At Ease

Software Requirements and Design Document

Group Members

Lucius Guthrie

Mark Koren

Jesse Parker

Change History

Revision #	Description	Author	Date
1	Document outline created	Mark K.	9/8/2015
2	Added Use case diagrams	Mark K.	9/8/2015
3	Added Activity diagrams	Jesse P.	9/9/2015
4	Added ER diagram	Jesse P.	9/10/2015
5	Added Purpose/Motivation and Scope	Lucius G.	9/14/2015
6	Added Class diagram	Jesse P.	9/14/2015
7	Added Functional Requirements	Mark K.	9/15/2015
8	Added Project Goals	Lucius G.	9/15/2015
9	Added Non-Functional Requirements	Mark K.	9/15/2015
10	Added Project Description	Lucius G.	9/15/2015
11	Edited Project Description and Goals	Lucius G.	10/1/2015
12	Edited All activity diagrams and descriptions	Jesse P.	10/1/2015
13	Edited Use Cases and Requirements	Mark K.	10/1/2015
14	Edited Use Case for Rent	Jesse P.	10/26/2015
15	Added Sequence diagrams for Messaging	Lucius G.	11/2/2015

16	Added Sequence diagrams for Payments and Class Diagrams.	Jesse P.	11/2/2015
17	Added Sequence diagrams for Maintenance	Mark K.	11/2/2015

Table of Contents

1. Introduction

1.1 Key Definitions

Manager - a user who is charged with overseeing all of the operations of a rented property
Tenant - a user who occupies land or property rented from a property manager
User - a person with one of two roles, tenant or manager, who uses the At Ease application
Maintenance Request/Work Order - a specific instruction submitted by a tenant user that outlines the details of a maintenance defect in their property that does not meet proper standards for the purpose of getting it fixed. **Note:** Work Order and Maintenance Request are used interchangeably throughout the document, but they refer to the same thing, defined above.
Property/Room - One living unit that is occupied by a tenant and managed by a manager
Floor - A group of rooms. While this is expected to be on a floor of a building, it may be defined differently by a manager.
Complex - A group of buildings.
Building - A group of floors.
System - A group of complexes. This is the highest level of grouping allowed.

1.2 Purpose and Motivation

The motivation for this project comes from the problems presented by renting a property from an Independent Rental Owner (IRO). Today, tenants face many unwanted problems that all mostly stem from poor communication between tenants and their property manager. Our app, At Ease, seeks to remedy these problems by attacking their root, and creating an environment where tenants can be at ease about the place they live.

1.3 Scope

The At Ease application is designed for the Google Android Platform. It is intended to be used by small to medium sized property managers who, despite their small size, still desire an efficient approach to manage and communicate to their tenants. Tenants who rent from property managers that communicate via At Ease will have access to the app on any android device. Their account will be connected to their manager's account, and they will have access to all the features provided to them by the At Ease application.

1.4 Project Goals

The overall goal of At Ease is to comfort both tenants and managers throughout the rental process. This is reached by providing them with an environment in which all of their needs pertaining to renting a property are handled in one intuitive and easy-to-use application. We want our app to be easy to maneuver and work at a high level.

One major goal of this project is effectively managing our time to successfully implement all the features that we desire. We think the three main features (rent, communication, and maintenance requests) are the minimum of what we need to create an app that could compete on the real market. Trying to add all of these features is a major risk to getting this project done and getting it done well. We plan to manage this by effectively collaborating on the major issues of the project, but also dividing up the smaller tasks to get done outside of our meeting times. We believe the app we are trying to create will stretch us to not only do things that we have maybe never done before, but also learn more about time management and effective work in the process.

2. Project Description

2.1 Tenant Portal

Each tenant will access the app through the specific lens, or portal, of a tenant user. This will make the app less cluttered by taking out the options of the app that only pertain to managers. With the proposed features we have right now, the tenant will have the following options: communicating to their specified property manager; initiating, cancelling and checking the status of maintenance requests; and paying their rent.

2.2 Property Manager Portal

Each property manager will access the app through the specific lens, or portal, of a manager user. Like the tenant portal, the property manager portal will take out the features only pertaining to tenants. With the proposed features we have right now, the property manager will have the following options: communicating to their tenants; managing and closing maintenance requests; and managing the rent payments from their tenants.

2.3 Communication Portal

One main feature of At Ease is the communication portal between tenants and their property manager. The communication feature will not be much different than your average messaging environment. We plan to use Sinch in-app instant messaging to be an outline for our communication portal.

2.4 Maintenance Request Priority Queue

Another main feature of At Ease is the maintenance request feature, which provides a simple way for tenants to submit maintenance requests to the property managers, and in turn a simple way for the property manager to check these maintenance requests and clear them from the queue if they are done.

2.5 Rent Payment

Rent payment will be a feature open solely to the tenant. We plan to use PayPal as a payment processor in this feature. Tenants will have either a credit/debit card or a bank account linked to their profile that they will use to pay the property manager for rent. The goal is to eventually implement Stripe and Google Wallet to this feature as well.

2.6 Rent Management

Rent management will be a feature open solely to the property manager. Here the property managers will be able to set rent for each of the properties they manage, and also see the status on whether each of their tenants have paid rent for the specific rental time period.

2.7 Explanation and Justification of Third Party Tools

Parse - Parse is a third-party back-end system that streamlines the use of online database storage. While there is a paid version, the free version should suffice for the scope of this project. Using Parse will allow us to spend more time on creating features for the app, while also making it more portable across different platforms. In addition, Parse has built-in User control, which will allow easy integration with Facebook and Google log-ins.
Sinch - Sinch is a third party messaging API that allows in-app communication between different users. Sinch is designed to be simple and easily imported into Android apps.
PayPal/Stripe/Google Wallet - These third-party programs will allow us to safely and securely process payments from a User. We will only implement one at first, but the goal is to allow the use of all three.

3. Functional Requirements

3.1 Priority Legend

The legend describing the different priority levels found throughout the functional and nonfunctional requirements sections is found below, in table 3.1.

Priority	Description	Chance of completion (by Dec.)
Essential	One of the main goals of the project. Unless further research shifts the priorities of the app, these will be completed.	100% > x > 95%
High	A major component of the current plan, almost all of these should be completed, and they are often complementary to the essential tasks.	95% > x >75%
Medium	While these tasks are important to complete, they are not critical to the completion of the project. Completion will be mainly based on the time needed to implement.	75% > x > 40%
Low	Tasks that would be nice to have, but are not important at all for the completion of the app.	40% > x >15%
Stretch	Great goals for the future, but unless the plan is changed drastically, they will almost certainly not be implemented by December.	15% > x >0%

Table 3.1 Functional Requirement Priority Legend

3.2 Rent

Requirement	Description	Priority
Edit Rent	Allow the manager to edit the rent for a tenant. This includes the collection frequency and dates.	Essential
Pay Rent	System for a tenant to pay the required rent to the manager.	Essential
Rent Reminders	User specified reminders for upcoming rent deadlines. A push notification would be good. Ideally, clicking the notification takes the user to the pay rent screen.	Medium
Request Rent	Managers can manually or automatically request late rent from a tenant.	Medium
Rent Penalty	Automatically assess a set penalty to late rent, if the option is enabled.	Low

Table 3.2 Rent Functional Requirements

3.3 Maintenance

Requirement	Description	Priority
Initiate Maintenance Request	A tenant will create a maintenance request, which will be sent to the manager's inbox.	Essential
Cancel Maintenance Request	Allow a tenant to cancel a previously issued maintenance request.	Essential
Close Maintenance Request	Allow a manager and tenant to jointly close a maintenance request. A manager first marks a maintenance request as complete. The tenant must then approve this action. Once both approve the action, the request is considered closed.	Essential
Update Maintenance Request Progress	The manager can update the status of the request, and post progress updates to keep the tenant informed.	Medium

Ask for Progress Update	Allow the tenant to request a progress update on a stale request.	Low
Attach Picture	Tenants can attach a picture to a maintenance request.	Low

Table 3.3 Maintenance Functional Requirements

3.4 Communication

Requirement	Description	Priority
Create an Inbox	Have an inbox containing the messages of a user.	
Manager->Tenant direct message	A direct message from the manager to the tenant.	Essential
Tenant -> Manager direct message	A direct message from the tenant to the manager.	Essential
Manager -> Group message	A group message from the manager to a room, floor, building, complex, or system.	Medium
Building bulletin board	A place where users can place messages and announcements visible to the whole floor, building, complex, or system. Ads here would be useful as well.	Stretch
Tenant -> Room "door note"	A system for tenants to leave anonymous messages visible to everyone in a room. The manager will have access to see who posted the message, to prevent abuse.	Stretch

Table 3.4 Communication Functional Requirements

3.5 Reviews

Requirement	Description	Priority
User Profiles	A profile page for managers and tenants. Could include	Stretch

	contact info, a description, and various other common social media profile components.	
Building Pages	A profile page for a building. It should include a description, pictures, and a location.	Stretch
Tenant Reviews	Reviews of a tenant, covering their rent reliability and how easy they are to rent to.	Stretch
Manager Reviews	Manager reviews, covering how easy they are to rent from.	Stretch
Building Reviews	Building reviews, covering quality, features, and livability.	Stretch
Building Amenities	A list of amenities available at the building.	Stretch

Table 3.5 Reviews Functional Requirements

3.6 Matching System

Requirement	Description	Priority
Searchable Buildings	Allow users to search the database of buildings by name, location, or amenities available.	Stretch
Roommate Match	Match tenants to other tenants who would be compatible roommates, based on a survey.	Stretch
Manager Match	Match tenants to a nearby manager or building, based on a survey.	Stretch

Table 3.6 Matching System Functional Requirements

4. Non-Functional Requirements

4.1 Rent

Requirement	Description	Priority
Financial Reliability	The payment system should be financially reliable, meaning that if there is a failure of some sort, no money is moved or lost.	Essential
Use Stripe	Integrate Stripe as a payment option.	Medium
Use PayPal	Integrate PayPal as a payment option.	High
Use Google Wallet	Integrate Google Wallet as a payment option.	Medium

Table 4.2 Rent Functional Requirements

4.2 Maintenance

Requirement	Description	Priority
Use Parse	Store data in Parse.	Essential
Maintenance Request Form	Have a set form for tenants to enter data into to create a maintenance request.	Essential
Customizable Form	Allow managers to create their own custom maintenance request form.	Stretch
Store Pictures	Have a storage system for the pictures uploaded with maintenance requests.	Low

Table 4.3 Maintenance Functional Requirements

4.3 Communication

Requirement	Description	Priority
Use Parse/Sinch	Use a combination of Parse and	Essential

	Sinch to store and send messages securely.	
Use Parse Users	Control user access using the Parse User feature.	Essential
Bulletin Board Recyclerview	Use a recyclerview to show the Bulletin Board, and design the layout to look like a physical one.	Low
Door Note Graphics	Design the graphics of the door note system to actually look like a whiteboard on a door.	Low
Load Inbox Dynamically	When loading the inbox, dynamically show messages as they load.	Medium
100+ Item Storage in Inbox	Keep up to 100 items in an inbox of a user.	High

Table 4.4 Communication Functional Requirements

4.4 Reviews

Requirement	Description	Priority
Compute an Average Rating	Compute an average rating from all the ratings given to a user or place.	Stretch
Store 1000 character reviews	Written reviews by a user should be able to store up to 1000 characters, or more.	Stretch

Table 4.5 Reviews Functional Requirements

4.5 Matching System

Requirement	Description	Priority
Advanced Search Menu	Have a standard, easy to use form to allow advanced searching of users and buildings.	Stretch

Table 4.6 Matching System Functional Requirements

5. UML Diagrams

Note: Due to the scope of this project, only requirements that have been marked as "essential" will be included in the diagrams found within this section.

