ARent: A Location-Based Augmented Reality Mobile Application for Finding Property Rentals

A Capstone Project Presented to the Faculty of the

School of Computer Studies

University of San Jose-Recoletos

Cebu City, Philippines

In Partial Fulfillment

Of the Requirements for the Degree of

Bachelor of Science in Information Technology

by

Camacho, Edrian Gustin D.

Negro, Sheena Justine A.

Dotillos, Joshua Andrei Nicolai C.

Mr. Ahdzleebee L. Formentera

Adviser

June 2022

TABLE OF CONTENTS

ACKNOWLEDGEMENT

To our families and friends, Despite the hardships and obstacles, we experienced throughout the epidemic, our family and friends helped the researchers reach our objectives and become the software developers we are today.

To the School of Computer Studies' staff, administrators, and instructors for their assistance and encouragement to participate in Accenture's Program the Future 2021, which presented the researchers with a challenge but the desired experience of representing the University as one of the top five national finalists.

To the coworkers we met throughout our on-the-job training (OJT) who aided the researchers by sharing technology and application suggestions.

Special Mentions:

Dr. Gregg Victor Gabison, who aided the researcher's trip by recommending the technology used and providing researchers a thorough discussion of Microsoft Azure.

Mr. Gene Abello, who aided the researchers in the creation of our Entity-Relationship Diagram (ERD) and provided structural advice when the researchers used database-based strategies.

Dr. Jovelyn Cuizon, who oversaw the researchers, aided in the formation of our algorithm, and redirected the course of our study, was responsible for directing our research.

Mr. Eric Magto, who assisted the researchers with the publication, provided advice and an opportunity to develop this concept.

Mr. Ahdzleebee Formentera, as the researcher's advisor and mentor in teaching the technologies to use, although having a tight timetable, required us to self-learn the

work that our advisor assigned to us and deliver the stated tasks for each sprint that was assigned.

Mrs. Josephine Petralba, the researcher's capstone coordinator, helped proofread the article to a high quality, compelling the team to pursue a well-thought-out outcome.

The team, which never gave up and persevered through difficult times despite the COVID-19 outbreak and typhoon Odette, was able to overcome each challenge and provide good outcomes while upholding the values integrated by the University.

ABSTRACT

One issue that many people nowadays face is that the job or school where they wish to work, or study is situated a long distance away from their homes. They must go from one island, city, or province to another, and many are unfamiliar with the region. Others possess real estate holdings that they want to rent out as a secondary source of income but have no idea how to advertise them properly. ARent, an online and mobile application that links rental owners and renters, was born. With the use of Augmented Reality technology, this application will aid rental searchers in discovering property rentals while also improving their user experience and engagement. It is also a better way for individual homeowners to advertise their rental properties. The development method for the essential features is present and discussed via several application implementations, such as Google Maps and ARCore Geospatial API, Azure Database PostgreSQL, and a simple, tailored matching algorithm for an ease-of-use experience while searching for property.

ENDORSEMENT

The project study entitled ARent: A Location-Based Augmented Reality Mobile Application for Finding Property Rentals was prepared by the following students: EDRIAN GUSTIN CAMACHO, JOSHUA ANDREI NICOLAI DOTILLOS, and SHEENA JUSTINE NEGRO, in partial fulfillment of the requirements for the degree of BACHELOR OF SCIENCE IN INFORMATION TECHNOLOGY, has been endorsed and is recommended for the acceptance and approval for ORAL EXAMINATION.

GENE P. ABELLO, MBA
Member

JOSEPHINE E. PETRALBA, MBA
Member

AHDZLEEBEE L. FORMENTERA
Adviser

JOVELYN C. CUIZON, DMHRM
Chairperson, CS/IT Department

GREGG VICTOR D. GABISON, DIT
SCS Dean

Approved by the Tribune for Oral Examination with the grade of PASSED.

GENE P. ABELLO, MBA
Member

JOSEPHINE E. PETRALBA, MBA
Member

AHDZLEEBEE L. FORMENTERA
Adviser

JOVELYN C. CUIZON, DMHRM
Chairperson, CS/IT Department

GREGG VICTOR D. GABISON, DIT
SCS Dean

CHAPTER I

RATIONALE OF THE STUDY

Pokemon GO may be the first thing that comes to mind when considering Augmented Reality (AR). Augmented Reality, however, is not limited to games. This cutting-edge technology affects a wide range of industries. According to Lumusvision, while AR video games anticipate generating $11.6 billion in revenue by 2025, other industries such as real estate and home improvement are not far behind. It serves as an aggregator of listings in this industry, categorizing and mapping property listings. It also includes instruments for conducting advanced searches, filtering pertinent data, and displaying data on maps. Students and workers, especially those seeking rental properties in a new area, can take advantage of this to the fullest extent. This study aims to provide property owners with a more strategic method of advertising their unit and an immersive experience for rental seekers in terms of providing directions to the location.

There are numerous applications wherein booking applications are prominent. In its most basic form, Airbnb is an accommodation booking engine. It is where individuals seeking temporary housing (typically for vacations) can find a place to sleep for the night. Booking.com is another website that offers online hotel and home reservation services. Due to their user-friendly search, tapping, and payment processes, these applications have gained popularity among travelers. Regarding Augmented Reality, Onirix Places would be one of them. Onirix Places' ultimate objective is to enhance the user experience by providing contextual information about the surrounding environment via Augmented Reality.

Although these systems achieve their intended purpose of connecting buyers and sellers, the method used to locate available properties is tedious. In Web-based applications, a search engine creates for available properties, after which clients pay a fee without knowing the surrounding area. Furthermore, these applications consider short-term stay inappropriate for housing-seeking students and employees. Incorporating a superior, unique, and engaging element into the marketing of homes, prospective renters must seek out easily searchable homes based on amenities, price, property type, quality, and the desired market. Augmented Reality could be beneficial in this regard. It allows for platform development, the expansion of marketing methods for property owners, and an enhanced experience for renters.

This paper aims to develop ARent, a location-based Augmented Reality mobile application for finding rental properties. It is a mobile application that connects landlords and tenants looking for rentals. With the use of augmented reality technology, this app will make it easier and more fun for people looking for a place to rent.

REVIEW OF RELATED WORKS

RealAR is an Augmented Reality and Artificial Intelligence application that converts floor plans into virtual walkthroughs that can be accessed remotely or on-site. Augmented Reality technology transmits digital visual components, sound, or other sensory stimulation to create an enhanced version of the actual physical world. It is a growing trend among companies that deal with mobile computing and commercial applications. Because AR technology enables personal digital experiences of still-under-construction areas, it is particularly advantageous for real estate agents who

manage properties before completion [2]. This application uses Augmented Reality to display the user's future house in a real-world environment.

ToGo is a mobile application that helps users find an apartment in the location of their choice [3]. It starts by specifying apartment preferences such as location and rent price. When an apartment matching the user's criteria becomes available in the area, they will be alerted. The app employs augmented reality to show the exact location of the unit. In finding apartments, users will set an area of interest by drawing a path to the location of their choice. Augmented Reality is implemented by showing available areas and their prices.

MagicPlan is an augmented reality app for real estate that generates floor plans from photographs. When a realtor uses a smartphone or tablet camera to take photos of the property, including anchor points such as walls, floors, ceilings, and doors, the app generates a detailed floor plan. An anchor is a point at which a 3D asset is overlaid in the real-world environment [23]. Realtors can use floor plans to demonstrate to buyers and tenants how they might utilize the space after moving in to ensure it meets their needs [4]. This application aims to create and edit floor plans, generate job estimates, examine space in 3D, plan do-it-yourself projects, and furnish a home. Possible users include real estate agents, architects, home inspectors, interior designers, small businesses, furniture retailers, and do-it-yourselfers.

realtor.com launches the Street Peak Realtor App. It enables smartphone users to aim their Android device at residences and instantly retrieve property information from the realtor.com® database by combining the user's location and orientation with a custom algorithm [5]. Using this augmented reality feature, users can observe price

bubbles forming over homes in the Street Peek viewfinder. It provides information to homebuyers about properties on any street, including the listed or rental price, the recently sold price, the expected price, and the number of rooms.

Lime is exploring ARCore's Geospatial API to enhance e-bike and e-scooter parking to be implemented in its app [27]. This application aims to assist e-bike and e-scooter riders in locating appropriate parking spots on streets and sidewalks. It employs augmented reality-enabled Visual Positioning Service (VPS) technology via Google Maps and the ARCore Geospatial API to evaluate Google's global localization feature in places such as Tel Aviv, Madrid, San Diego, London, Bordeaux, and Paris. Riders may locate their exact location with Google's global localization augmented reality technology using the ARCore Geospatial API by launching the Lime app and rapidly scanning their surroundings with their phone's camera and sensors. Lime allows to pinpoint the vehicle's exact location and ensure it is parking in a pedestrian-free area.

Bird VPS, or Visual Parking System, is driven by Google's ARCore Geospatial API. This new innovative parking solution allows Bird to geo-localize parked scooters with pinpoint accuracy using years of Google 3D scanning, augmented reality (AR) technology, and global Street View data [28]. Using the ARCore Geospatial API, Bird VPS compares a rider's photos in real-time with Google's massive data repository and Street View images. This relatively close technique yields centimeter-level geolocation that enables Bird VPS to detect and prevent improper parking with great precision, ensuring that our vehicles are only parking in authorized zones.

AirBits is a Microsoft Azure-based web application development company. Microsoft Azure is a public cloud computing service that provides computing, data

analytics, storage, and network resources. End users or clients can choose from these computing services to build, innovate, maintain, and scale new applications and run or deploy existing applications in the cloud. In addition, it provides a pay-per-use option regarded as very inexpensive or reasonable for using cloud resources [20]. Utilizing the Microsoft Azure Cloud for an application is significantly more reliable and user-friendly. The project's architecture and how well the database fits into it worked o in the application.

A weighted scoring approach generates a team-specific, value-weighted score for potential outcomes [29]. It obtains a score that facilitates object comparison by carefully selecting and weighing the user's criteria. ARent used this to determine which attributes carry the most weight. Airfocus is a platform for product management that is compatible with numerous devices. Airfocus emphasizes workstation cooperation, which facilitates feedback and delivery in retrospect. The team employs a weighted scoring system to prioritize product development and road map creation tasks. As a result, the program makes data-driven, intelligent decisions. Each task utilized weighted grading to determine the relative importance of tasks and delegates.

Jaccard Similarity, widely known as Jaccard's Coefficient, is a measurement of asymmetric information on binary and non-binary variables [19]. This algorithm best utilized in our matching system yielded a promising outcome. In a recent investigation of supermarkets, Teknomo and Kardi (2015) utilized Jaccard's Coefficient methodologies. This information facilitated recommending products to his customers through simple product matching. It may not have been the ideal algorithm, but it provided people with a well-considered method for determining which topics may interest them.

According to a study by Niwattanakul, S., Singthongchai, J., Naenudorn, E., and Wanapu, S., they discovered similar findings using Jaccard's Coefficient on their search engines for their document retrieval system. This application executes huge queries from enormous archives of documents [22]. Their concentration was on measuring keyword similarity. Jaccard's Coefficient effectively assessed the similarity of terms in word comparison, as shown by the findings. It comprises words in which each letter may interchange locations and be considered the same. In ARent, a particular amenity listed in the final column of a property is identical to an amenity shown in the first column.

PROJECT OBJECTIVE

This study aims to create and develop an application connecting rental owners and searchers. With AR technology, this application will assist rental searchers in finding property rentals, enhancing their user experience and engagement.

Additionally, the researchers aim to:

1. Superimpose digital content such as location pins and directional signage for a better user experience and engagement using Augmented Reality.
2. Provide a 360-degree panoramic image view of the place.
3. Implement a search filter based on the user's location, property type, price, and amenities.
4. Integrate APIs for data in the backend and Google cloud platform such as maps, polygons, and geospatial.
5. Integration of the Optical Character Recognition for the KYC verification

6. Provide a communicative platform as a linkage between rental owners and rental searchers.

PROJECT SCOPE AND LIMITATION

The application requires users to have good access to the internet and location services or turn on GPS. The application will only support Android devices running on version 7.0 (Nougat) or newer, specifically API Level 24 development, because of its AR capabilities. Location-based Augmented Reality applications do not fit a nearly perfect approach since it is still a developing technology. This technology poses the following challenges: Virtual objects should stay in place even when users move their cameras; the property rental location pin's altitude and posture because the Maps API only provides latitude and longitude data of the pinned property of the rental owner; Ascertain that the Global Positioning System (GPS) works with ARCore, building a software development kit by Google that allows augmented reality applications with little to no inaccuracy. ARCore's Geospatial API -- launched recently in May 2022, researchers have been unable to overlay multiple anchors due to a lack of online sources.

The web application is still running only in .NET 5.0, and some features are not available in.NET 5.0, as some of the packages need to be in a particular version will make the application work. OCR has limitations on the application where parts need to update their version and some that are restricted in letting the application work for its procedures, making the verification manual validating the use cases.

CHAPTER II

SOFTWARE REQUIREMENTS AND DESIGN SPECIFICATION

The researchers elaborate on the functional and non-functional needs for the "ARent" web-mobile application. Included are the application's overview idea, the use case diagram, the use case narratives matching to each use case of the Use Case Diagram (Use Case Diagram), the architectural diagram, the use case narratives, the activity diagram, the class diagram, the user interface, and the design.

APPLICATION OVERVIEW

ARent is a mobile application that connects rental owners and rental searchers. Search suitable Property Rental for the user based on the location/area, property type, price, and amenities. With AR technology, this application will assist rental searchers in finding property rentals, enhancing their user experience and engagement.

ARCHITECTURAL DIAGRAM

The architectural diagram shows the application architecture view, which helps the reader identify the applications, database, services, and interactions.

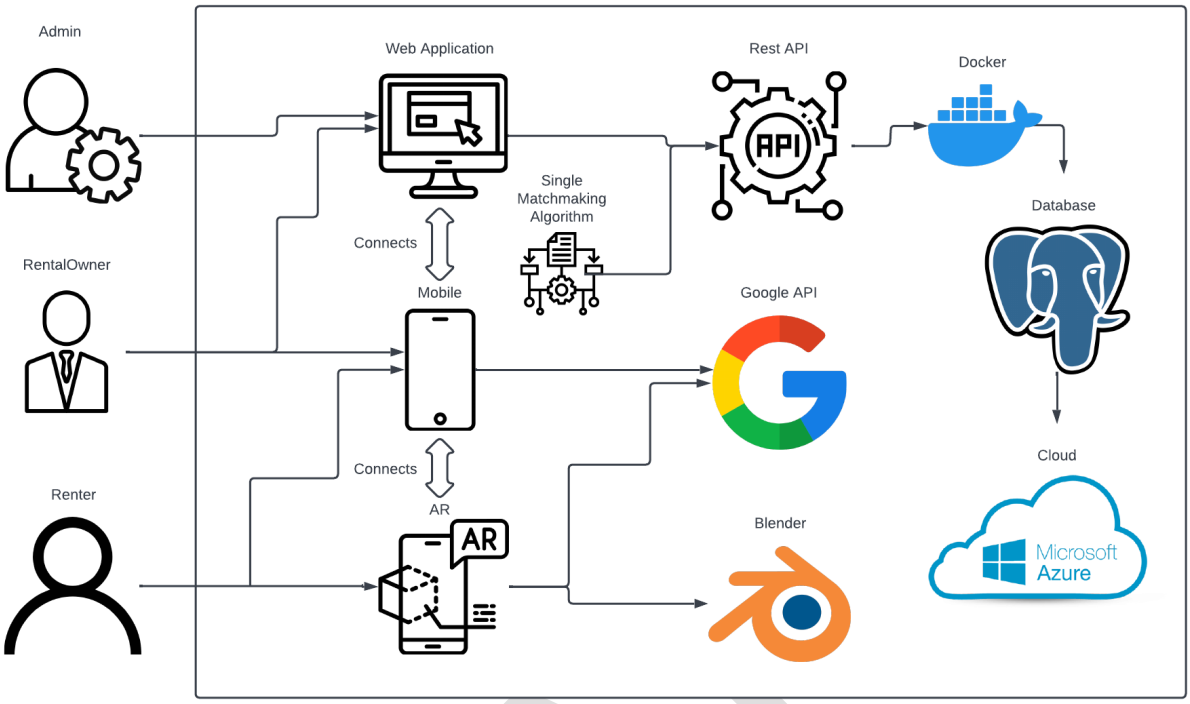


Figure 1: Architectural Diagram

The illustration depicts the architecture of the application. It demonstrates the link between users and the Web, Mobile, and AR integration of the application. The Web Application is the basis that enables the mobile device to connect to its functionality, obtaining the REST API for its services and the data modification offered by each controller. The Mobile Application is where Owners and Renters may cooperate with one another by exchanging the information necessary for both parties to understand their respective relationships. The renter can utilize the AR integration section, which demonstrates the capability of interacting with a real-world experience. The Google API is utilized by both mobile and augmented reality, despite the necessity for distinct features. Mobile utilizes the Maps and Polygons API to make its processes' features functional. AR employs maps and geographical data as required by the following techniques to ensure its success. AR utilizes Blender for producing assets. The web

application utilizes Docker to coordinate with the database and enable the database to connect to the Azure cloud via the application's endpoint.

USE CASE DIAGRAM

ARent's Use Case Diagram depicts the interaction between rental owners and rental seekers. It illustrates the primary functionality of the program and how both players carry out the application's required implementation or use cases.

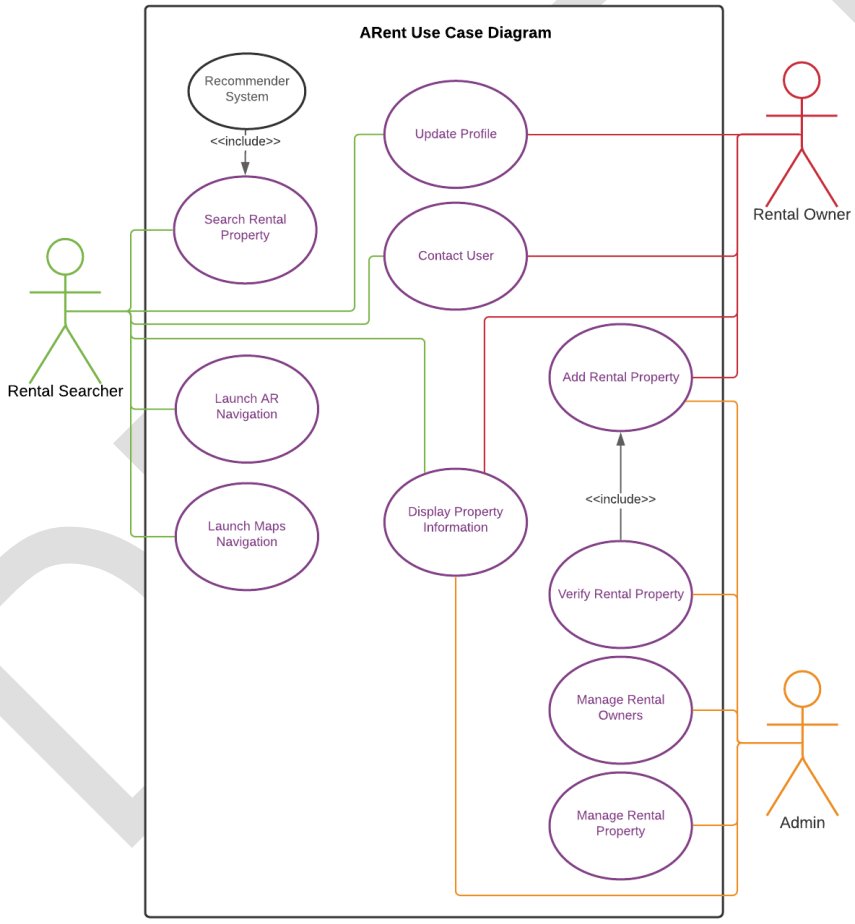


Figure 2: Use Case Diagram

The use case diagram represents the application's primary functionalities. There are two actors in mobile: the rental searcher and the rental owner. The rental searcher

can search rental property, including the recommended properties, based on the current location/area, type of property, price, and amenities, and may go to directions then display the property information. The rental searcher and owner can view chats and call history as they view messages and update their profile. The rental owner can add a rental property that includes Rental Property verification. There are two actors on the web: the rental admin and the rental owner. This diagram depicts what the actors can do with the system for the rental owner and can add the rental property as it views/edits the rental property. The rental admin includes three options: verification, view, or modify a rental property, which gives the capability of adding a rental owner, and lastly, it can add a rental property as it views or modifies rental property.

USE CASE NARRATIVES

The Use Case Narratives of ARent show the specific flow of web and mobile to aim at a specific goal. It explains how each use case functions on the entire application.

Search Rental Property Use Case Narrative

Use case:	Search Rental Property
Actors:	Rental Searcher
Type:	Essential
Precondition:	The user has logged in the system.
Postcondition:	Displays property information.
FLOW OF EVENTS	
Actor Action	System Response
1.) User inputs on the search bar.	2.) Displays rental properties based on the user's location.
3.) Taps Filter by option	4.) Displays "sorted by location/area, type of property, price or amenities" options.

5.) Choose a certain option.	6.) Displays all rental properties nearby sorted by the option chosen by the user.
ALTERNATIVE FLOW OF EVENT – Location turned off	
Actor Action	System Response
	1.) Displays the message “Location is turned off” shows the button to turn on the Location
2.) A) Pressed “Turn On” B) Pressed “Cancel”	3.) A) The properties will be shown in the screen B) The properties will not display

Table 1.1: Search Rental Property Use Case Narrative

Launch Maps Navigation Use Case Narrative

Use case:	Launch Maps Navigation
Actors:	Rental Searcher
Type:	Essential
Precondition:	The actor must be in Maps
Postcondition:	Displays property information and AR signages
FLOW OF EVENTS	
Actor Action	System Response
1.) Taps a rental property on the Maps.	2.) Shows the route and the distance from the user's location to the property.
3.) Clicks “Go there”	4.) Opens the camera and displays a 3D pin on the property. It will also display the property name and distance and a button “Show direction” at the bottom of the screen.
5.) Clicks “Show direction”	6.) 3D elements such as AR navigation signages will be flashed in the screen of the user guiding it to arrive at the destination. It will also update the distance as the user moves closer to the property rental.

Table 1.2: Launch Maps Navigation Use Case Narrative

Launch AR Navigation Use Case Narrative

Use case:	Launch AR Navigation

Actors:	Rental Searcher
Type:	Essential
Precondition:	The actor must be in Camera
Postcondition:	Displays property information and AR signages
FLOW OF EVENTS	
Actor Action	System Response
1.) Tap “Camera”	1.) System displays existing rental property via 3D pins within the vicinity.
3.) Directs the camera to a 3D pin	4.) System returns the name and distance of the pin. In this part, the user is far from the place. It will also display a button “Show Direction” at the bottom of the screen.
	5.) Conditional statement if the user wants to go to that pin or not.
	6.) If yes, 3D elements such as AR Navigation signages will be flashed at the screen of the user guiding it to the destination. It will also update the distance as the user moves close to the property rental.

Table 1.3: Launch AR Navigation Use Case Narrative

Contact User Use Case Narrative

Use case:	View Messages
Actors:	Rental Searcher, Rental Owner
Type:	Essential
Precondition:	The user has logged in the system.
Postcondition:	Displays chat and call history.
FLOW OF EVENTS	
Actor Action	System Response
1.) Click the “Chats” tab.	2.) Displays chat and call history.
3.) Clicks a certain actor.	4.) Displays the conversation and a “call” icon.
5.) Clicks the “call” icon	6.) Call the certain rental owner/rental searcher.

Table 1.4: Contact User Use Case Narrative

Update Profile Use Case Narrative

Use case:	Update Profile
Actors:	Rental Searcher, Rental Owner
Type:	Essential
Precondition:	The user has logged in the system.
Postcondition:	The changes in the user's profile will be displayed
FLOW OF EVENTS	
Actor Action	System Response
1.) Click the "Profile" tab.	2.) Directed to the "Profile" interface displaying personal information.
3.) Click the edit icon	4.) Displays user information with editable fields.
5.) Edit information	6.) Displays the updates in user's profile
7.) Click "Update Profile" button	8.) Displays the message "Successfully updated your profile"

Table 1.5: Update Profile Use Case Narrative

Add Rental Property Use Case Narrative

Use case:	Add Rental Property
Actors:	Rental Owner
Type:	Essential
Precondition:	The user has logged in the system as the rental owner.
Postcondition:	Successfully created a post for the added property rental.
FLOW OF EVENTS	
Actor Action	System Response
1.) Click "Create a post".	2.) Flashed on the screen the Property Description to be inputted by the rental owner such as Type of rental, rental name, location with a "Maps" icon, floor, and lot area, what's nearby along with its distance away from the property rental, amenities and terms and conditions.
3.) Clicks "Maps" icons	4.) Redirect to Maps
5.) Pins the location of the rental property.	6.) Returns to the "Create a post" screen displaying the latitude and longitude with its address on the location of the property rental.

7.) Click the” Continue” button.	8.) Flashed on the screen the details of the floors. It will prompt the rental owner to input how many floors and upload images of the outdoor and indoor of the house. The rental owners can add 360-view of the rooms.
9.) Click the “Continue” button.	10.) Displays the Verification Requirement based on what type of property.
11.) Uploads required documents/images for the verification of rental property.	
12.) Click the “Continue” button.	13.) Displays the overview of the post. Prompts a message that the post will be on the verification process.
14.) Click the “Confirm” button.	14.) Saves the data to the database. It will prompt a message “Post created successfully!”.
	15.) Once the post is verified by the admin, it gives a notification to the rental owner that the post was verified. Details of the rental property will now be visible to the rental searchers.

Table 1.6: Add Rental Property Use Case Narrative

Display Property Information Use Case Narrative

Use case:	Display Property Information
Actors:	Rental Admin / Rental Owner / Rental Searcher
Type:	Essential
Precondition:	The user has logged in the system
Postcondition:	The system will display full information of the selected rental property
FLOW OF EVENTS	
Actor Action	System Response
1.) Tap chosen rental property	2.) Database gets data from the chosen rental property
	3.) Displays full information of the rental property including the distance, type of property, price, amenities, and others. It also has actions: Add favorites, Contact Owner, Reserve Slot

4.) Choose an action	5.) System adds the chosen rental property to the Favorites list
	6.) Redirects to the Contact Owner screen and creates an instant message inquiring about the property
	7.) Sends a Reserve Slot request to Rental Owner
	8.) Users receive a notification regarding the action. Tap Ok.

Table 1.7: Display Rental Property Use Case Narrative

Verify Rental Property Use Case Narrative

Use case:	Verification
Actors:	Rental Admin
Type:	Essential
Precondition:	The user has logged in the system
Postcondition:	The system will verify the Rental Owner and the Rental Property in a list
FLOW OF EVENTS	
Actor Action	System Response
9.) Click “Verification” Tab	10.) The system will display the registered Rental Owners and the Rental Properties
11.) a.) Click “Rental Owners” Tab b.) Click “Rental Properties” Tab	12.) a.) Displays the list of “Rental Owners” b.) Displays the list of “Rental Properties”
13.) a.) Can Verify or not to the specified Rental Owner b.) Can Verify or not to the specified Rental Property	14.) The system will reflect what option you chose

Table 1.8: Verify Rental Property Use Case Narrative

Manage Rental Owners Use Case Narrative

Use case:	Manage Rental Owner
Actors:	Rental Admin
Type:	Essential

Precondition:	The user has logged in the system
Postcondition:	The system will display the Rental Owners in a list and also can add a Rental Owner to the list
FLOW OF EVENTS	
Actor Action	System Response
1.) Click “Rental Owner” Tab	2.) Display Rental Owners
3.) a.) Click “Add Rental Owner” b.) Click “Edit Rental Owner” icon	4.) a.) The system will display the information of the Rental Owners that has registered to the list b.) The system will add the information that the admin created and be registered to the list
5.) a.) User will choose if the information is confirmed, once confirmed it will then **submit** to the registered list b.) The admin has the option to Add a Rental Property and delete (archive) a Rental Owner	6.) a.) The system then added to the database b.) The system will delete (archive) or add a Rental Owner to what the admin has chosen for the option
	7.) b.) The system will update the database

Table 1.9: Manage Rental Owners Use Case Narrative

Manage Rental Property Use Case Narrative

Use case:	View/Edit Rental Property
Actors:	Rental Admin / Rental Owner
Type:	Essential
Precondition:	The user has logged in the system
Postcondition:	The system will display the Rental Properties in a list
FLOW OF EVENTS	
Actor Action	System Response
1.) Click “Rental Property” Tab	2.) The system will display the information of the Rental Properties that has registered to the list
3.) The admin has the option to Add a Rental Property and	4.) The system will delete (archive) or add a Rental Owner to what the admin has chosen for the option

delete (archive) a Rental Property	
	5.) The system will update the database

Table 1.10: Manage Rental Property Use Case Narrative

ACTIVITY DIAGRAM

The activity diagrams here present the series of actions and flows of the most prominent features, describing how they interact step-by-step.

Search Rental Property

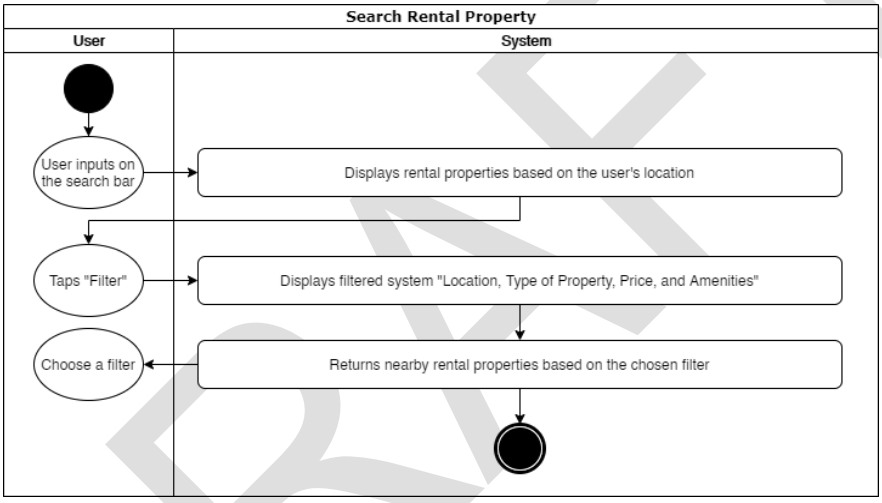


Figure 3.1: Activity Diagram for Search Rental Property

Figure 3.1 demonstrates that when a user enters information into the search field, the application subsequently presents rental listings depending on the user's location. There will be options for sorting by location, home type, price, and amenities. Utilizing filters, it provides neighboring rental homes depending on the sorting choice selected by the user.

