Project Name:
Software Architecture
version 1.0

HomePairs
Computer Science Department
California Polytechnic State University
San Luis Obispo, CA USA

November 7, 2019

Contents

Credits

Name	Date	Role	Version
Thomas Bergmann	November 6, 2019	Author of ER diagram, adding appliance activity diagram, Section 1 and 2	1.0
Sarah MacDougall	November 6, 2019	Lead Author of Deployment Diagram and UI Dialog Map Descriptions	1.0
Cesar Chacon	November 6, 2019	Author of Class Diagram and Chatroom Use Case	1.0
Eeron Grant	November 6, 2019	Author of Job Request Use Case and State Diagrams	1.0
Adam Berard	November 6, 2019	Author of Job Request Activity Diagram and Appliance Management Use Case Diagram	1.0

1 Introduction

This document characterizes the architecture of our tenant/property manager Roopairs-powered application by describing the different components involved in the solution as well as the technologies that will be used during development. The document also goes into detail about the relationship between different components, such as how the Roopairs API will interact with our application. This document includes information about how Google Cloud Platform is being incorporated, the database schema, and diagrams detailing how each major component will be implemented.

2 Problem Description

HomePairs wants to streamline property managers abilities to make repairs for their properties. By using the HomePairs app, repairs should be faster to complete, catalog, and organize and can even be be referenced from the app. HomePairs will also make the process of requesting a repair much more transparent for both the property manager and the tenant. This will be accomplished by creating a stand-alone app for property managers and their tenants that uses Roopairs' pre-existing service provider connections and technology. Property managers will have access to these service providers to easily find the service provider needed for a given repair. In addition, to allowing users to easily find service providers, the app will also offer status updates for each repair that both the property manager and tenants can see, and which the service provider will update. If development time permits, the app will also have chat capabilities so property managers, tenants, and service providers can message each other about service requests as needed.

As many users will have little experience with technology, the user-experience is geared towards making the app intuitive and frustration-free.

3 Solution

3.1 Overview

The HomePairs application will be a native iOS and Android application deployed through the app store as well as a web application. It will allow the sending and storing of messages through a database via RESTful calls to a server. It will also be using Roopairs' Service Provider RESTful API which stores the service provider information and functionality that will be displayed to the user's iPhone, Android, and/or computer. Each of the diagrams below describe in detail the components of the software architecture "under the hood" and general processes of data and functionalities.

3.2 Components

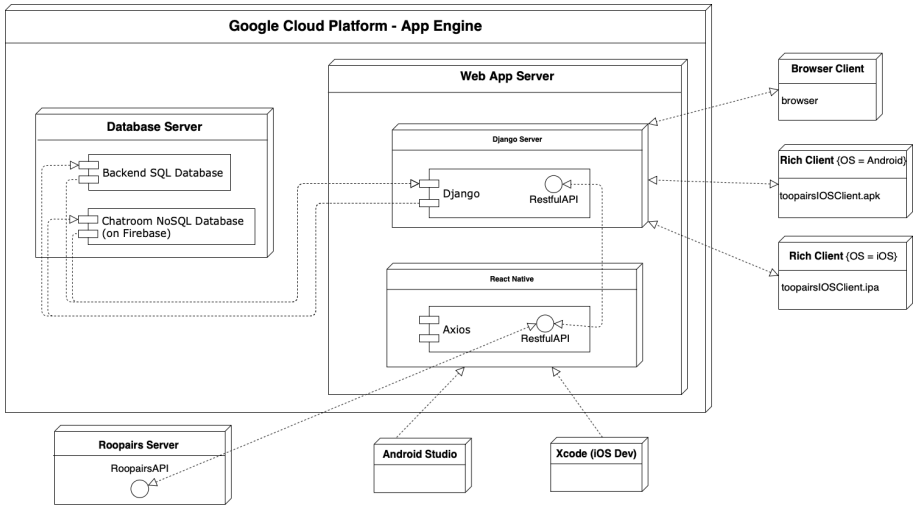

Figure 1: Deployment Diagram (Sarah MacDougall)

This diagram illustrates the proposed fundamental components for our project. 3 of these components are potential client devices: web browser, Andriod, and iOS. For Andriod and iOS, React Native is able to convert our web application into native applications for these devices.

Our applications will be developed from 2 primary components: Django and React Native. Django will support our backend logic and React Native our frontend. Our only confirmed library currently is Axios, which would we use to make HTTP Requests. Our web application could ideally be hosted on Google Cloud Platform's App Engine, as there are free tiers for both Node.js (React Native) and Python (Django).

Our application will be communicating with two main sources of information: databases created by ourselves and the Roopairs API. The Roopairs API is already separately hosted by our clients. We would create and host our own databases on Google Cloud Platform. Presently, we only require a SQL database for backend logic specific to our product. If we are able to develop chatroom capabilities, a separate NoSQL fire base database would be hosted as well. We are currently unsure of the exact cost of our databases, but believe we would have a sufficient free tier spending limit.