5.1 Use Case Diagrams

5.1.1 Top-Level Use Case Diagram

The Top-Level use case diagram is shown in Figure 5.1. This shows a general overview of the features that Actors of the system can access. The core features will be the main focus, while the additional features and the stretch goal features will only be added if time permits, and the core features are fully working.

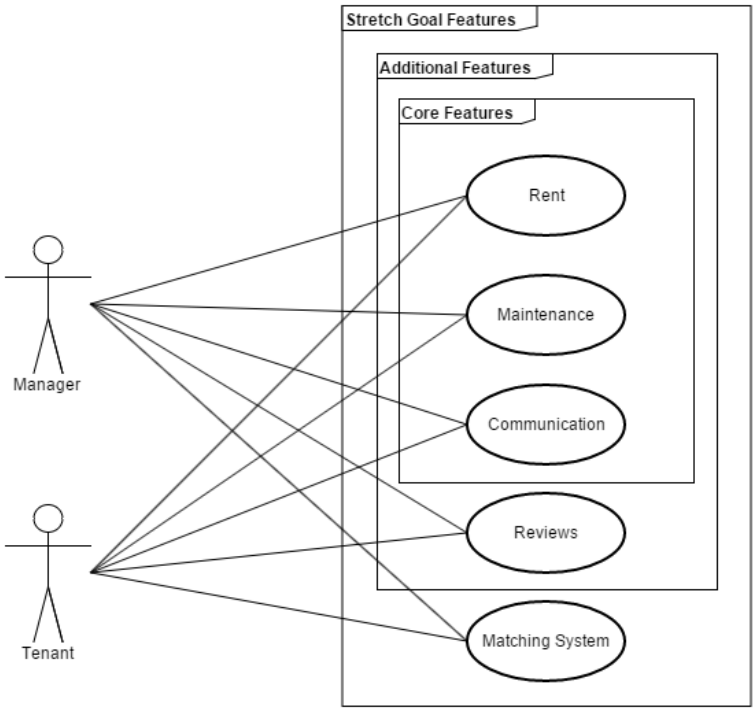

Figure 5.1

5.1.2 Rent Use Case Diagram

The rent use case is shown in Figure 5.2. A Tenant can Pay Rent and Check Rent. A Manager can Check Rent and Edit Rent. Pay Rent interacts with a third-party payment processing system, such as PayPal.

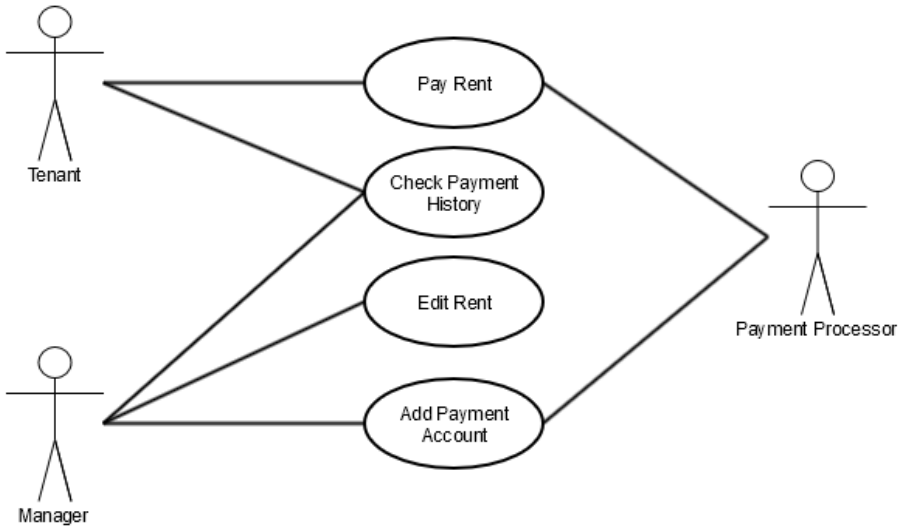

Figure 5.2

5.1.3 Maintenance Use Case Diagram

The maintenance use case is shown in Figure 5.3. A tenant can initiate requests and cancel requests that are not needed. Managers can Close completed requests, which will then allow tenants to jointly close the completed request.

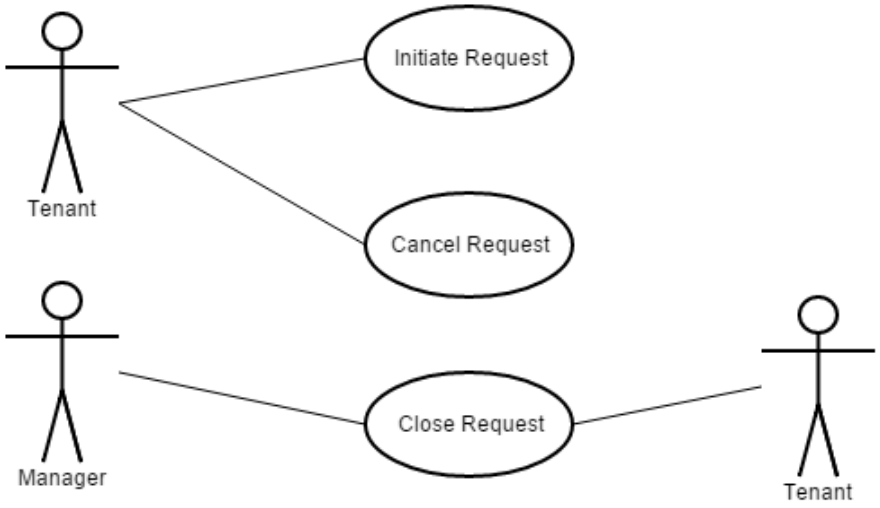

Figure 5.3

5.1.4 Communication Use Case Diagram

The communication use case diagram is shown in Figure 5.4. Both tenants and managers can send messages, receive messages, and archive messages.

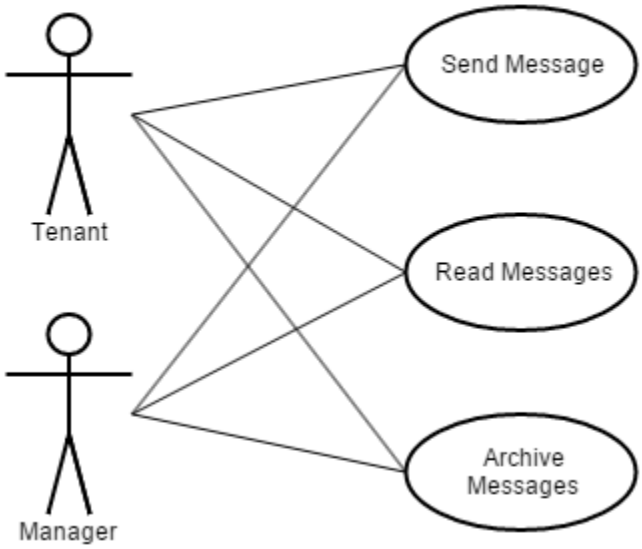

Figure 5.4

5.2 High Level Class Diagram

The high level class diagram is shown in Figure 5.5. The diagram shows some preliminary relationships between classes that will exist in our system. We have not included full implementation details in this version of the class diagram.

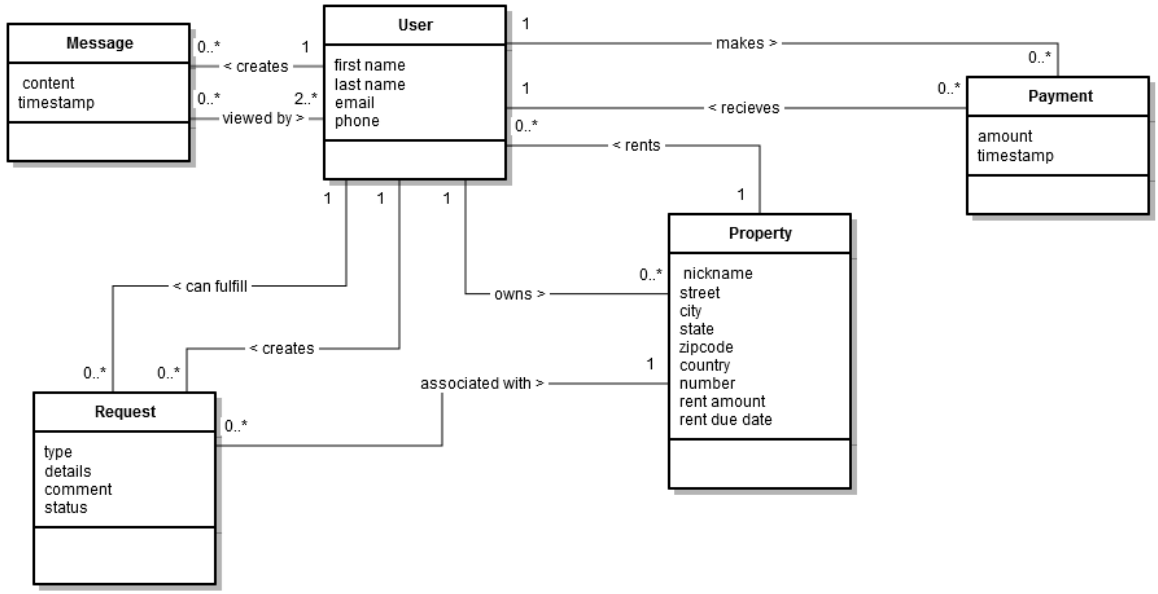

Figure 5.5

5.3 Activity Diagrams

5.3.1 Top-Level Activity Diagram

The top-level activity is shown in Figure 5.6. This activity outlines the way a User will navigate into the other activities. The User will also be identified here as either a Tenant or a Manager.