Launch Maps Navigation

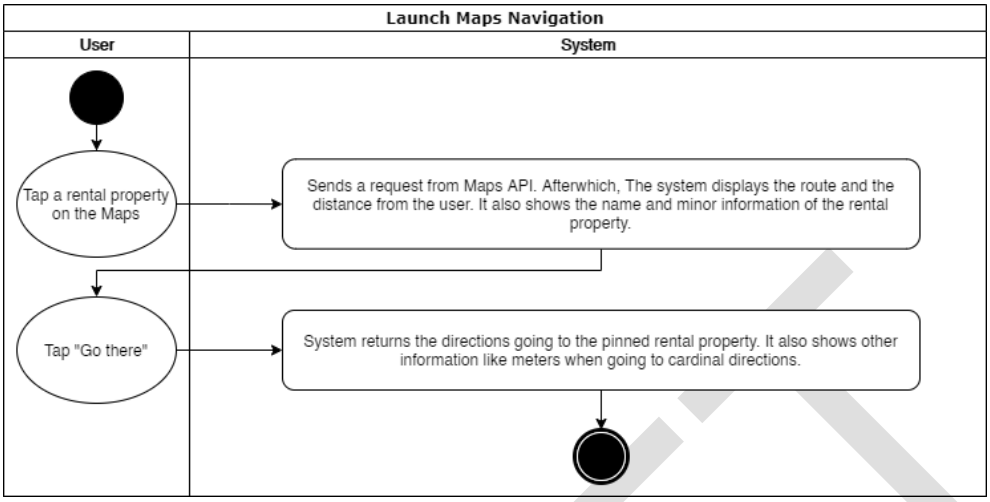


Figure 3.2: Activity Diagram for Launch Maps Navigation

Figure 3.2 depicts the UI when the Maps option is launched. The technology automatically shows the rental property's name and distance from the user's current location when the user clicks on a rental property on Maps. When the "Go there" button is tapped, the system provides a route from the user's current position to the selected rental property on the map.

Launch AR Navigation

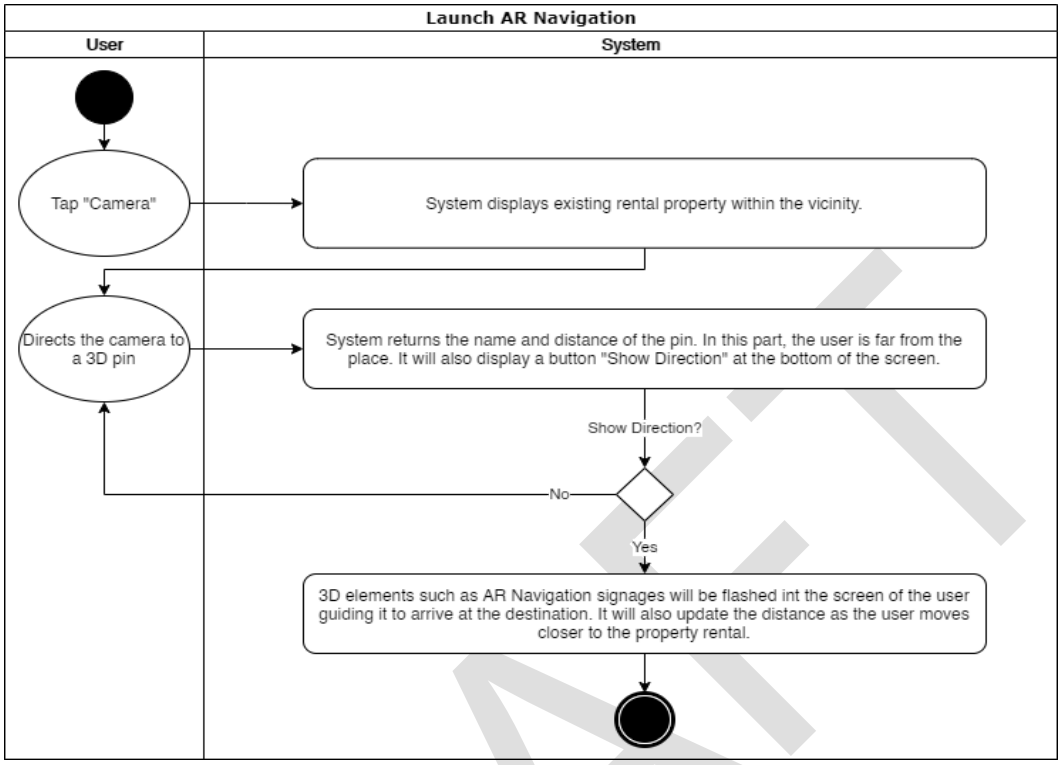


Figure 3.3: Activity Diagram for Launch AR Navigation

Figure 3.3 enables the program to utilize the camera when augmented reality navigation is implemented. When the user hits "Camera," a 3D pin is shown on the rental properties in the neighborhood. As the user points the camera to a 3D pin, the system provides the property's name and distance from the user's present position in real time. When a user hits "display direction," augmented reality (AR) navigation signs will take the user to the rental property.

Contact User

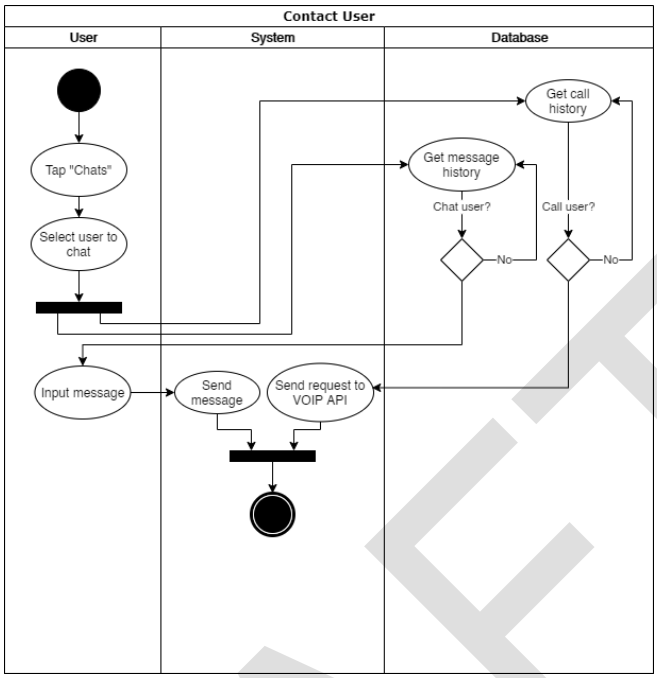


Figure 3.4: Activity Diagram for Contact User

Figure 3.4 demonstrates that the application enables rental searchers and rental owners to communicate through chat about the chosen rental property. Both parties have access to chat logs and phone logs.

Update Profile

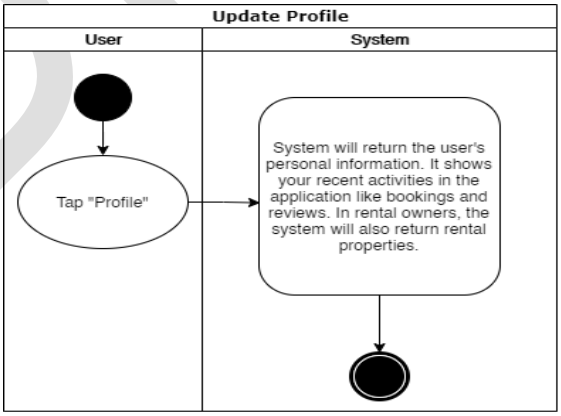


Figure 3.5: Activity Diagram for Update Profile

Figure 3.5 enables the user to update their application profile. When the user hits "Profile," the system displays their personal information, which may be edited and updated.

Add Rental Property

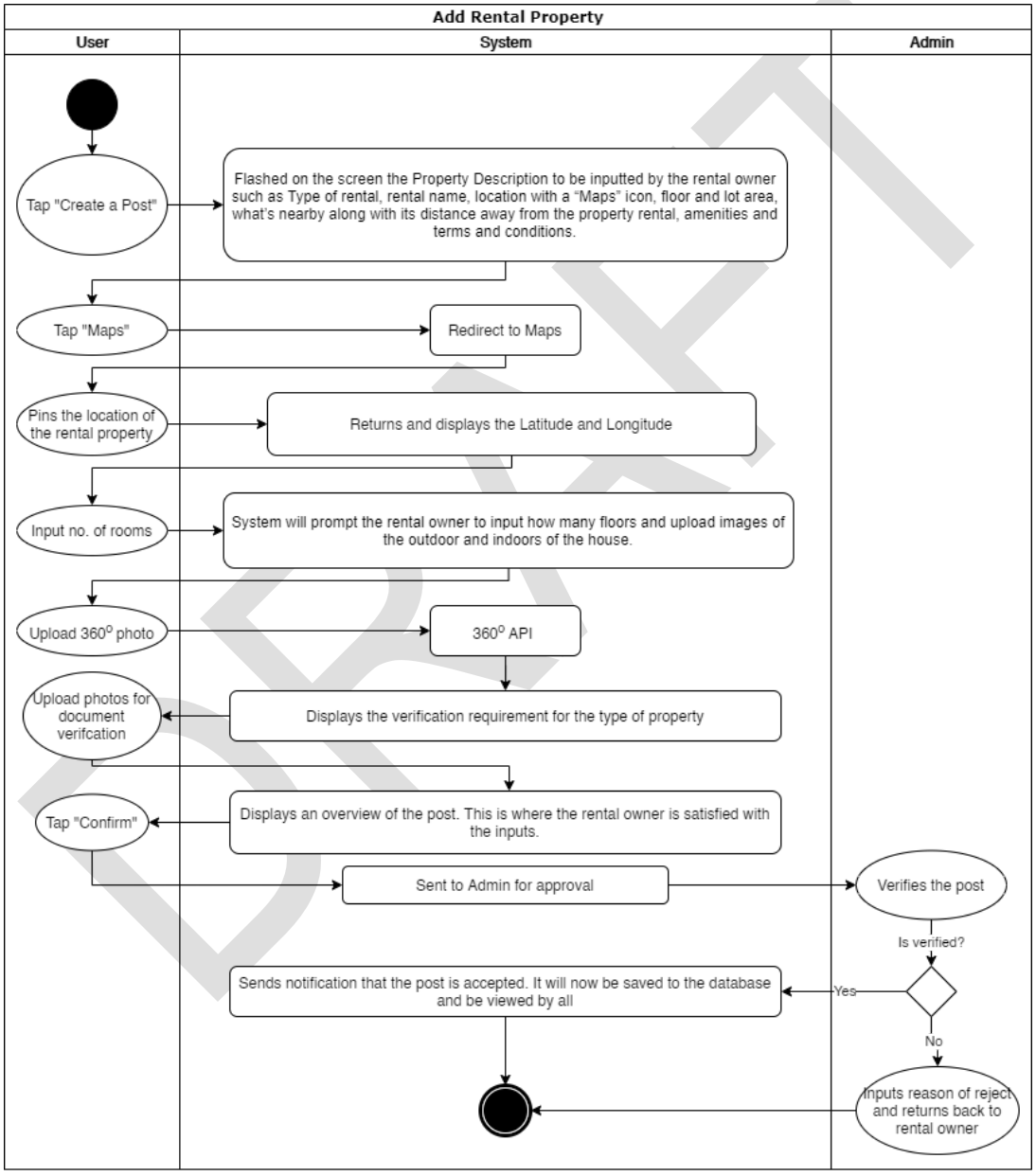


Figure 3.6: Activity Diagram for Add Rental Property

Figure 3.6 depicts the process by which a rental owner adds a rental property to the system. In writing a post, the user will include the kind of rental, the name of the rental, its location, floor, and lot size, as well as its distance from the property and its amenities. The system will provide the rental property's latitude and longitude when the user marks its position on "Maps." As the user progresses, it will enter information about the floor and submit photographs of it. The user is needed to supply papers or photographs for the verification of the rental property after giving property information. Once validated, the system will tell the rental owner that the post has been validated, and it will become available to rental seekers.

Display Property Information

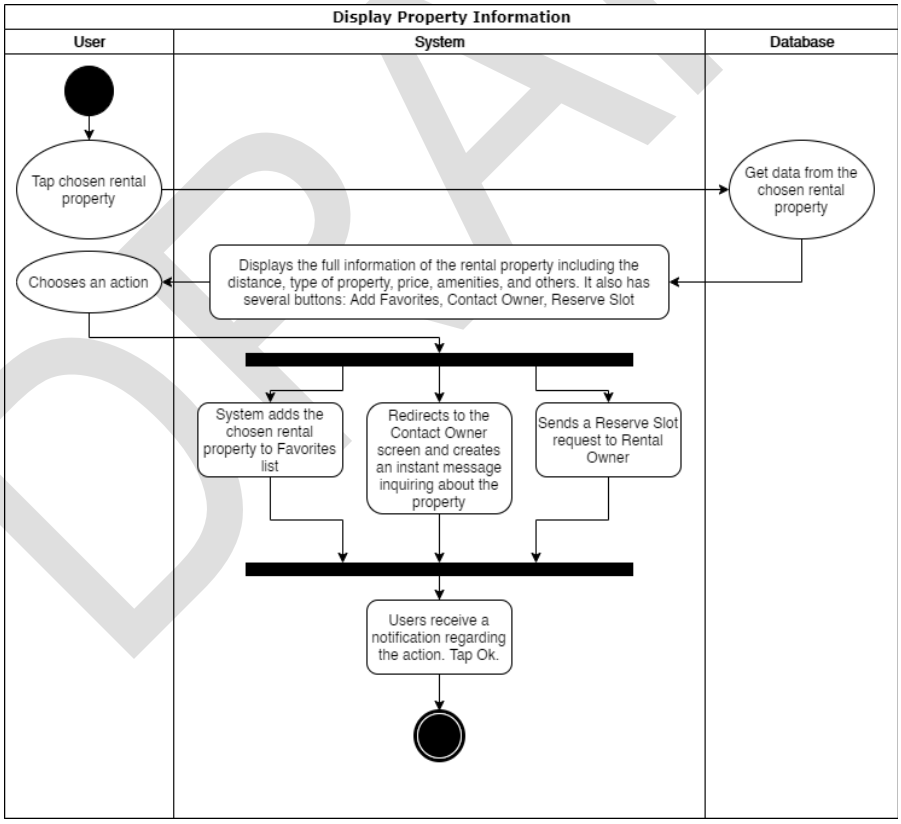


Figure 3.7: Activity Diagram for Display Property Information

Figure 3.7 illustrates the system's capacity to present property information to the process as its functionality is communicated. After selecting the rental property, the relevant rental property's information is shown. The rental seeker has the option of adding the listing to their favorites, contacting the owner, or reserving a spot. After selecting an option, the user will be notified of the resulting action.

Verify Rental Property

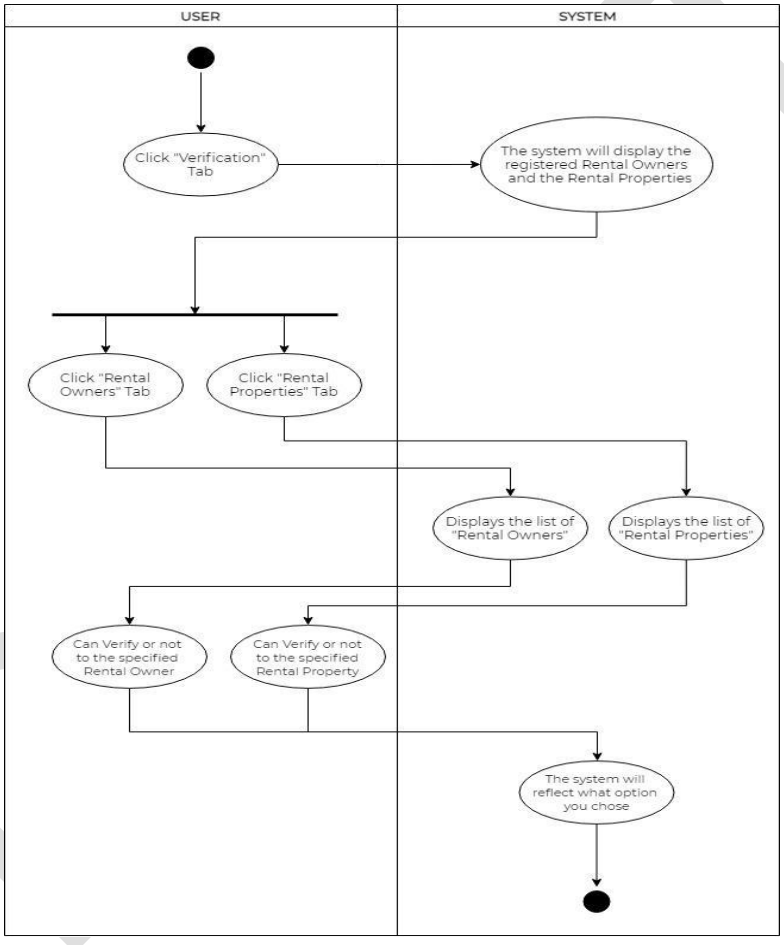


Figure 3.8: Activity Diagram for Verify Rental Property

Only the Rental Admin may check the rental owner and rental property information, as seen in Figure 3.8. The administrator will check the rental owner's and

rental property's documentation. It indicates whether to verify a certain owner or property.

Manage Rental Owner

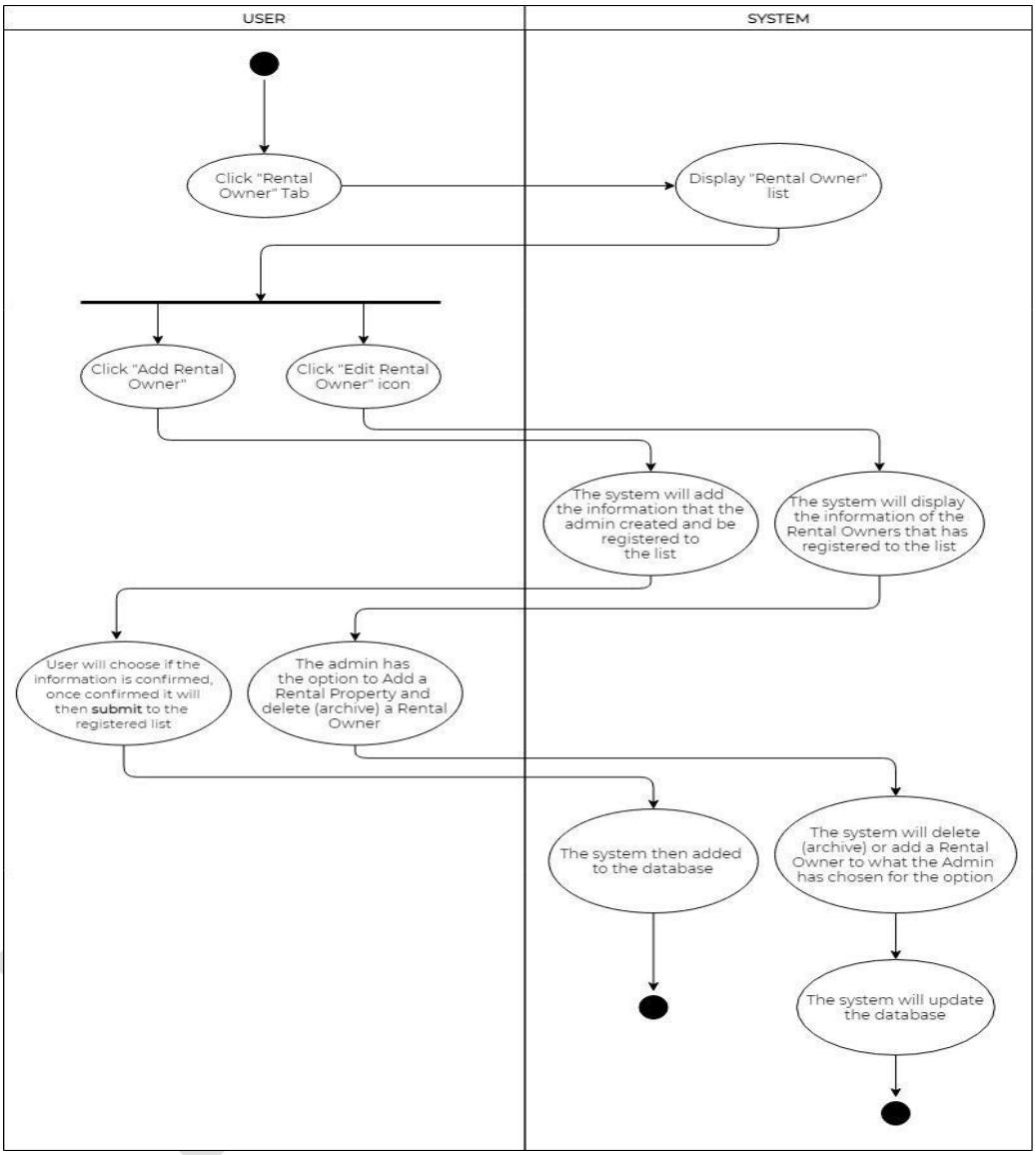


Figure 3.9: Activity Diagram for Manage Rental Owner

The procedure of adding rental owner information via the Rental Admin is shown in Figure 3.9. It will present the list of prospective database additions for rental property owners. The administrator can verify the rental owner's details. The two processes

indicate the tabs to be pushed and processed in accordance with their implementation flow.

Manage Rental Property

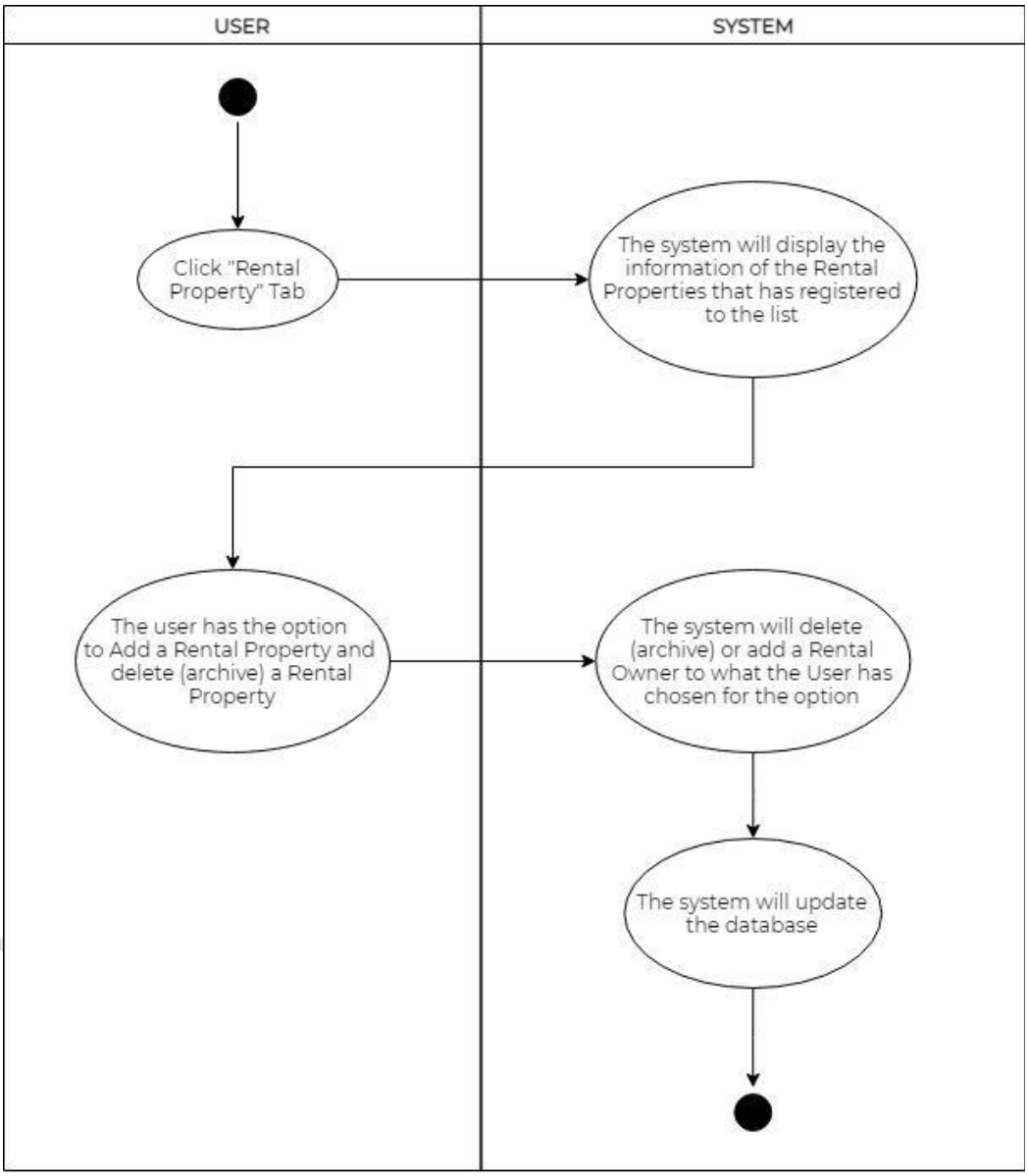


Figure 3.10: Activity Diagram for Manage Rental Property

Figure 3.10 depicts how rental property owners may handle rental property data inside the system. The system provides a list of rental properties that have been

uploaded to the database, allowing the user to review and verify the property information.

CLASS DIAGRAM

Figure 3.11 illustrates the link between the application's functionalities. It contains our data's entity relationship diagram.

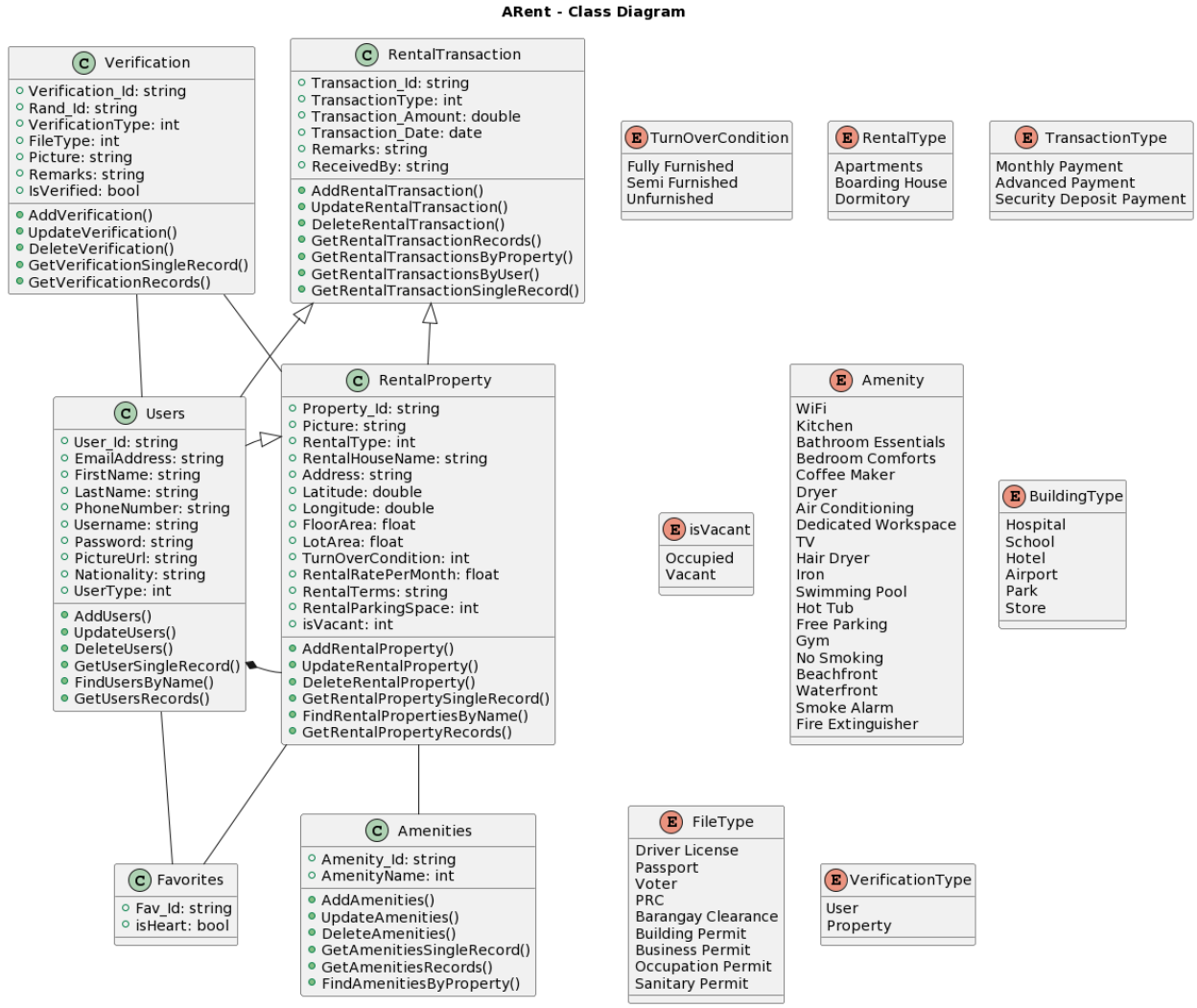


Figure 3.11: Class Diagram

Users

The entity is the user, categorizing them as a rental owner or a rental searcher. This entity is legally associated with the rental properties that they will own. This class stores the user's basic information related to authentication and is described in the verification entity. After coordinating the renter and rental property with the Rental Transaction entity, that entity will receive all transactions.

Rental Property

The user with a relationship aggregated links to a rental property through this connection. Property owner removes property in the application when it deletes their rental. It correlates in two different ways, which are as follows: One requirement is that the owner of the rental property must also own the rental property. The second factor, which serves as the rental property's rental transaction correlation, is that the tenant has previously rented from that property.