3.3 Design

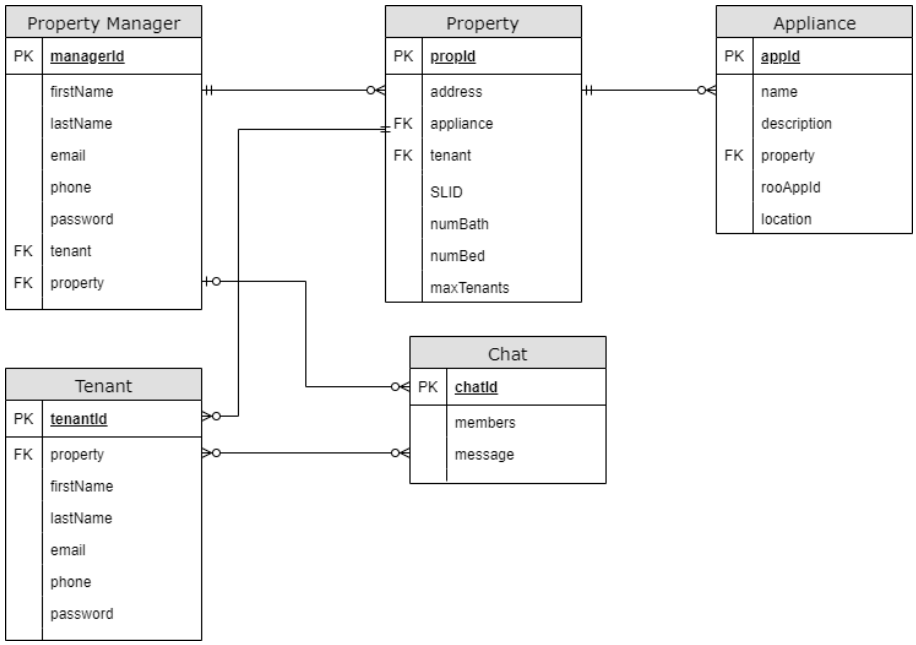

Figure 2: Entity-Relationship Diagram(Thomas Bergmann)

The figure above is the entity-relationship diagram that describes our database schema. An important consideration for this diagram is that the Roopairs database will be handling all information dealing with service providers. Our schema also contains references to primary keys in their database such as the rooAppId column in the Appliance table. Besides for these considerations, the entities and relationships are self-explanatory in the diagram.

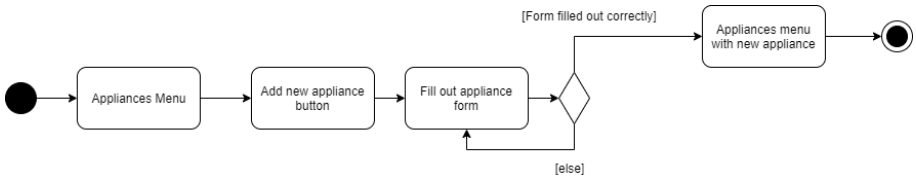

Figure 3: Activity diagram for adding an appliance(Thomas Bergmann)

This is the activity diagram for when a user wants to add an appliance to a property. The feature of adding an appliance along with many of the other features for our app is purposely meant to be simple to accommodate for users with little experience with technology. The page where the user fills in the form for the appliance requires the user to give basic information about the appliance such as what kind of appliance it is (for service category) and its name.

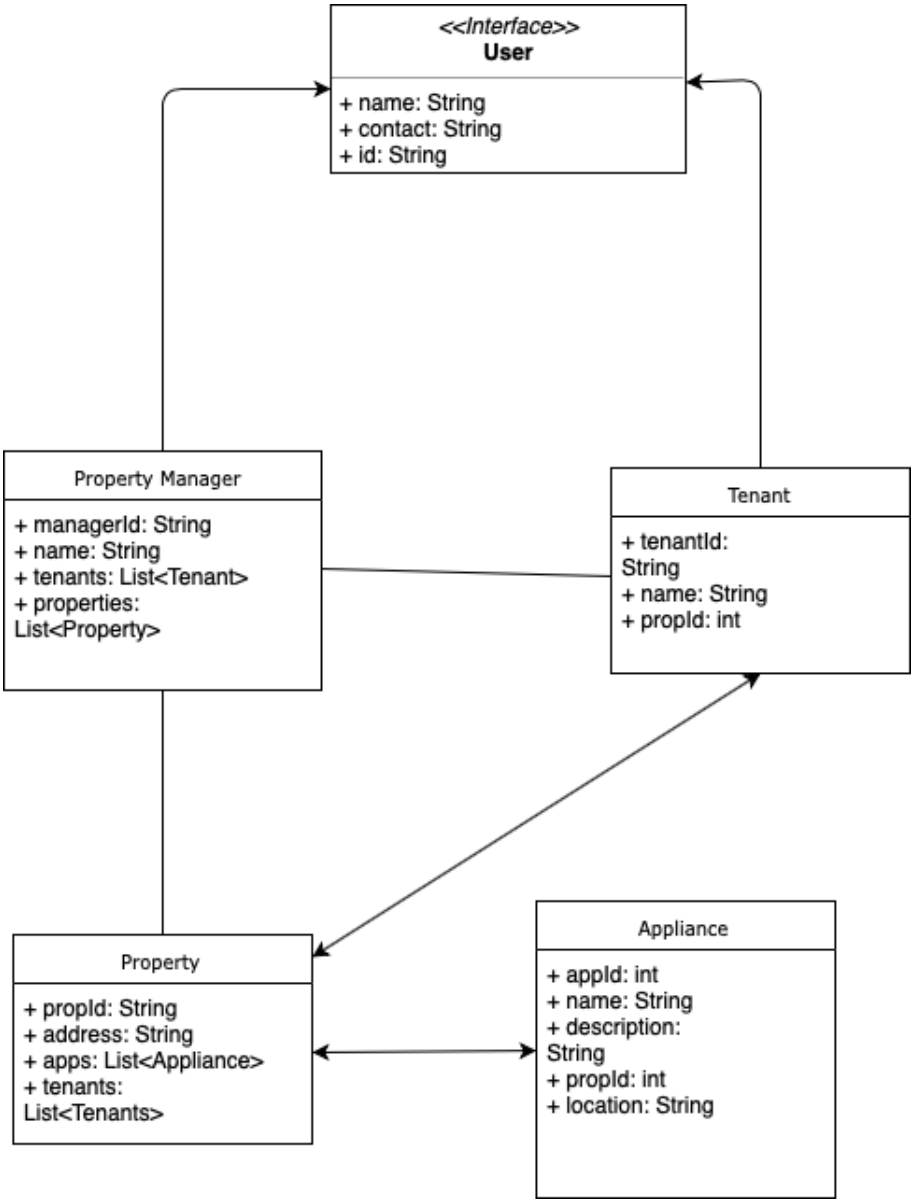

Diagram.png

Figure 4: Class diagram(Cesar Chacon)