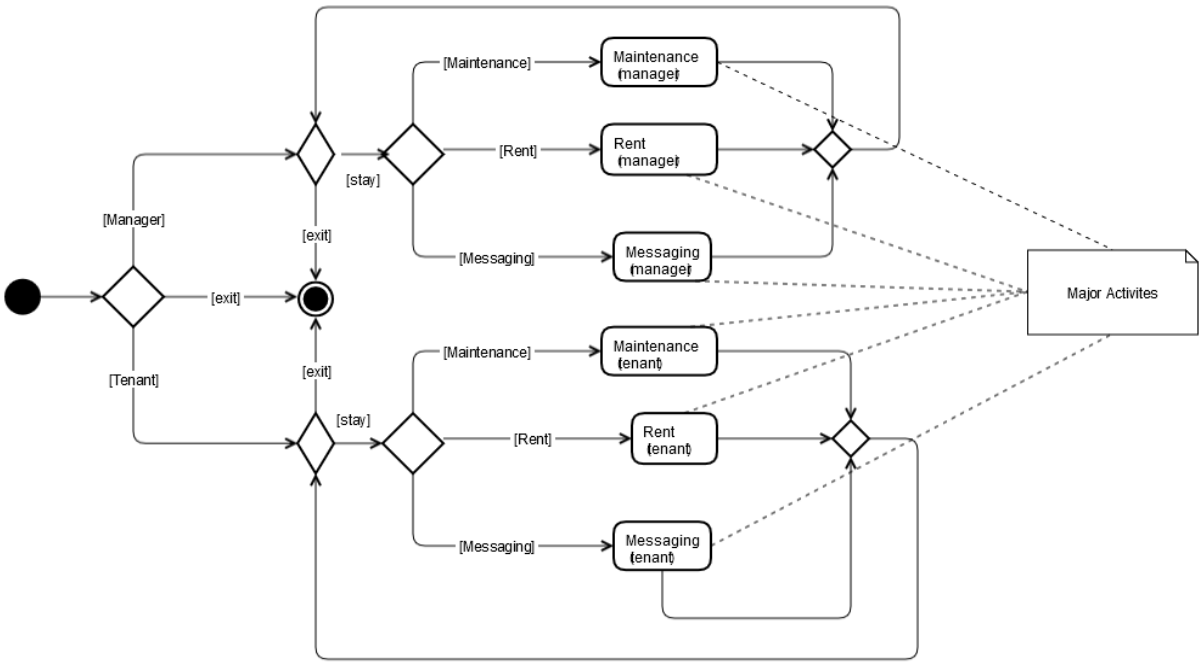

Figure 5.6

5.3.2 Maintenance (Manager) Activity Diagram

The Maintenance (Manager) activity is shown in Figure 5.7. This diagram outlines how a manager can view maintenance requests, mark them as done, or add a comment to a maintenance request.

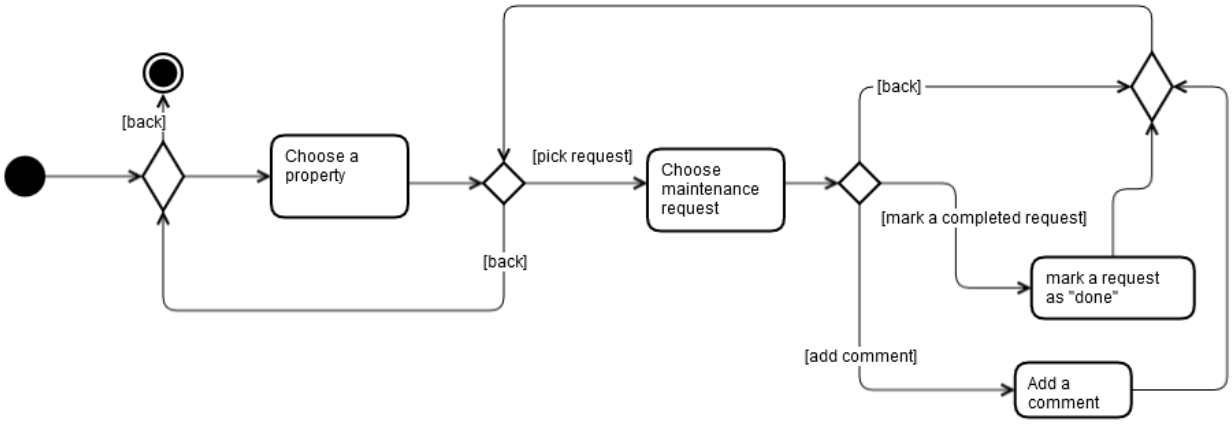

Figure 5.7

5.3.3 Rent (Manager) Activity Diagram

The Rent (Manager) activity is shown in Figure 5.8. This diagram outlines how a manager can set rent for a particular property, and view payment history of their tenants.

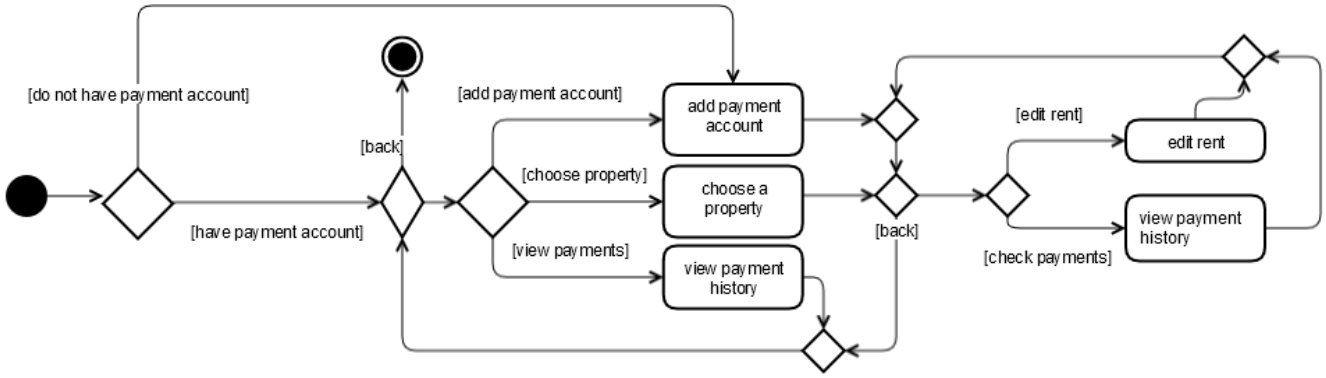

Figure 5.8

5.3.4 Messaging (Manager) Activity Diagram

The Messaging (Manager) activity is shown in figure 5.9. This outlines the process by which a Manager can read, delete, and send messages to a Tenant.

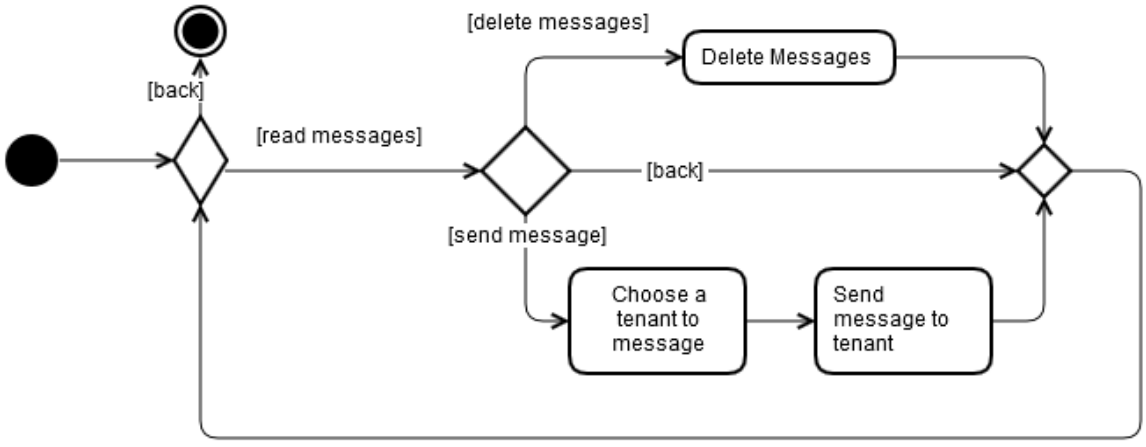

Figure 5.9

5.3.5 Maintenance (Tenant) Activity Diagram

The Maintenance (Tenant) activity is shown in Figure 5.10. This illustrates how a Tenant can view the different maintenance requests he has submitted, cancel or check the status of the request, initiate a new request, or accept a request as completed.

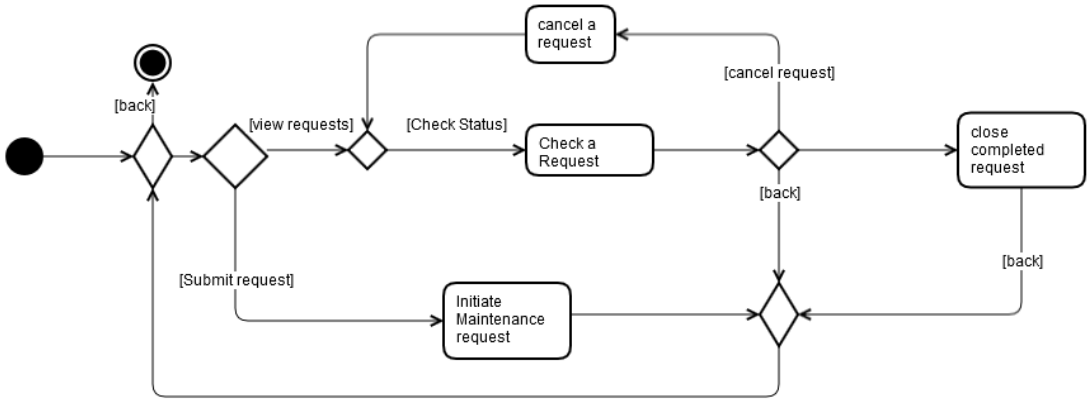

Figure 5.10

5.3.6 Rent (Tenant) Activity Diagram

The Rent (Tenant) activity is shown in Figure 5.11. It outlines how a tenant can pay for their rent, add a payment account, or check their payment history.

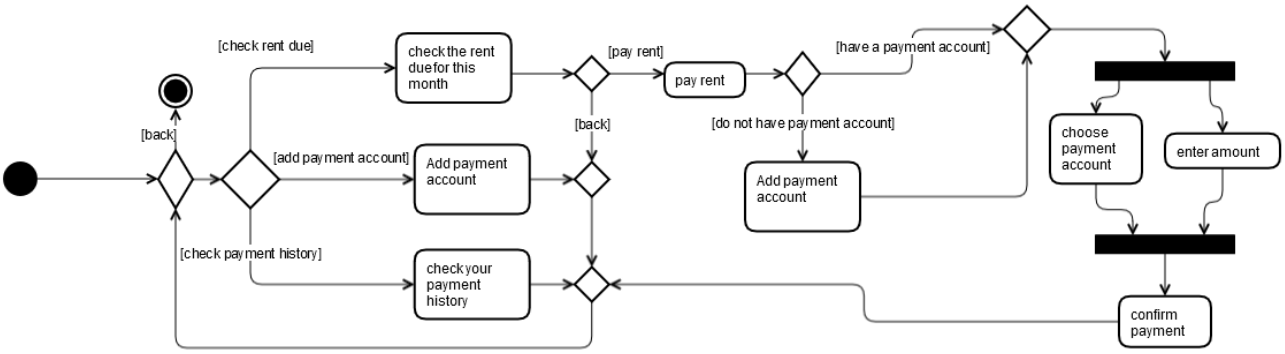

Figure 5.11

5.3.7 Messaging (Tenant) Activity Diagram

The Messaging (Tenant) activity is shown in Figure 5.12. The activity describes how a Tenant can read, delete, and send messages to a manager.