Rental Transaction

The transaction between the renter and property owner is a handshake.

Verification

Before allowing a user or rental property to use the application, it conducts verification on the user and the rental property. The verification process is separate for rental owners and rental properties. Submitting their documents is required, as it applies to verifying users and rental properties. Users must possess certain documents, including driver's licenses, passports, voters' permits, and PRCs. A barangay clearance, building permit, business permit, occupancy permit, and sanitary permit are all included in the rental property.

Amenities

The rental property uses it exclusively for the entity and stores data on the amenities delivered by a single property.

Favorites

The user may save all their preferred properties, giving them fast access to favorites on the dashboard. The user can favorite any property that is available on the site.

USER INTERFACE AND DESIGN

This section provides the user interface and user experience design for ARent, exhibiting all of the application's screens for the prospective user involvement of the program.

Web

Figure 4.1: Landing Page

Figure 4.1 depicts the application's homepage, which is the default or startup page of a browser and has a login button that navigates to the login page.

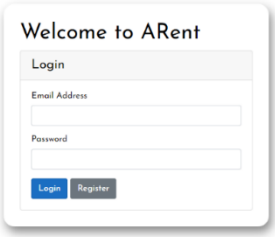


Figure 4.2: Login Page

Figure 4.2 depicts the application's login page. The user must provide their email address and password to access the system.

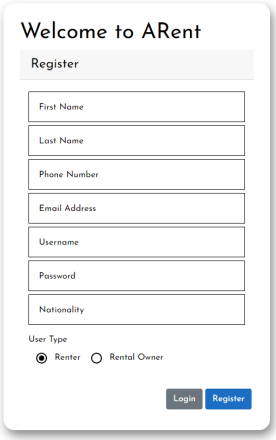


Figure 4.3: Registration Page

Figure 4.3 depicts the application's registration page. It displays the user's required information fields. The user must submit the required information to complete the job on the page. There are two types of information that the user must input: First, we will utilize personal information such as first and last names, phone numbers, and

addresses. Username, password, email address, and user type comprise the second section of account information.

a) Admin User Interface

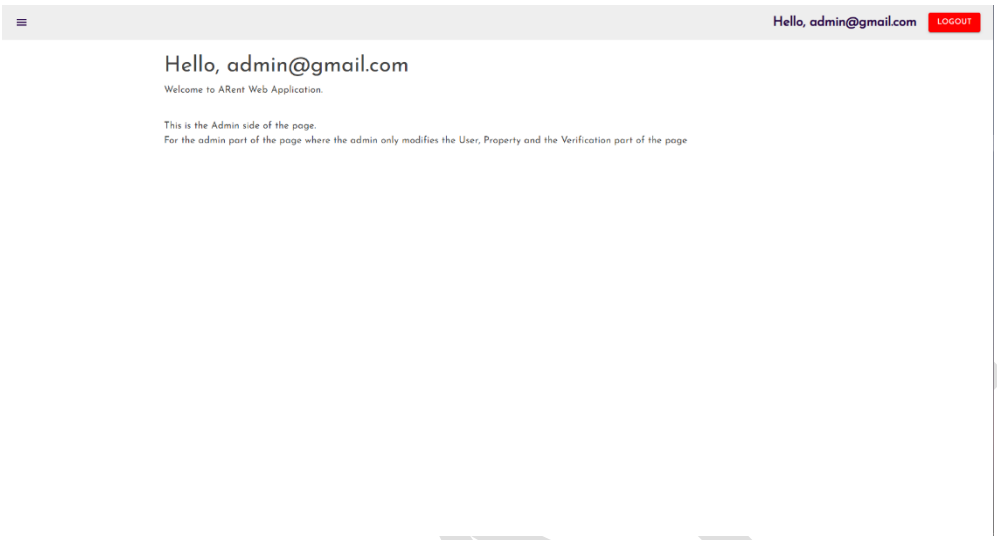


Figure 4.4: Admin – Dashboard

Figure 4.4 depicts the application's admin Dashboard. It is the default page when an administrator logs in with their credentials.

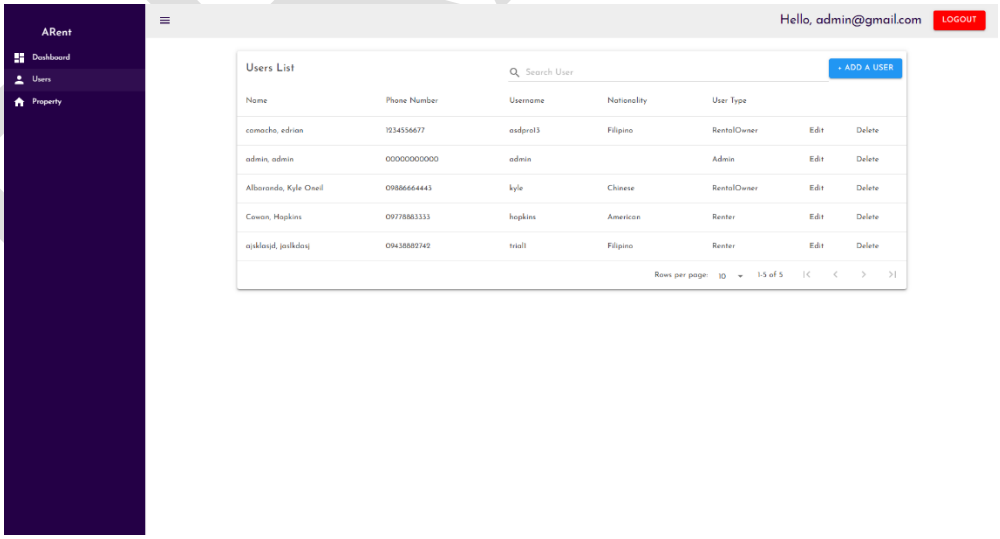


Figure 4.5: Admin – Users Page

Figure 4.5 depicts the Admin Users page of the program; a user's information is shown in the table and may be modified or removed by an Administrator. You will be able to add a user for specified reasons through a button.

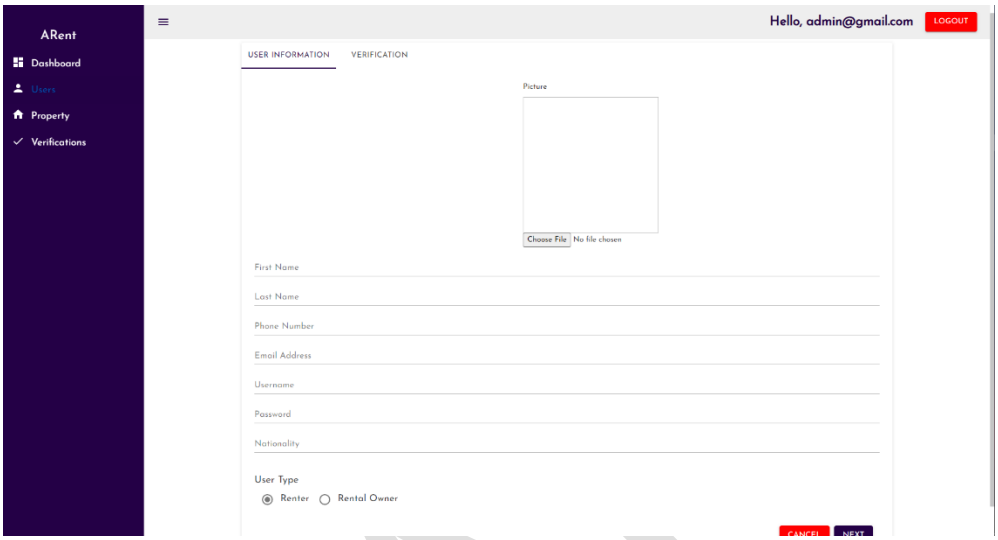


Figure 4.5.1: Admin – Add User Page

Figure 4.5.1 depicts the admin add user page of the application, where you will add the users specified in the previous figure, which contains the information and identifies the user or user type for that instance.

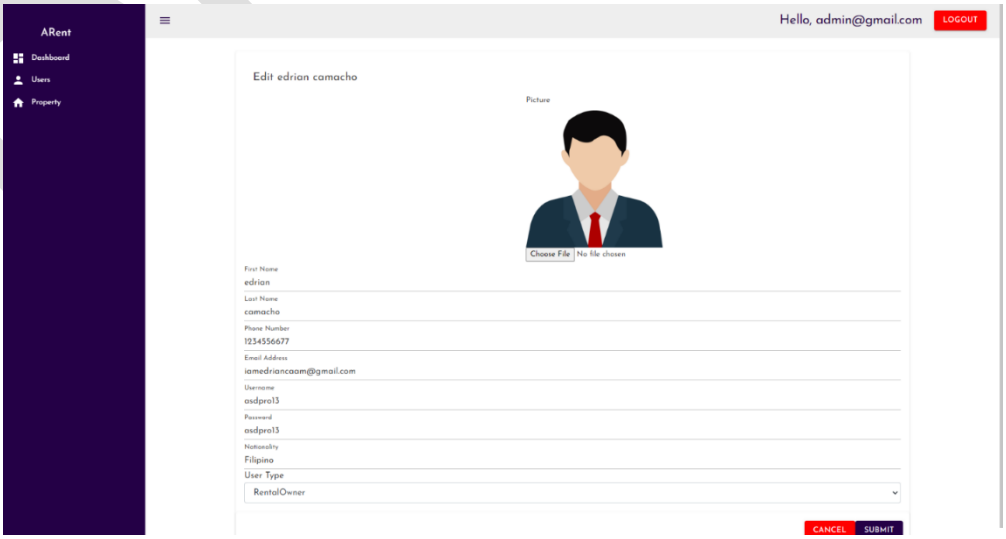


Figure 4.5.2: Admin – Edit User Page

Figure 4.5.2 depicts the admin edit user page of the application, in which the administrator may modify the user's credentials and information.

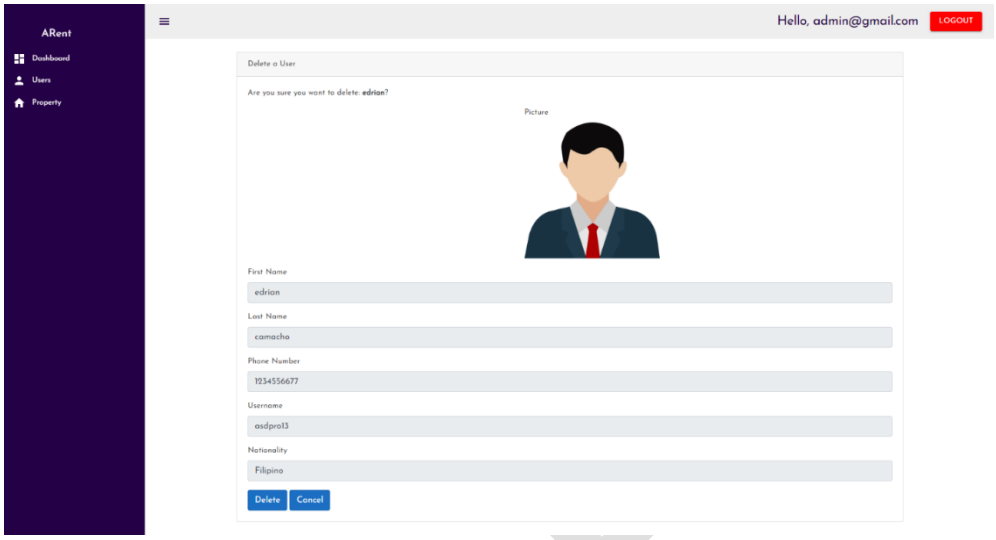


Figure 4.5.3: Admin – Delete User Page

Figure 4.5.3 depicts the admin delete page of the program, where the administrator may remove a user who is no longer in use or has been deleted for other reasons.

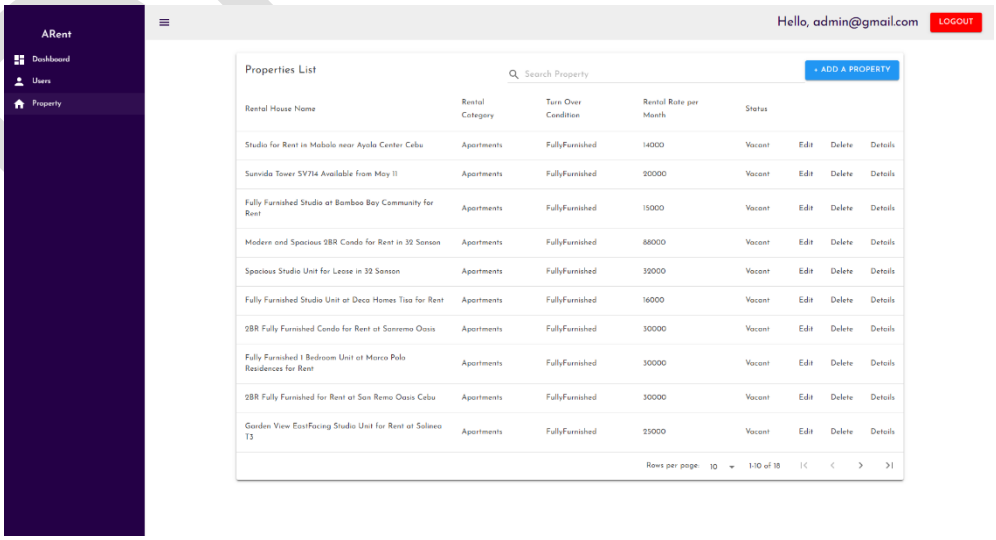


Figure 4.6: Admin – Property Page

Figure 4.6 illustrates the admin property page of the program; the information of a property is shown in a table and may be modified, detailed, or removed by an administrator. There will be a button that allows you to add a certain property.

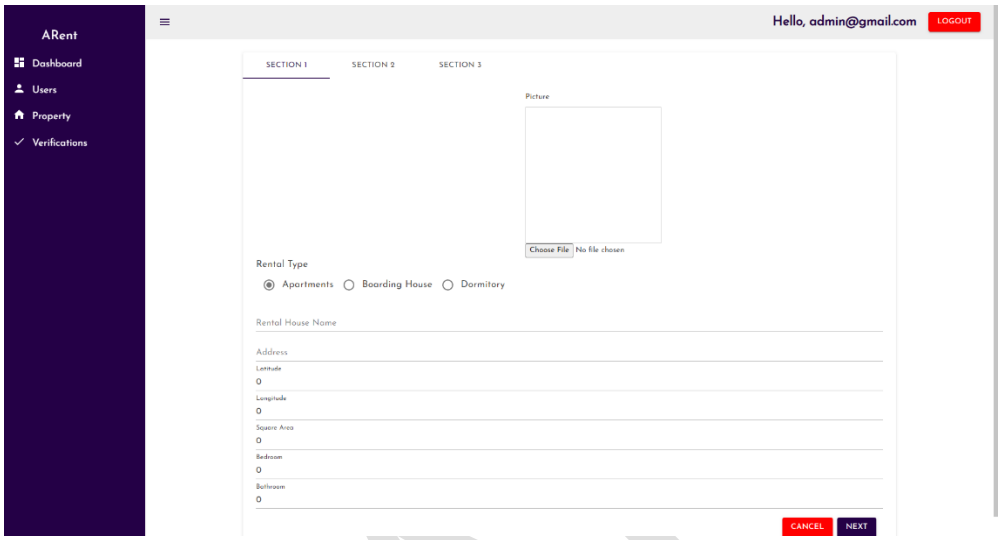


Figure 4.6.1: Admin – Add Property

Figure 4.6.1 depicts the admin add property page of the application, where you will add a property specified in the previous figure, which contains the information and specifies what sort of property or rental property this instance will be.

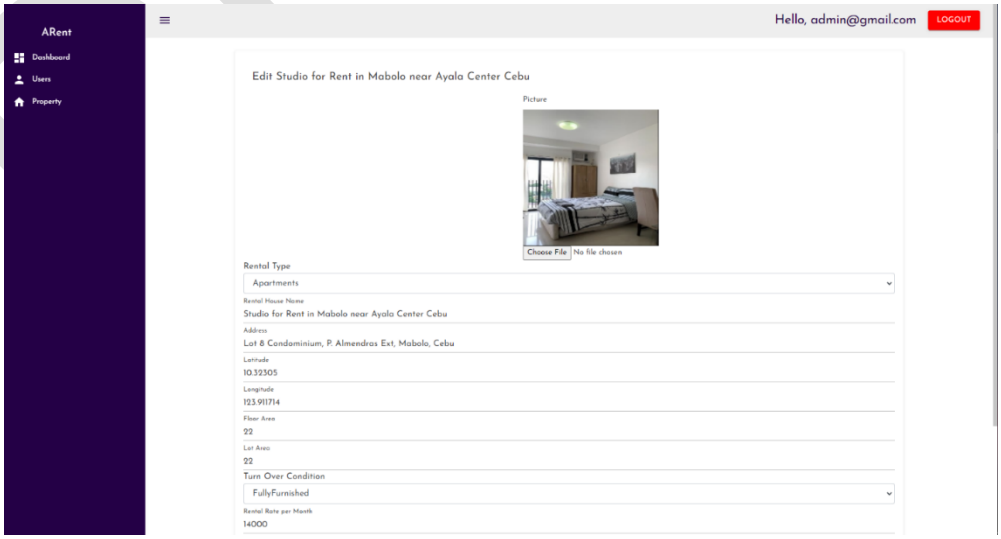


Figure 4.6.2: Admin – Edit Property Page

Figure 4.6.2 depicts the admin edit property page of the application, where the administrator may modify property information..

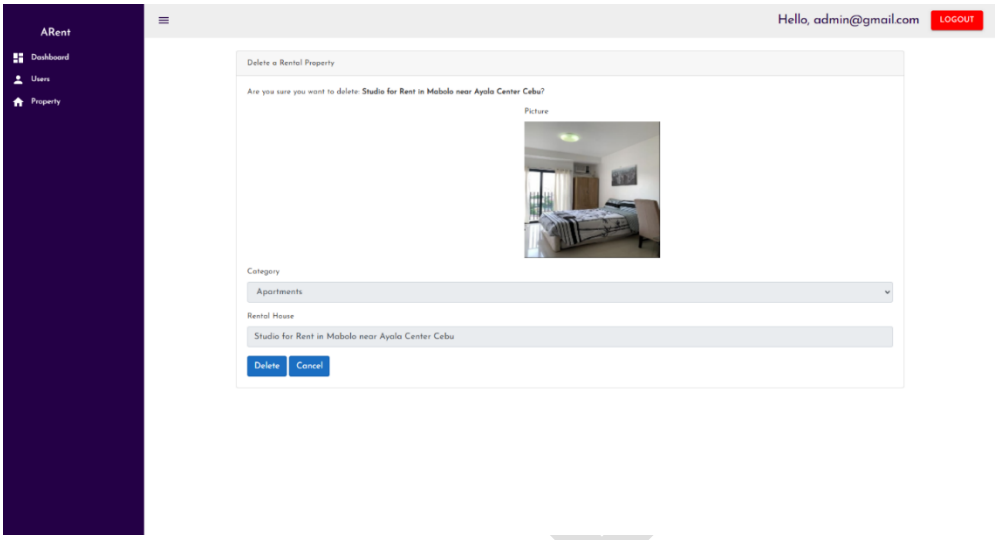


Figure 4.6.3: Admin – Delete Property Page

Figure 4.6.3 depicts the page in the application where an administrator may remove a property.

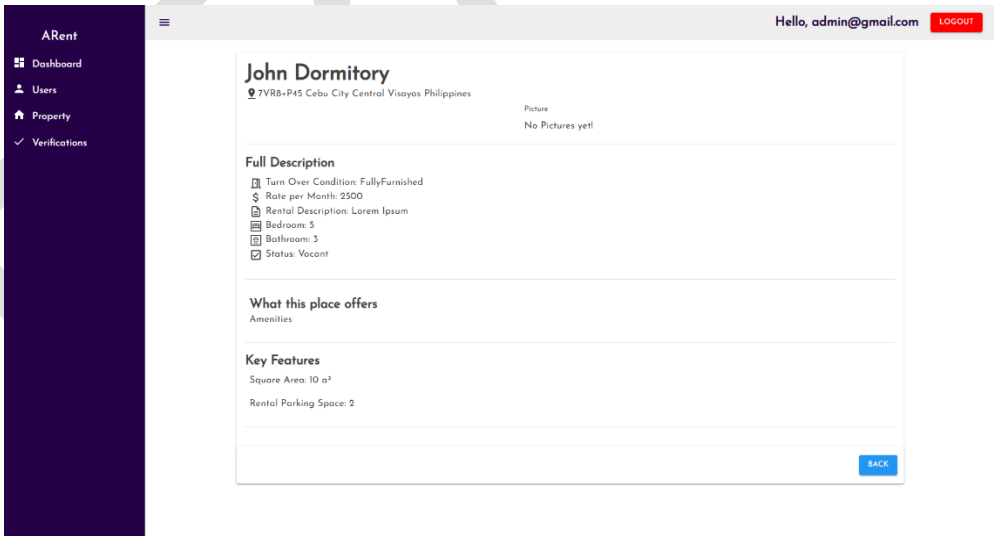


Figure 4.6.4: Admin – Property Details Page

Figure 4.6.4 depicts the admin property details screen of the program, where an administrator may access specific property information.

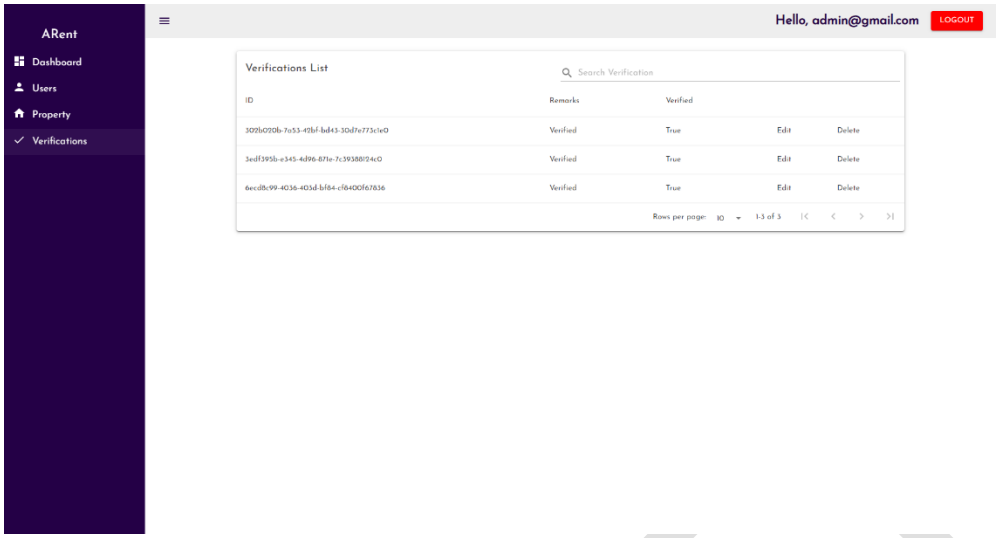


Figure 4.7: Admin – Verifications Page

Figure 4.6 depicts the admin verifications page of the program; each verification's information is shown in the table and may be changed or removed by an administrator.

b) Owner User Interface

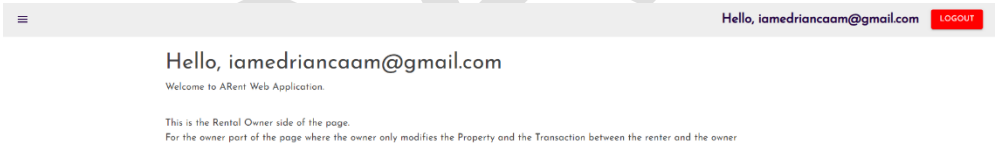


Figure 4.7: Owner – Dashboard

The Owner Dashboard page of the program is seen in Figure 4.7. It is the default page when an owner logs in with their credentials.

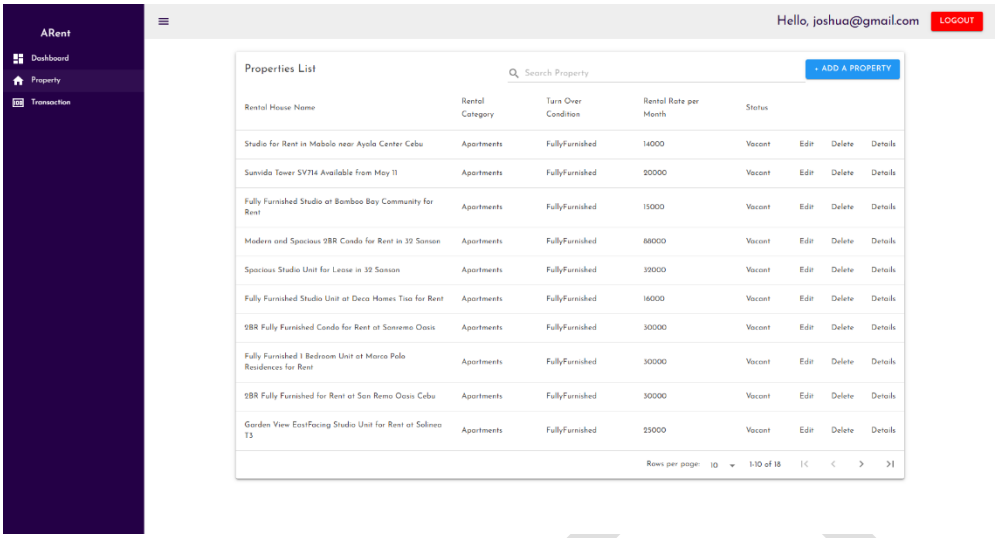


Figure 4.8: Owner – Property

Figure 4.8 depicts the owner property page of the application; the information on a specific property that the current owner owns will be shown in the table and may be modified, detailed, or removed by the owner. There will be a button that allows you to add a certain property.

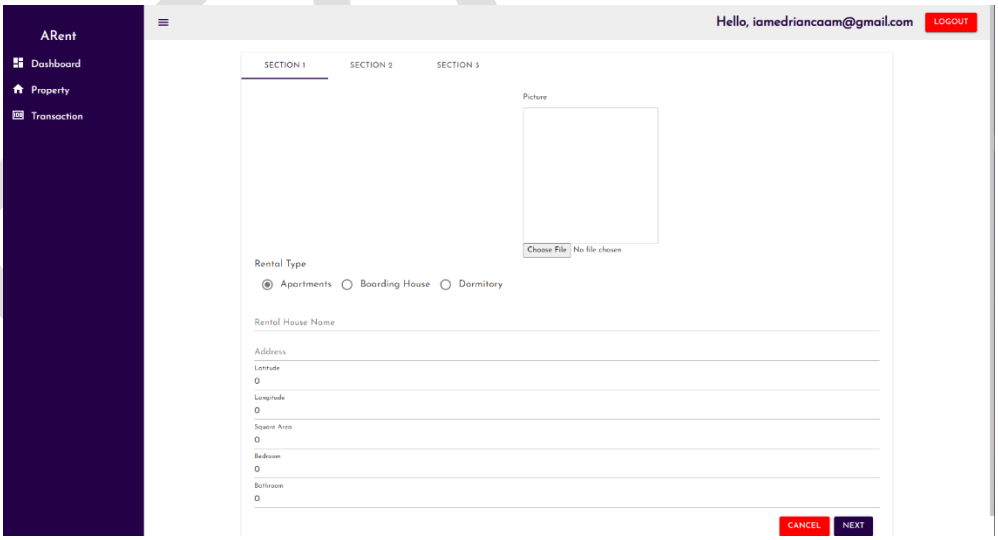


Figure 4.8.1: Owner – Add Property Page

Figure 4.8.1 depicts the owner add property page of the program, where you will add the property specified in the previous figure, which contains the information and specifies what sort of property or rental property it will be.

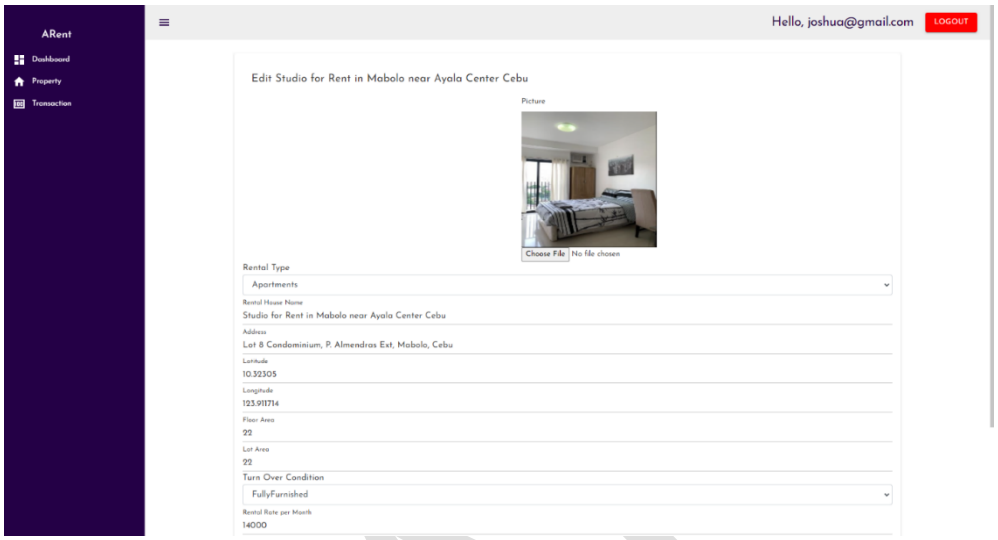


Figure 4.8.2: Owner – Edit Property Page

Figure 4.8.2 depicts the application's owner edit property page, where the owner will modify the property's details.