Figure 4 is a high-level class diagram that describes the model that we would use to separate frontend display and functionality across the different users and objects. Each object in the model has unique attributes that describes their scope of knowledge as well as how they connect to other objects in the model. We decided to label property managers and tenants as Users to encapsulate their common attributes and functionalities. Property and Appliance are standalone objects, at the moment, that compose the attributes of users.

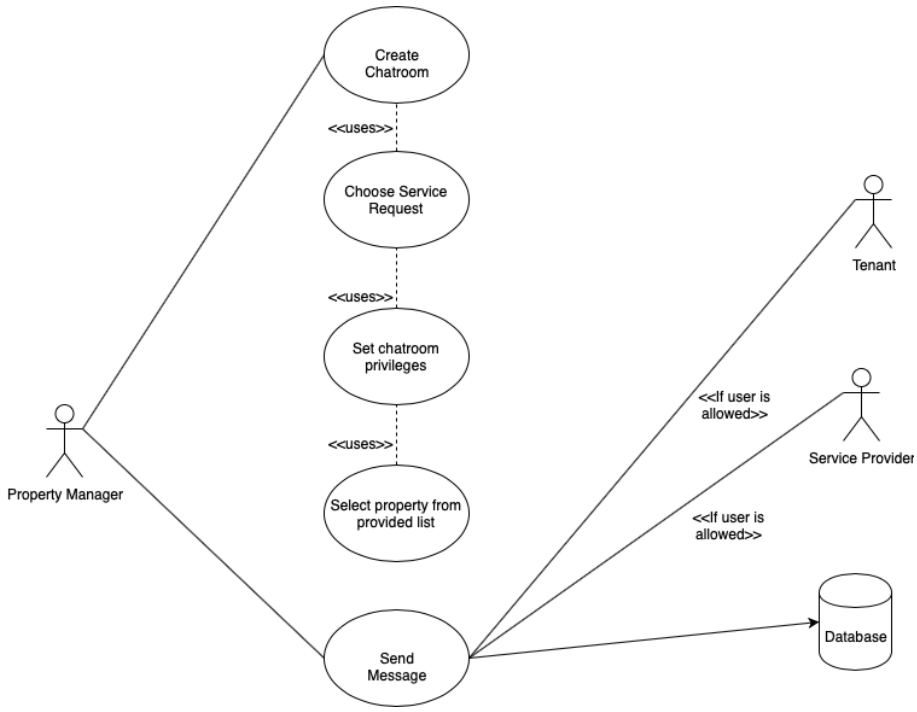

Figure 5: Use Case diagram for Chat Communication(Cesar Chacon)

Figure 5 describes the use case of a property manager creating a group chat within the application that directly corresponds to a specific service request. The property manager can set certain privileges or settings for the other users inside the chatroom. All the users are able to send messages (if they are allowed by the PM) which are sent to the database and updated to all users within that chatroom.

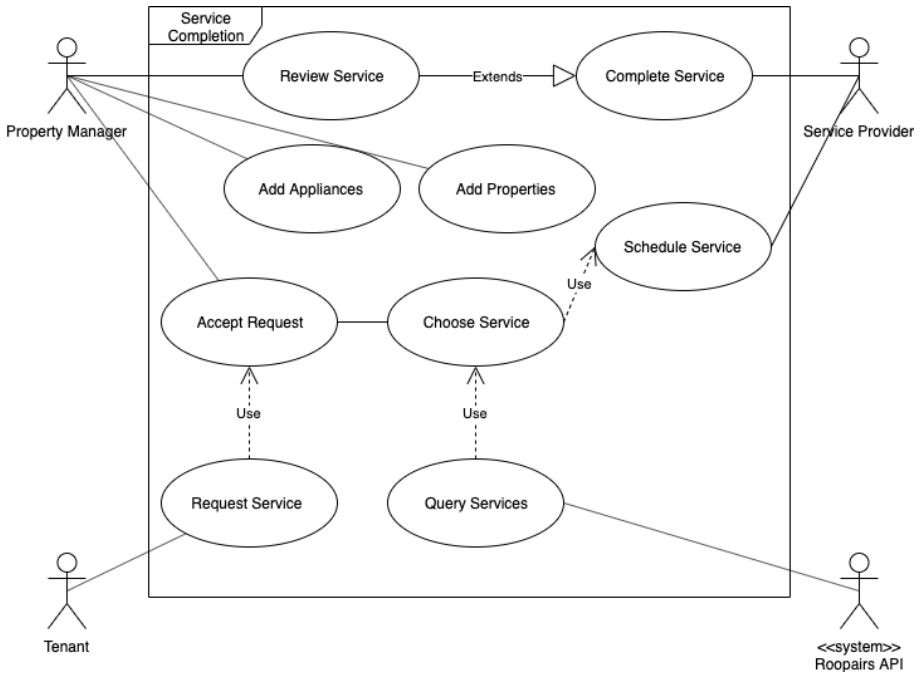

Figure 6: Job Request Use Case Diagram(Eeron Grant)

Figure 6 shows how the different Users interact with the primary use cases for a Job's lifetime and how these use cases interact with each other. Four actors are defined, although it should be noted that the Roopairs API is a system whose only involvement is for being a Point of Sales for context of these use cases.