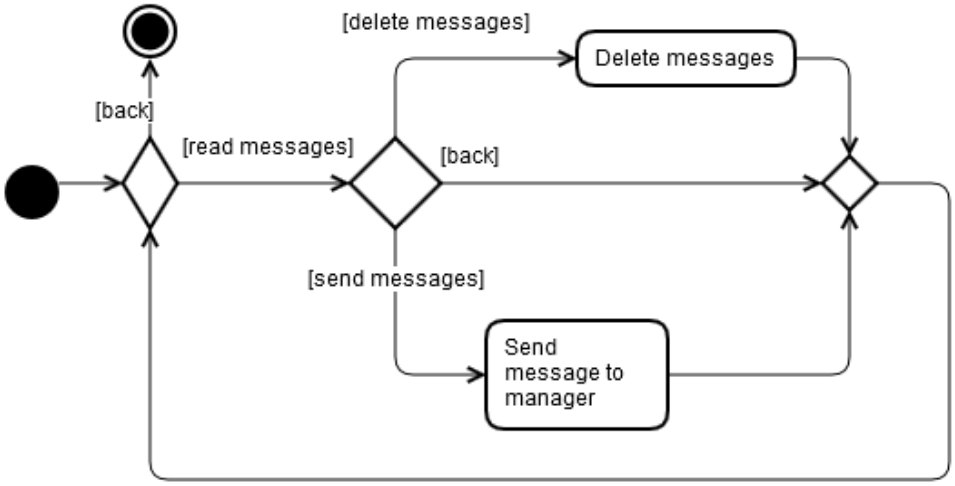

Figure 5.12

5.4 Sequence Diagrams

5.4.1 Pay Rent Sequence Diagram

The Tenant will open the Rent Pay Activity, then they will have to enter their credit card information as well as the amount they want to pay. When the Tenant is done, they will click the confirm button, which will start the process of making the Stripe.Card object with their card information. The card will be validated and a Stripe.charge object will be created. The charge object will hold the card object as well as the amount that the Tenant is paying. It will then process the charge and the payment will be stored in our database. The sequence diagram for paying rent is shown in Figure 5.13.

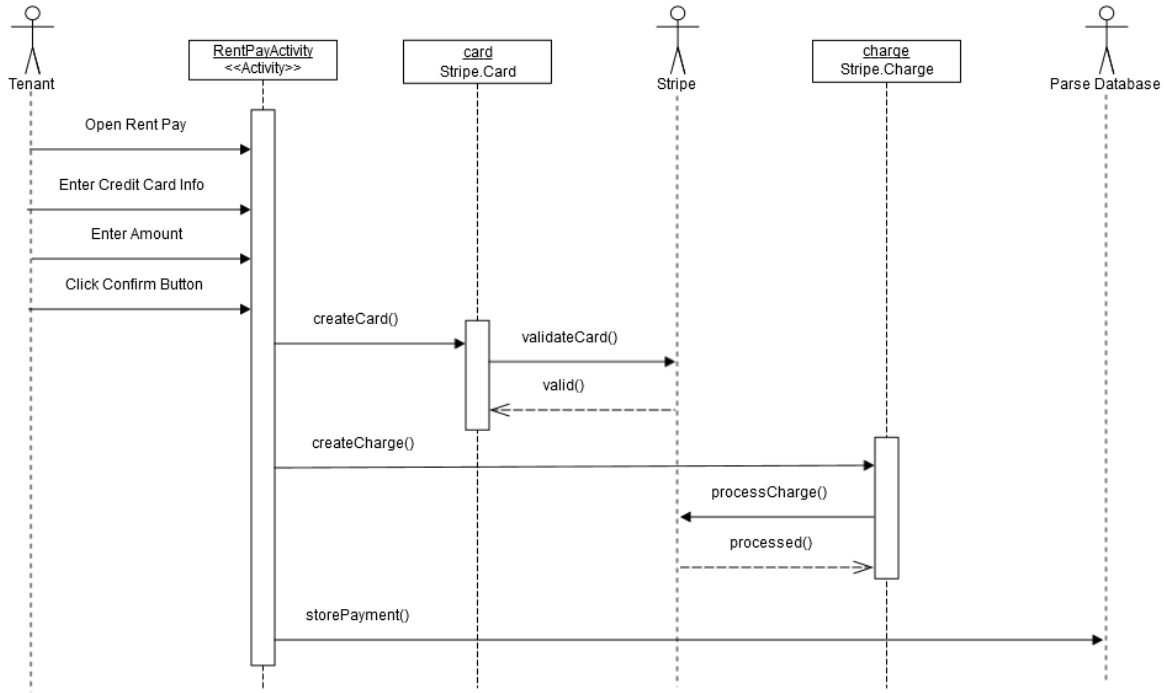

Figure 5.13

5.4.2 View Payment History Sequence Diagram

To View Payment History, the User (Tenant or Manager) will click the Payment History Activity. The activity will get the payments from the database and display them in a list format through the Payment History Recycler View. After the view is populated, all of the payments will be available for viewing. The sequence diagram for viewing payment history is shown in Figure 5.14.

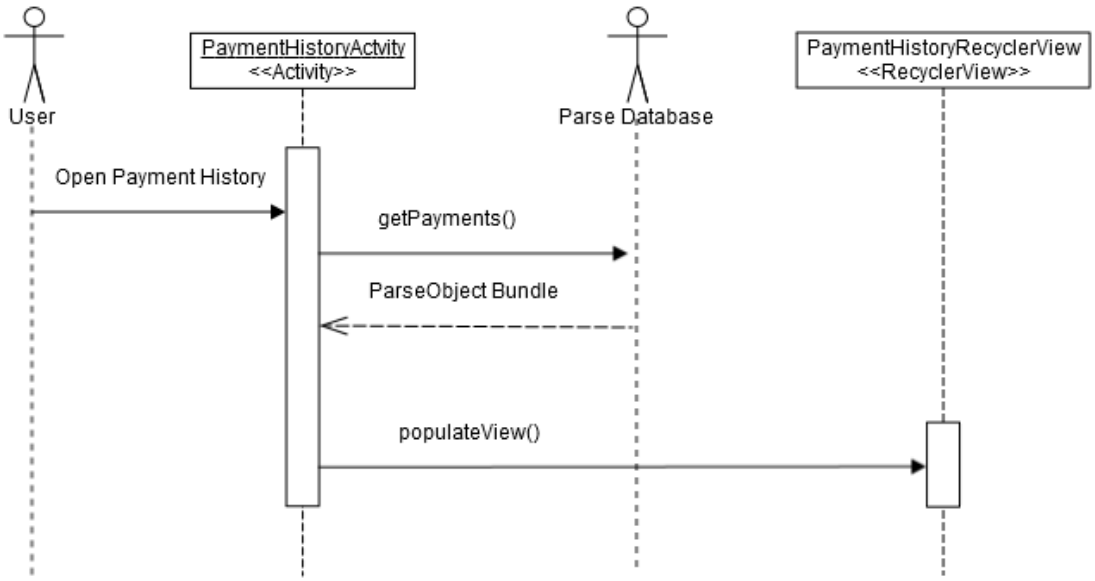

Figure 5.14

5.4.3 Edit Payment Sequence Diagram

To Edit Payment Settings, the manager will open the Edit Payment Settings Activity. This activity will have editable fields for the Manager to use to change their various settings. When the Manager is finished editing their various settings, they can click the save button to save their new settings. When the save button is pressed, the manager settings will be stored in the database and will be effective immediately. The sequence diagram for editing payment is shown in Figure 5.15.

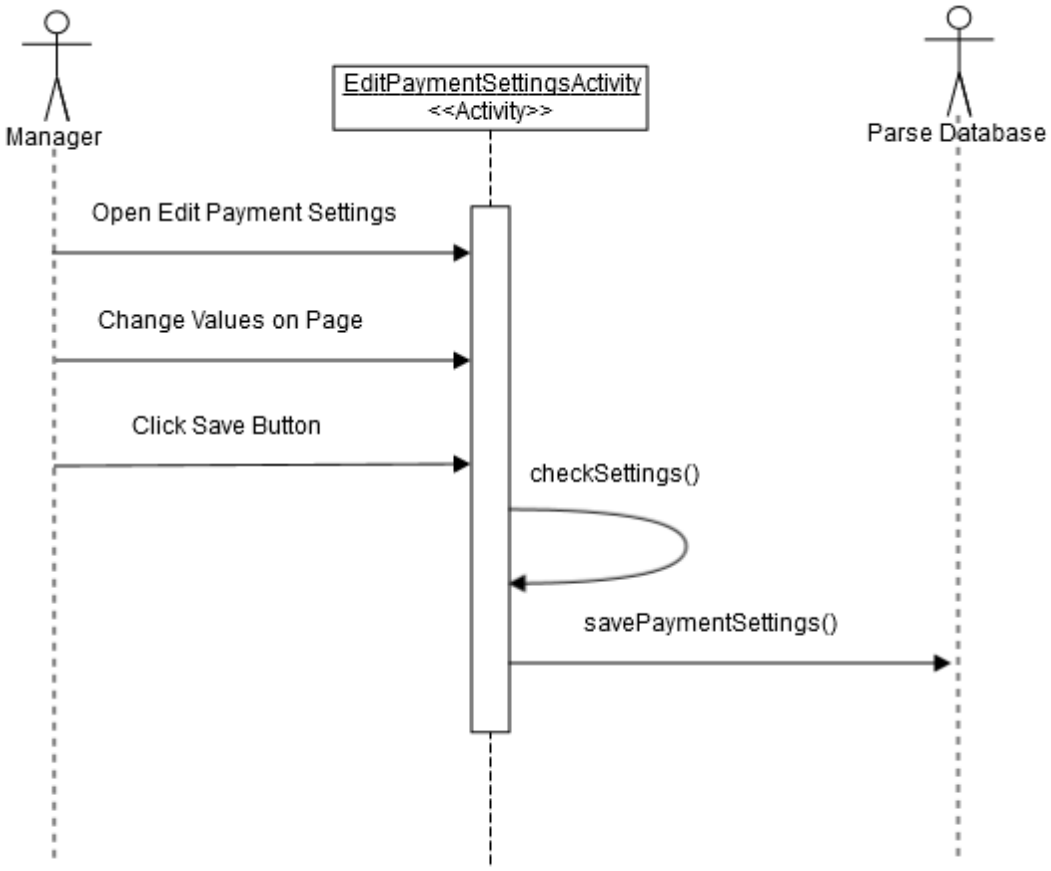

Figure 5.15

5.4.4 Add a Payment Account Sequence Diagram

To add a payment account, a Manager must open the Stripe Connect Activity. The Stripe Connect Activity will first set a new WebViewClient to the StripeWebClient. Once the StripeWebClient is set, the Stripe Connect Activity will load the URL for the Stripe Connect Login webpage into the webView that is displayed. The Manager must then enter their stripe information and successfully login to their stripe account. Once successfully logged in to their stripe account, the StripeWebClient will process the API response from Stripe and extract the authorization key. The client then sends an authorization request to Stripe with this authorization key. A response will come back from stripe containing the access Token information for the manager's stripe account. With this information At-Ease will be able to charge tenants on behalf of the manager and send the money to the manager's account. The access token information is stored in the database, and a payment account has been successfully added to At-Ease. The sequence diagram for adding a payment account is shown in Figure 5.16.