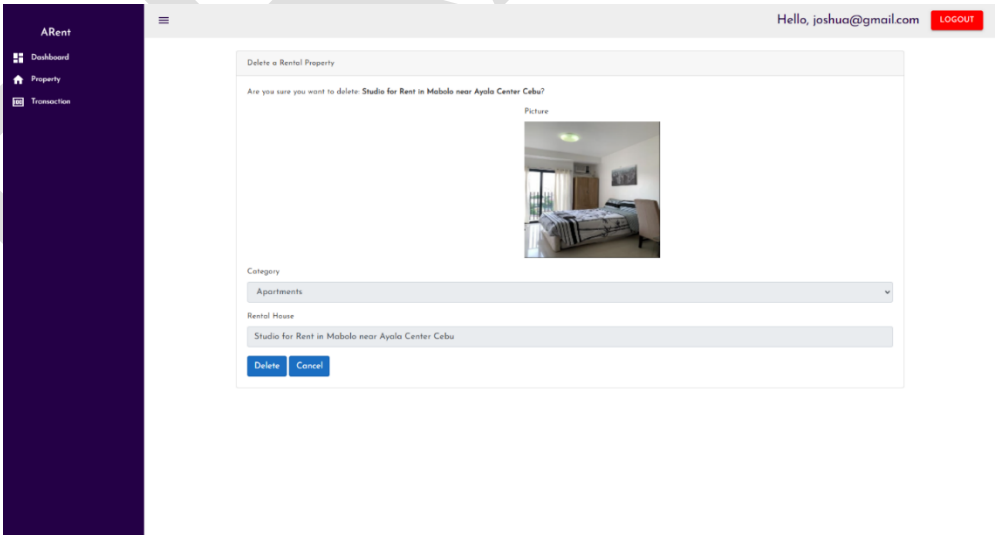


Figure 4.8.3: Owner – Delete Property Page

Figure 4.8.3 depicts the owner delete page of the program, where the owner may remove a property that is held by another individual.

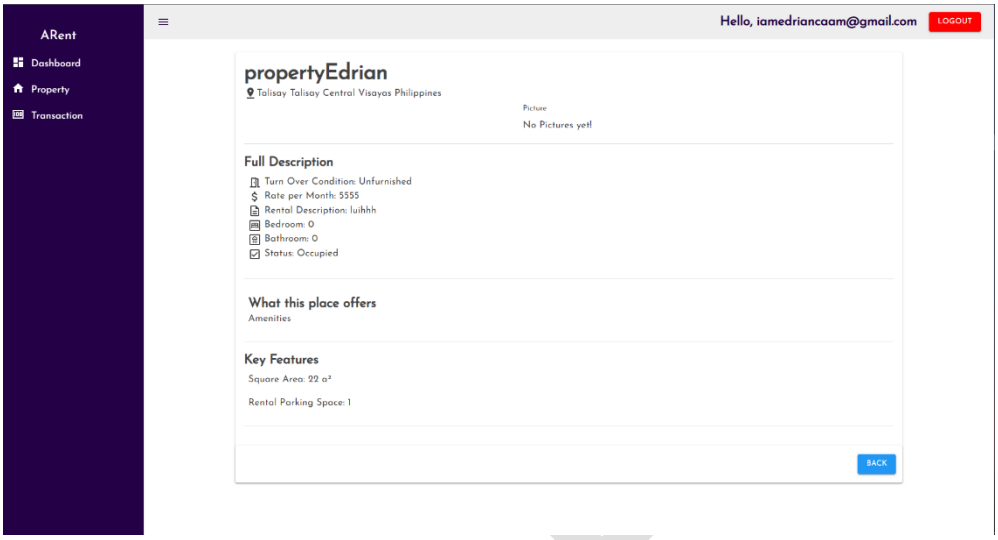


Figure 4.8.4: Owner – Property Details Page

Figure 4.8.4 depicts the owner's property details page of the application, where the owner may examine information on their owned property.

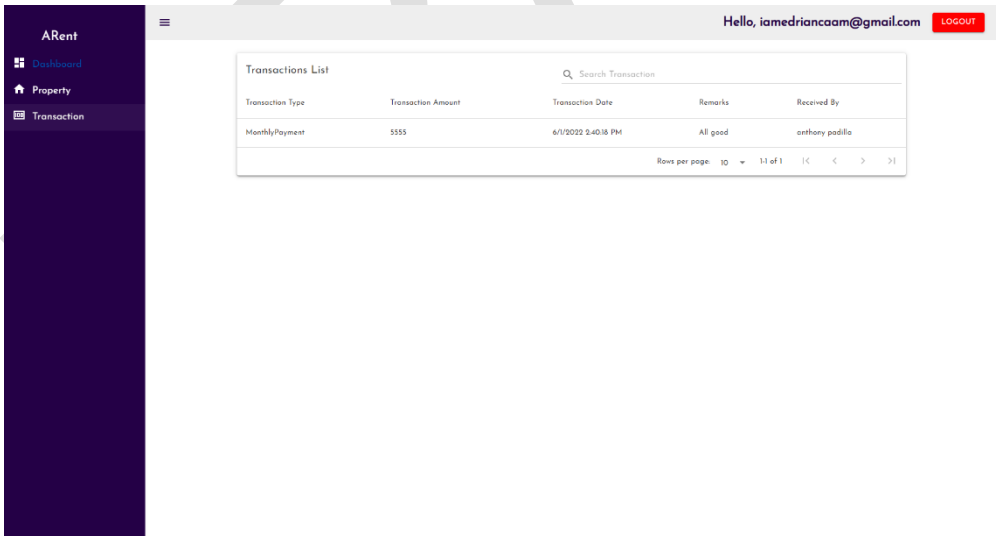


Figure 4.8.5: Owner – Transaction

Figure 4.8.5 depicts the owner transaction screen of the program, where the owner may examine a renter's transactions.

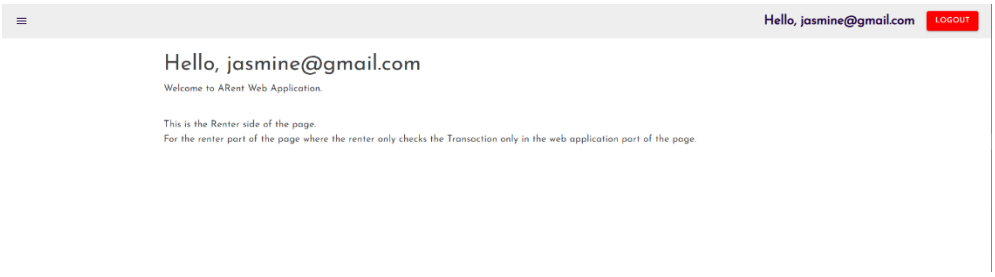


Figure 4.9: Renter – Dashboard

Figure 4.9 illustrates the web application's landing page. It holds the user's email address once the login is successful.

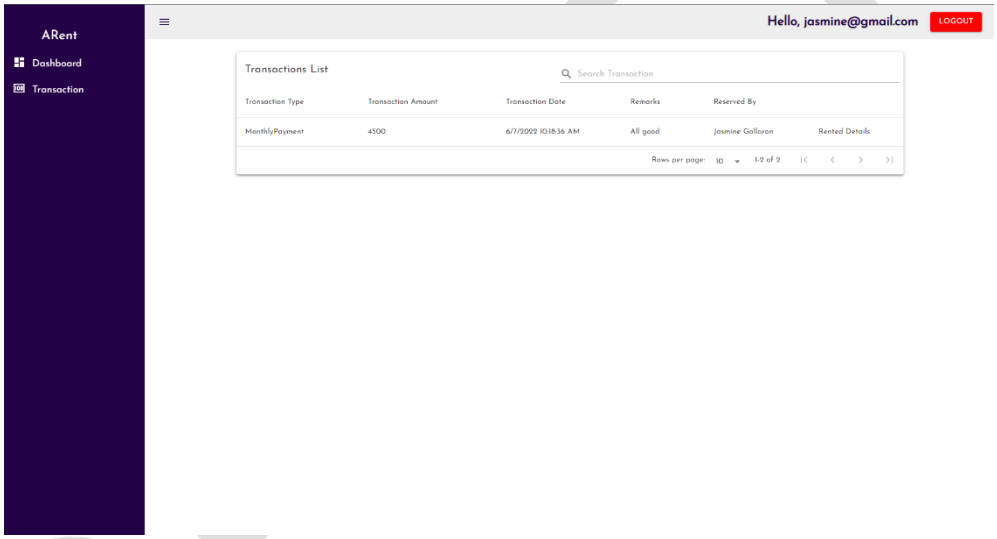


Figure 4.10: Renter – Transaction

Figure 4.10 depicts the renter transaction screen of the application, in which the renter may observe the transactions of a particular property for which the present renter will conduct a transaction.

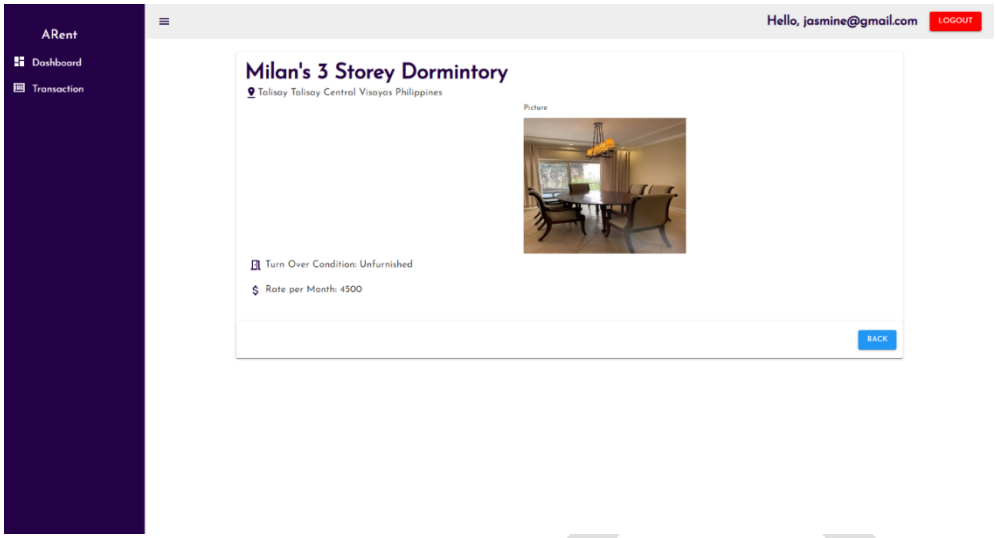


Figure 4.10.1: Renter – Transaction Details

Figure 4.10.1 depicts the renter's transacted property information page of the application, where the renter may examine the specifics of a property with which he or she has transacted.

Mobile

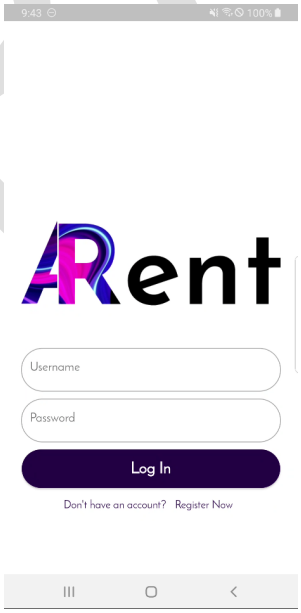


Figure 5.1: Login Screen

Figure 5.1 enables Rental Searchers and Rental Owners to Login to the mobile platform. This necessitates the submission of the user's login and password. The user establishes an account by clicking the "Register Now" button if they do not already have one.

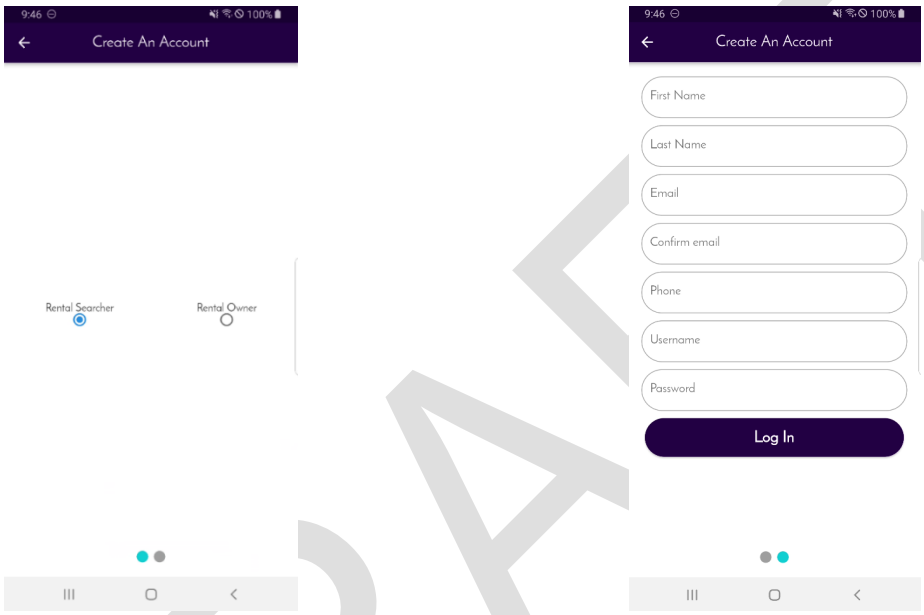


Figure 5.2: Registration Page

Figure 5.2 depicts the account creation process for our system. On the initial screen, users are prompted to choose the kind of account they want to establish. On the second screen, individuals must provide credentials such as login and password. Other information includes Name, Email, and Contacts.

A. Rental Owners User Interface

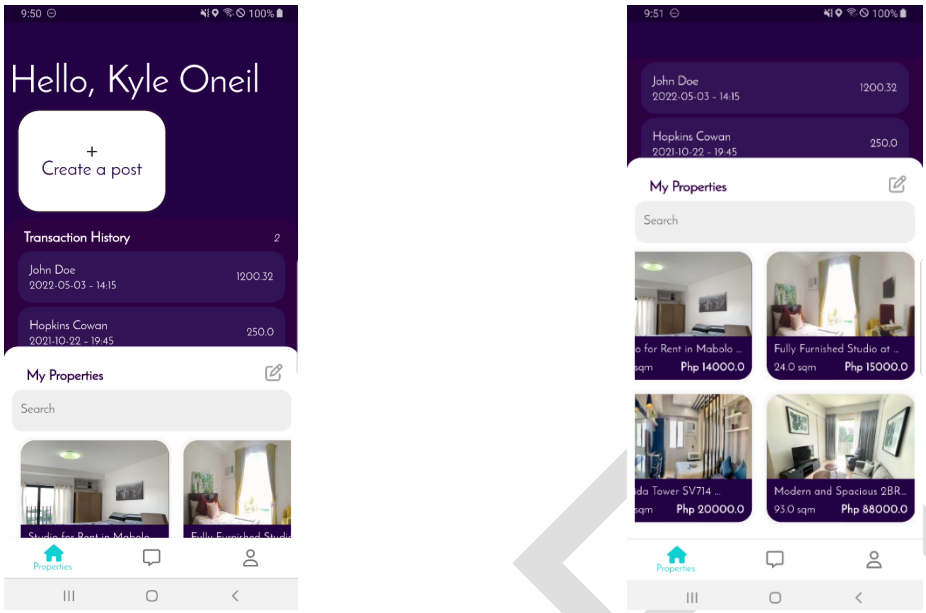


Figure 5.3: Home Page (Owners)

Figure 5.3 depicts the Rental Owners homepage. This is to handle all property data and the most recent transactions. Rental property owners may upgrade their particular properties' management features.

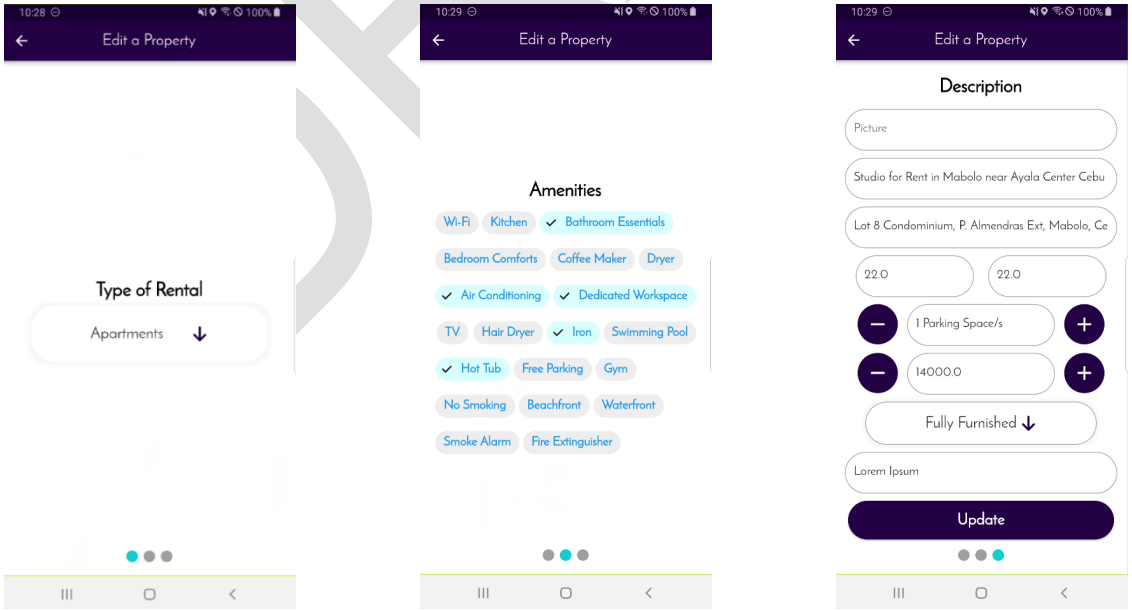


Figure 5.4: Manage Property

Figure 5.4 depicts a property management update page. On the first page, owners may choose among flats, boarding houses, and dorms. On the second page, owners must choose the desired amenities for their homes. On the third page, you will see the property's information. These comprise the property's location, name, lot size, square footage, parking space, turnover condition, and description.

B. Rental Searcher User Interface

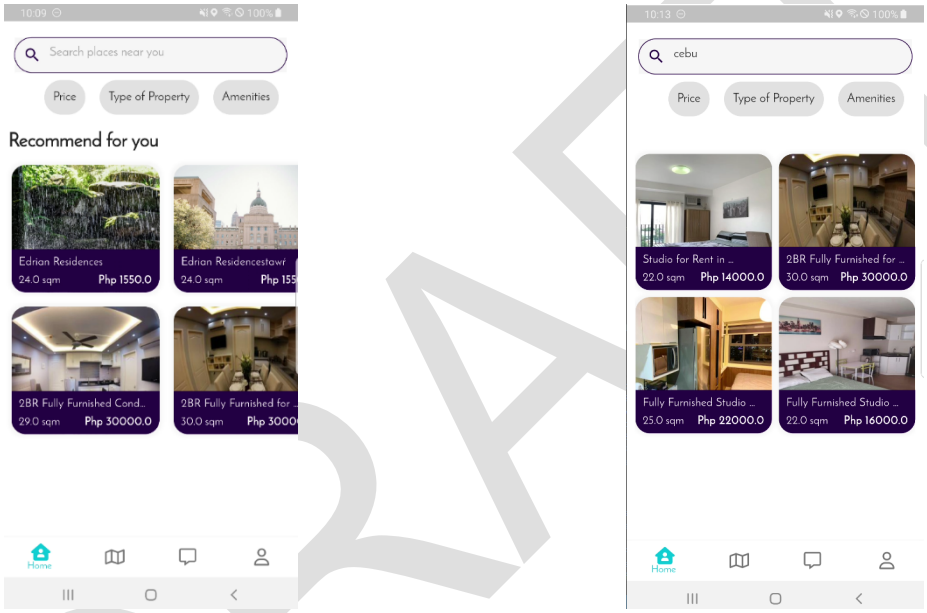


Figure 5.5: Home Page (Searcher)

Figure 5.5 depicts the Rental Searchers homepage. Initially, clients are provided with suggestions for the most suitable property. There is also a search filter for enhanced preference searching, which covers price, home type, and amenities. When searching, they are again presented with suggestions prior to the results of the search.

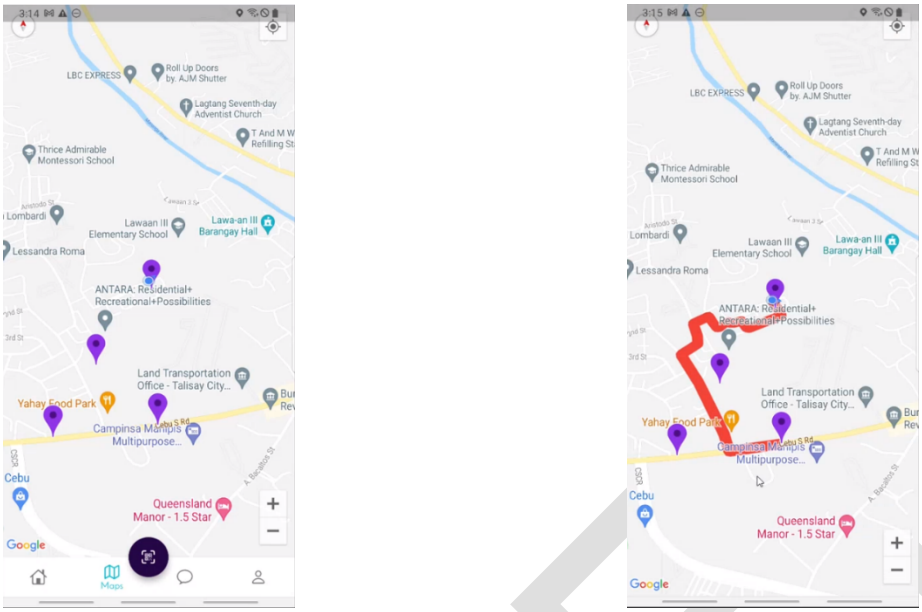


Figure 5.6: Navigation Page

Figure 5.6 depicts our system's Navigation Page. In addition to Augmented Reality, customers have the option to choose any property they like. The algorithm then provides the fastest route to the allocated pin and takes you to the chosen pinned property in real time.

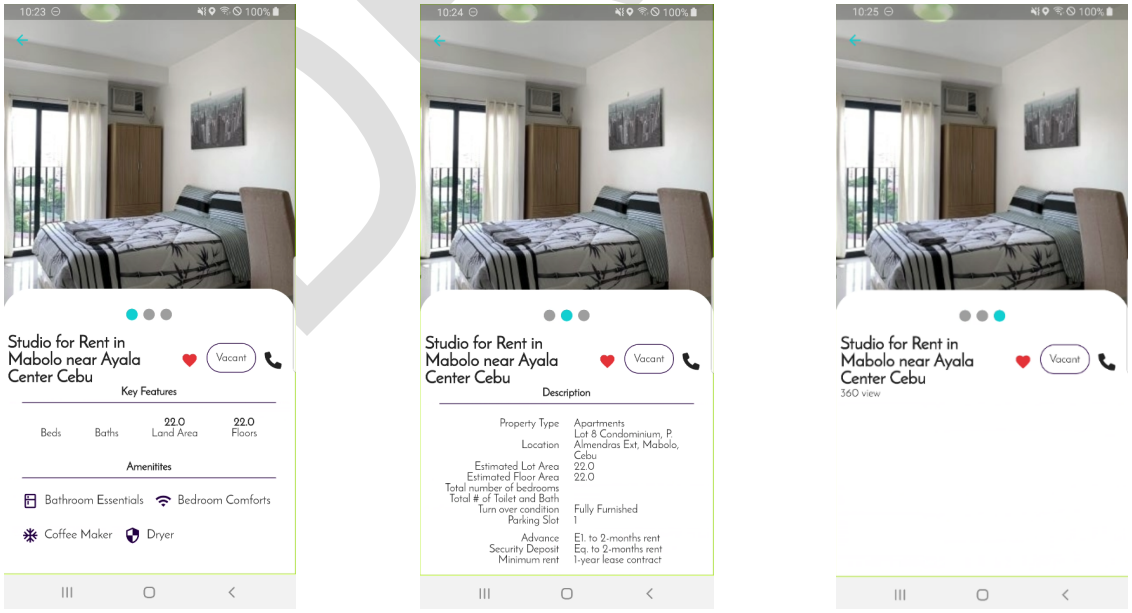


Figure 5.7: Property Detail Page

Figure 5.7 offers further details on the characteristic. This contains the name, availability, contact information, amenities, bed and bath, square footage, and much more. Additionally, this affords us a panoramic perspective of the land.

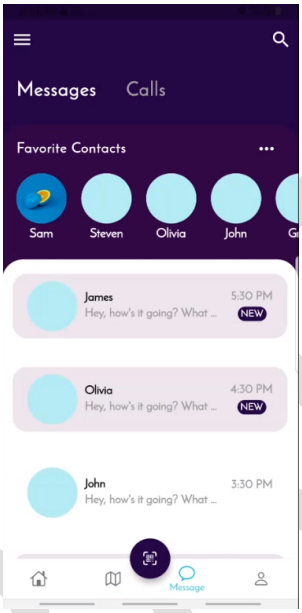


Figure 5.8: Chat Page

Figure 5.8 illustrates our application's chat mechanism. This is where people speak with one another on property-related questions.

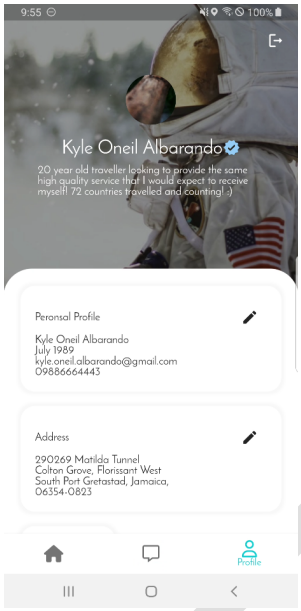


Figure 5.9: Profile Page

Figure 5.9 displays the user information for rental owners and renters. They were able to modify their information. If they are not already confirmed, they may request verification by selecting "Verify Now.".

Augmented Reality

Figure 5.10: User's Current Geospatial Data

Figure 5.10 shows a purple marker that indicates the user's current location and points to where it is heading. The upper portion of the screen displays the camera view of the user. The user's current latitude, longitude, altitude, and heading are shown in the lower part of the screen.

Figure 5.11: The property's location pin superimposed on the scene

Figure 5.11 shows the property that the user wants to check overlayed with the AR location pin. The 3D asset is positioned according to the rental owner's specified latitude and longitude.

CHAPTER III

SOFTWARE DEVELOPMENT AND TESTING

This chapter summarizes how the researchers develop the platform, integrate different environments, use management tools for development, and test the process. It breaks down the research's importance by identifying the application's root. It also

explains the functionalities and process of development of how the researchers implemented such a development process.

DEVELOPMENT SOFTWARE PLATFORMS, DEVELOPMENT ENVIRONMENTS, AND PROJECT MANAGEMENT TOOLS

The researchers utilized Visual Studio Code and Android Studio for mobile development and Visual Studio Enterprise 2022 for web development. In mobile, the researchers are using Android Studio as their primary programming language for mobile. Moreover, Android Studio has a Gradle-based build system, emulator, code templates, and GitHub integration. Every Android Studio project has at least one mode containing source code and resource files [10]. For web development, ARent is using Visual Studio. Microsoft's Visual Studio is an IDE. Visual Studio's code editor supports refactoring and IntelliSense (a code completion feature). With this, source and machine code may be in debugging. This package also offers code profiling and design tools for applications with a graphical user interface. Subversion and Git are supported by plug-ins, as are various programming language-specific toolkits [11]. Microsoft Visual Studio enables web development in addition to several other platforms. We used the IDE's web development and API services features.

The researchers were using Flutter as their primary programming language for mobile phones. Google developed Flutter in May 2017 and released it as a free, open-source software development kit [12]. It is also known as a user interface framework, a collection of changeable user interface components that comprise an application modifiable during development. The programming language Dart is the foundation for Flutter [13]. We implemented the GetX microframework to improve state administration. GetX is a microframework solution for Flutter that is both lightweight and effective. Moreover, The Dart Package community created it with the promise of

superior state management, dependency injection, and route management performance. These factors were the most important in deciding to install it. [14].

In developing the web framework, ARent is using .Net. Microsoft.NET is a proprietary application designed for use with the programs it supports. It is well-known for its reliability and quickness. It utilizes just-in-time compilation, which enhances speed and performance [15]. *Web services* are servers explicitly designed to satisfy a website's or other application's demands. Client applications connect with web services using application programming interfaces (APIs). A Web API is a web service interface directly receiving client queries and responding to them. Moreover, It is a Web API conforms to the REST architectural style and creates APIs for current online services. [21].

Blender is a free, open-source 3D modeling program [1]. Also supported are video editing and game development, modeling, rigging, animation, simulation, rendering, compositing, and motion tracking. *Blender* is a program that works on Linux, Windows, and Mac computers. It uses OpenGL to create its user interface to offer a consistent user experience. Moreover, it was using it to construct the 3D pieces that would be overlaid on the app when pressing the camera button. The 3D elements are rendered and exported as an OBJ file, which is mobile device friendly. These assets are location pins and directional signs in rental homes to lead users to the desired rental unit.

ARCore is Google's foundation for producing augmented reality experiences. ARCore enables ARent to observe its surroundings, understand the world, and interact with data through various APIs [16]. To combine virtual material with the actual world as

viewed on mobile, ARCore allows our application to leverage Augmented Reality since it utilizes three crucial capabilities: motion tracking, environmental awareness, and light estimation.

In May 2022, Google released the ARCore Geospatial API for Android and iOS. The new API will allow developers to create immersive experiences in real-world locations across 87 countries without physically visiting them [24]. The ARCore Geospatial API uses data from Google Earth 3D models and Google Maps Street View image data to allow the app to provide seamless, global-scale, location-based augmented reality experiences [25]. This API makes it possible to create immersive, location-based AR experiences as developers may now attach anchors by latitude, longitude, and altitude without being there or scanning the physical space, saving significant time and resources.

Kotlin is a statically typed open-source programming language for the JVM, Android, JavaScript, and Native platforms [26]. Kotlin was used to build the Augmented Reality component of the project. It is easy, secure, and compatible with Java and other languages, with several choices for productive code reuse between platforms.

When it comes to ARent, we need a number that is both trustworthy and resource-efficient in order to administer our database servers. *Docker Desktop* is an open-source program that enables users to create and distribute grouped-up applications and microservices [6]. Containers, unlike virtual machines, do not transport the payload of a whole instance of a software system or a hypervisor.; instead, they carry just the OS processes and dependencies required to execute the code. With that said, Operating systems are there to operate on virtual computers. Moreover,

Containers are similar to virtual machines in that they share portability, minimal resource use, and dependency on the host operating system.

Ubuntu is a Linux distribution based on the Debian architecture, one of the oldest operating systems based on the Linux kernel [7] and consisting primarily of free and open-source software [8]. Cloud servers and developers construct and manage online services (WSL) using the Windows Subsystem for Linux, which is programmed to run our Docker Engine container.