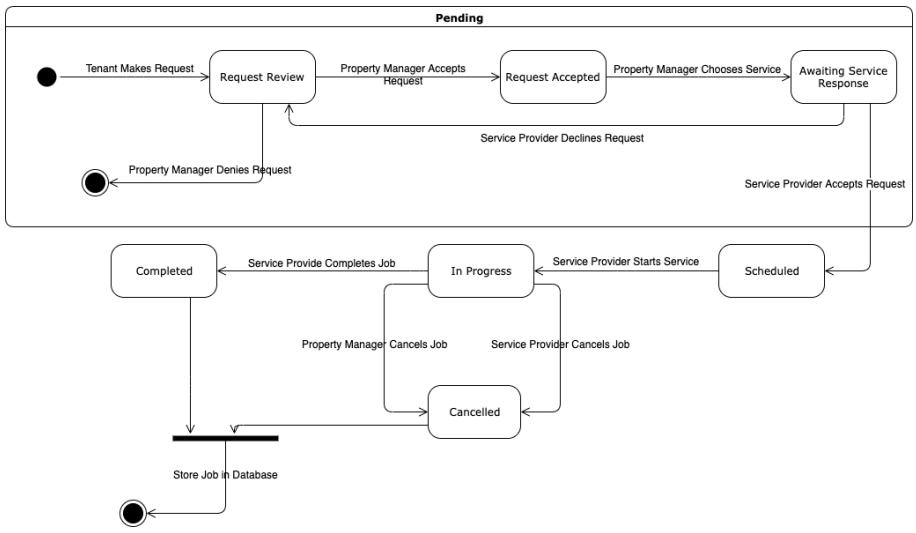

Figure 7: Job Request State Diagram(Eeron Grant)

Figure 7 showcases the different states that a specific Job will partake throughout its

lifetime. All Jobs will begin with a Super State, Pending, and will only change from this state when a Service Provider has scheduled for this job to be completed. Each state will provide different functionality for each user. For example, a Job with an In Progress State will allow the Property Manager or Service Provider to cancel. Although featured in this diagram, Review functionality may not be supported.

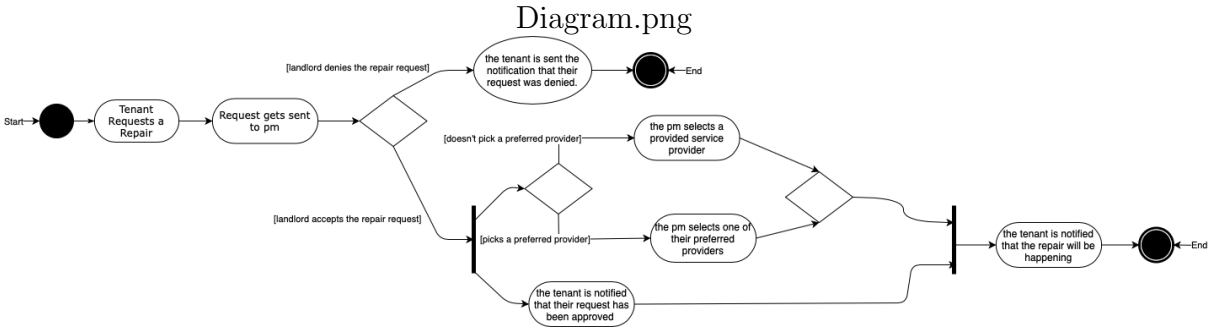

Figure 8: Job Request Activity Diagram (Adam Berard)

Figure 8 shows the activity diagram for one of the most common features of our application, when a tenant requests a repair. The feature starts with the tenant going into their application/web app and requesting maintenance on one of their appliances. That request is then sent to their property manager, who has the option to agree to the service or deny the repair. If the repair is denied, then the process is terminated at that point. If the repair is accepted, the tenant is notified and the property manager can decide to either choose from their preferred service providers or be given a list of five service companies. Once one is selected, the tenant is again then notified of who will be providing the service and when. That completes the repair request process.