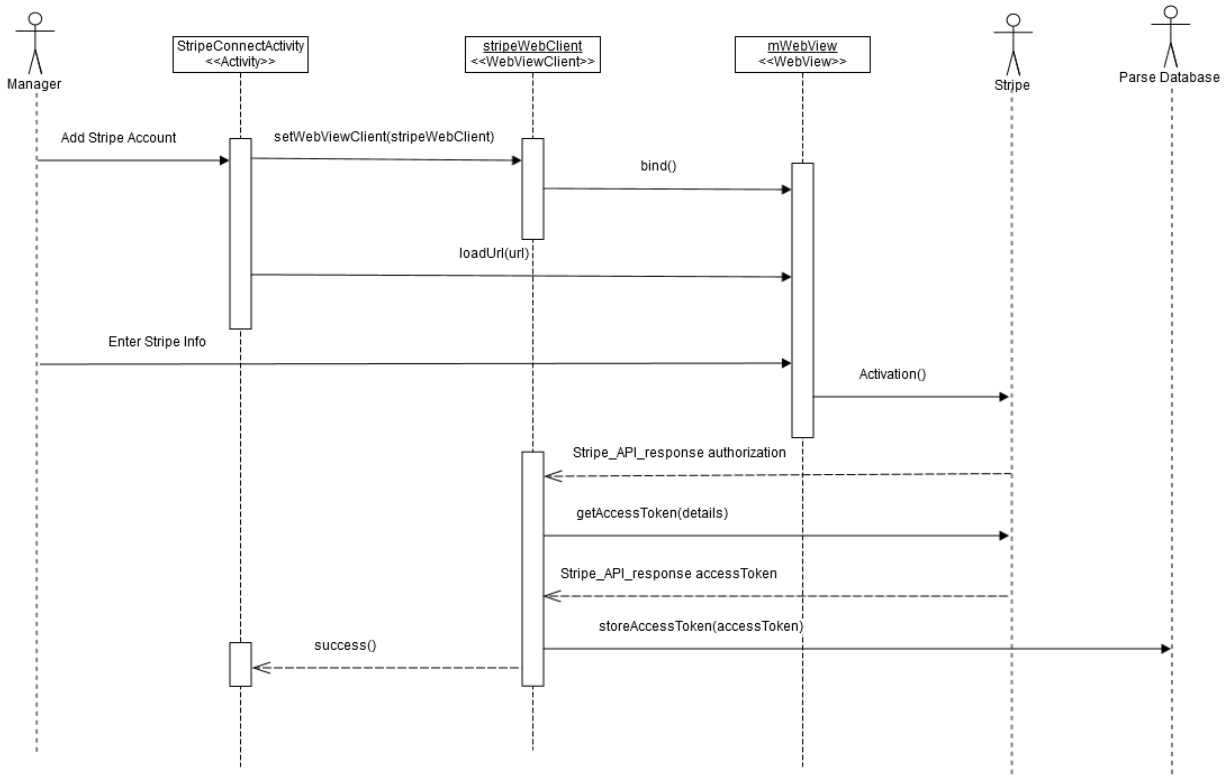

Figure 5.16

5.4.5 Send Message Sequence Diagram

To send a message, a user (both tenant and manager), opens the List User Activity, which will list all of the other AtEase users that they have access to communicate with. From the user perspective, it runs very similarly to an average messaging component. The user chooses the message thread of whom they wish to communicate, which opens the Message Activity. The Message Activity presents a screen very similar to a typical messaging component, with an area to type a message at the bottom of the screen. After the user types a message and sends it, the Message Service communicates with the Sinch client that adds the message to the message thread for both the sender and the receiver. The sequence diagram for sending a message is shown in Figure 5.17.

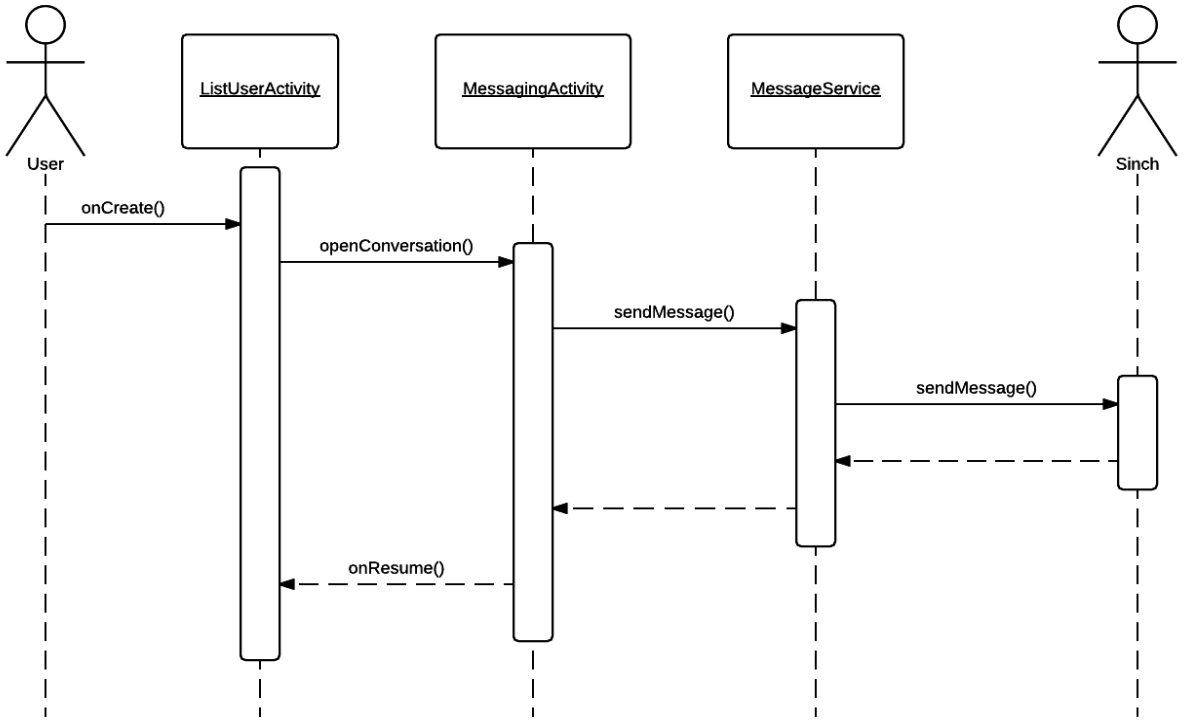

Figure 5.17

5.4.6 Read Message Sequence Diagram

Reading a message is very similar to sending a message, it just stops before the action of sending a message. The user chooses the message thread from List User Activity, and the Message Activity makes a call to the Parse database to get the history of all messages (read and unread) between the two users. The Message Adapter formats the messages into the Message Activity that makes it into a typical message thread (i.e. sender messages on the right, and receiver messages on the left). The sequence diagram for reading a message is shown in Figure 5.18.

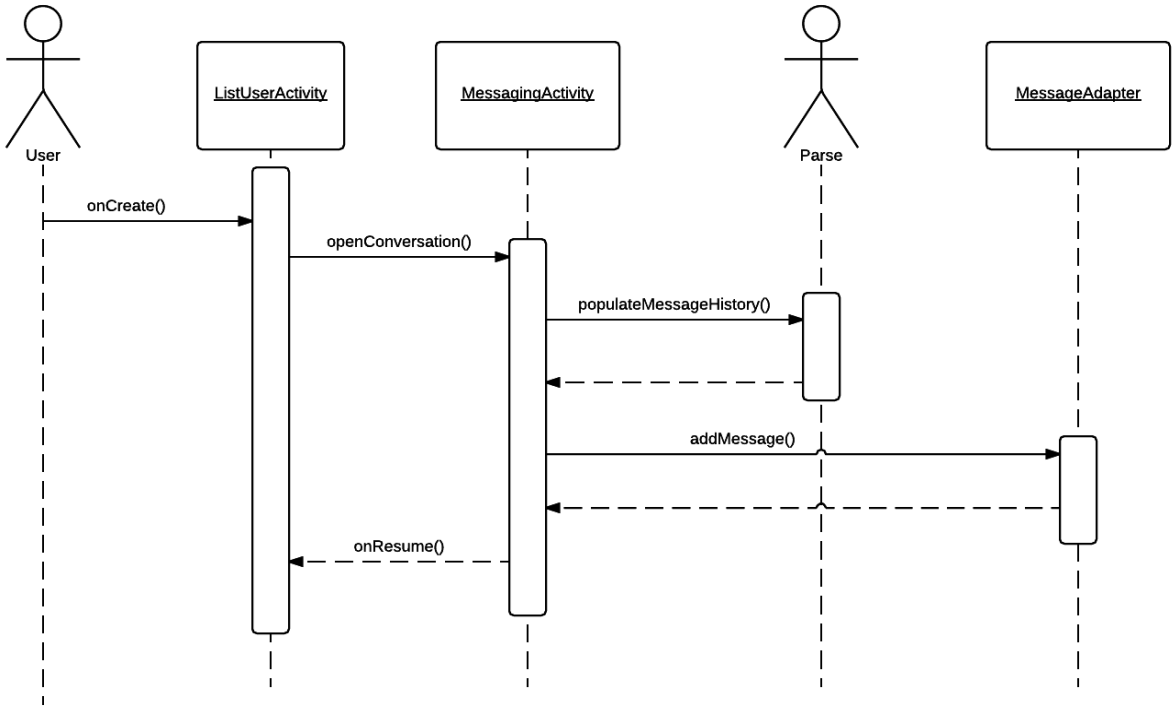

Figure 5.18

5.4.7 Initiate Maintenance Request Sequence Diagram

Figure 5.19 shows the sequence diagram for initiating a new maintenance request. This is a very simple workflow. The user first initiates a new work order, which creates a NewWorkOrderExpandableActivity. The new activity loads up with a screen for the user to enter data into. Once the user has entered all of the data, they may submit. The user is prompted to ensure that they want to take this action, and then the work order is saved to the Parse database. At this point, the activity is finished.

