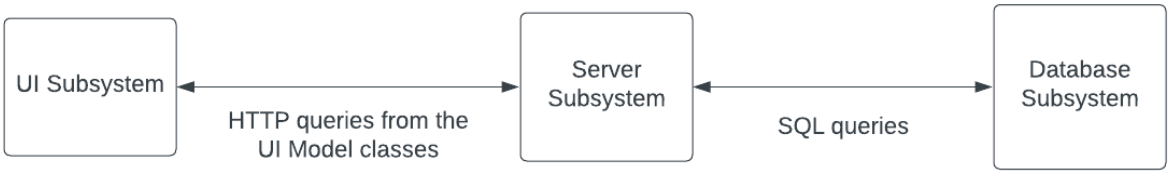
Roomies Arch & Design

Purpose of Document

The purpose of this document is to clearly explain the design decisions that are being made to execute this project. The primary subsystems of this product are the UI, Database, and Server. In each section, a brief explanation is given as to why design decisions were chosen (Design Decisions), and then concrete details of those decisions follow. With this document, we will be able to separate tasks and be clear on what each subsystem is expected to deliver.

Systems Diagram

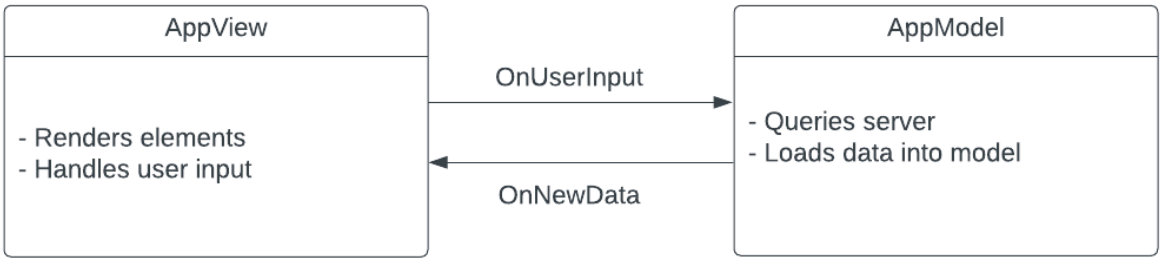


UI

Design Decisions

- Utilizing a multilayered UI we intend to create a user interface that is reliable along with scalable as more system features are added.
- The design pattern simulates a React model-view-presenter framework, consisting of multiple layers to ensure performance and functionality.
- In comparison to the model-view-presenter pattern, due to the size of the application being small and future iterations being configured in React or other frameworks, we decided to contain only the view and model layers of the application. For current usage we feel like this will allow the application to be easily deployed and maintained while lowering the cost of hours to complete the project.
- In the view layer of the application, the customer facing user interface consists of information, widgets, components that the user will be interacting with. In the model layer of the application, the deterministic logic from the view layer is defined.

- This two layer system will allow the user interface and logic to be contained in distinct structures and aid in future compatibility.



Views

FeedView

- This will be the landing page of the app, and will contain a scrollable element of all of the events that were recently completed by the user or other relevant accounts.

AppView

- This view will have a scrollable page of widgets that will take the user to other views. The widgets that are present can be added/removed according to the user's liking.

GroceryWidgetView

- This is an example of one of the widgetViews we will provide. The main element of this view is a scrollable list of items on the roommates' shared grocery list. Any individual in the group will be able to add/remove from the list with button elements.

PeopleView

- In this view, the there will be a scrollable element containing contacts that are relevant to the user. The user can click on these elements to open up a new view of the contact page of that individual.