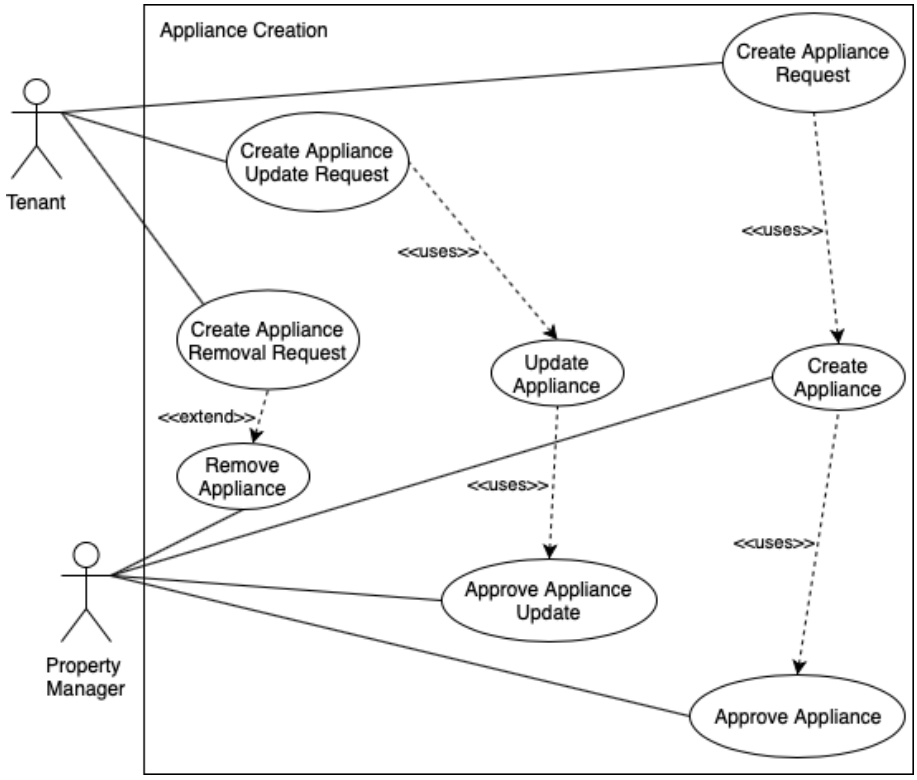

Figure 9: Appliance Management Use Case Diagram (Adam Berard)

Figure 9 shows the use case diagram for managing the appliances at a certain property. There are two actors in this situation which are the tenant and the property manager. For this use case the logic is quite simple. The tenant cannot outright add/update/remove appliances, but they can create a request for all of those things. Property managers are the only ones that can add/update/remove appliances, but they must put in all the information just like the tenant would, and then are shown the same approval screen that they would be had a tenant made the request. This keeps the property manager in charge, and in the know of what their properties appliances are.

3.4 UI Dialog Maps

Figures 10-16 show the flow through the user interface (UI) requested of our final product. The horizontal prototype used in this UI visualization was created by our Roopairs clients. All UI visualizations post authentication are from the perspective of a theoretical property manager. Tenants' screens will look similar but with reduced functionality.

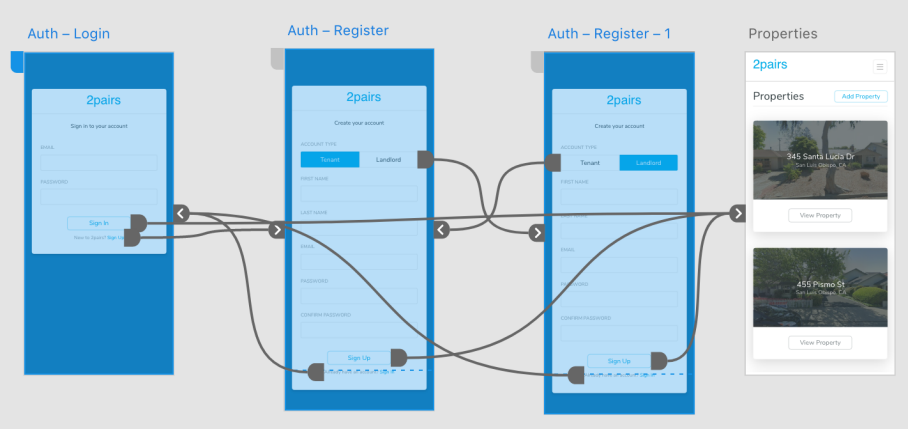

Figure 10: Authentication

Users start at the login screen upon launching our application. They can either register for a Roopairs account or login directly. Once they login, they will see the properties home page.

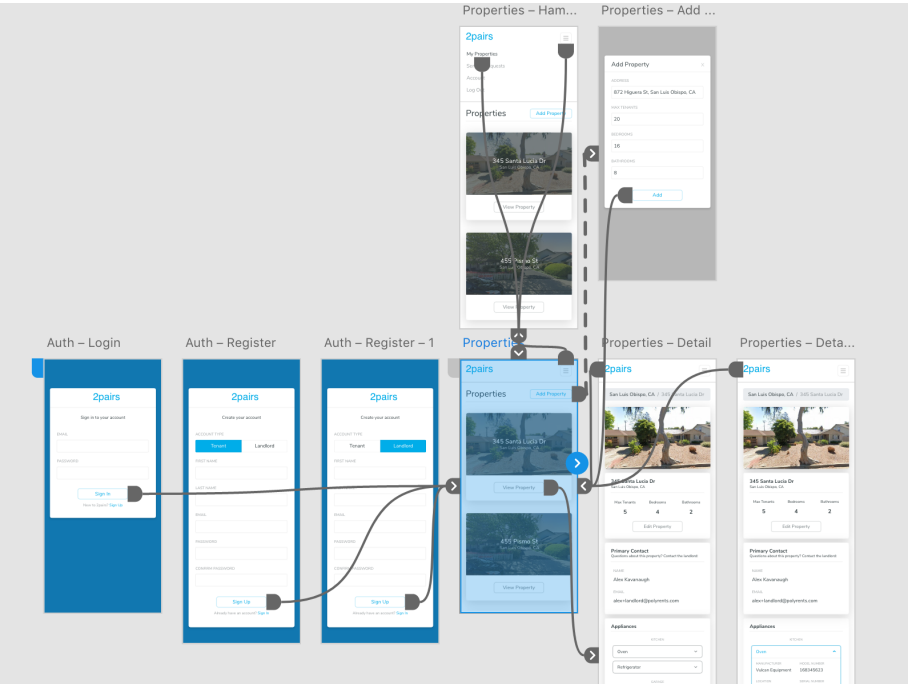

Figure 11: Properties Home

The properties home page lists a user's current properties. From here, users can select a current property to view or add a new property. Each current property displays a map card from Google, an image of said property.