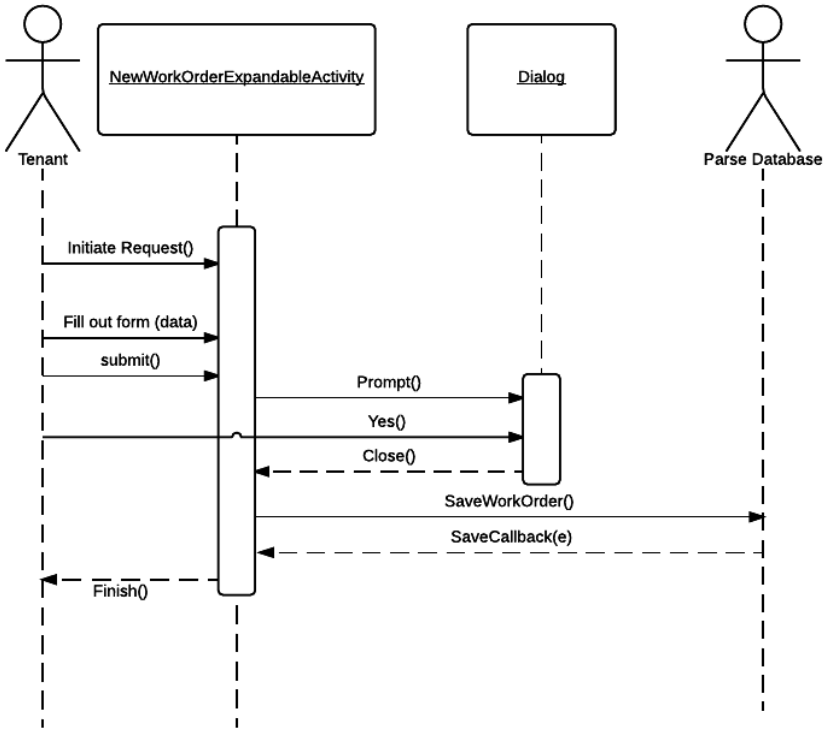

Figure 5.19

5.4.8 View Maintenance Request Sequence Diagram

The sequence diagram for the viewing of a Maintenance Request (Work Order) is shown in Figure 5.20. The first part of the sequence shows how the inbox is populated, which will be repeated in all of the other maintenance request sequence diagrams as well. When a user opens the inbox, the program searches the Parse database for all relevant work orders, and then displays them in a recyclerview. A maintenance request is relevant if the current user is either the manager or the tenant on said request. To view a maintenance request in detail, the user clicks on the respective item in the recyclerview. The work order is then passed to the ViewWorkOrderActivity. This activity is almost the same as the NewWorkOrderExpandableActivity, except all editable fields are now un-editable, so this form is just to display all of the data in a maintenance request. When the user is finished, they may close the form and leave the inbox, finishing all involved activities.

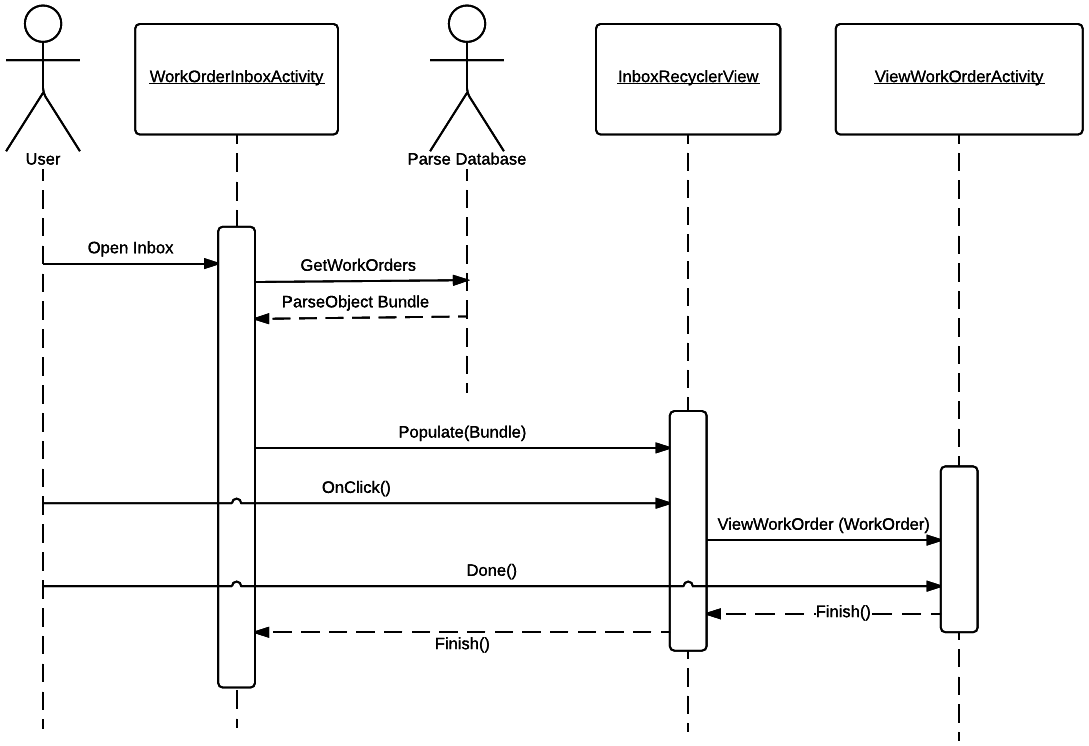

Figure 5.20

5.4.9 Close Maintenance Request Sequence Diagram

The sequence diagram for closing a maintenance request is shown in Figure 5.21. In this sequence, the inbox is first populated as previously stated. Then, the user, specifically a manager in this case, swipes an item in the list to the right. The manager is prompted to ensure he wants to do this, and then an alert is sent to the tenant who opened the request. If the tenant agrees that the maintenance request was resolved, then the work order is deleted.

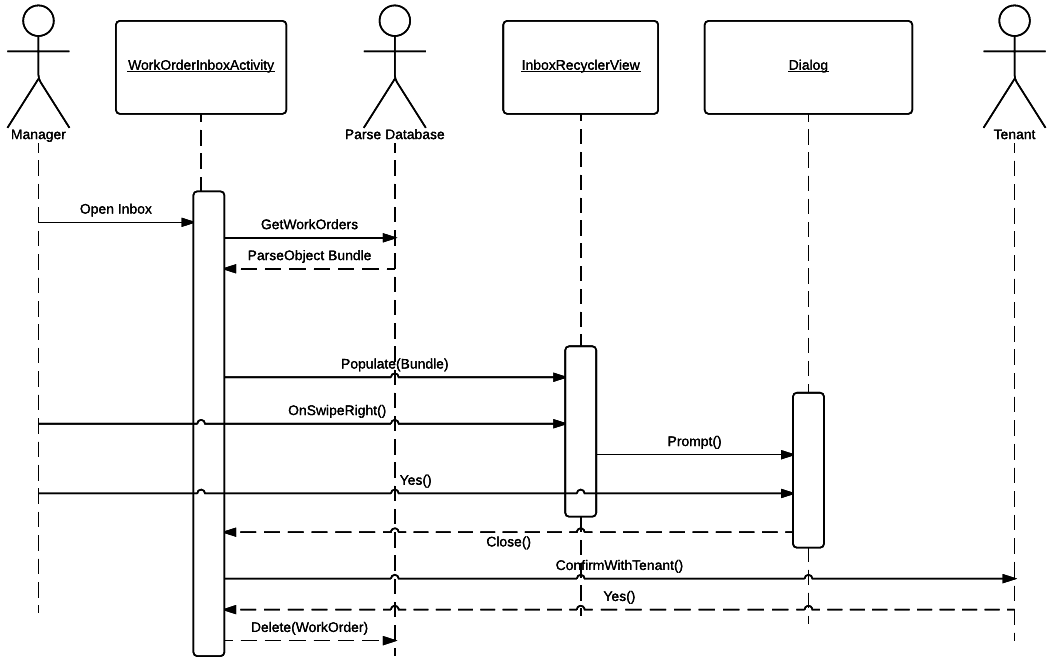

Figure 5.21

5.4.10 Cancel Maintenance Request Sequence Diagram

The sequence diagram for canceling a maintenance request is shown in Figure 5.22. The inbox is first populated as previously stated. Then a user, specifically a tenant in this case, swipes an item to the right. The user is then prompted to ensure that they want to delete a maintenance request. If so, the work order is then deleted from the Parse database.

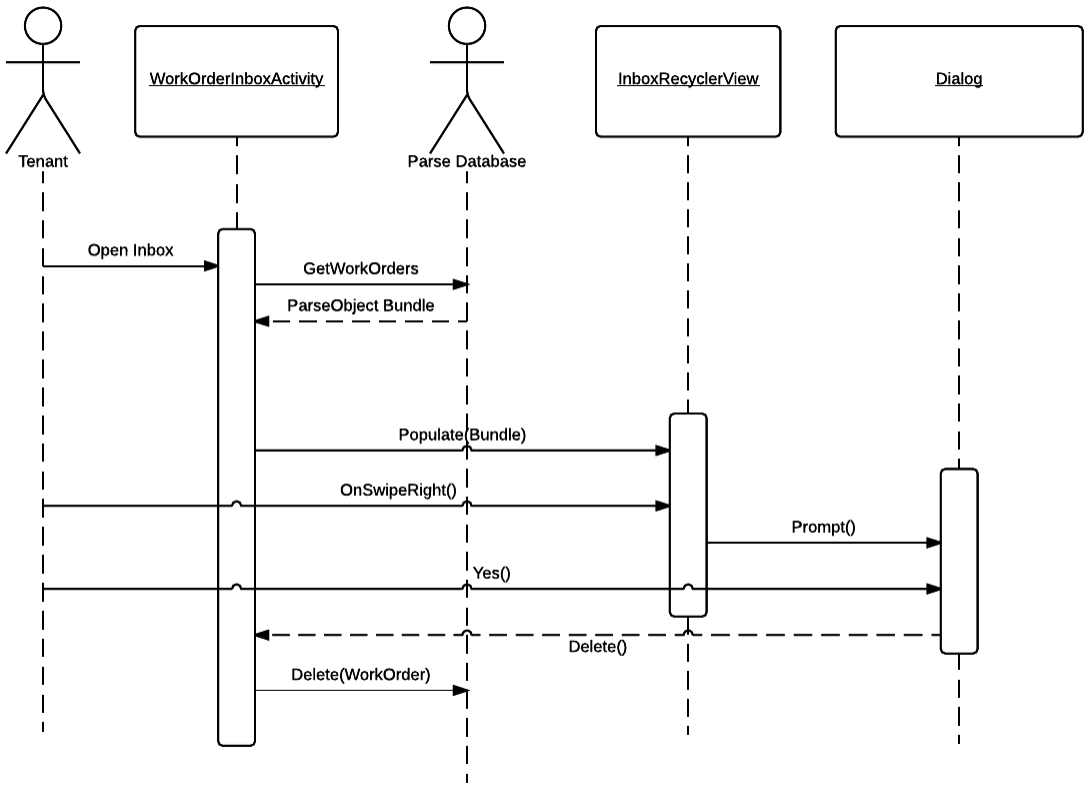

Figure 5.22

5.5 Detailed Class Diagrams

5.5.1 Entity Classes

This class diagram shows all of the entities of our System. Each class contained here will extend the ParseObject in order to be easily mapped directly to the Parse Database. Other classes will have access to each of these Entity Classes but those connections have been omitted for readability. The entity class diagram is shown in Figure 5.23.