Models

FeedModel

- Contains the Logic for the various UI actions from the FeedView
- Sends a request to the server to get the SQL returned from database
- Returned request contains complete list of user notifications
- Loads data into the model for the Feed-represented as the various notifications for the user.
- Sends this data to the FeedView

AppModel

- Contains the Logic for the various UI actions from the AppView
- Sends a request to the server to get the SQL returned from database
- Returned request contains complete list of user notifications
- Loads data into the model for the Feed-represented as the various notifications for the user.
- Sends this data to the AppView

GroceryWidgetModel

- Contains the Logic for the various UI actions from the GroceryStoreWidgetView
- Sends a request to the server to get the SQL returned from database
- Returned request contains complete list of user notifications
- Loads data into the model for the Feed-represented as the various notifications for the user.
- Sends this data to the GroceryWidgetModel

PeopleModel

- Contains the Logic for the various UI actions from the PeopleView
- Sends a request to the server to get the SQL returned from database
- Returned request contains complete list of user notifications
- Loads data into the model for the Feed-represented as the various notifications for the user.
- Sends this data to the PeopleView

Databases

Design Decisions

- Databases will be designed and accessed using MySQL
- MySQL offers a secure database management system that is reliable and easy to use.
- Along with reliability, MYSQL in comparison to other services is capable of high speeds and ease of functionality.
- MySQL offers multiple packages and usages that allow for users to connect to the database and perform SQL operations quickly and efficiently
- MySQL is compatible with the GOLang environment and is a very suitable choice for queries in our application including creating users, creating feed items, and creating roommate groups.

Users

Column Name	**Data Type**	**Constraints**	**Description**
userID	VARCHAR(255)	PRIMARY_KEY, UNIQUE	Unique identifier for the user
groupID	VARCHAR(255)	SORT_KEY	ID of the group the user belongs to
firstName	VARCHAR(255)		
lastName	VARCHAR(255)		
username	VARCHAR(255)	UNIQUE	
password	VARCHAR(255)		

Groups

Column Name	**Data Type**	**Constraints**	**Description**
groupID	VARCHAR(255)	PRIMARY_KEY	

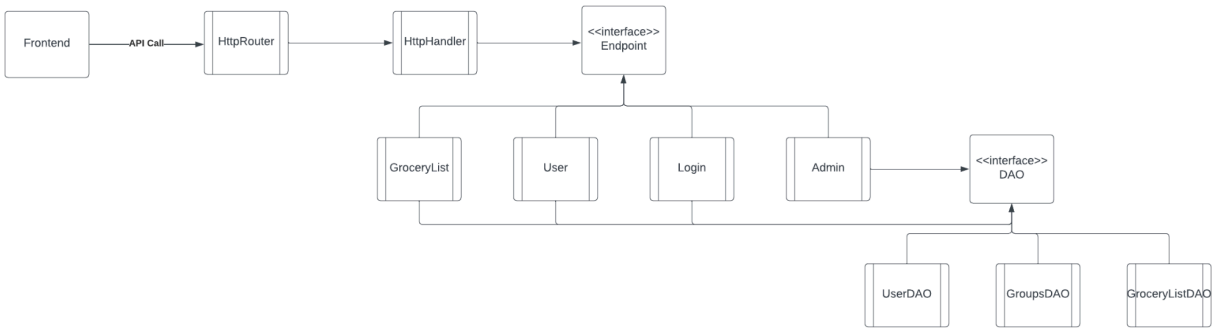
Grocery List (Shared)

Column Name	Data Type	Constraints	Description
itemID	INT	PRIMARY_KEY, UNIQUE	Unique item identifiers
groupID	VARCHAR(255)	SORT_KEY	Unique identifier for the user
listItem	VARCHAR(255)		The item in the list {“light bulbs”)
addedByID	VARCHAR(255)		ID of the user that added the item
recurring	BOOL		Item gets re-added to list automatically
recurringDate	DATETIME		When an item should be added again
timestamp	DATETIME		Timestamp when added

Grocery List (Private)

Column Name	Data Type	Constraints	Description
userID	VARCHAR(255)	PRIMARY_KEY, UNIQUE	Unique identifier for the user
listItem	VARCHAR(255)		The actual item on the list (ex: lightbulbs)
timestamp	DATETIME		Timestamp when added

Server



Design Decisions

- Have a central http router that every endpoint will be routed through. This allows for a single go executable to be run on the server.
- Each endpoint is represented by a struct containing core logic and DAOs as needed.
- Each DAO is implemented following core DAO interfaces and will be updated to connect to a database when ready. For now, we will have a DAO that gives spoofed data.

