ISO 27001 -

The ISO 27001 standard outlines the basic guidelines to ensure the confidentiality, integrity, and availability of sensitive data. The requirements of this standard ensure that risks within data management have been identified and mitigated to ensure data is correctly stored and managed. This will be important for our system as we will be storing user transaction data which is considered very sensitive data.

Trade Offs: ISO 27001 requires additional risk analysis to be performed to make sure user data is correctly managed. This is especially important for sensitive information like user transactions. This will require additional testing efforts as well to make sure that users cannot access data that does not belong to them.

ISO/IEC/IEEE 29119:2022 -

The ISO/IEC/IEEE 29119:2022 standard outlines general concepts in software testing solutions. This will be applicable to our project as we will be performing the following test types:

- Unit
- Coverage
- Requirements
- Load
- Performance
- Reliability

This will ensure we are correctly meeting requirements and that our application is sufficiently usable for our target customers.

Trade Offs: By implementing software testing, we are needing much more resources during the development lifecycle. For small applications, it could be more cost-effective and efficient to test the software in production. However, since our application is dealing with financial and personal data, it is in our best interest to make sure the software works as intended before reaching a production state.

NIST SP 800-144:2011 -

The NIST SP 800-144:2011 provides guidelines for ensuring the security of cloud applications. Since our application will be provisioned on AWS, we must ensure proper IAM (Identity and Access Management), Network Configurations (VPCs, Subnets, Routing), and software configuration measures are enforced to make sure our data is secure for consumers. This involves ensuring that only the intended permissions are provided to each of the infrastructure components and that the system is resilient to attempts of exfiltrating data.

Trade Offs: Conforming to NIST SP 800-144:2011 requires additional measures in designing our cloud infrastructure to make sure that our application is secure for our users. This means it will have additional complexity that requires more time and testing in the development process.

This process will make sure that our deployed application maintains industry standard protection processes to ensure that our system is resilient to attackers.

ISO 5055 -

The ISO 5055 quality standard are set of measurable factors for ensuring the internal structure of a software product in the following areas:

- Security
- Reliability
- Performance Efficiency
- Maintainability

It is important for our software project to be developed with these factors in mind in order to:

- Provide a secure environment for managing user data
- Ensuring the web application is resilient to large volumes of requests
- Ensuring the software project is maintainable to be handed off to future project maintainers

These factors are also useful for identifying risk and accessing the level of technical debt within our application.

Trade Offs: By conforming to ISO 5055, we have to put additional effort into identifying software risks and technical debt. This process will take additional development time, however, conforming to this standard allows our codebase to have a longer lifespan and is easier for future maintainers to work with.