Researchers thought the Android Debug Bridge might address debugging and usability difficulties with port forwarding for our cloud database. With that said, ADB is a command-line program for connecting with Android devices. It can execute various instructions, including port forwarding and device installation. Using a reverse proxy in port forwarding, an HTTP server operating on the mobile device receives connections through USB Debugging.

DEVELOPMENT PROCESS AND TESTING

Development Process

In this section, the development process of making the core features is presented and detailed through different implementations for the application. It explains the main functionality for the correct execution of every function.

System Architecture

For our safety, it was vital for the researchers to operate remotely due to the pandemic. The researchers compromised on the optimal cooperation tools to make ARent a reality. As the researchers constructed the application, these were the prerequisites.:

- All REST API requests must coexist on one platform – web
- The cloud server must always be accessible for the researchers to be involved.
- Use a physical mobile device to run the application
    - Require USB debugging in developer mode

REST API Service

The researchers developed their REST API with their request and response for HTTP requests to each controller, which are structured using the standard GET, POST, PUT, and DELETE methods. Each HTTP request has a unique implementation and data retrieval strategy. Since it allows mobile and augmented Reality to communicate and produce their data requests, the REST API is one of the project's most essential tools. It contains *providers* that will coordinate via the SQL services, facilitating communication between the controller and a provider's service.

- **Search Rental Property**

  Provider:

```
/// <summary>
/// Search for rental properties with the specified <paramref name="name"/>.
/// </summary>

public IEnumerable<RentalProperty> FindRentalPropertiesByName(string name) =>
    m_dbContext.RentalProperties.Where(p => p.RentalHouseName.ToLower().Contains(name.ToLower()));
```

Figure 6.1: Search Rental Property Provider

  Controller:

```
/// <summary>
/// Get for Rental Property
/// </summary>
[HttpGet]
0 references | - changes | -authors, -changes
public IEnumerable<RentalProperty> Get([FromQuery] string searchName) =>
    searchName == null
        ? m_dataAccessProvider.GetRentalPropertyRecords()
        : m_dataAccessProvider.FindRentalPropertiesByName(searchName);
```

Figure 6.2: Search Rental Property Controller

The usage of Figure 6.1 and Figure 6.2 snippets for retrieving the rental property is where the provider and controller are used as the foundation for searching for the rental property using the HTTP GET technique.

- **Update Users Profile**

  Provider:

```
/// <summary>
/// Updating a User from Database
/// </summary>
2 references | joshuadotillos, 262 days ago | 1 author, 2 changes
public void UpdateUsers(Users users)
{
    m_dbContext.Users.Update(users);
    m_dbContext.SaveChanges();
}
```

Figure 6.3: Update Users Profile

  Controller:

```
/// <summary>
/// Edit Users
/// </summary>
[HttpPut]

public void Edit([FromBody] Users users)
{
    if (ModelState.IsValid)
    {
        m_dataAccessProvider.UpdateUsers(users);
    }
}
```

Figure 6.4: Update Users Profile Controller

The HTTP method PUT is used by Figures 6.3 and 6.4 to update the profile of a user who requires the provider and the controller with the usage of its API to update its profile.

- **Add Rental Property**

  Provider:

```
/// <summary>
/// Adding a Rental Property to Database
/// </summary>

public void AddRentalProperty(RentalProperty renters)
{
    m_dbContext.RentalProperties.Add(renters);
    m_dbContext.SaveChanges();
}
```

Figure 6.5: Add Rental Property Provider

Controller:

```
/// <summary>
/// Add a Rental Property
/// </summary>
[HttpPost]

public void Create([FromBody] RentalProperty rentalProperty)
{
    if (!ModelState.IsValid)
        return;

    var obj = Guid.NewGuid();
    rentalProperty.Property_Id = obj.ToString();
    m_dataAccessProvider.AddRentalProperty(rentalProperty);
}
```

Figure 6.6: Add Rental Property Controller

The usage of Figures 6.5 and 6.6 when adding a rental property that requires a provider and the controller when adding a property that corresponds with the HTTP POST method are shown in Figures 6.5 and 6.6.

- **Display Property Information**

Provider:

```
/// <summary>
/// Getting a specified Rental Property from the Database
/// </summary>
2 references | joshuadotillos, 265 days ago | 2 authors, 2 changes
public RentalProperty GetRentalPropertySingleRecord(string id) =>
    m_dbContext.RentalProperties.First(rp => rp.Property_Id == id);
```

Figure 6.7: Display Property Information Provider

Controller:

```
/// <summary>
/// Get ID for a specified Rental Property
/// </summary>
[HttpGet("{id}")]

public RentalProperty Details(string id) => m_dataAccessProvider.GetRentalPropertySingleRecord(id);
```

Figure 6.8: Display Property Information Controller

Figures 6.7 and 6.8 are used to show property information using the provider and controller corresponding to the GET HTTP method.

- **Verify Rental Property**

  Provider:

```
/// <summary>
/// Search for Verification with the specified <paramref name="name"/>.
/// </summary>

public IEnumerable<Verifications> FindVerificationsByRandId(string name) =>
    m_dbContext.Verifications.Where(o => o.rand_Id.Contains(name.ToLower()));
```

Figure 6.9: Verify Rental Property Provider

  Controller:

```
/// <summary>
/// Get ID for verifications
/// </summary>
[HttpGet("{id}")]

public Verifications Details(string id) => m_dataAccessProvider.GetVerificationSingleRecord(id);
```

Figure 6.10: Verify Rental Property Controller

When utilizing Figures 6.9 and 6.10 to verify the rental property, which employs the provider and the controller, the GET HTTP method will be utilized where it must be stated to determine whether the verification occurs.

- **Manage Rental Owner**

  Provider:

```
2 references | joshuadotillos, 263 days ago | 1 author, 2 changes
void AddUsers(Users users);
2 references | joshuadotillos, 263 days ago | 1 author, 2 changes
void UpdateUsers(Users users);
2 references | joshuadotillos, 265 days ago | 1 author, 1 change
void DeleteUsers(string id);
2 references | joshuadotillos, 265 days ago | 1 author, 1 change
Users GetUserSingleRecord(string id);
2 references | joshuadotillos, 265 days ago | 1 author, 1 change
IEnumerable<Users> FindUsersByName(string name);
2 references | joshuadotillos, 265 days ago | 1 author, 1 change
IEnumerable<Users> GetUsersRecords();
```

Figure 6.11: Manage Rental Owner Provider

Controller:

```
/// <summary>
/// Get for Users
/// </summary>
[HttpGet]

public IEnumerable<Users> Get([FromQuery] string searchName) =>
    searchName == null
        ? m_dataAccessProvider.GetUsersRecords()
        : m_dataAccessProvider.FindUsersByName(searchName);
```

Figure 6.12: Manage Rental Owner GET Controller

```
/// <summary>
/// Add a User
/// </summary>
[HttpPost]
0 references | joshuadotillos, 262 days ago | 1 author, 2 changes
public void Create([FromBody] Users users)
{
    if (!ModelState.IsValid)
        return;

    var obj = Guid.NewGuid();
    users.User_Id = obj.ToString();
    m_dataAccessProvider.AddUsers(users);
}
```

Figure 6.13: Manage Rental Owner POST Controller

```
/// <summary>
/// Edit Users
/// </summary>
[HttpPut]
0 references | joshuadotillos, 262 days ago | 1 author, 2 changes
public void Edit([FromBody] Users users)
{
    if (ModelState.IsValid)
    {
        m_dataAccessProvider.UpdateUsers(users);
    }
}
```

Figure 6.14: Manage Rental Owner PUT Controller

```
/// <summary>
/// Delete a Users
/// </summary>
[HttpDelete("{id}")]
0 references | joshuadotillos, 262 days ago | 1 author, 2 changes
public void DeleteConfirmed(string id) => m_dataAccessProvider.DeleteUsers(id);
```

Figure 6.15: Manage Rental Owner DELETE Controller

It requires the providers and controllers that will use the HTTP methods POST, PUT, GET, and DELETE for managing the owners. The location where the API will organize and manage its owners' data.

- **Manage Rental Property**

Provider:

```
void AddRentalProperty(RentalProperty rentalProperty);

void UpdateRentalProperty(RentalProperty rentalProperty);

void DeleteRentalProperty(string id);

RentalProperty GetRentalPropertySingleRecord(string id);

IEnumerable<RentalProperty> FindRentalPropertiesByName(string name);

IEnumerable<RentalProperty> GetRentalPropertyRecords();
```

Figure 6.16: Manage Rental Property Provider

Controller:

```
/// <summary>
/// Get for Rental Property
/// </summary>
[HttpGet]

public IEnumerable<RentalProperty> Get([FromQuery] string searchName) =>
    searchName == null
        ? m_dataAccessProvider.GetRentalPropertyRecords()
        : m_dataAccessProvider.FindRentalPropertiesByName(searchName);
```

Figure 6.17: Manage Rental Property GET Controller

```
/// <summary>
/// Add a Rental Property
/// </summary>
[HttpPost]
0 references | joshuadotillos, 265 days ago | 2 authors, 3 changes
public void Create([FromBody] RentalProperty rentalProperty)
{
    if (!ModelState.IsValid)
        return;

    var obj = Guid.NewGuid();
    rentalProperty.Property_Id = obj.ToString();
    m_dataAccessProvider.AddRentalProperty(rentalProperty);
}
```

Figure 6.18: Manage Rental Property POST Controller

```
/// <summary>
/// Edit Rental Property
/// </summary>
[HttpPut]
0 references | Ahdzlee Formentera, 279 days ago | 1 author, 2 changes
public void Edit([FromBody] RentalProperty rentalProperty)
{
    if (ModelState.IsValid)
    {
        m_dataAccessProvider.UpdateRentalProperty(rentalProperty);
    }
}
```

Figure 6.19: Manage Rental Property PUT Controller

```
/// <summary>
/// Delete a Rental Property
/// </summary>
[HttpDelete("{id}")]

public void DeleteConfirmed(string id) => m_dataAccessProvider.DeleteRentalProperty(id);
```

Figure 6.20: Manage Rental Property DELETE Controller

It requires the providers and controllers that will use the HTTP methods POST, PUT, GET, and DELETE for managing properties. Where the API will be used to organize and manage its data's attributes.

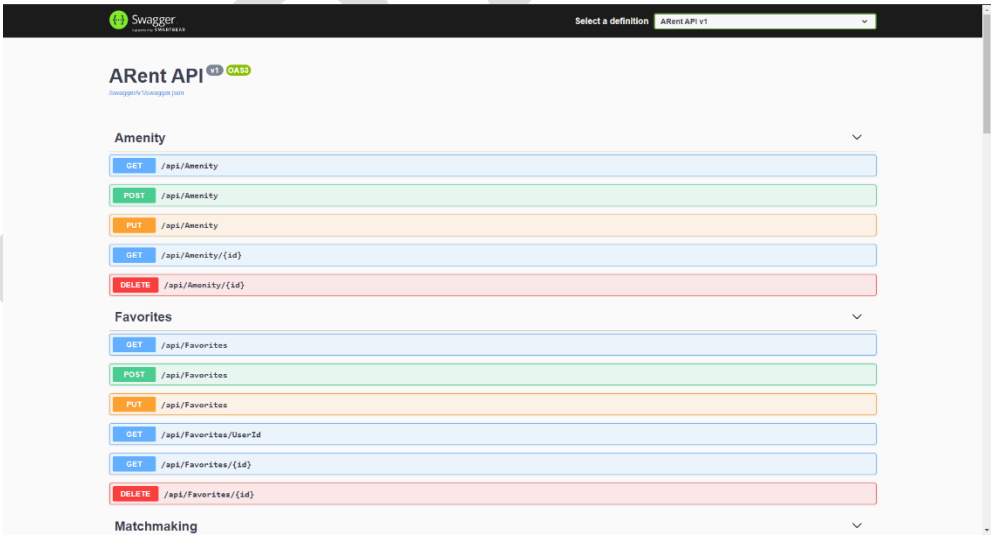


Figure 6.21 Swagger - API Documentation

Figure 6.21 depicts the API documentation that has been merged into Swagger. Swagger is solely used to determine which API has been implemented and as a source

that knows how to handle all JSON-formatted data. These are the resources for using HTTP GET, POST, PUT, or DELETE with any API operation.

The Cloud Service

The researchers used Microsoft Azure for the cloud-based architecture. It alleviates researchers' concerns about performance difficulties and security breaches since the program gives safe, dependable access to cloud-hosted data. Specifically, the researchers used relational and non-relational database services and storage. Researchers also used Docker Containers to provide cloud-based client-side database access. The researchers subscribed to one of Microsoft Azure's database services,' Azure Database for PostgreSQL flexible server,' as it caters multiple assurances to consumers:

- **Flexibility**: Allows ARent to cater to multiple users without latency issues
- **Cost**: Pay-As-You-Go model for cost-efficiency
- **Security**: Uses SSL Encrypted controls for data privacy
- **Disaster Recovery**: Calamities, like the recent Typhoon Odette, would not affect the data.

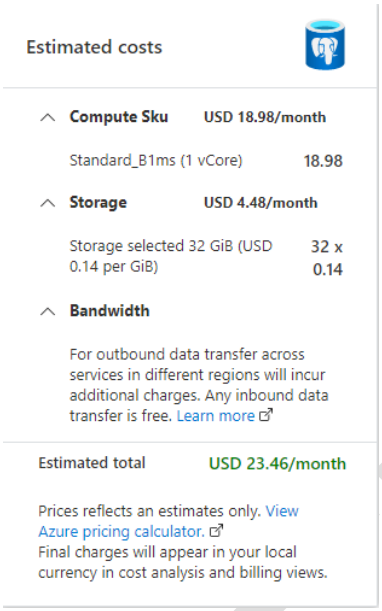


Figure 7.1 Azure Cloud – Estimated Costs

Figure 7.1 shows that this will be the payment for using the PostgreSQL flexible database powered by the Azure Cloud service, as it allows the project to access the data and retrieve it from the service.

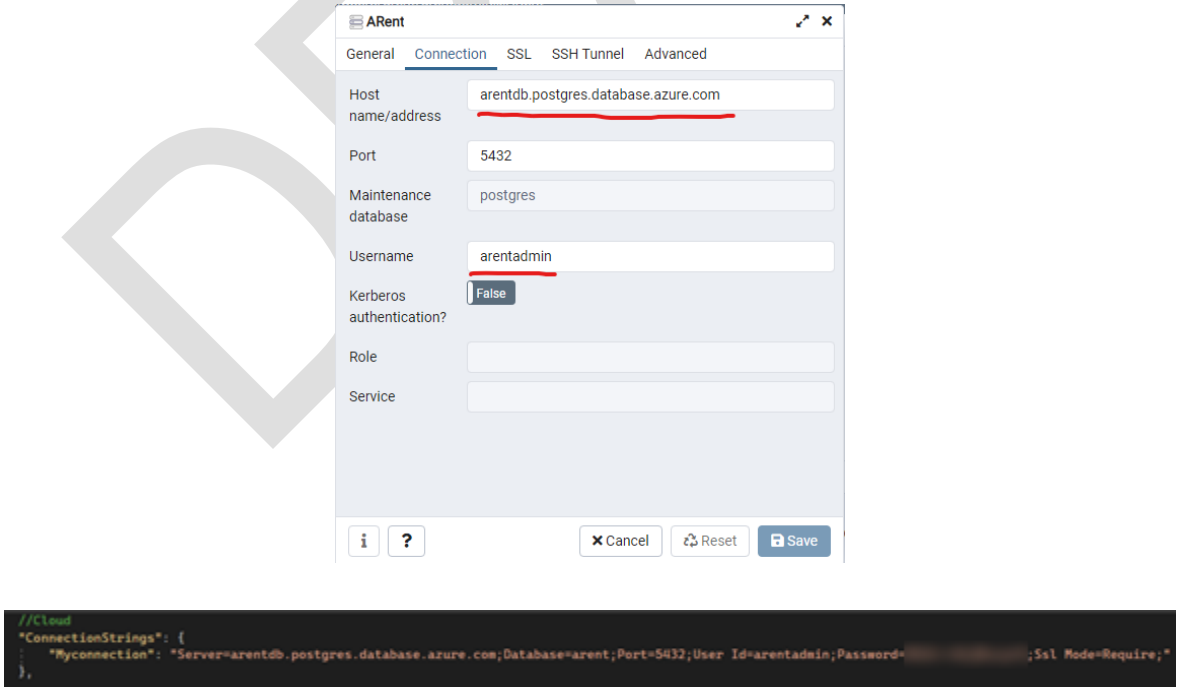


Figure 7.2 Docker Properties – Azure Cloud connection

The connection between Docker and Azure Cloud is shown in Figure 7.2, which shows the researchers highlighting the crucial fields to be filled up using the database held in the Azure Cloud service, which acts as the link between Docker and Azure Cloud. In the project, it must be specified as the connection between the cloud and the other connection string needs listed in the figure. This integration is used by accessing just the data stored in the Azure Cloud.

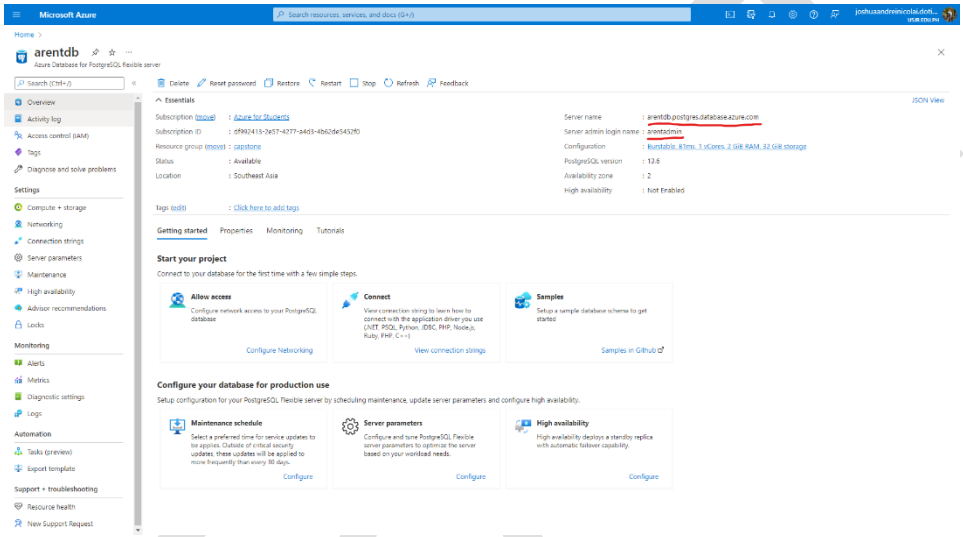


Figure 7.3 Azure Cloud PostgreSQL Flexible Database

Figure 7.3 depicts the cloud service that was created and stored in the cloud, with each credential's key components highlighted by the researchers. It is preferable to utilize the cloud service as one of the methods for storing data so that other mobile or AR integrations may use it to their applications.

The Database

The researchers used relational and non-relational databases for distinct storage purposes and use cases. The researchers first used free database providers to save money. Free database services present several concerns, including the fact that not only is it hazardous, but it cannot conduct mobile-web synchronizations since it was

restricted to free users. The primary objective for ARent was to develop a cloud-based database solution so that we could receive REST queries through the API. To consolidate our database across several platforms, the researchers subscribed to Azure Database for PostgreSQL Flexible Server for relational databases. It ensured that any differences across platforms would quickly address debugging issues. We used Azure Cosmos DB for the non-relational database. Moreover, the chat system uses this database.

```
public ARentDBContext()
{
}

public ARentDBContext(DbContextOptions<ARentDBContext> options)
    : base(options)
{
}

public virtual DbSet<Users> Users { get; set; }

public virtual DbSet<NearbyLocation> NearbyLocation { get; set; }

public virtual DbSet<RentalProperty> RentalProperties { get; set; }

public virtual DbSet<RentalTransaction> RentalTransactions { get; set; }

public virtual DbSet<Amenities> Amenities { get; set; }

public virtual DbSet<Verifications> Verifications { get; set; }
```

Figure 8.1: Setting up - Database Context

Figure 8.1 shows how to initialize the database context for the SQL integration of the service. Making the database ready to be set by migrating the context to the server.

Figure 8.2 Docker implementation

Figure 8.2 depicts the docker implementation for the project based on what docker can perform for the application. It is where the credentials are stored and the docker version is sent to the application's structure.

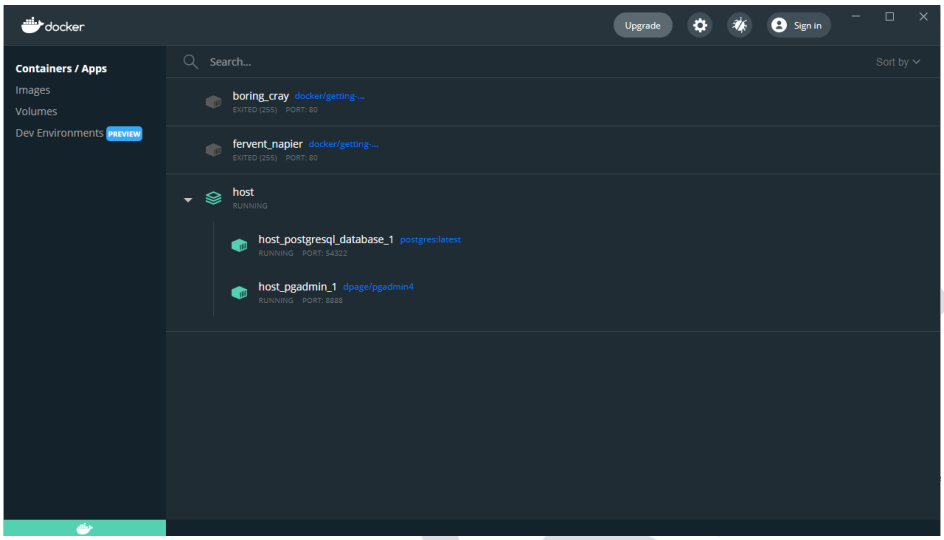


Figure 8.3 Docker application

Figure 8.3 depicts the connection to the server's localhost, where you may access the database using the application's specified address.

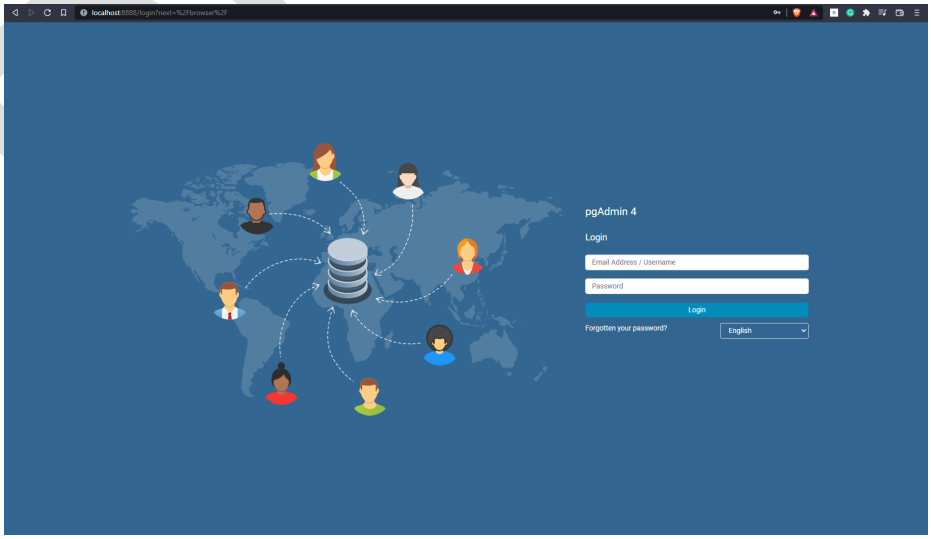


Figure 8.4 Docker landing page

Figure 8.4 is where the credentials to be input are given by the information that is given in Figure 8.2 that gives the credentials to be input for logging in to the database.

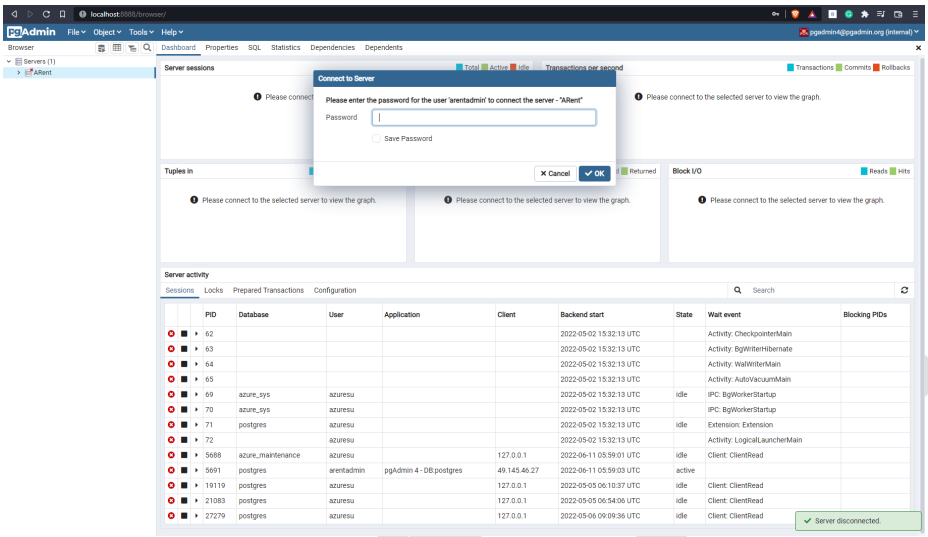


Figure 8.5 Docker connection – password input

Figure 8.5 shows the Docker connection as logging in. Then it needs to be logged in with another credential, which will be the password input, as it is found on the connection that is indicated in Figure 7.2.

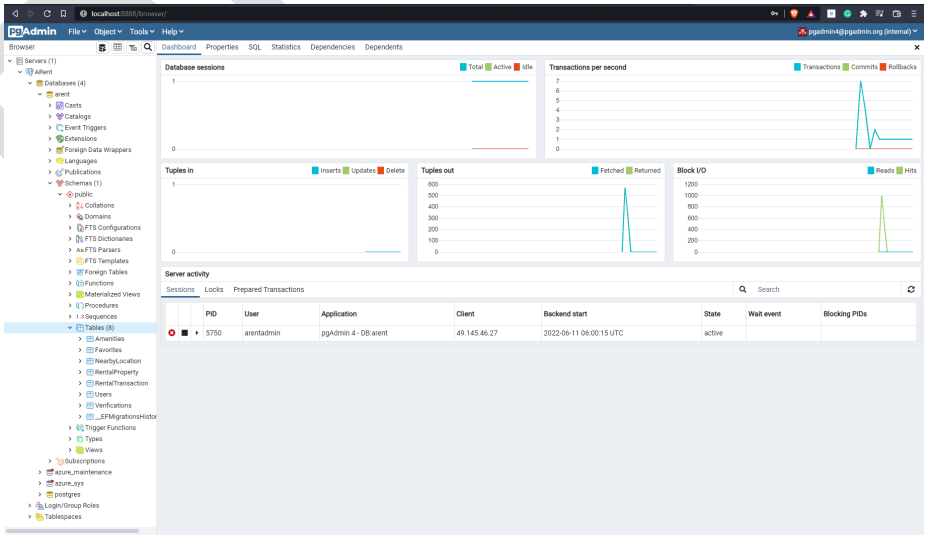


Figure 8.5.1 Docker connection – accessing the server

Figure 8.5.1 depicts the connection for contacting the server after the input for reaching the server has been completed. Then, you will have access to the server's cloud database.

Version Control

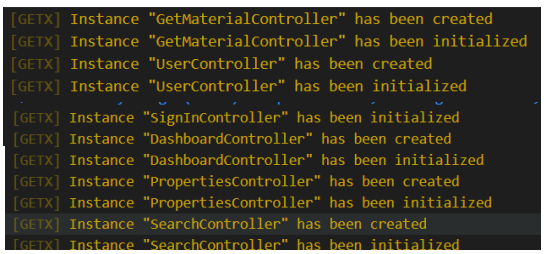
The researchers utilized Git to monitor and manage application modifications. Git is a version control system that is open source [17]. Git provides all contributors in every git command. For our hosting platform's graphical user interfaces, we used GitHub and GitKraken (GUI).

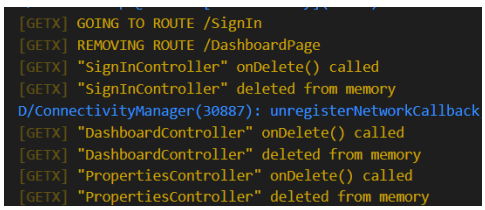
Mobile Microframework

Getx

Although the Flutter package executes well from state management (Stateful and Stateless), dependency injection, and route management, the mobile researcher still experiences inconsistencies regarding how variables execute each widget. Although it is possible to extend everyone as Stateful, the question relies on the app's efficiency, for there are instances wherein only a Text is regarded as stateful and not the whole Scaffold. Getx, regarded as the no.1 most liked flutter package, according to *pub.dev* [14], is a flutter dependency that boosts efficiency and lightweight that covers three principles: Productivity, Performance, and Organization.

- **Performance**





```
Widget build(BuildContext context) {
  var pc = navi.Get.put(PropertiesController());
  pc.getPropertyByOwner();

  return navi.GetBuilder<PropertiesController>(
    builder: (controller) {
      return controller.resultPropertybyOwner.isNotEmpty
          ? _showContents(controller)
          : centerLoadingCircle();
    },
  );
}
```

```
Future getPropertyByOwner() async {
  var uc = navi.Get.put(UserController());
  await propertiesService
      .getPropertyByOwner("", uc.userSingle.user_Id)
      .then((value) {
    resultPropertybyOwner.clear();
    resultPropertybyOwner = value.body!;
    if (kDebugMode) {
      print('getPropertyByOwner: ${resultPropertybyOwner.length}');
    }
  }).whenComplete(() => update());
}
```

Figure 9.1: Resource Control of GetX

An application needs to capture resources as they are required. Typically, similar functions exist in which a function listens to a widget; however, this consumes excessive background data. *GetX* initializes these controllers, so developers never have to worry about it. Moreover, it is similar to a "pay-as-you-go" model of Microsoft Azure, in which you only pay for what you consume. It is only necessary to call the controller by its name (i.e., *UserController*). One of which is the *.put()* function. Normally on Flutter, if we declare changes to our widget states, we use *SetState()* to update our variables. There are instances wherein the developer wants to display all properties in each widget tree. It is cumbersome and inefficient to declare the widget tree stateful because all of its children do not need to be changed. That is where *.put()* comes in. initialized telling the system that you want to use this controller followed by its method.