Grocery List

Endpoint: Get Shared List

HTTP Method: GET
URL Path: /grocery-list/shared?groupID=”dasfasdf”
Description: Gets all items associated with groupID

Request Parameters(none):

Parameter	Location	Type	Required	Description
groupID	Query param	string	yes	

Responses:
Array of groceryListItems (see below)

Endpoint: Create Shared List Item

HTTP Method: POST
URL Path: /grocery-list/shared/{groupID}
Description: Add a new item

Request Parameters:

Parameter	Location	Type	Required	Description
listItem	BODY			
addedByID	BODY			
recurring	BODY			
recurringDate	BODY			
timestamp	BODY			

Expected Responses:

Status Code	Description
200	Good
400	Error

Endpoint: Delete Shared List Item

HTTP Method: DELETE
URL Path: /grocery-list/shared/{groupID}/{itemID}
Description: Delete an item from grocerylist databases

Request Parameters(none):

Parameter	Location	Type	Required	Description

Expected Responses:

Status Code	Description

200	Good
400	Error

Login

Endpoint: Request to Login

HTTP Method: GET
URL Path: /login
Description: Attempts to login

Request Parameters:

Parameter	Location	Type	Required	Description
userName	Body	string	yes	
password	Body	string	yes	Maybe should be hashed?

Response:

We probably should do authTokens and stuff, but for now, I'd just say if the login is successful (a username/password match is found) return a 200, otherwise return a different code and I'll use that to log the user in.

Parameter	Location	Type	Required	Description
groupID	Body	string	yes	The groupID the user is part of (created by the apartment managers and is input by the user at registration)
userID	Body	string	yes	The uniqueID of the user (generated by the server at

				registration)

Register

Endpoint: Request to Register

HTTP Method: GET
URL Path: /register
Description: Attempts to register

Request Parameters:

Parameter	Location	Type	Required	Description
userName	Body	string	yes	
password	Body	string	yes	Maybe should be hashed?
groupID	Body	string	yes	
firstName	Body	String	yes	
lastName	Body	String	yes	

Response:

When the user registers, the info should be saved to the database, and a new userID should be generated for the user. GroupID will be provided by the apartment manager and sent in the request

Parameter	Location	Type	Required	Description
userID	Body	string	yes	

Feed

Endpoint: Request to get feed

HTTP Method: GET

URL Path: /feed?groupID=”sdfklasdf”
Description: Get’s to most recent 20 items for the group’s feed (we can add pagination later we want but let’s not worry about it for now)

Request Parameters:

Parameter	Location	Type	Required	Description
groupID	Query param	string	yes	

Response:

An array of Feed Items, that each contain the following

Parameter	Location	Type	Required	Description
itemID	Body	string	yes	Generated by the server in the POST /feed endpoint
itemText	Body	string	yes	Describes the event that took place (Ex: Added eggs to grocery list)
addedByID	Body	string	yes	userID of the user who added the item to the feed
timeStamp	Body	time.Time	yes	When the item was added to feed

Endpoint: Add feed item

HTTP Method: Post
URL Path: /feed?groupID=”sdfklasdf”
Description: Adds an item to the group’s feed

Request:

Parameter	Location	Type	Required	Description
groupID	Query param	string	yes	
itemText	Body	string	yes	Describes the event that took place (Ex: Added eggs to grocery list)
addedByID	Body	string	yes	userID of the user who added the item to the feed
timeStamp	Body	time.Time	yes	When the item was added to feed

Response:

Parameter	Location	Type	Required	Description

People

ServiceRequest

// Probably can just ignore this one for now since we aren't going to have an admin view

Announcements