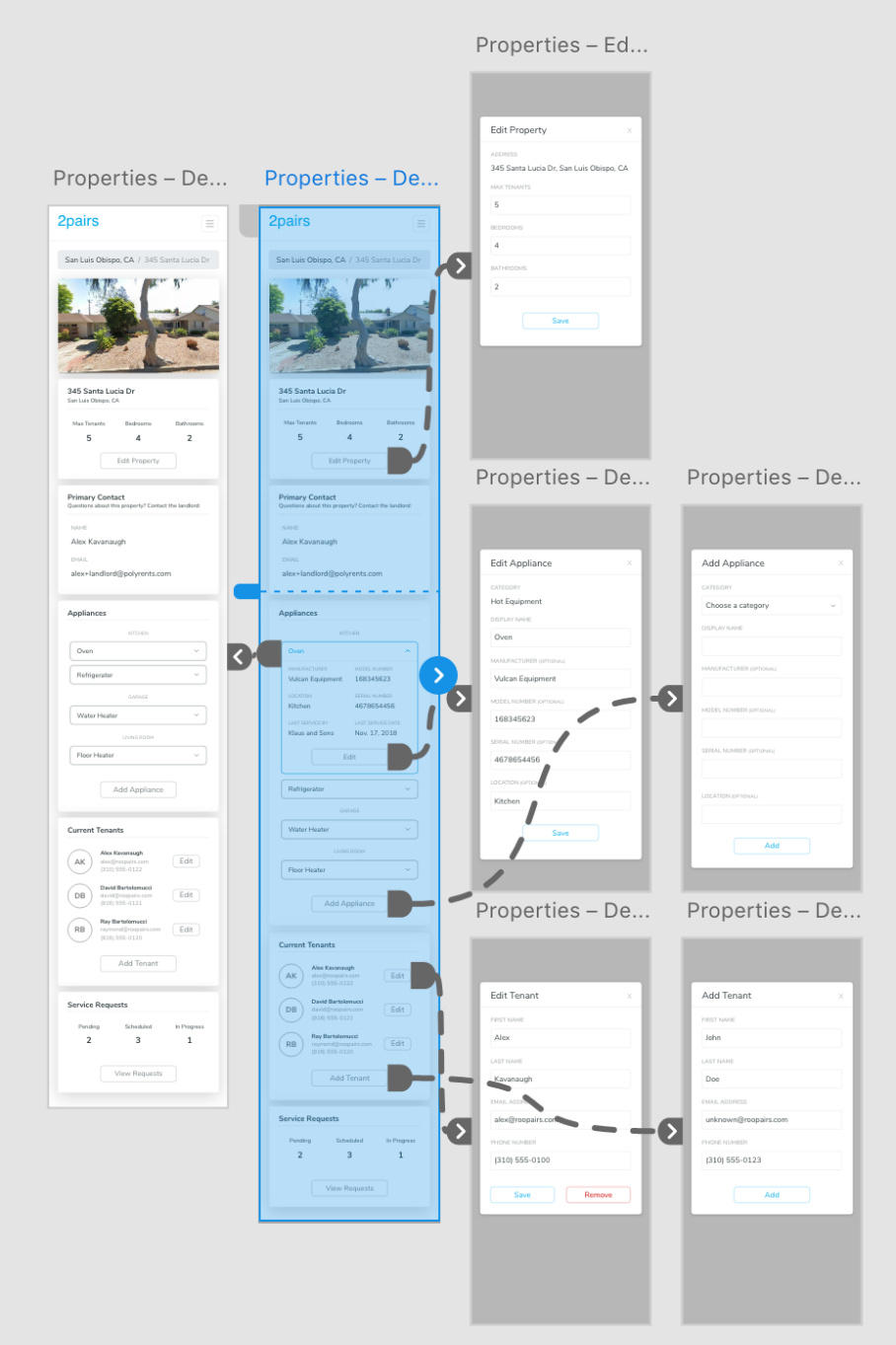

Figure 12: Property Details

When the user chooses to view a specific property, at the very top they can see that property's number of tenants, bedrooms, and bathrooms. Below that is the property's primary contact.

Below that basic information is a list of appliances logged for the property. Property

managers can add new appliances and edit current ones whereas tenants can always suggest new appliances for the property manager's approval.

Below appliances is a list of the current tenants. Property managers can edit current tenants and add new ones.

Finally, at the bottom of the property's details page, is a summary of current service requests for the property, which can be viewed in greater detail (it will send you to the corresponding section in Service Requests; see later figures).