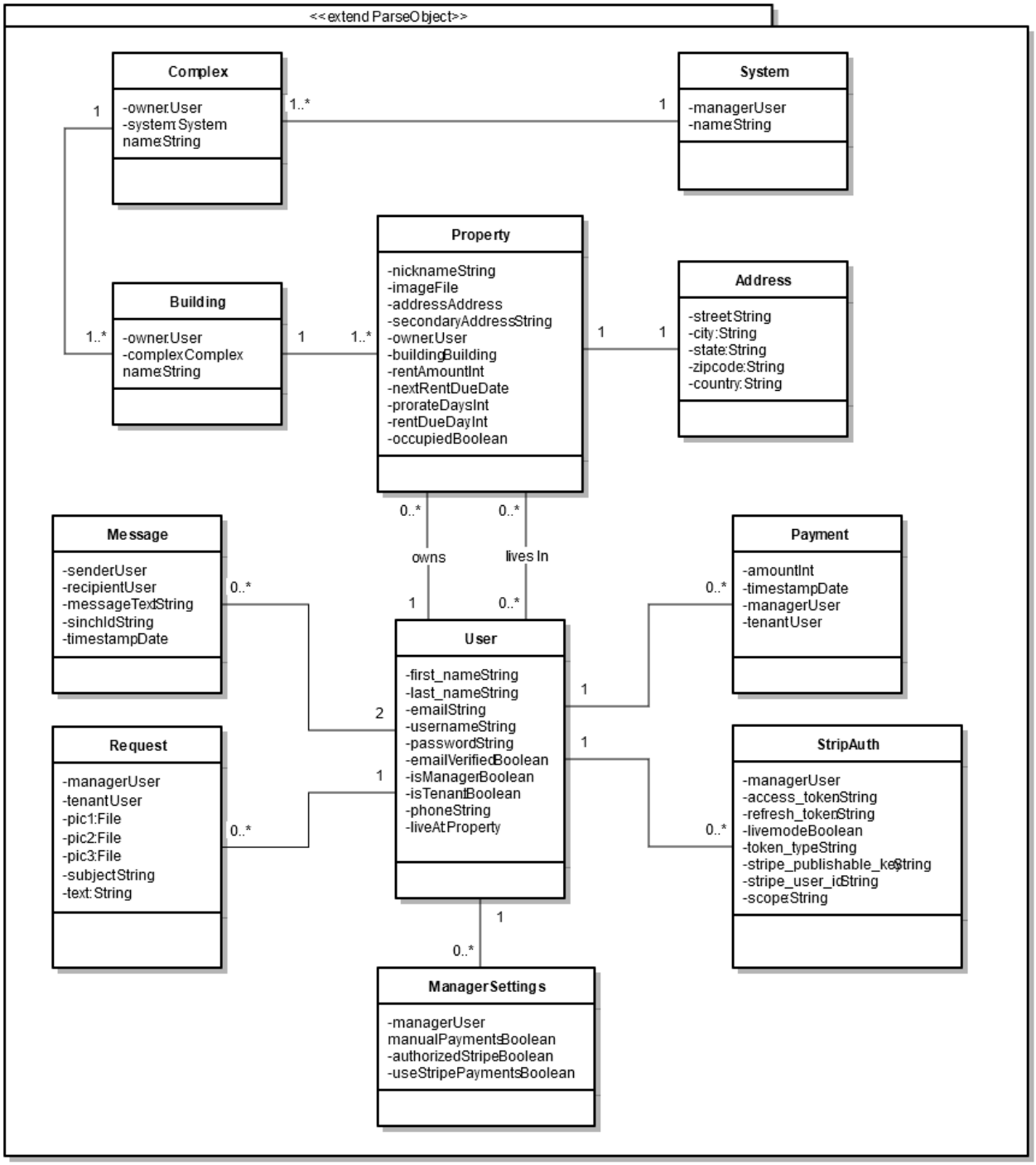
<<extend ParseObject>>
Complex
-ownerUser
-systemSystem
nameString
System
-managerUser
-nameString
Property
-nicknameString
-imageFile
-addressAddress
-secondaryAddressString
-ownerUser
-buildingBuilding
-rentAmountInt
-nextRentDueDate
-prorateDaysInt
-rentDueDayInt
-occupiedBoolean
Building
-ownerUser
-complexComplex
nameString
Address
-streetString
-cityString
-stateString
-zipcodeString
-countryString
Message
-senderUser
-recipientUser
-messageTextString
-sinchIdString
-timestampDate
owns
lives In
Payment
-amountInt
-timestampDate
-managerUser
-tenantUser
User
-first_nameString
-last_nameString
-emailString
-usernameString
-passwordString
-emailVerifiedBoolean
-isManagerBoolean
-isTenantBoolean
-phoneString
-liveAtProperty
Request
-managerUser
-tenantUser
-pic1File
-pic2File
-pic3File
-subjectString
-textString
StripAuth
-managerUser
-access_tokenString
-refresh_tokenString
-livemodeBoolean
-token_typeString
-stripe_publishable_keyString
-stripe_user_idString
-scopeString
ManagerSettings
-managerUser
manualPaymentsBoolean
-authorizedStripeBoolean
-useStripePaymentsBoolean

Figure 5.23

5.5.2 Detailed Overview of Classes

This is a class diagram that details all of the classes we expect to have in our System. Currently, these Classes should give us all of the functionality that we want in our application. A few of the Entity classes are used throughout this diagram, it is assumed they would have an association, but they have been omitted for readability. This diagram is broken into three large fragments, which can be found in Figures 5.24, 5.25, and 5.26. The detailed overview of all the classes is shown in Figure 5.23.

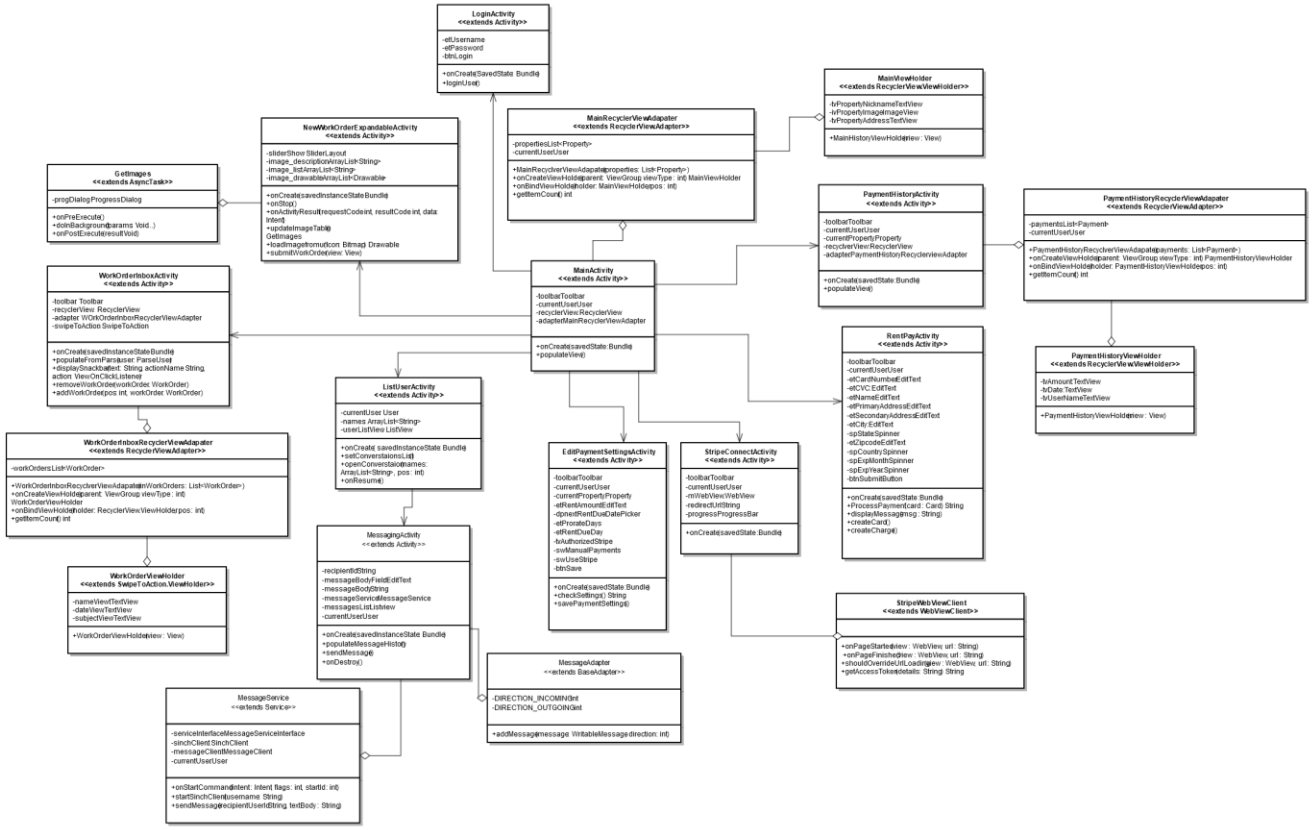

Figure 5.23

5.5.3 Class Diagram for Payments

This is a close-up of the Payment Activity Classes from the Detailed Class Diagram. Each of the Payment Activities that is associated with the Main activity corresponds to use cases and they each serve an independent purpose. The class diagram for payments is shown in Figure 5.24.