- **Productivity**

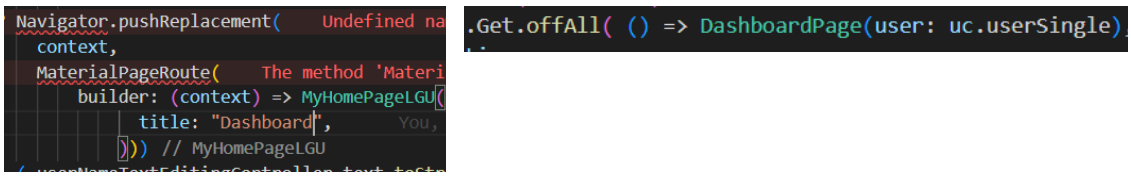


Figure 9.2: Comparison on Navigation

The Navigator function has two parameters: *BuildContext* and a *MaterialPageRoute,* which also requires a *BuildContext*. The researchers also want to point out that the widget must be Stateful which means that the underlying child of that widget is utilizing all for a user to navigate onto the next screen. *.to()* does it even better. As simple as a one-liner code, researchers can now pass a navigation widget to it, after which it is good to go.

- **Organization**

```
navi.GetBuilder<PropertiesController>(
  builder: (controller) => FlutterSwitch(
      showOnOff: true,
      activeColor: navi.Get.theme.colorScheme.secondary,
      value: controller.isVacant,
      onToggle: (val) {
        controller.isVacantButton(val);
      }), // FlutterSwitch
), // navi.GetBuilder
const SizedBox(width: 10),
controller.isVacant ? const Text('Vacant') : const Text('Occupied')
```

Figure 9.3: A widget enclosed by a GetBuilder function

GetX separates the front-end logic and the business logic. We used *GetXControllers* as it performs as our state management extension, wherein any changes required within its variables can be easily updated and remains the same state unless deprecated on command. On the front-end, any widgets that require updating will be enclosed by a *GetBuilder* function and pass its controller variable.

Maps Display

For our researchers to view and plot data onto a map and determine real-time navigation from users and properties, we used Google Maps Services for it caters to both determining real-time navigation and plotting pins concerning ARent Property Geopoints.

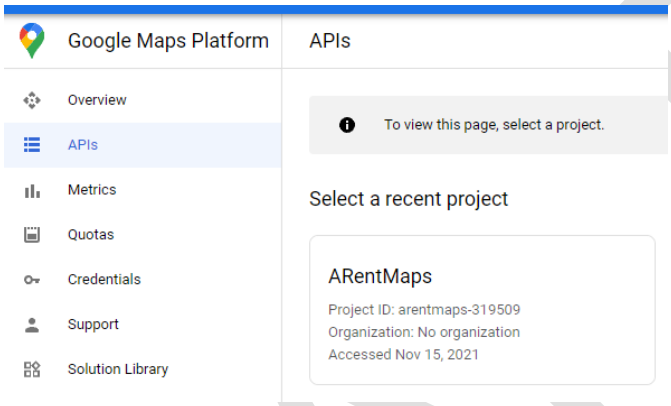


Figure 9.4: ARent subscribing to Google Cloud Platform via USJ-R Email

To determine our users' position in real-time, we used a flutter dependency called *flutter_location, enabling* ARent to identify the mobile device's geolocation in real-time. For the maps to integrate into the app, we used a flutter dependency called *google_maps_platform* and instantiated a widget called *GoogleMaps*.

```
getUserLocation() async {
  bool serviceenabled;
  PermissionStatus permissiongranted;

  serviceenabled = await location.serviceEnabled();
  if (!serviceenabled) {
    serviceenabled = await location.requestService();
    if (!serviceenabled) return;
  }       You, 4 weeks ago • Created Registration

  permissiongranted = await location.hasPermission();
  if (permissiongranted == PermissionStatus.denied) {
    permissiongranted = await location.requestPermission();
    if (permissiongranted != PermissionStatus.granted) return;
  }
  location.onLocationChanged.listen((event) {
    locationPosition = Rx(LatLng(event.latitude!, event.longitude!));
  });
}

void finalLoc() {
  location.onLocationChanged.listen((event) {
    locationPosition = Rx(LatLng(event.latitude!, event.longitude!));
    update();
  });
}
}
```

Figure 9.5: Function that determines the location of a user

The system initially requests permission to access the GPS on the user's mobile device. The system then accesses the *getUserLocation* function within the GetxController() named 'MapsController' and instantiates the device's location, which locates in the *LatLng* variable *locationPosition*. This function is executed only once due to GetX. The only thing a developer would do to access the variable would be to call *'MapsController().locationPosition'* from the maps controller.

Custom Matchmaking Algorithm

ARent's matchmaking algorithm was pre-determined using these four form attributes: **Type of Property, Location, Price, and Amenities**. As the team discussed finding the best algorithm for ARent, we decided to build a standard matchmaking algorithm which means creating an algorithm for each attribute and combining them into one. With these, we could determine the best property for our users.

The simple matchmaking algorithm integrated into the custom API makes it easy for the researchers to access the data natively. Since matchmaking is an API, we can

easily integrate these onto different platforms like the web whenever ARent decides to release it on a web platform.

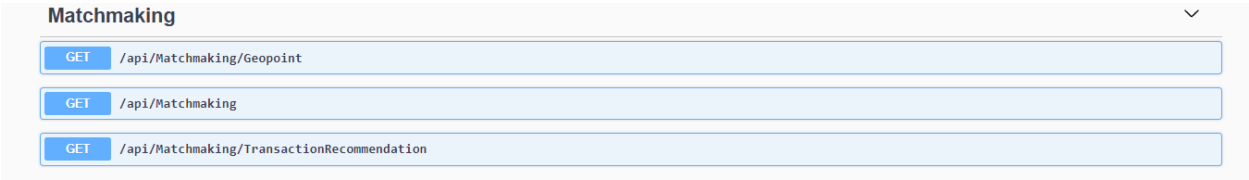


Figure 9.6.1: Custom API for ARent's Simple Matchmaking

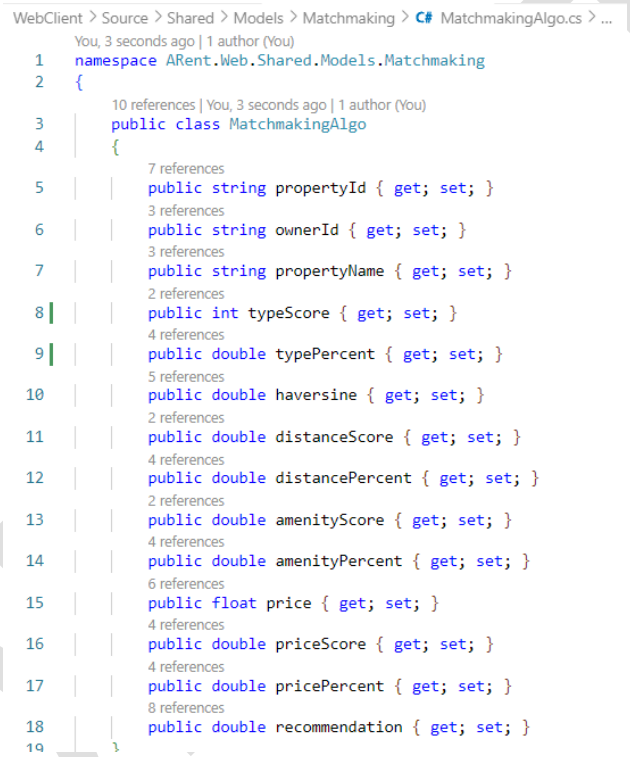


Figure 9.6.2: ARent's Matchmaking Algo Model

```
223         var getAllPropertyRecords = m_dataRentalPropertyProvider.GetRentalPropertyRecords();
224         var initData = new List<MatchmakingAlgo>();
```

The system gets all the properties in the database, precisely their latitude and longitude, and stores them into a variable named 'getAllPropertyRecords.' This returns a collection of properties. The system also creates an '*initData'* List of MatchmakingAlgo as this will use to store the matchmaking object.

	propertyId	ownerId	Latitude	longitude
3	0b5e7794-7cce-4db8-926e-fbc1ec9d55e7	3edf395b-e345-4d96-871e-7c39388124c0	10.270096360718572	123.87247168182333
4	b958699c-6d9c-4f06-8b05-7bd1dfba08db	3edf395b-e345-4d96-871e-7c39388124c0	10.269917	123.872515
1	04cc7ad0-332c-11ec-8e84-1d64de333cc4	31a77da7-6031-4335-b512-1da06be228a0	10.2633507	123.82784579999999
2	56a53b40-46ef-11ec-8929-d55d537c80a8	dc10407a-b609-4a26-ae7b-e70241c54186	10.2633441	123.8278352
5	41a67755-a823-47d4-ba8c-4dc1fe73ac9a	97ad359b-fd8c-4169-a5f5-e79134c91332	10.291801	123.865481

Table 4.1: Example values inside getAllPropertyRecords

For the location, the researchers used Haversine Formula to calculate the distance between two geopoints –latitude and longitude- on a sphere's surface.

```
30      //Haversine Formula to calculate distance from user to property
        2 references | ···
31      private static class Haversine
32      {       You, 2 months ago • Created Haversine Formula and Levenshein Algo
33          //gets the Linear Distance from LatLng A to LatLng B
            2 references
34          public static double calculate(GeoPoint p1, GeoPoint p2)
35          {
36              double r = 6372.8; // In kilometers
37              double dLat = toRadians(p2.Latitude - p1.Latitude);
38              double dLon = toRadians(p2.Longitude - p1.Longitude);
39              p1.Latitude = toRadians(p1.Latitude);
40              p2.Latitude = toRadians(p2.Latitude);
41
42              double a = Math.Sin(dLat / 2) * Math.Sin(dLat / 2) + Math.Sin(dLon / 2) * Math.Sin(dLon / 2) * Math.Cos(p1.Latitude) * Math.Cos(p2.Latitude);
43              double c = 2 * Math.Asin(Math.Sqrt(a));
44              return r * c;
45          }
46
47          //transforms latlng to radians
            4 references
48          public static double toRadians(double angle) => Math.PI * angle / 180.0;
49      }
```

$$\operatorname{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2}$$

From the Haversine calculate function, the system retrieved the distance concerning the user's location.

	propertyId	ownerId	haversineResult (in km)
3	0b5e7794-7cce-4db8-926e-fbc1ec9d55e7	3edf395b-e345-4d96-871e-7c39388124c0	0.0278 km
4	b958699c-6d9c-4f06-8b05-7bd1dfba08db	3edf395b-e345-4d96-871e-7c39388124c0	0.0282 km
1	04cc7ad0-332c-11ec-8e84-1d64de333cc4	31a77da7-6031-4335-b512-1da06be228a0	4.944 km
2	56a53b40-46ef-11ec-8929-d55d537c80a8	dc10407a-b609-4a26-ae7b-e70241c54186	4.946 km
5	41a67755-a823-47d4-ba8c-4dc1fe73ac9a	97ad359b-fd8c-4169-a5f5-e79134c91332	5.183 km

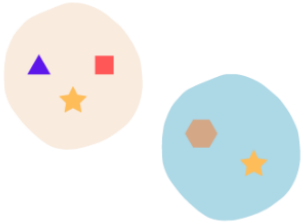
Table 4.2: Example values of haversineResult inside initData

Based on the data, the system uses a scaling method wherein if the distance is near, or when the *haversineResult* is relatively near to zero, the scale will be close to 1. The formula used for this is $\frac{1}{(1+haverSineResult)}$

	propertyId	haversineResult (in km)	haversineScore
3	0b5e7794-7cce-4db8-926e-fbc1ec9d55e7	0.0278 km	0.9728
4	b958699c-6d9c-4f06-8b05-7bd1dfba08db	0.0282 km	0.9724
1	04cc7ad0-332c-11ec-8e84-1d64de333cc4	4.944 km	0.16822
2	56a53b40-46ef-11ec-8929-d55d537c80a8	4.946 km	0.16821
5	41a67755-a823-47d4-ba8c-4dc1fe73ac9a	5.183 km	0.16171

Table 4.3: Example values of haversineScale inside initData

For amenities, researchers used the concept of Jaccard Similarity as the best approach to determining the scale of amenities. By definition, it is a similarity of two sets of data in which the members share distinct data [19].



$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

$$\frac{|\bigstar|}{|\blacktriangle \bigstar \blacksquare \hexagon|} = \frac{1}{4}$$

Figure 9.7: Basic Concept of Jaccard Similarity

Figure 9.7 briefly explains the researchers understanding of the algorithm. A property would represent as a set of Letters (i.e., J(A), J(B)) and each set of Letters has distinct shapes with each representing an amenity (i.e., Air-conditioning, Kitchen, Wi-Fi, etc.).

As an example, let User A = {Dryer, Air-Conditioning}

	propertyId	Amenities	jaccardScore
3	0b5e7794-7cce-4db8-926e-fbc1ec9d55e7	B = {}	0
4	b958699c-6d9c-4f06-8b05-7bd1dfba08db	B = {}	0
1	04cc7ad0-332c-11ec-8e84-1d64de333cc4	B = {Dryer, Air-Conditioning}	1
2	56a53b40-46ef-11ec-8929-d55d537c80a8	B = {Dryer, Air-Conditioning}	1
5	41a67755-a823-47d4-ba8c-4dc1fe73ac9a	B = {Dryer}	0.5

Table 4.4: Amenities in Each Property

Table 4.4 tells us that each property has these types of amenities. The Jaccard formula can be computed as the size of the intersection divided by the union of two sets. $\frac{A\ intersect\ B}{A\ union\ B}$ and it is stored on the *jaccardScore* variable.

```
1 reference
private static double GetAmenitiesScore(string Id, List<string> listOfAmenities, IAmenityProvider x)
{
    double total = 0.0;
    var getAllAmenityByProperty = x.FindAmenitiesByProperty(Id);

    //checks all Amenities of selected property
    foreach (var ap in getAllAmenityByProperty)
    {
        string stringAmenity = null;
        var amenityDisplayName = ap.AmenityName.GetAttribute<DisplayAttribute>();
        _ = amenityDisplayName == null ? stringAmenity = ap.AmenityName.ToString() : stringAmenity = amenityDisplayName.Name.ToString();
        foreach (string stringap in listOfAmenities)
        {
            if (stringap.ToLower().Contains(stringAmenity.ToLower()))
            {
                total++;
            }
        }
    }
    return total;
}
```

For the price, we get a minimum price from the set of properties and divide it by each property's price. For this example, given from the set of properties, the minimum value is 2000. The formula used is $\frac{minPrice}{selectedProperty.price}$

Example no.	propertyId	price	priceScore
3	0b5e7794-7cce-4db8-926e-fbc1ec9d55e7	Php 3,600	0.55
4	b958699c-6d9c-4f06-8b05-7bd1dfba08db	Php 8,000	0.25
1	04cc7ad0-332c-11ec-8e84-1d64de333cc4	Php 4,500	0.44
2	56a53b40-46ef-11ec-8929-d55d537c80a8	Php 8,000	0.25
5	41a67755-a823-47d4-ba8c-4dc1fe73ac9a	Php 2,000	1.00

Table 4.5: Example values of priceScore inside initData

```
73        /// Gets the priceScore of the selected property.
          1 reference
74        private static double GetPriceScore(MatchmakingAlgo item, double min) => min / item.price;
75
```

After computing each attribute from its own set of accustomed algorithms, the system computes the percentile of the result. ARent satisfies the use of a Weighted-scoring method. Weighted scoring calculates by multiplying the resulted value and its corresponding weight. The matchmaking will use the following weighted scores:

ATTRIBUTE	WEIGHT
Type	10%
Location	20%
Amenities	30%
Price	40%

Table 4.6: Weighted Scoring for Each Attribute

The system then calculates each result of attributes concerning their given weights. The formula used is $Score * Weight$

					20%	30%	40%	10%
	propertyId	haversineScore	jaccardScore	priceScore	haversineWeight	amenitiesWeight	priceWeight	typeWeight
3	0b5e7794-7cce-4db8-926e-fbc1ec9d55e7	0.9728	0	0.55	**0.03364**	**0**	**0.222**	**0.1**
4	b958699c-6d9c-4f06-8b05-7bd1dfba08db	0.9724	0	0.25	**0.03363**	**0**	**0.1**	**0**
1	04cc7ad0-332c-11ec-8e84-1d64de333cc4	0.16822	1	0.44	**0.1945**	**0.3**	**0.177**	**0.1**
2	56a53b40-46ef-11ec-8929-d55d537c80a8	0.16821	1	0.25	**0.1944**	**0.3**	**0.1**	**0**
5	41a67755-a823-47d4-ba8c-4dc1fe73ac9a	0.16171	0.5	1.00	**0.0323**	**0.15**	**0.4**	**0.1**

Table 4.7: Applied Weighted Scoring from Each Property

```
//populate initData
foreach (var item in getAllPropertyRecords)
{
    //init
    var locA = new GeoPoint { Latitude = lat1, Longitude = long1 };
    var locB = new GeoPoint { Latitude = item.Latitude, Longitude = item.Longitude };
    double haversineResult = Haversine.calculate(locA, locB);
    double distScore = 1 / (1 + haversineResult);
    double amenityScoreResult = GetAmenitiesScore(item.Property_Id, listOfAmenities, m_dataAmenityProvider) / listOfAmenities.Count;

    //add initData
    initData.Add(new MatchmakingAlgo
    {
        propertyId = item.Property_Id,
        ownerId = item.Owner_Id,
        propertyName = item.RentalHouseName,
        //distance from user to property in KM
        haversine = haversineResult,
        distanceScore = distScore,
        //percentile of location
        distancePercent = distScore * locationP,
        amenityScore = amenityScoreResult,
        //percentile of amenity
        amenityPercent = amenityScoreResult * amenityP,
        price = item.RentalRatePerMonth,
        priceScore = 0.0,
        pricePercent = 0.0,
        recommendation = 0.0,
    });
}
```

Once everything has been pre-populated from *initData*, the system determines the best property based on our weighted scores. ARent uses the concept of Percentage-based Scoring to determine each property's final 'matchmaking' score. To find the summation of scores multiplied by 100. The formula used is

$$matchmaking = (typeWeight + locWeight + jaccWeight + priceWeight) * 100$$

	propertyId	typeWeight	haversineWeight	amenitiesWeight	priceWeight	Matchmaking
3	0b5e7794-7cce-4db8-926e-fbc1ec9d55e7	0.1	0.03364	0	0.222	**35.59%**
4	b958699c-6d9c-4f06-8b05-7bd1dfba08db	0	0.03363	0	0.1	**13.36%**
1	04cc7ad0-332c-11ec-8e84-1d64de333cc4	0.1	0.1945	0.3	0.177	**77.24%**
2	56a53b40-46ef-11ec-8929-d55d537c80a8	0	0.1944	0.3	0.1	**59.45%**
5	41a67755-a823-47d4-ba8c-4dc1fe73ac9a	0.1	0.0323	0.15	0.4	**68.23%**

Table 4.8: Applied Percentage-based Scoring from Each Property

```
197             //update initData
198             foreach (var item in initData)
199             {
200                 _ = initData.Where(x => x.propertyId == item.propertyId).Select(y =>
201                 {
202                     double temp = GetPriceScore(item, min);
203                     //updates the priceScore
204                     y.priceScore = temp;
205                     //updates the price and returns a percentile of it.
206                     y.pricePercent = temp * priceP;
207                     //calulate percentage of matchmaking
208                     y.recommendation = (item.distancePercent + item.amenityPercent + (temp * priceP)) * 100;
209                     return y;
210                 }).ToList();
211             }
212
213             //sort object descending with top 5 applied
214             initData.Sort((e1, e2) => e2.recommendation.CompareTo(e1.recommendation));
215
216         }
217         catch (Exception e)
218         {
219             Console.WriteLine("Error in matchmakingAlgo " + e);
220         }
221         return initData;
222     }
```

Based on the matchmaking result, Property Example no. 1 was the most matched, scoring 71%, followed by Property Example no. 5 with 68% and Property Example no. 2 with a score of 61%. The system shows if assessing each attribute affects a property's matchmaking score.

Matchmaking by Transaction

In contrast, ARent's matching algorithm uses transactions as the foundation for user recommendations since they include user-preferred criteria. The researchers were thus able to develop a system that "*matches*" the user has desired property information. The accounts that we will be using are the following:

No.	User_Id	Transaction_Id
1	6ecd8c99-4036-403d-bf84-cf8400f67836	6370b6c5-4178-4074-a989-700a721babff
2	ea9608fd-5ace-4c99-bf6d-64ab77bf814b	882c3a3f-2fbe-44a3-9023-a536fb46d66f

Table 4.9: Accounts with Transactions

Since basic matchmaking is in the backend, the researchers could get data from the backend by implementing different methods. We get the related property because of the transaction. *GetRentralTransactionByProperty* is a function developed by the researchers that returns a transaction property.

```
65        public IEnumerable<RentalTransaction> GetRentalTransactionsByProperty(string name) =>
66          m_dbContext.RentalTransactions.Where(p => p.Property_Id.Contains(name));
```

The researchers developed the *GetSingleProperty* method, which takes property Id as an argument. The system then cross-checks with the Properties services and provides all related property information.

```
56        public RentalProperty GetRentalPropertySingleRecord(string id) =>
57            m_dbContext.RentalProperties.First(rp => rp.Property_Id == id);
```

The researchers also designed a method called *FindAmenitiesByProperty*, which takes *property_Id* as input since the Amenities have unique relationships. The system then cross-references the Amenities service and provides all connected property information.

```
63        public IEnumerable<Amenities> FindAmenitiesByProperty(string name) =>
64            m_dbContext.Amenities.Where(p => p.Property_Id.Contains(name));
```

The system is still unaware of the facilities with which it is related. The researchers developed a second method called *getAmenitiesofTransaction*, which provides a list of transaction-specific amenities. This method takes two arguments: *amenityProvider and transaction_Id*.

```
83      private static List<String> GetAmenitiesConversion(IAmenityProvider x, string transactionId)
84      {
85          var result = new List<String>();
86          string stringProperty;
87          var getAllAmenityOfProperty = x.FindAmenitiesByProperty(transactionId);
88          if (getAllAmenityOfProperty.ToArray().Length > 0)
89          {
90              foreach (var item in getAllAmenityOfProperty)
91              {
92                  var amenityDisplayName = item.AmenityName.GetAttribute<DisplayAttribute>();
93                  //If there is no attibute, pass the enum method.
94                  _ = amenityDisplayName == null ? stringProperty = item.AmenityName.ToString() : stringProperty = amenityDisplayName.Name;
95                  result.Add(stringProperty);
96              }
97          }
98          return result;
99      }
100
```

```
7       public enum AmenityName
8       {
9           //Essentials
10          [Display(Name = "Wi-Fi")]
            1 reference
11          WiFi,
```

Researchers were having trouble finding the *"display name"* of a given amenity while examining the model structure of our Amenities. The researchers discovered a way around the problem. Using a custom function named *GetAttribute*, it turns each attribute into a string as it iterates over the attributes, essentially a tool that returns the name *"display name"* of that enum entry. Researchers developed this function for this purpose.

```
506     public static TAttribute GetAttribute<TAttribute>(this Enum enumValue)
507             where TAttribute : Attribute
508     {
509         return enumValue.GetType()
510                         .GetMember(enumValue.ToString())
511                         .First()
512                         .GetCustomAttribute<TAttribute>();
513     }
```

As we did in our custom matchmaking service, once everything has been pre-populated from initData, the system determines the best property based on our weighted scores. ARent uses the concept of Percentage-based Scoring to determine each property's final 'matchmaking' score. To find the summation of scores multiplied by 100. The formula used is

$$matchmaking = (typeWeight + locWeight + jaccWeight + priceWeight) * 100$$

```
339             //populate initData
340             foreach (var item in getAllPropertyRecords)
341             {
342                 //init
343                 var locA = new GeoPoint { Latitude = getSingleProperty.Latitude, Longitude = getSingleProperty.Longitude };
344                 var locB = new GeoPoint { Latitude = item.Latitude, Longitude = item.Longitude };
345                 double haversineResult = Haversine.calculate(locA, locB);
346                 //Console.WriteLine("haversineResult:" + haversineResult);
347                 double distScore = 1 / (1 + haversineResult);
348                 //Console.WriteLine("distScore:" + distScore);
349                 int typeResult = getTypeScoreTransaction(rentalType: getSingleProperty.RentalType, selectedProperty: item);
350                 //Console.WriteLine("typeResult:" + typeResult);
351                 double amenityScoreResult = 0.0;
352                 if (getAmenitiesSingleProperty.ToArray().Length >= 1)
353                 {
354                     amenityScoreResult = (GetAmenitiesScore(Id: item.Property_Id, listOfAmenities: getAmenitiesOfTransaction,
355                     x: m_dataAmenityProvider) / getAmenitiesSingleProperty.ToArray().Length);
356                     Console.WriteLine("amenityScoreResult:" + amenityScoreResult);
357                 }
358                 //add initData
```

Based on the transactions of those renters, the system was able to return the information of those properties related to the transaction.

User_Id	6ecd8c99-4036-403d-bf84-cf8400f67836	ea9608fd-5ace-4c99-bf6d-64ab77bf814b
Transaction_Id	6370b6c5-4178-4074-a989-700a721babff	882c3a3f-2fbe-44a3-9023-a536fb46d66f
Property_name	3 Bedroom Townhouse in a Gated Community, near Ateneo De Cebu	Furnished 4 Bedrooms Townhouse For Rent Located In Central Cebu City Near San Carlos Main
Location	10.3429073	10.330847
	123.9286125	123.899348
Type	Dormitory	Boarding House
Price	P6,899.00	P13,444.00
Amenities	{19, 17,16,14,13,10,11,9,7,4,3,6,1,2,0}	{7,1,0,19,17,16,10,9,13,11,6,4,3,2}

Table 4.10: Data Retrieved from GetRentalTransactionByProperty()

The system returns the suggested property rentals for a particular searcher. The sorting of properties occurs on the backend due to the presence of database providers on the backend, which includes data parsing, data initialization, conversion of properties, obtaining distances between properties, amenities, and price comparison, among others, during the development of the custom matchmaking API. The iteration of properties establishes openness between the database and the matching process.

Simply creating functions to execute each essential relational object was all the researchers performed. As it filters through all the attributes, it cross-checks each one so that we can establish their recommendation score, which is a factor in determining our recommendation result.

In this section, we will see an in-depth result of the recommended properties of each of our two users.

	Property1	Property2	Property3	Property4	User1
Apartment					
Boarding House					
Dormintory	1	1	1	1	1
typeScore	1	1	1	1	
typePercent	**0.1**	**0.1**	**0.1**	**0.1**	

Table 4.11: Type result for 6ecd8c99-4036-403d-bf84-cf8400f67836

	Property1	Property2	Property3	Property4	User2
Apartment			1		
Boarding House	1				1
Dormintory		1		1	
typeScore	1	0	0	0	
typePercent	**0.1**	**0**	**0**	**0**	

Table 4.12: Type result for ea9608fd-5ace-4c99-bf6d-64ab77bf814b

Tables 4.11 and 4.12 shows us the results of each recommendation from their respective type of property. This was achieved by using the simple scoring method wherein if a property contains this user-preferred type of property, then it is scored by 1.

		Property1	Property2	Property3	Property4	User1
0	Wi-Fi	1	1		1	1
1	Kitchen	1	1		1	1
2	Bathroom Essentials	1	1		1	1
3	Bedroom Comforts	1	1		1	1
4	Coffee Maker	1	1		1	1
5	Dryer		1		1	
6	Air Conditioning	1	1		1	1
7	Dedicated Workspace	1	1		1	1
8	TV		1		1	
9	Hair Dryer	1	1		1	1
10	Iron	1	1		1	1
11	Swimming Pool	1	1		1	1
12	Hot Tub				1	
13	Free Parking	1	1		1	1
14	Gym	1	1		1	1
15	No Smoking	1		1	1	
16	Beachfront	1	1		1	1
17	Waterfront	1	1		1	1
18	Smoke Alarm			1	1	

19	Fire Extinguisher	1	1		1	1
	amenityScore	15	15	0	15	
		1	1	0	1	
	amenityPercent	**0.3**	**0.3**	**0**	**0.3**	

Table 4.13: Amenity result for 6ecd8c99-4036-403d-bf84-cf8400f67836

		Property1	Property2	Property3	Property4	User2
0	Wi-Fi	1	1		1	1
1	Kitchen	1	1		1	1
2	Bathroom Essentials	1	1		1	1
3	Bedroom Comforts	1	1		1	1
4	Coffee Maker	1	1		1	1
5	Dryer					
6	Air Conditioning	1	1		1	1
7	Dedicated Workspace	1	1		1	1
8	TV	1			1	
9	Hair Dryer	1	1		1	1
10	Iron	1	1		1	1
11	Swimming Pool	1	1		1	1
12	Hot Tub	1			1	
13	Free Parking	1	1		1	1
14	Gym	1	1		1	
15	No Smoking	1	1	1	1	
16	Beachfront	1	1		1	1
17	Waterfront	1	1		1	1
18	Smoke Alarm	1		1	1	
19	Fire Extinguisher	1	1		1	1
	amenitiyScore	14	14	0	14	

		1	1	0	1
	amenityPercent	**0.3**	**0.3**	**0**	**0.3**

Table 4.14: Amenity result of ea9608fd-5ace-4c99-bf6d-64ab77bf814b

Tables 4.13 and 4.14 illustrate the scoring methodology for each property's information. This was made possible by the Jaccard Similarity principle, which assigns a score to asymmetric binary data. If user1 chooses "WiFi" and property1 offers "WiFi," then they have a score of 1.