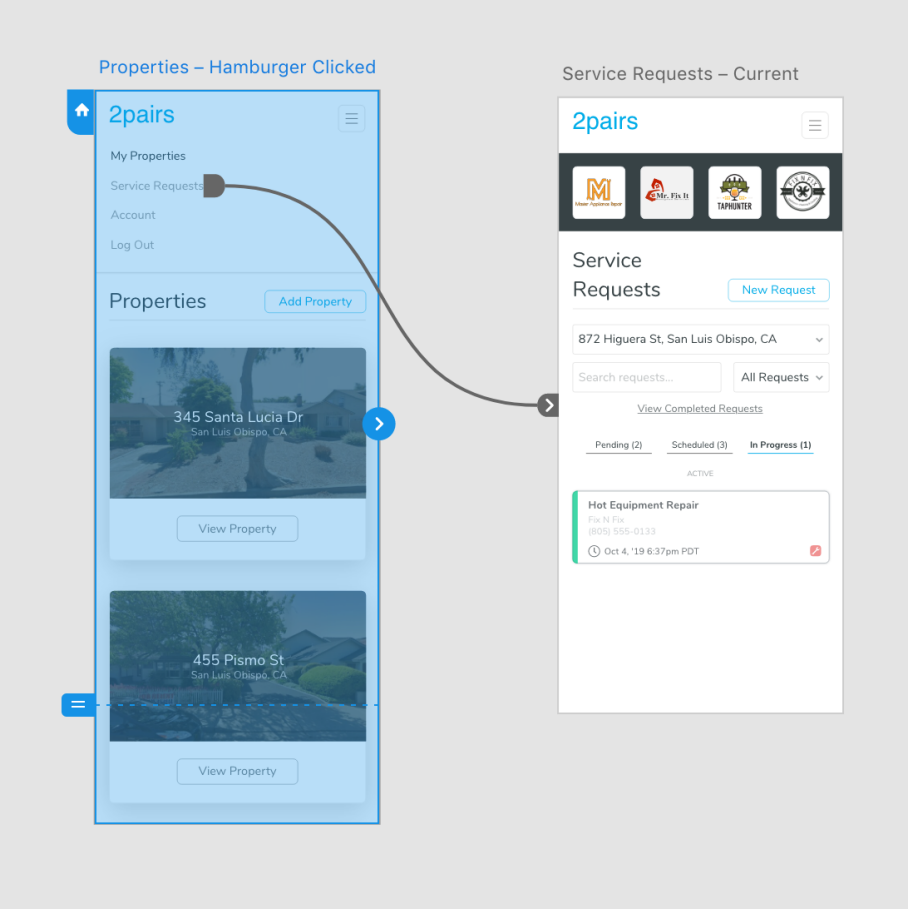

Figure 13: Properties to Service Requests

By clicking the hamburger icon, constantly at the top of the screen, users can navigate between the primary pages of the application: "My Properties", "Service Requests", and "Settings". Users can log out from here as well. Settings are not detailed at this time.

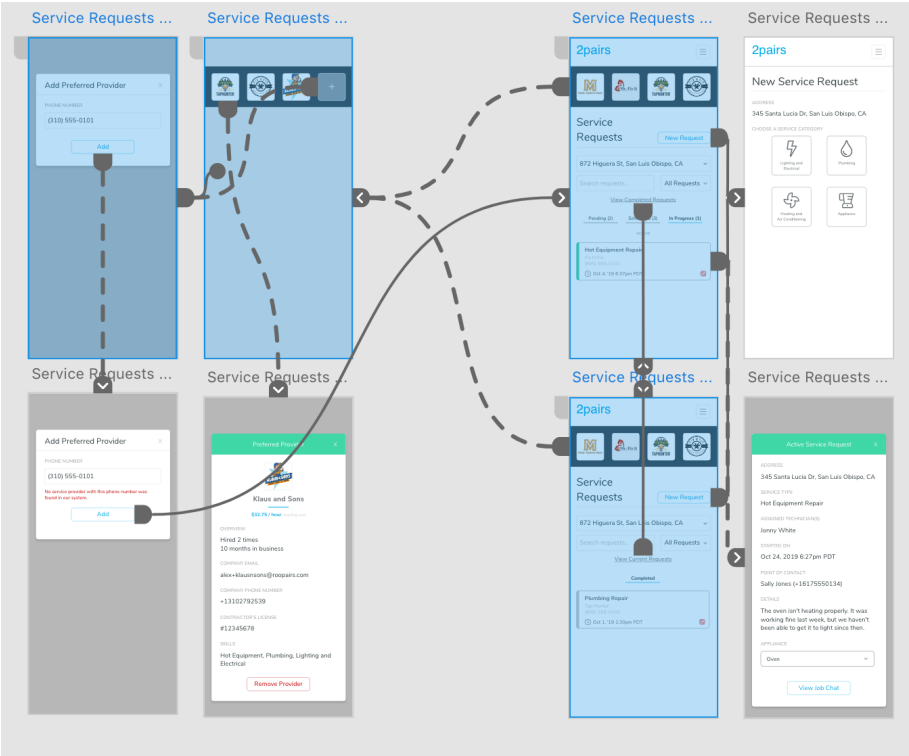

Figure 14: Service Requests Home

From service requests, property managers can see their favorite service providers, represented in a scroll-able bar of icons at the top of the screen. Here, they can manage their favorite service providers, view their service history, and make a new service request.