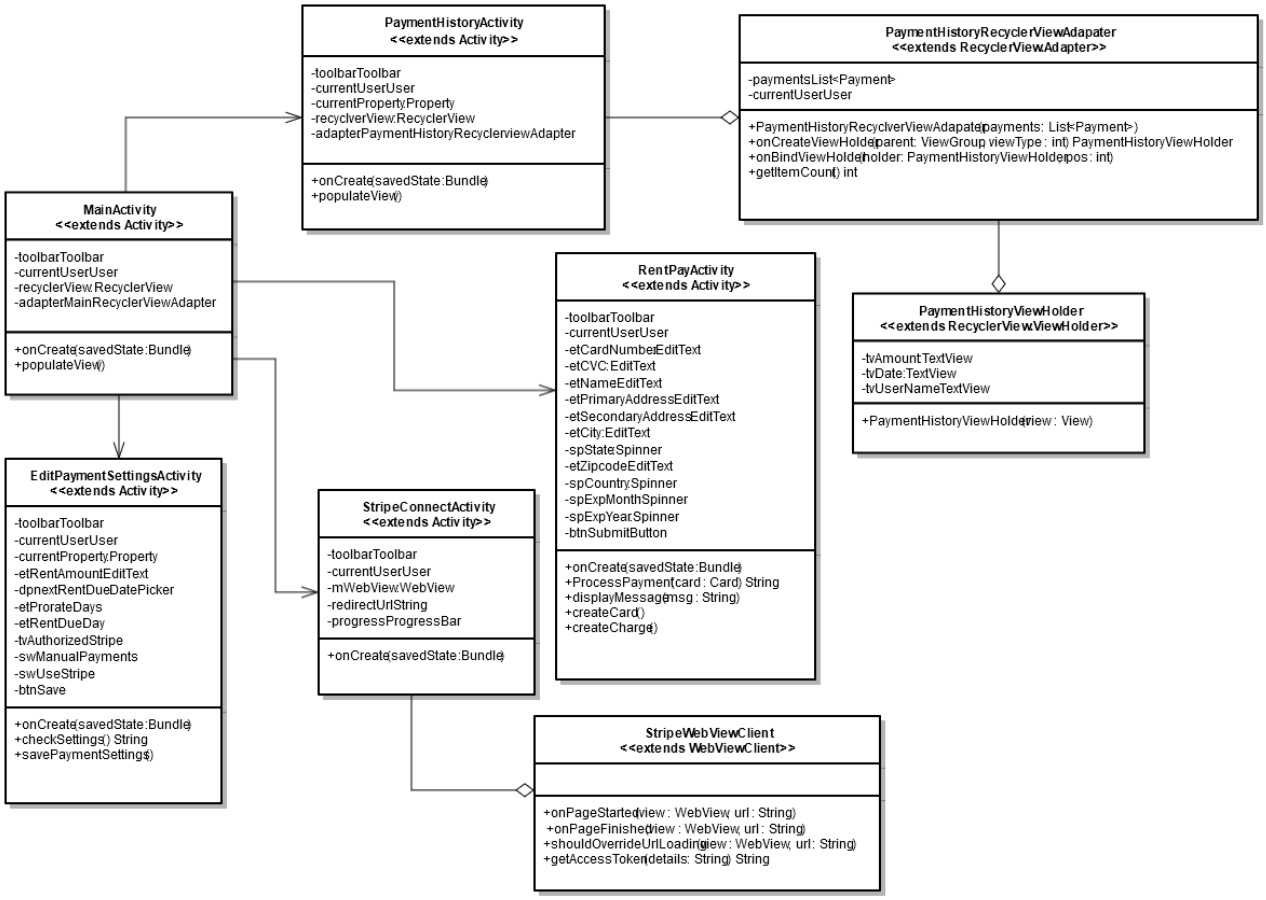

Figure 5.24

5.5.4 Class Diagram for Messaging

This is a close-up of the Messaging Activity Classes from the Detailed Class Diagram. The List User Activity is connected to the main activity, and the activities and entities related to messaging are connected to List User Activity. The class diagram for messaging is shown in Figure 5.25

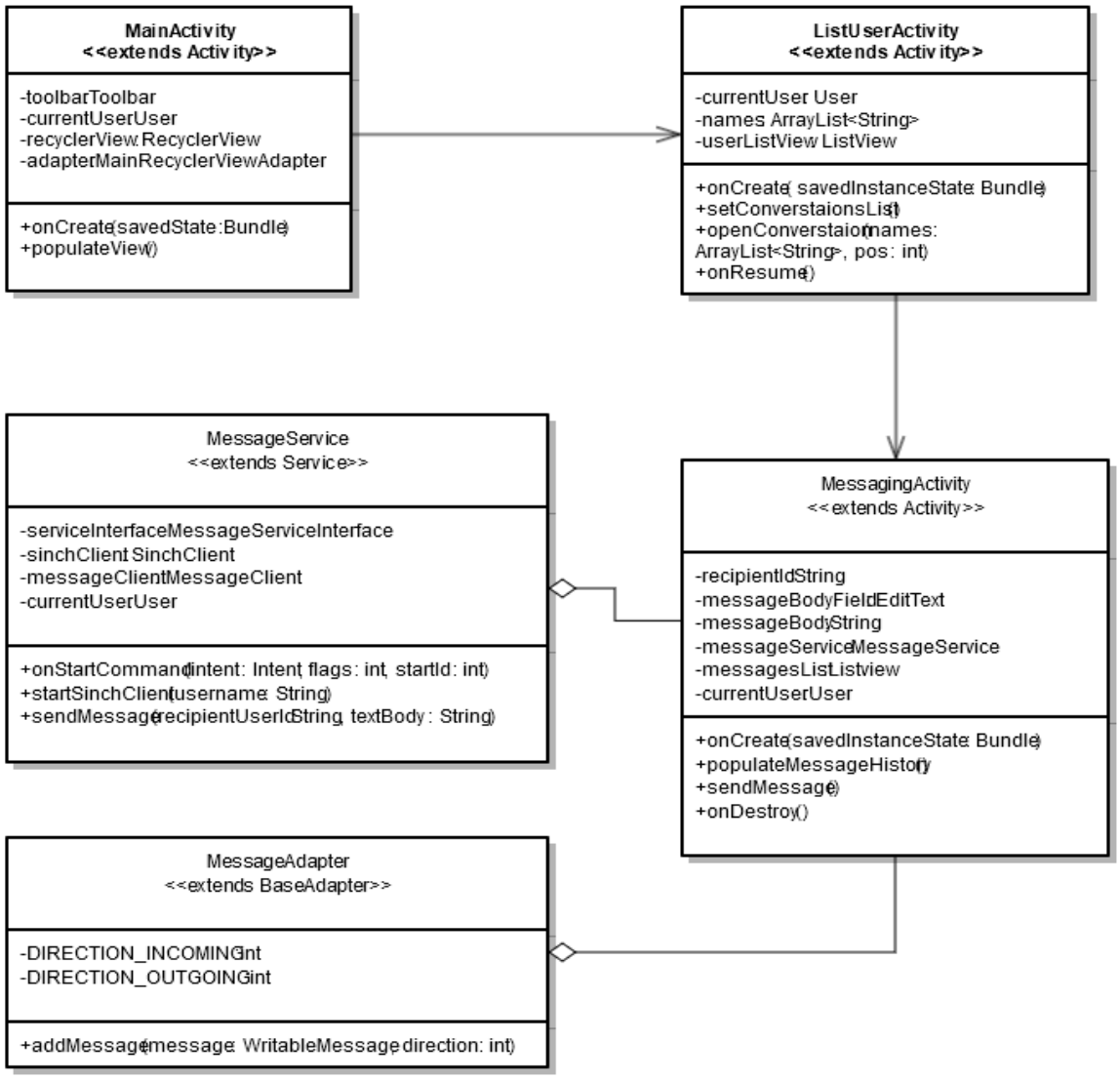

Figure 5.25

5.5.5 Class Diagram for Work Orders

This is a close-up of the Work Order Activity Classes from the Detailed Class Diagram. The Work Order Inbox and the New Work Order Activity both connect to the Main Activity. The class diagram for work orders is shown in Figure 5.26

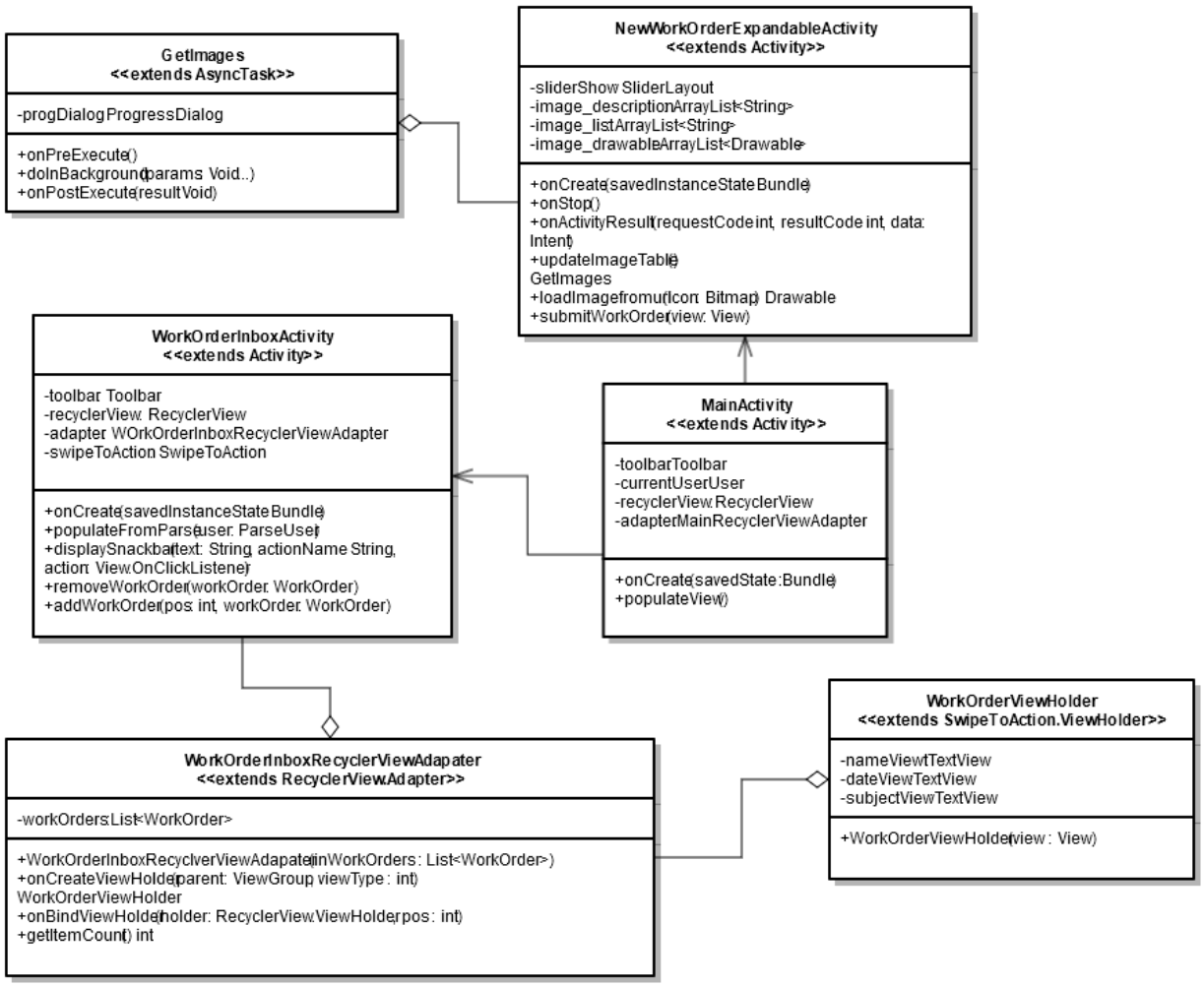

Figure 5.26