0.1			**0.2**		**0.3**			**0.4**		
typeScore	typePercent	haversine	distanceScore	distancePercent	amenityScore	amenityPercent	price	priceScore	pricePercent	
1	0.1	3.482848389	0.223072456	0.044614491	1	0.3	5778	0.4325026	0.173001038	**61.76**
1	0.1	8.946356368	0.100539329	0.020107866	1	0.3	7566	0.3302934	0.132117367	**55.22**
1	0.1	5.270722755	0.159471251	0.03189425	0	0	2499	1	0.4	**53.18**
1	0.1	7.594730351	0.116350363	0.023270073	1	0.3	10000	0.2499	0.09996	**52.32**

Table 4.15: Recommendation result for 6ecd8c99-4036-403d-bf84-cf8400f67836

0.1			**0.2**		**0.3**			**0.4**		
typeScore	typePercent	haversine	distanceScore	distancePercent	amenityScore	amenityPercent	price	priceScore	pricePercent	
1	0.1	2.288171472	0.304120393	0.060824079	1	0.3	12354	0.2022827	0.080913065	**54.17**
0	0	3.221430534	0.236886523	0.047377305	1	0.3	5788	0.4317554	0.172702142	**52.00**
0	0	0.872065465	0.534169354	0.106833871	0	0	2499	1	0.4	**50.68**
0	0	5.633228249	0.150756157	0.030151231	1	0.3	6899	0.3622264	0.144890564	**47.50**

Table 4.16: Recommendation result for ea9608fd-5ace-4c99-bf6d-64ab77bf814b

Tables 4.15 and 4.16 detail the recommendation's outcomes. In the last column, we can see a range of scores for each property, ranked according to the four characteristics. User1 has a recommendation score of 61 percent, whereas User2 has a recommendation score of 54 percent. This difference in outcomes is correlated with each user's transactions. Consequently, we can see that the four criteria contribute to ARent's straightforward grading mechanism. The backend will update its data and deliver a fresh set of suggestions when users do transactions. The 'Matchmaking by Transaction' custom API has the same functions as the 'Custom Matchmaking' custom API. The way of contacting each table differs by constructing a distinct set of procedures and supplying arguments to feed our system with matchmaking data.

Augmented Reality

```
val earth = session.earth
if (earth?.trackingState == TrackingState.TRACKING) {
  val cameraGeospatialPose = earth.cameraGeospatialPose
```

Figure 10.1: Requests for geospatial information from ARCore

This GeospatialPose includes a location, represented in latitude and longitude, along with an estimation of the location's accuracy; elevation, with an estimate of the elevation's accuracy; heading, an estimation of the direction the device is facing, with an estimate of the heading's accuracy.

```
  activity.view.mapView?.updateMapPosition(
    latitude = cameraGeospatialPose.latitude,
    longitude = cameraGeospatialPose.longitude,
    heading = cameraGeospatialPose.heading
  )
}
if (earth != null) {
  activity.view.updateStatusText(earth, earth.cameraGeospatialPose)
}
```

Figure 10.2: Get the user's current geospatial data

The figure above used the GeospatialPose stored in cameraGeospatialPose. The map will display a purple marker that rotates and moves as the user views and walks across its setting. It updates the current user's location on the map using the Geospatial API's returned values.

```
private val retrofit = Retrofit.Builder()
    .client(unsafeOkHttpClient)
    .addConverterFactory(MoshiConverterFactory.create(moshi))
    .baseUrl(BASE_URL)
    .build()

interface ARentApiService {
    @GET("properties/{id}")
    fun getPropertyInfoAsync(@Path("id") id: String): Call<PropertyInfo>
}

object ARent {
    val api: ARentApiService by lazy {
        retrofit.create(ARentApiService::class.java)
    }
}
```

Figure 10.3: Consuming Web Services

The API is linked to the Augmented Reality component of the project, as depicted in the figure above. This allows retrieval of data from the ARent Database. It can access the property information by calling a specific property by its ID.

```
val propertyId = "082ac0a7-cdc0-4869-b7c7-8d2dc18c2313"
val call = ARent.api.getPropertyInfoAsync(propertyId)
```

Figure 10.4: Calling a specific property from the ARent Database

```
val earth = session?.earth ?: return
if (earth.trackingState != TrackingState.TRACKING) {
  return
}
earthAnchor?.detach()
call.enqueue(object : Callback<PropertyInfo> {
  override fun onResponse(call: Call<PropertyInfo>, response: Response<PropertyInfo>) {
    if (response.isSuccessful) {
      val propertyInfo = response.body()
      Log.d(HelloGeoActivity.TAG,  msg: "propertyInfo.latitude=" + propertyInfo?.latitude)
      Log.d(HelloGeoActivity.TAG,  msg: "propertyInfo.longitude=" + propertyInfo?.longitude)

      val altitude = earth.cameraGeospatialPose.altitude + 3
      val latitude = propertyInfo?.latitude ?: 0.0
      val longitude = propertyInfo?.longitude ?: 0.0
      val qx = 0f
      val qy = 0f
      val qz = 90f
      val qw = 1f

      earthAnchor =
        earth.createAnchor(latitude, longitude, altitude, qx, qy, qz, qw)
```

Figure 10.5: Placing an anchor using Earth's coordinates of the property

Figures 10.4 and 10.5 superimpose an anchor on a real-world setting using the property's latitude and longitude fetched from the database. The Earth object's TrackingState is set to TRACKING so that the position of the Earth is known. createAnchor renders the 3D asset with the geodetic coordinates of the selected property and a specified rotation into the scene. This anchor aims to remain steady and fixed at the coordinates and altitude supplied.

TESTING PROCESS AND CASES

There will be two test cases: the JMeter testing and the Project Application. Each phase has its test methods. JMeter will indicate which HTTP methods will test the

project application to see whether the test results will be passed or not for each test case.

JMeter Test Cases:

Test Case 1: Users

- **HTTP GET**

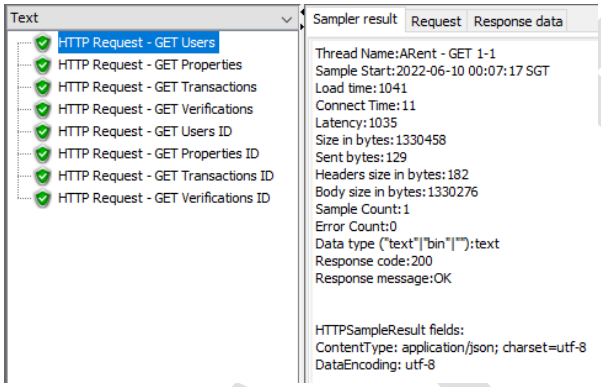


Figure 11.1.1: GET Users Test

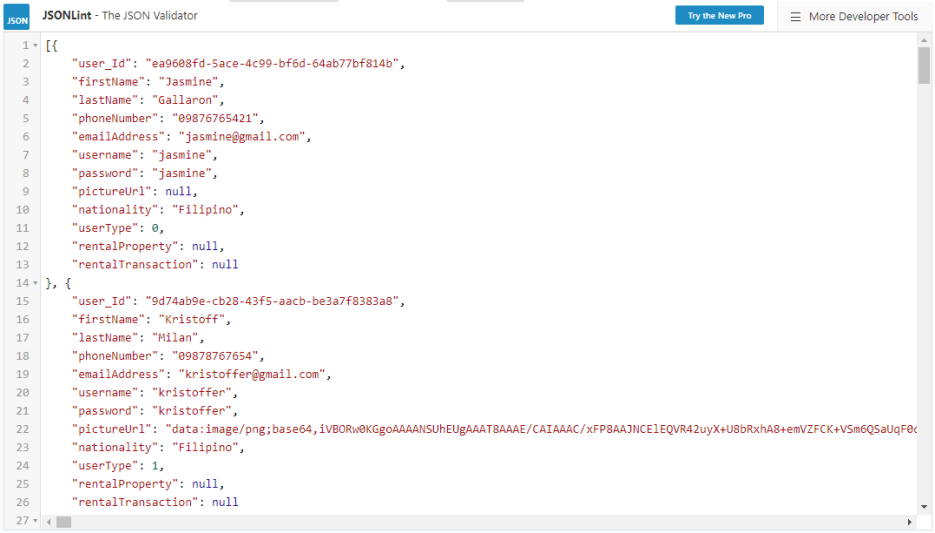


Figure 11.1.2: GET Users Test – Result

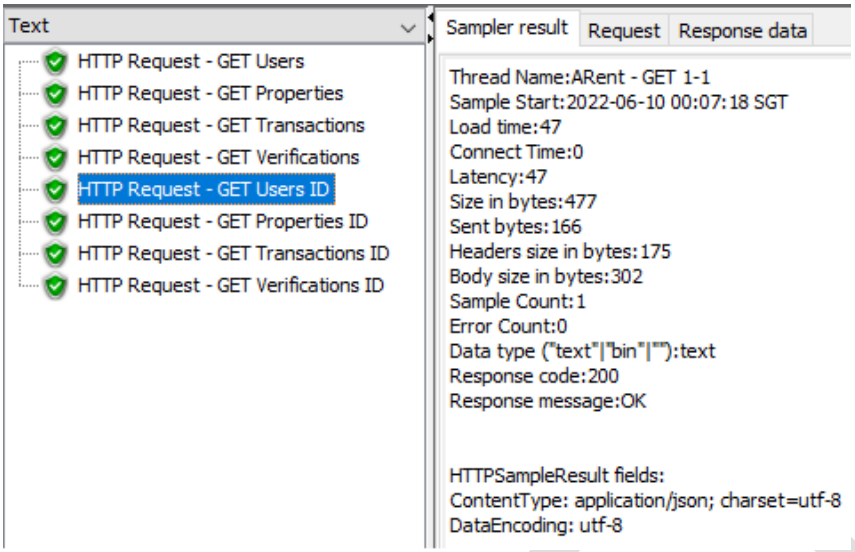


Figure 11.2.1: GET User ID Test

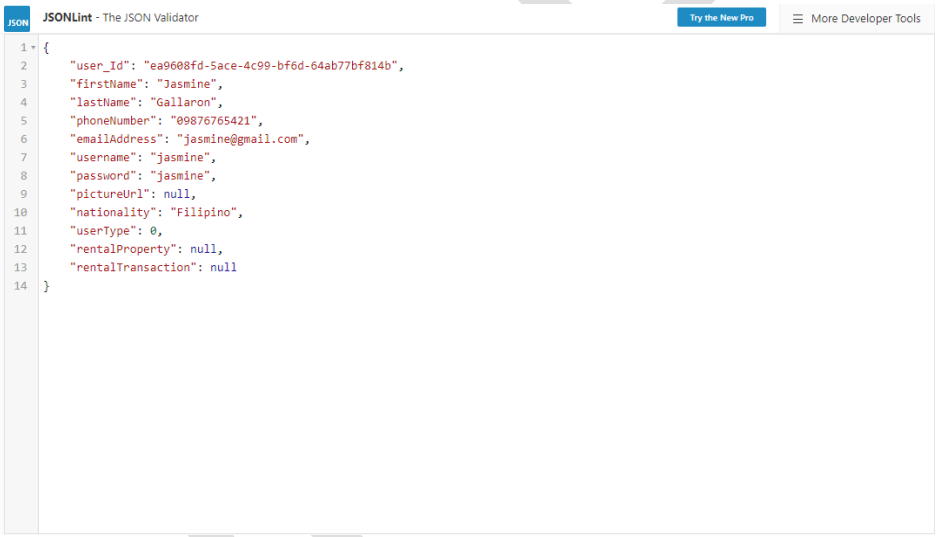


Figure 11.2.2: GET User ID Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
GET Users	Getting the Users by putting the address "~/api/Users" in the URL path	Display in the test that all Users will be returned by what is requested	Figure 11.1.2 displays the data result of the Users

GET User ID	Getting the User by putting specified ID in the address “~/api/Users/{id}” in the URL path	Display in the test that it is passed, and the User ID will be returned by what is requested of its specified ID	Figure 11.2.2 displays the data result of the User of its respected ID which results to be passed

- **HTTP POST**

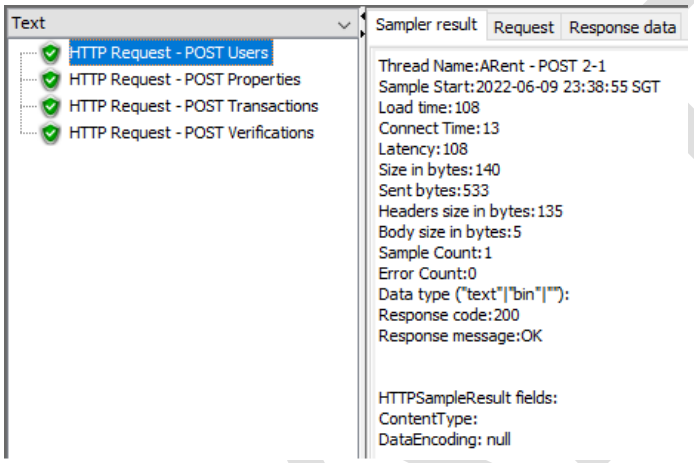


Figure 11.3.1: POST User Test

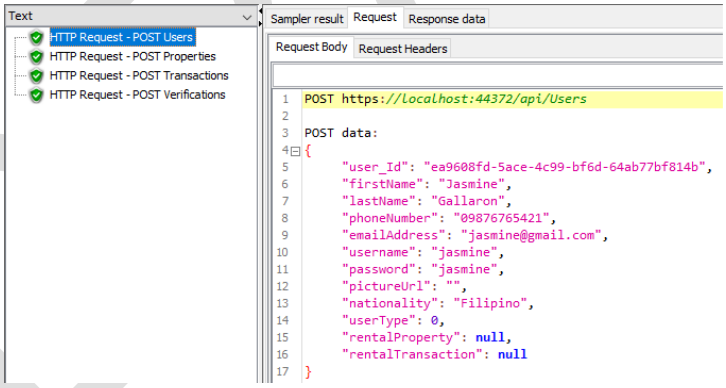


Figure 11.3.2: POST User Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
POST User	Adding a User in the JSON format of its respected data to its URL path “~/api/Users”	The User has been added in the API and passed through the database and it results to passed in the test	Figure 11.3.2 shows the result that has been passed and added to the Users records

- **HTTP PUT**

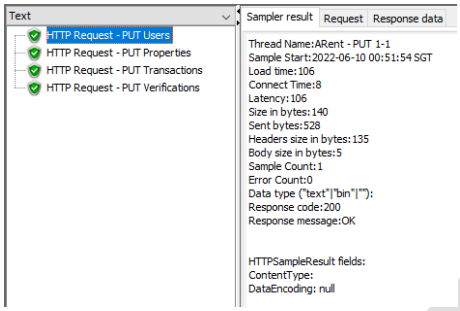


Figure 11.4.1: PUT User Test

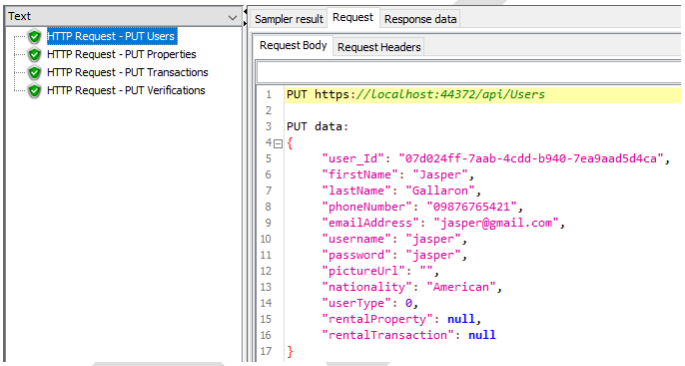


Figure 11.4.2: PUT User Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
PUT User	Edit the data of a specified User that is added in the URL Path “~/api/Users” using the “User_Id”	Successfully edited of a certain User that corresponds to the ID that has been specified	Figure 11.4.2 shown the data result of a certain User that is edited which results to passed

- **HTTP DELETE**

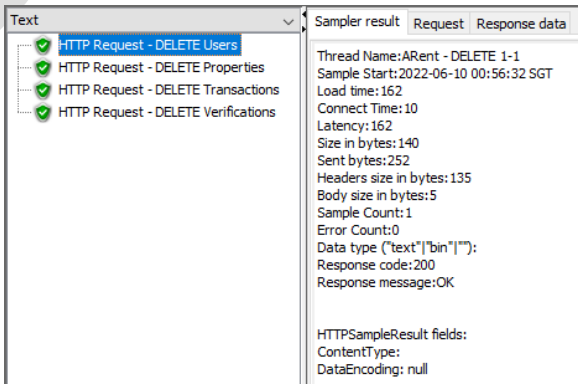


Figure 11.5.1: DELETE User Test

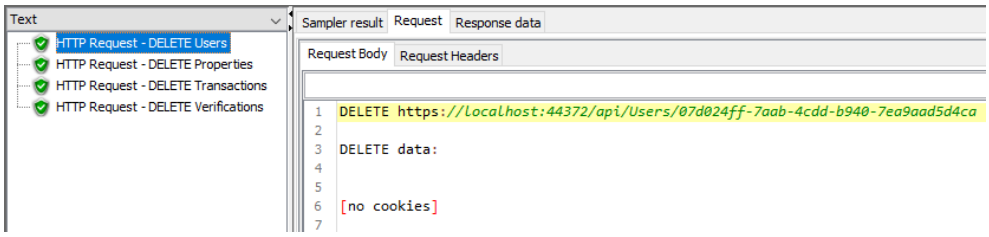


Figure 11.5.2: DELETE User Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
DELETE User	Delete a data of a specified User that is added in the URL Path "~/api/Users/{id}" which corresponds to User ID	Successfully deleted of a certain User that corresponds to the ID that has been specified	Figure 11.5.2 shown the data result of a certain User that is deleted which results to passed

Test Case 2: Property

- **HTTP GET**

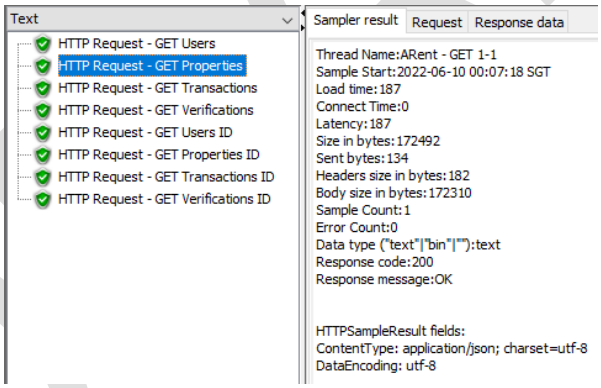


Figure 12.1.1: GET Properties Test

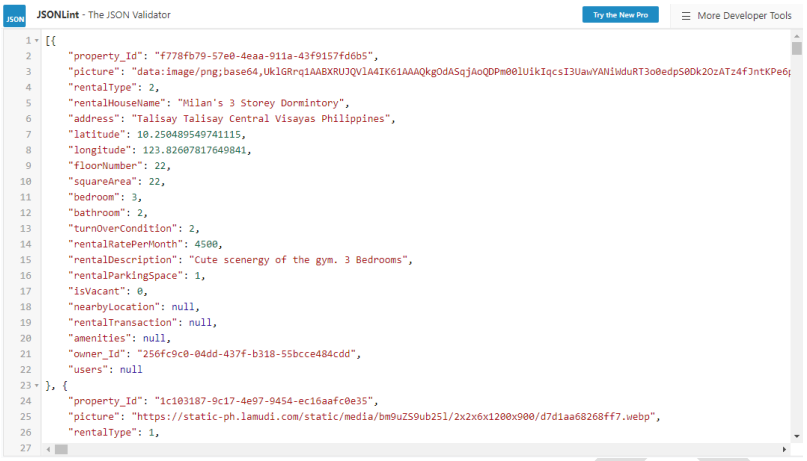


Figure 12.1.2: GET Properties Test – Result

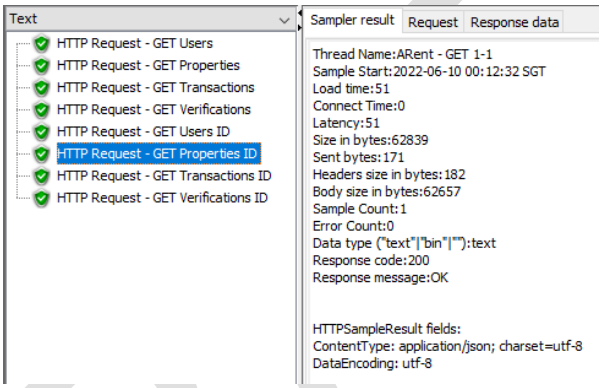


Figure 12.2.1: GET Property ID Test

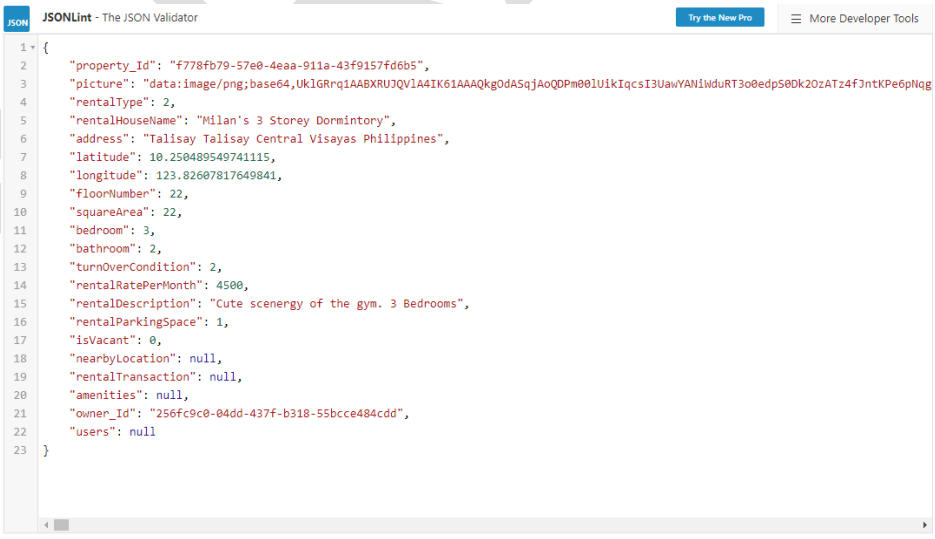


Figure 12.2.2: GET Property Test – Result

Test Module	Test Scenario	Expected Result	Actual Result

GET Properties	Getting the Properties by putting the address “~/api/properties” in the URL path	Display in the test that all Properties will be returned by what is requested	Figure 12.1.2 displays the data result of the Properties
GET Property ID	Getting the Property by putting specified ID in the address “~/api/properties/{id}” in the URL path	Display in the test that it is passed, and the Property ID will be returned by what is requested of its specified ID	Figure 12.2.2 displays the data result of the Property of its respected ID which results to be passed

- **HTTP POST**

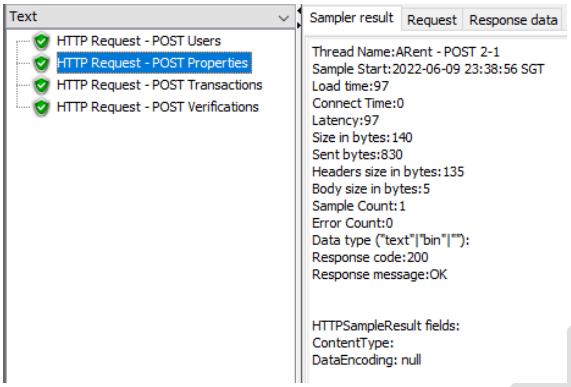


Figure 15.3.1: POST Property Test

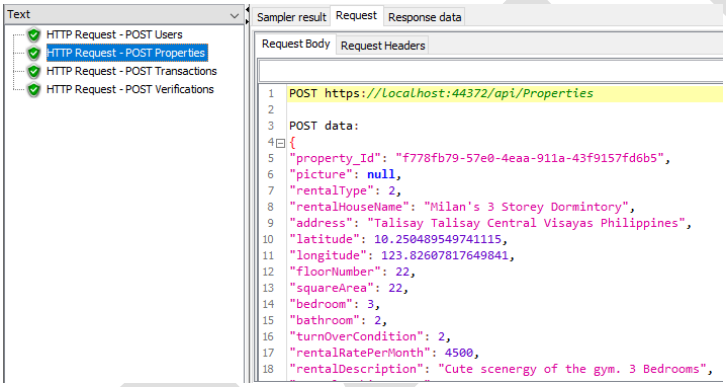


Figure 12.3.2: POST Property Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
POST Property	Adding a Property in the JSON format of its respected data to its URL path “~/api/Properties”	The Property has been added in the API and passed through the database and it results to passed in the test	Figure 12.3.2 shows the result that has been passed and added to the Property records

- **HTTP PUT**

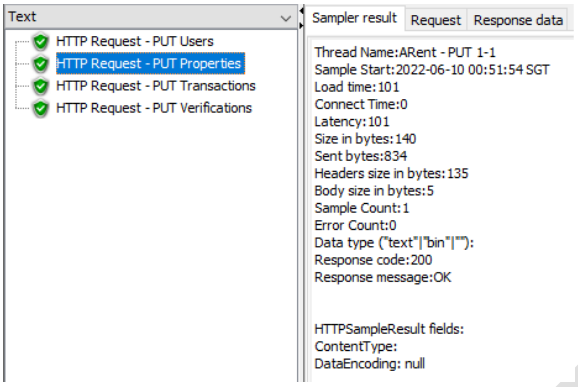


Figure 12.4.1: PUT Property Test

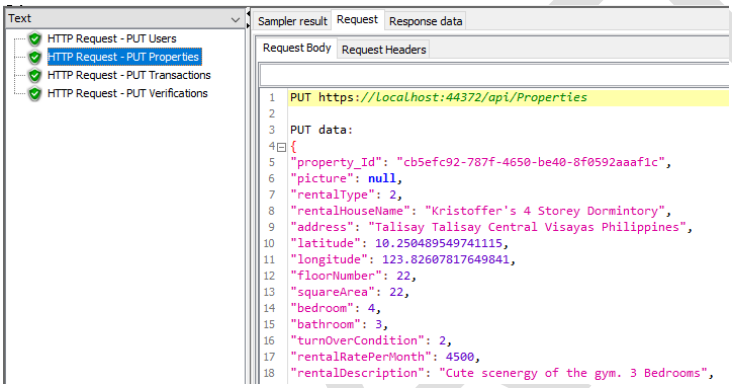


Figure 12.4.2: PUT Property Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
PUT Property	Edit the data of a specified Property that is added in the URL Path “~/api/Properties” using the “Property_Id”	Successfully edited of a certain Property that corresponds to the ID that has been specified	Figure 12.4.2 shown the data result of a certain Property that is edited which results to passed

- **HTTP DELETE**

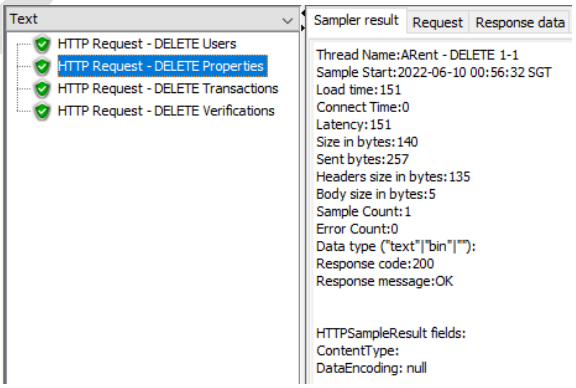


Figure 15.5.1: DELETE Property Test

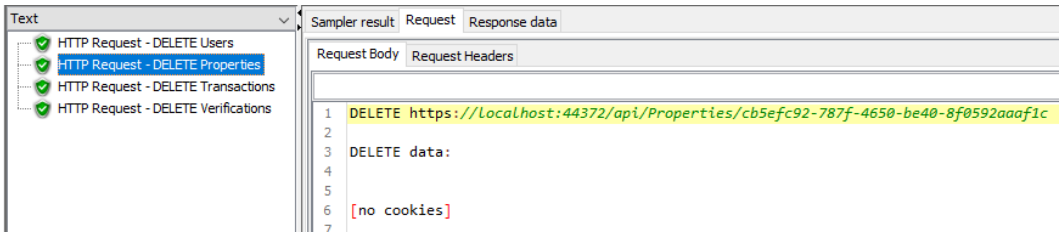


Figure 12.5.2: DELETE Property Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
DELETE Property	Delete a data of a specified Property that is added in the URL Path “~/api/Properties/{id}” which corresponds to Property ID	Successfully deleted of a certain Property that corresponds to the ID that has been specified	Figure 12.5.2 shown the data result of a certain Property that is deleted which results to passed