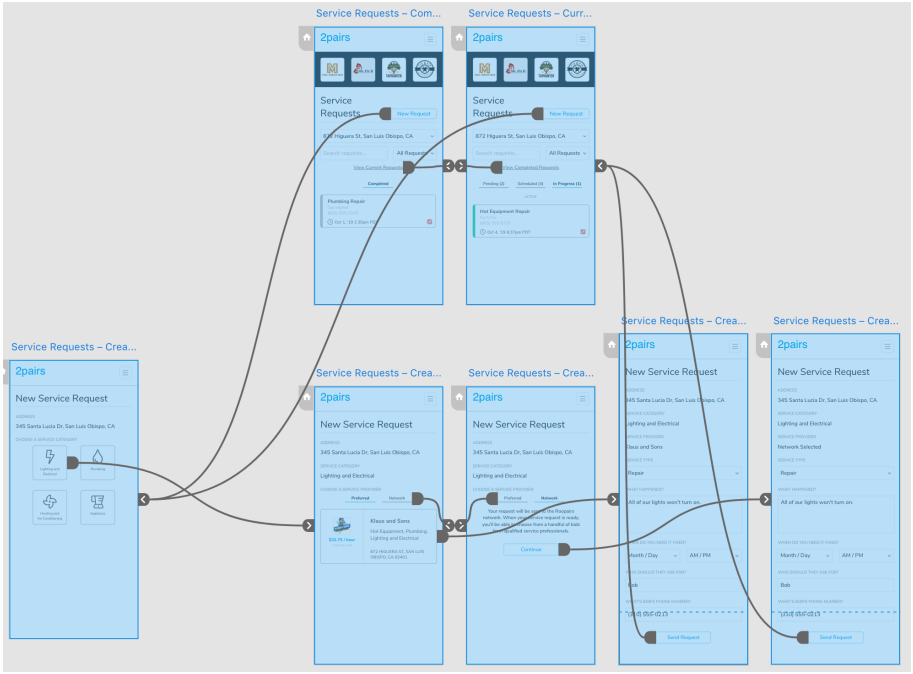

Figure 15: New General Service Request

When making a service request, users will choose from one of four service categories. If this category has no appliances logged underneath it for the property, they will fill out a general service request. For any service request, property managers may either select a service provider from their favorites, or choose from a list of five generated by the Roopairs network. Upon completion, they are returned to an updated service requests home page.

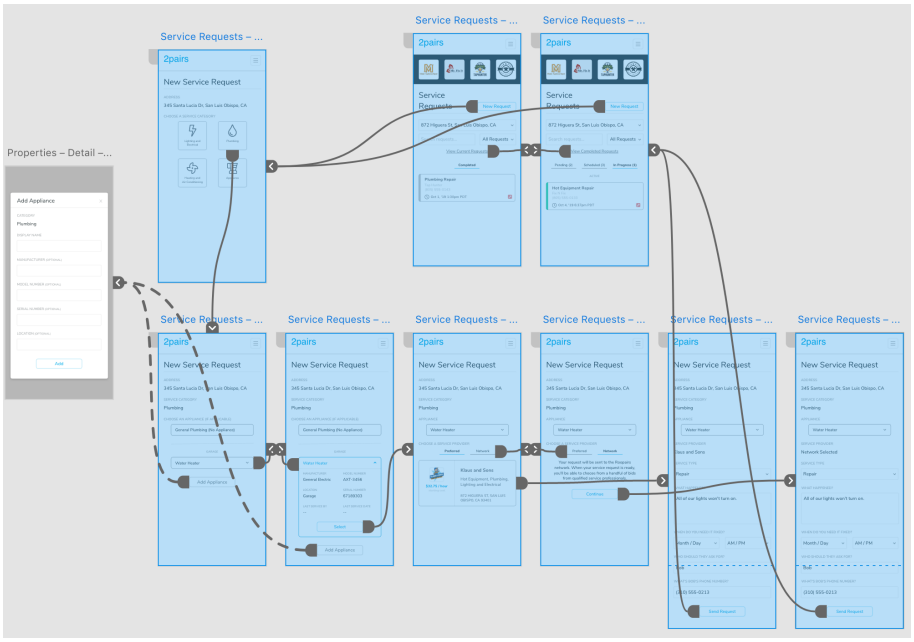

Figure 16: New Appliance Service Request

Alternative, when making a service request, if for the chosen category there are appliances for the chosen property, then the user may specify an appliance to be repaired instead. If the appliance to be repaired is not logged yet, a new appliance may be created from this screen as well.

4 Issues

1. What steps will the financial transaction between the service provider have to implement? Is that something that HomePairs will be providing a front end for or will that be taken care of entirely by Roopairs?

2. Will the service provider also have an account with the HomePairs application? The UI/UX walkthrough only stated that PM's and tenants would be users, but if that is the case, then Service Providers would not be able to communicate through our chatrooms unless that was implemented into Roopairs, where they already have an account set up.

A Glossary

Appliance (aka Equipment): A device or piece of equipment designed to perform a specific task in a building unit.
Building System: A foundation installed into a building unit that provides a specific

need. Examples include electrical systems and plumbing systems.
Building Spaces: Spaces in a building reserved for a common purpose. Kitchens, bedrooms, and washrooms are all examples of Building Spaces.
Chat: The communication/messages between two users.
Chat Rooms: A defined list of users that are capable of sending messages amongst each other.
DashBoard: The focus of the Toopairs home page. It show cases current most recent statuses of jobs associated with the user. It also is the main page for navigation throughout the Toopairs application.
Job (aka Service): A task that has been received, accepted, and currently be addressed by a Service Provider.
Job Request (aka Service Request): A proposal for receiving service sent from the tenant to Property Manager, and from the Property Manager to service. These can be approved or denied from the receiving party.
Landlord: A user of the Toopairs application whom is the owner of a given property. By default they are given the status of a Property Manager.
Navigation (NAV) Bar: A component that sits at the top or the bottom of a user interface that contains various buttons that navigate a user to different components of the application.
Notification: A message or alert presented onto a device indicating that something has happened.
PolyRents: An application whose primary use cases are to simplify the leasing process nearby Cal Poly by digitizing the housing application process.
Progressive Web App (PWA): A mobile friendly website application that has the capability of functioning offline.
Property (aka Service Location): An address specifying where service for a Job will occur.
Property Manager: The user who will determine to accept requests from tenants for services. These users can be the landlord be work on behalf of the landlord.
Real-time Status: The actual time during which something takes place. In the Toopairs app status will show Pending, Progress, and Completed.
Residential Sector: The potential market for this application. This market demographic mostly consists of homeowners and renters.
Roopairs API: The application programming interface that will conduct a majority of the business logic as constrained by the our customer -Roopairs
Service Provider: A company/individual that will give service to users. They will be listed from the Roopairs API.
Tenant: A user of the Toopairs application that is leasing a property from a Landlord. They typically communicate with the Property Manager.
Web Application: An application that is built on a device which has a UI built specifically for web browsers. It is divided into front-end (browsers) and back-end (servers) to allow remote usability on the web.