Test Case 3: Transaction

- **HTTP GET**

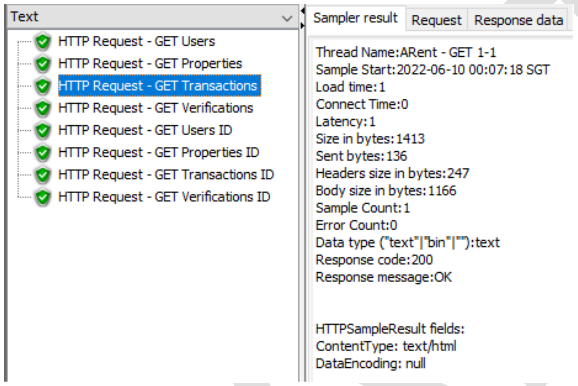


Figure 13.1.1: GET Transactions Test

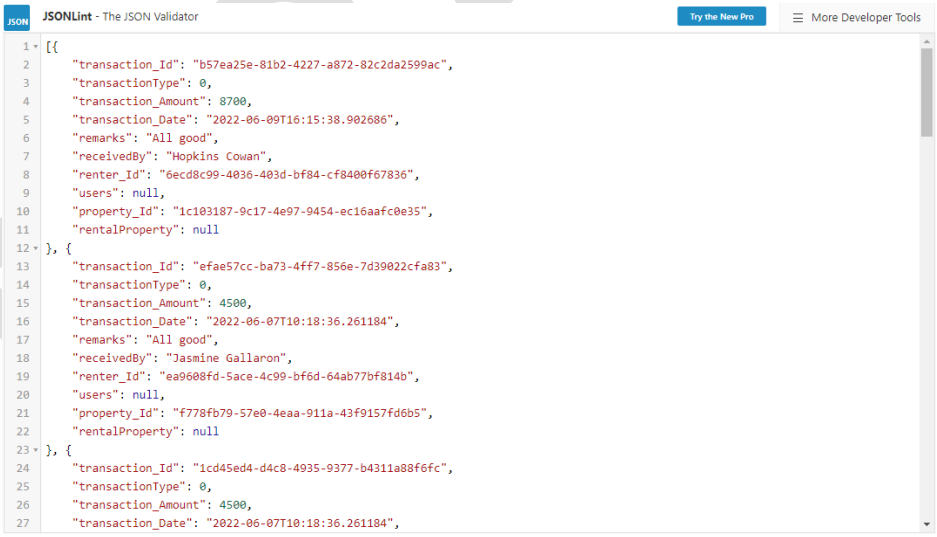


Figure 13.1.2: GET Transactions Test – Result

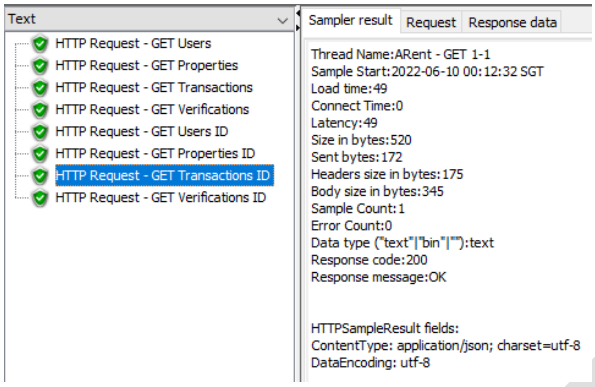


Figure 13.2.1: GET Transaction ID Test

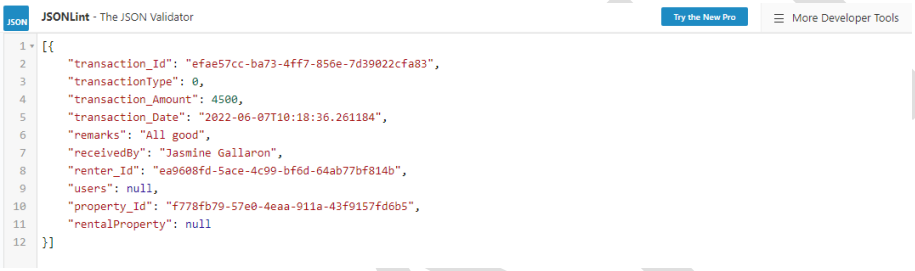


Figure 13.2.2: GET Transaction ID Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
GET Transactions	Getting the Transactions by putting the address “~/api/transaction” in the URL path	Display in the test that all Transactions will be returned by what is requested	Figure 13.1.2 displays the data result of the Transactions
GET Transaction ID	Getting the Transaction by putting specified ID in the address “~/api/transaction/{id}” in the URL path	Display in the test that it is passed, and the Transaction ID will be returned by what is requested of its specified ID	Figure 13.2.2 displays the data result of the Transaction of its respected ID which results to be passed

- **HTTP POST**

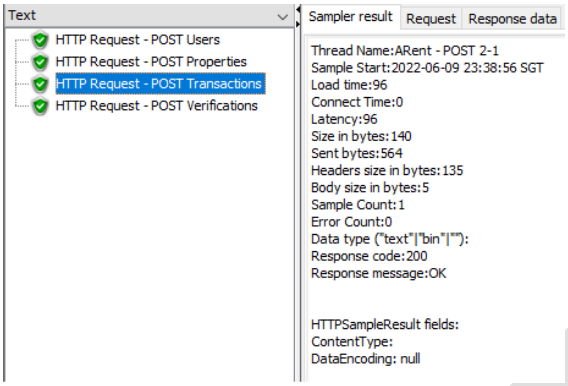


Figure 13.3.1: POST Transaction Test

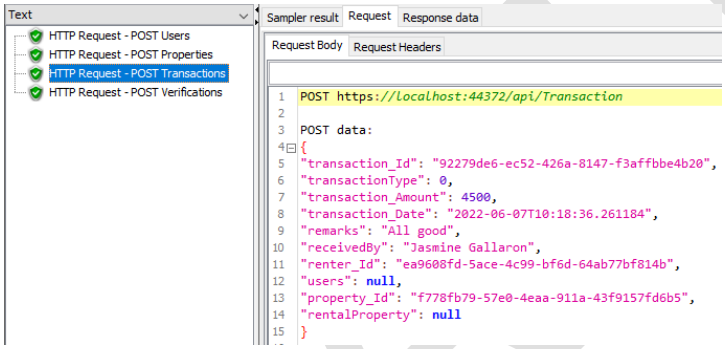


Figure 13.3.2: POST Transaction Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
POST Transaction	Adding a Transaction in the JSON format of its respected data to its URL path "~/api/Transaction"	The Transaction has been added in the API and passed through the database and it results to passed in the test	Figure 13.3.2 shows the result that has been passed and added to the Transaction records

- **HTTP PUT**

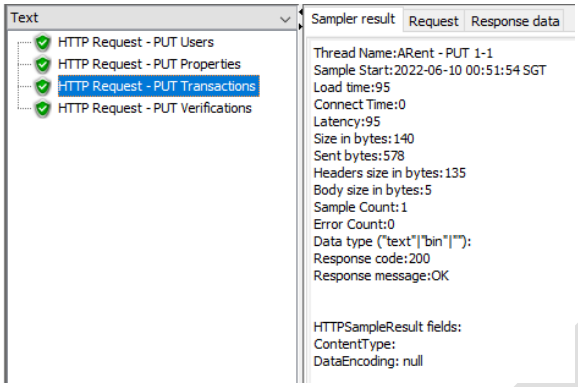


Figure 13.4.1: PUT Transaction Test

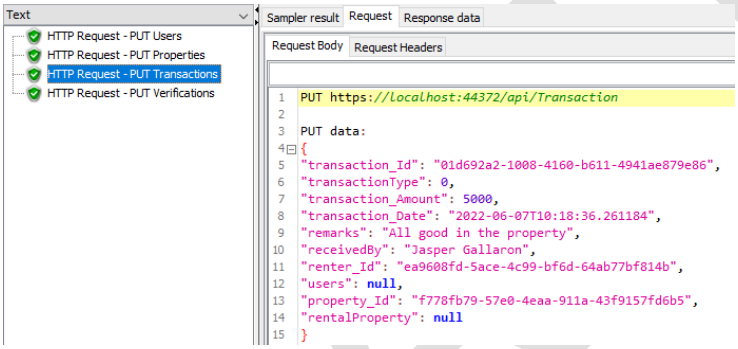


Figure 13.4.2: PUT Transaction Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
PUT Transaction	Edit the data of a specified Transaction that is added in the URL Path “~/api/Transaction” using the “Transaction_Id”	Successfully edited of a certain Transaction that corresponds to the ID that has been specified	Figure 13.4.2 shown the data result of a certain Transaction that is edited which results to passed

- **HTTP DELETE**

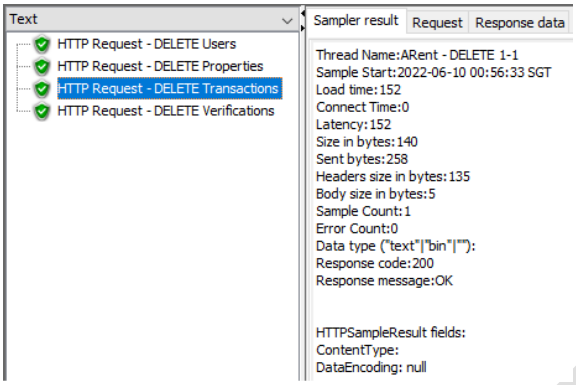


Figure 13.5.1: DELETE Transaction Test

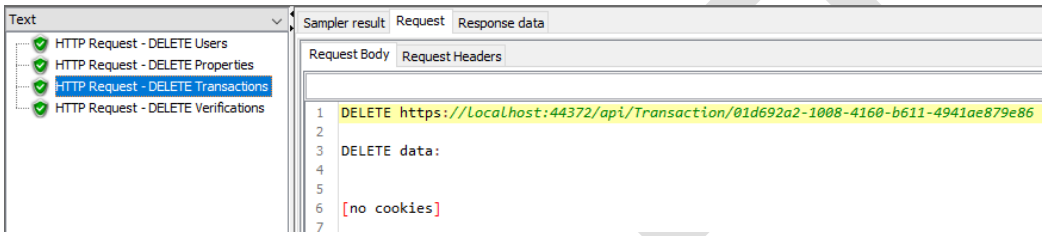


Figure 13.5.2: DELETE Transaction Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
DELETE Transaction	Delete a data of a specified Transaction that is added in the URL Path "~/api/Transaction/{id}" which corresponds to Transaction ID	Successfully deleted of a certain Transaction that corresponds to the ID that has been specified	Figure 13.5.2 shown the data result of a certain Transaction that is deleted which results to passed

Test Case 4: Verification

- **HTTP GET**

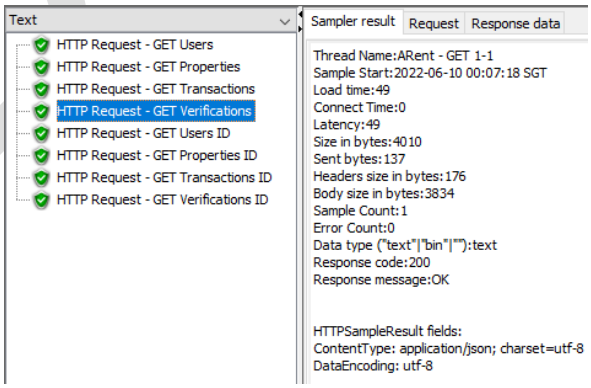


Figure 14.1.1: GET Verifications Test

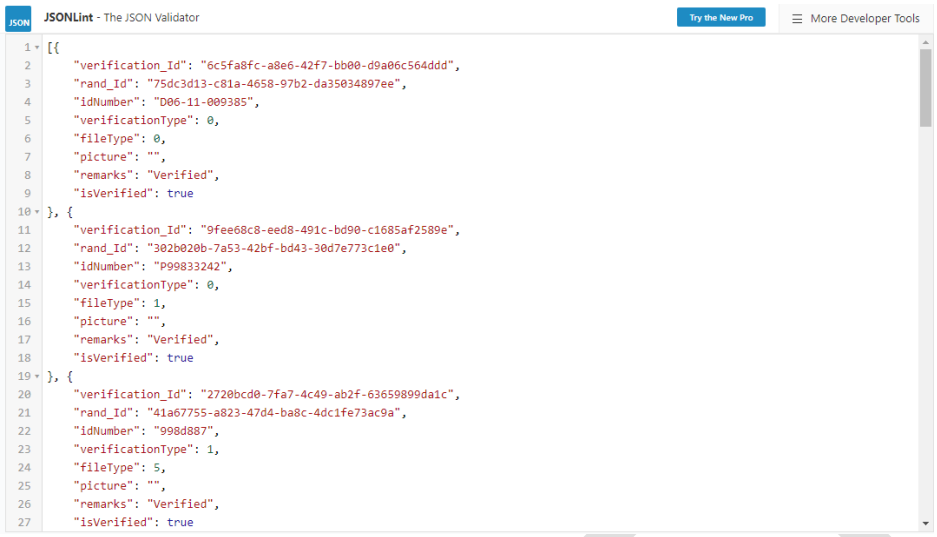


Figure 14.1.2: GET Verifications Test – Result

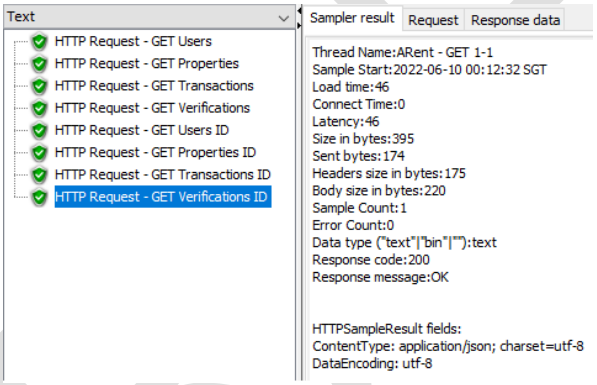


Figure 14.2.1: GET Verification ID Test

Figure 14.2.2: GET Verification ID Test – Result

Test Module	Test Scenario	Expected Result	Actual Result

GET Verifications	Getting the Verifications by putting the address “~/api/verifications” in the URL path	Display in the test that all Verifications will be returned by what is requested	Figure 14.1.2 displays the data result of the Verifications
GET Verification ID	Getting the Verification by putting specified ID in the address “~/api/verifications/{id}” in the URL path	Display in the test that it is passed, and the Verification ID will be returned by what is requested of its specified ID	Figure 14.2.2 displays the data result of the Verification of its respected ID which results to be passed

- **HTTP POST**

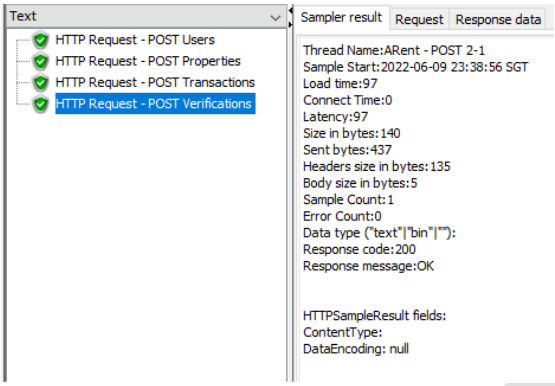


Figure 14.3.1: POST Verification Test

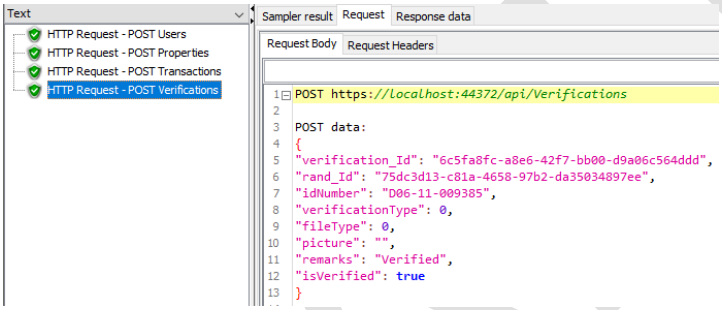


Figure 14.3.2: POST Transaction Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
POST Verification	Adding a Verification in the JSON format of its respected data to its URL path "~/api/Verifications"	The Verification has been added in the API and passed through the database and it results to passed in the test	Figure 14.3.2 shows the result that has been passed and added to the Verification records

- **HTTP PUT**

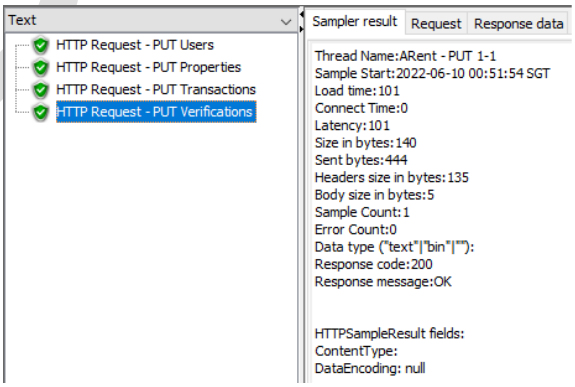


Figure 14.4.1: PUT Verification Test

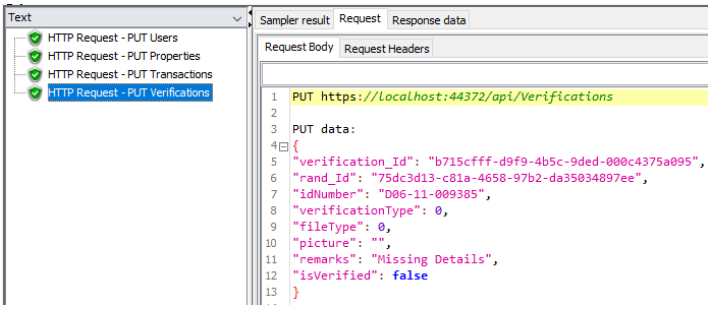


Figure 14.4.2: PUT Verification Test – Result

Test Module	Test Scenario	Expected Result	Actual Result
PUT Verification	Edit the data of a specified Verification that is added in the URL Path "~/api/Verifications" using the "Verification_Id"	Successfully edited of a certain Verification that corresponds to the ID that has been specified	Figure 14.4.2 shown the data result of a certain that is edited Verification which results to passed

- **HTTP DELETE**

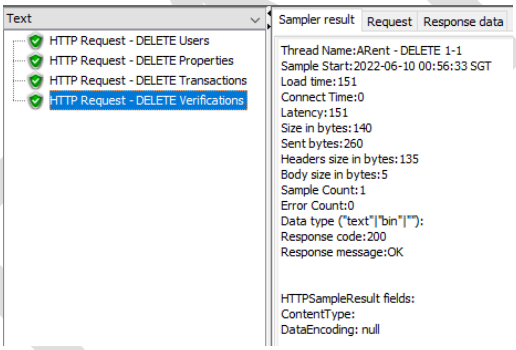


Figure 14.5.1: DELETE Verification Test

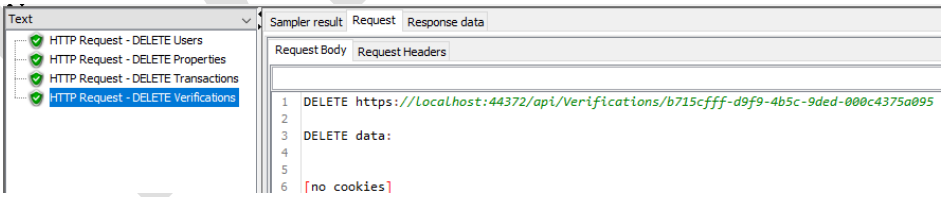


Figure 14.5.2: DELETE Verification Test – Result

Test Module	Test Scenario	Expected Result	Actual Result

DELETE Verification	Delete a data of a specified Verification that is added in the URL Path “~/api/Verifications/{id}” which corresponds to Verification ID	Successfully deleted of a certain Verification that corresponds to the ID that has been specified	Figure 14.5.2 shown the data result of a certain Verification that is deleted which results to passed

ARent Project Application Testing:

Test Case 1: Rental Searcher

Test Module	Test Scenario	Expected Result	Actual Result
Search Rental Property	Search a value that can be a name, or a category specified of a certain data in the table	Displays the searched name that is input into search field and immediately displays current property searched	Displays the searched name that is input into search field and immediately displays current property searched
Launch Maps Navigation	Pressing the map button for accessing the google maps	Displays the google maps in the screen	Displays the google maps in the screen
Launch AR Navigation	Pressing the map button and once the property is found it will launch the AR Navigation route	Display the arrow sign to where it is located for the navigation route of a certain property	Display the arrow sign to where it is located for the navigation route of a certain property
Contact User	Pressing a property that will have a chatting system for a certain owner	Can chat with the owner with corresponds to the details of the owner’s property	Can chat with the owner with corresponds to the details of the owner’s property
Display Property Information	Displaying the property information after logging-in	Displays the properties in the system of its application	Displays the properties in the system of its application

Test Case 2: Rental Owner

Test Module	Test Scenario	Expected Result	Actual Result

Search Rental Property	Search a value that can be a name, or a category specified of a certain data in the table	Displays the searched name, or a category that is input into search field and immediately displays the data in the table	Displays the searched name, or a category that is input into search field and immediately displays the data in the table
Manage Property	Add a property, after edit a certain property then deleting a property	Can modify in adding the property, editing the property, and deleting a certain property	Can modify in adding the property, editing the property, and deleting a certain property
Display Property Information	Wanted to display in the application the property details when selecting a certain property	Displays the certain property details of its page	Displays the certain property details of its page
Manage Transactions	Deleting a certain transaction or editing a certain transaction	Can modify in deleting a certain transaction, and editing a certain transaction	Transaction can't be deleted but can be edited only for references

Test Case 3: Admin

Test Module	Test Scenario	Expected Result	Actual Result
Manage Property	Add a property, after edit a certain property then deleting a property	Can modify in adding the property, editing the property, and deleting a certain property	Can modify in adding the property, editing the property, and deleting a certain property
Manage Users	Add a user, edit a certain user, then deleting a user	Can modify in adding the user, editing the user, and deleting a certain user	Can modify in adding the user, editing the user and deleting a certain user

Verify Rental Property and User	Verify a certain rental property and user that uses OCR for complete verification	Can make the property or user verified with the use of the edit button to make them verified or not verified	No integration for the verification due to some limitations of the OCR and its verification status

CHAPTER IV

SUMMARY, CONCLUSION, AND RECOMMENDATIONS

This chapter describes the issues and conclusions encountered by the project's creators during its development, which addresses hardships and how project plans and methods were adjusted.

SUMMARY OF FINDINGS

Initial plans for this project's Augmented Reality component involved using the Unity Game Engine and Mapbox SDK. Mapbox SDK has World-scale AR utilizing Unity's AR Interface and location services to superimpose 3D maps and position data using latitude and longitude over the AR camera feed. However, there was a conflict between Mapbox SDK version 2.1.1 and Unity 2020.3.11 because Mapbox will only allow imports into Unity if all AR components are removed, defeating the point of using Mapbox for the project. The researchers attempted to import the Mapbox SDK into Unity 2018.4.36, which successfully imports the data without deleting the AR components of Mapbox. However, it failed to run because ARCore has packages only available on the updated version of Unity. The researchers' advisor suggested that it is preferable to use an updated version of the software to avoid conflicts when integrating AR into the mobile and web components of the application. Due to version-related compatibility issues with ARCore, Unity, and Mapbox, the researchers decided to switch to ARCore flutter. The researchers successfully inserted shapes into the real-world setting, and while further studying flutter, in May 2022, Google released ARCore's Geospatial API. When creating a location-based Augmented Reality application, researchers have a few options: Use built-in sensors on users' phones, such as the GPS and compass; Use

ARCore Cloud Anchors API or the Geospatial API to create spatial anchors. It gets a global scale with GPS but with lower accuracy and variable reliability. Because it explicitly needs to map space, the Cloud Anchors API provides high accuracy within a limited scale. It also gets the best of both worlds with the Geospatial API, which allows it to place content anywhere in the world while providing near-ubiquitous, accurate localization and heading without manually mapping the space. Thanks to its decade of ground-level imagery, Google has mapped the world in minute detail. Collaborating with Location Services can take a reasonable GPS estimate and make it great and ready for AR.

The ARCore Geospatial API provides a developer guide for Android (Java/Kotlin), Android NDK, and Unity (AR Foundation). Kotlin featured a video tutorial of the processes for developing an app that shows geographical data and inserts content in actual places using the API, something Android NDK and Unity lacked. Kotlin's information aided the researchers in analyzing the API and implementing the required functionality for the project given the time. An anchor was placed in the real world using the property's latitude, longitude, and altitude. This good API can superimpose anchors to an accurate location worldwide without being there. Unlike the previous ways of superimposing 3D assets in the real world, it is dependent on surface detection.

Optical Character Recognition has issues regarding the version needed in the application's framework. Some OCR integration works, but it results in an error that the file path is unrecognized or that the application cannot support its current version.

Because the project application is still using.NET Framework 5.0, some features of the current framework are not working due to the new version of the framework, which was last updated on November 8, 2021, and there will be conflicts if researchers upgrade to the latest version.

CONCLUSION

The study has achieved its goal of providing a 360-degree panoramic view of the place and recommending properties based on the user's location, type of property, price, and amenities using a custom-made algorithm. Researchers utilize APIs for backend data like maps, polygons, and geospatial using swagger and the Google cloud platform. The app connected rental landlords and renters. The app's online component maximizes Azure cloud services.

The researchers encountered difficulty implementing Augmented Reality into the application, primarily since the technology is still in development and has early access recently this May 12, 2022. In addition, Kotlin is the primary programming language for Augmented Reality with which the researchers are unfamiliar. Augmented Reality was integrated into the mobile component and consumed the application's web services. However, it could only superimpose a location pin on a property and could not place multiple anchors in the real world. As a result of application testing, ninety percent of the application's functionalities have been built and are operational. The program is compatible with all Android devices and may access via a website with a responsive screen size.

RECOMMENDATIONS

The researchers have devised a list of additional enhancements to improve the functionality and capabilities of ARent.

For future research on location-based Augmented Reality, utilize the recently released ARCore's Geospatial API to maximize its capabilities. It provides global-scale coverage, precise location, orientation, remote placement, and guidance at global localization. For an optimized augmented reality experience, Android phones must be running Android 7.0 (Nougat) or later. ARCore's design works best on these phones with its motion tracking, environmental understanding, and light estimation capabilities.

For better experience and features in Web applications, there needed to be a.NET 6.0 version of its framework to make the other features more functional.

CHAPTER V

BIBLIOGRAPHY

[1] Blender Foundation. 2020. About — blender.org. blender.org. Retrieved from https://www.blender.org/about/

[2] easy. 2020. RealAR® Walkthrough property plans as easy as physical spaces. RealAR® Walkthrough property plans as easy as physical spaces. from https://www.realar.com/

[3] 2020. Augmented Reality: Real Estate App Concept | SuperSuper — CX research, design and consulting for premium brands. Supersuperagency.com. Retrieved from https://www.supersuperagency.com/blog/augmented-reality-real-estate-app-concept

[4] 2020. magicplan | Construction & Floor Plan App For Contractors. Magicplan.app. Retrieved from https://www.magicplan.app/

[5] 2017. Street Peek and Sign Snap: Use Augmented Reality and Image Recognition to Find Your Home - Home Made Blog. Home Made Blog. Retrieved from https://www.realtor.com/homemade/streek-peek-and-sign-snap-use-augmented-reality-and-image-recognition-to-find-your-home/

[6] 2022. Docker Desktop overview. Docker Documentation. Retrieved from https://docs.docker.com/desktop/

[7] Wikipedia Contributors. 2022. Debian. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Debian
[8] Wikipedia Contributors. 2022. Ubuntu. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Ubuntu
[9] 2022. Android Debug Bridge (adb) | Android Developers. Android Developers. Retrieved from https://developer.android.com/studio/command-line/adb#howadbworks
[10] Cynthia Harvey. 2021. 11 Best Android IDEs for Developers of 2022 | Developer.com. Developer.com. Retrieved from https://www.developer.com/mobile/top-android-ides-fordevelopers/#:~:text=Android%20Studio%3A%20Google's%20official%20IDE, IDE%20for%20Java%20desktop%20apps!
[11] brbrot brbrot. 2022. Visual Studio 2022 version 17.2 Release Notes | Microsoft Docs. Visual Studio 2022 version 17.2 Release Notes | Microsoft Docs. Retrieved from https://docs.microsoft.com/en-us/visualstudio/releases/2022/release-notes
[12] Gaël Thomas. 2019. What is Flutter and Why You Should Learn it in 2020. freeCodeCamp.org. Retrieved from https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/
[13] andrew fulmer. 2021. What is Flutter in a Nutshell? - Surf. Surf. Retrieved from https://surf.dev/what-you-should-know-about-flutter/
[14] getx.site. 2019. get | Flutter Package. Dart packages. Retrieved from https://pub.dev/packages/get
[15] Rick-Anderson. 2022. Overview of ASP.NET Core. Microsoft.com. Retrieved from https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0
[16] 2022. Overview of ARCore and supported development environments | Google Developers. *Google Developers*. from https://developers.google.com/ar/develop
[17] 2022. Git. Git-scm.com. from https://git-scm.com/
[18] Vigneshwaran. 2022. Calculate Haversine Distance in Java. Blogspot.com. Retrieved from https://reflectvicky.blogspot.com/2013/04/calculate-haversine-distance-in java.html
[19] 2022. Fatih Karabiber. Jaccard Similarity. Retrieved from https://www.learndatasci.com/glossary/jaccardsimilarity/#:~:text=The%20Jaccard%20similarity%20is%20calculated,the%20union%20of%20two%20sets.
[20] Sahil Taneja, Mani Karthik, Mohit Shukla, and Hitesh Sharma. 2017. AirBits: A Web Application Development Using Microsoft Azure. Retrieved from https://socrd.org/wp-content/uploads/2017/12/12.-AirBits-A-Web-Application-Development-Using-Microsoft-Azure.pdf

[21] Mark Massé. 2012. REST API Design Rulebook. O'Reilly. Retrieved from https://pepa.holla.cz/wp-content/uploads/2016/01/REST-API-Design-Rulebook.pdf
[22] Suphakit Niwattanakul*, Jatsada Singthongchai, Ekkachai Naenudorn and Supachanun Wanapu. 2013. Using of Jaccard Coefficient for Keywords Similarity. Retrieved from https://www.researchgate.net/profile/Ekkachai-Naenudorn/publication/317248581_Using_of_Jaccard_Coefficient_for_Keywords_Similarity/links/592e560ba6fdcc89e759c6d0/Using-of-Jaccard-Coefficient-for-Keywords-Similarity.pdf
[23] 2021. Anchor | ARCore | Google Developers. Google Developers. Retrieved from https://developers.google.com/ar/reference/java/com/google/ar/core/Anchor
[24] Sam Sprigg. 2022. Google announces new ARCore Geospatial API for building Augmented Reality experiences in real-world locations in 87 countries | Auganix.org. Auganix.org. Retrieved from https://www.auganix.org/google-announces-new-arcore-geospatial-api-for-building-augmented-reality-experiences-in-real-world-locations-in-87-countries/
[25] 2022. Build global-scale, immersive, location-based AR experiences with the ARCore Geospatial API | Google Developers. Google Developers. Retrieved from https://developers.google.com/ar/develop/geospatial#android-kotlinjava
[26] 2016. Kotlin Help. Kotlin Help. Retrieved from https://kotlinlang.org/docs/faq.html
[27] 2022. Lime pilots Google's augmented reality technology to improve e-bike…. Lime Micromobility. Retrieved from https://www.li.me/blog/lime-pilots-googles-augmented-reality-technology-to-improve-e-bike-and-e-scooter-parking
[28] 2022. Bird VPS, powered by Google, Will Change Scooter Parking Forever. Bird · Enjoy the ride. Retrieved from https://www.bird.co/blog/bird-vps-powered-google-change-scooter-parking-forever/
[29] 2019. Weighted Scoring. ProductPlan.com. Retrieved from https://www.productplan.com/glossary/weighted-scoring/