**Capstone: Asset Management Application**

**Task 2 - Section C**

**Western Governors University**



**Ryan Michael Patterson**

**Project Table of Contents**

Application Design Documentation 3

Database Design 3

Class Design 5

Architecture & UI Design 6

Application Testing 8

Test Plan Purpose 8

Test Plan Overview 8

Test Plan 9

Test Code Example 10

Unit Test Pass/Fail Conditions 11

Unit Test Results 11

Maintenance Guide 12

Maintenance Prerequisites 12

Opening the Application Code 13

End User Guide 15

Navigating the Asset Management Application 15

Create a New Entity 16

Viewing Entity Details 17

Modifying an Entity 18

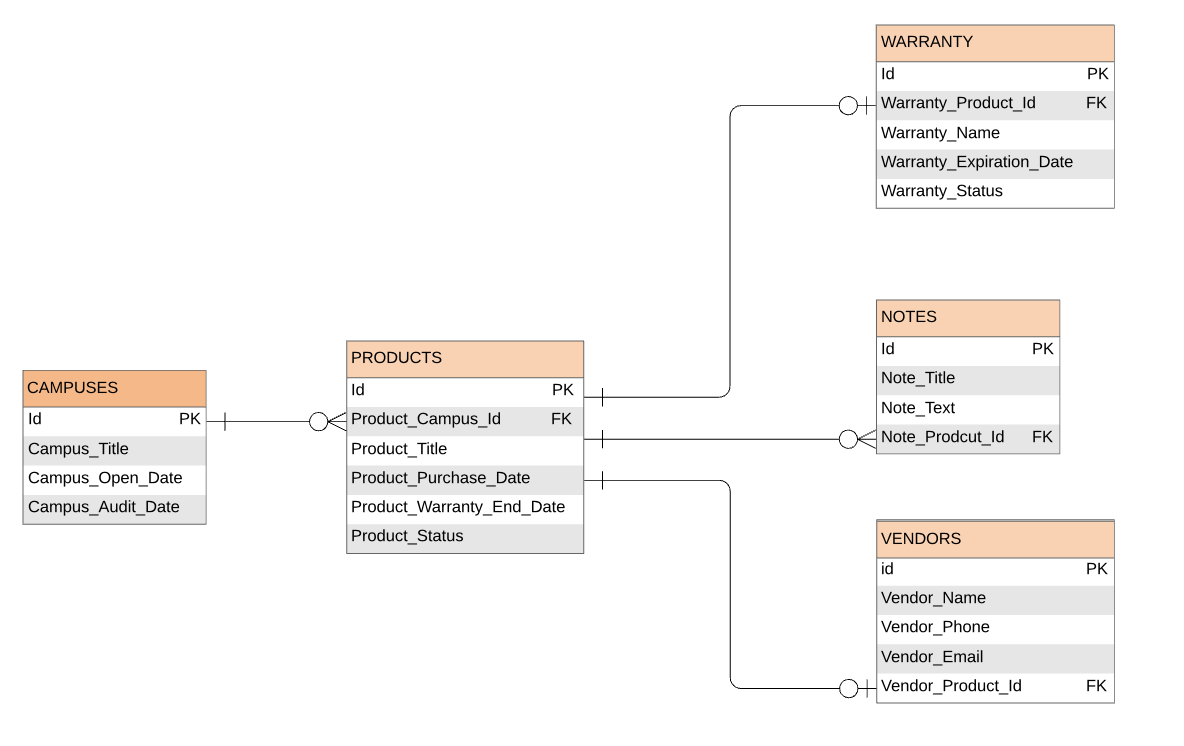
Deleting an Entity 19

Working with Pre-Defined Attributes 20

Linking Entities to One Another 21

# Application Design Documentation

## **Database Design**

The Entity Relationship Diagram (ERD) below represents the table structure for the Asset Management Application. The database will be using the open source variation of SQL called SQLite. The database will be made up of five different tables, each corresponding to one of the unique entities that form the core of the application. The primary key for each table is represented by the PK notation and any foreign keys within the table are noted by the FK notation in the corresponding column.

**Campus Table**: The campus table illustrates the buildings that Pinnacle Technologies has assets deployed in. A campus has the ability to house many different products, so the campus table will have a zero-to-many relationship with entities created in the Products table. However, a product must exist on a specific campus to be included in the company’s inventory.

**Product Table:** After collecting all of the information from the spreadsheet, the product table is subdivided into four tables through the processes of database normalization. The core product table will contain the information that relates to the asset itself: the name of the product, the date that the product was purchased, the standard manufacturer’s warranty, and the product status.

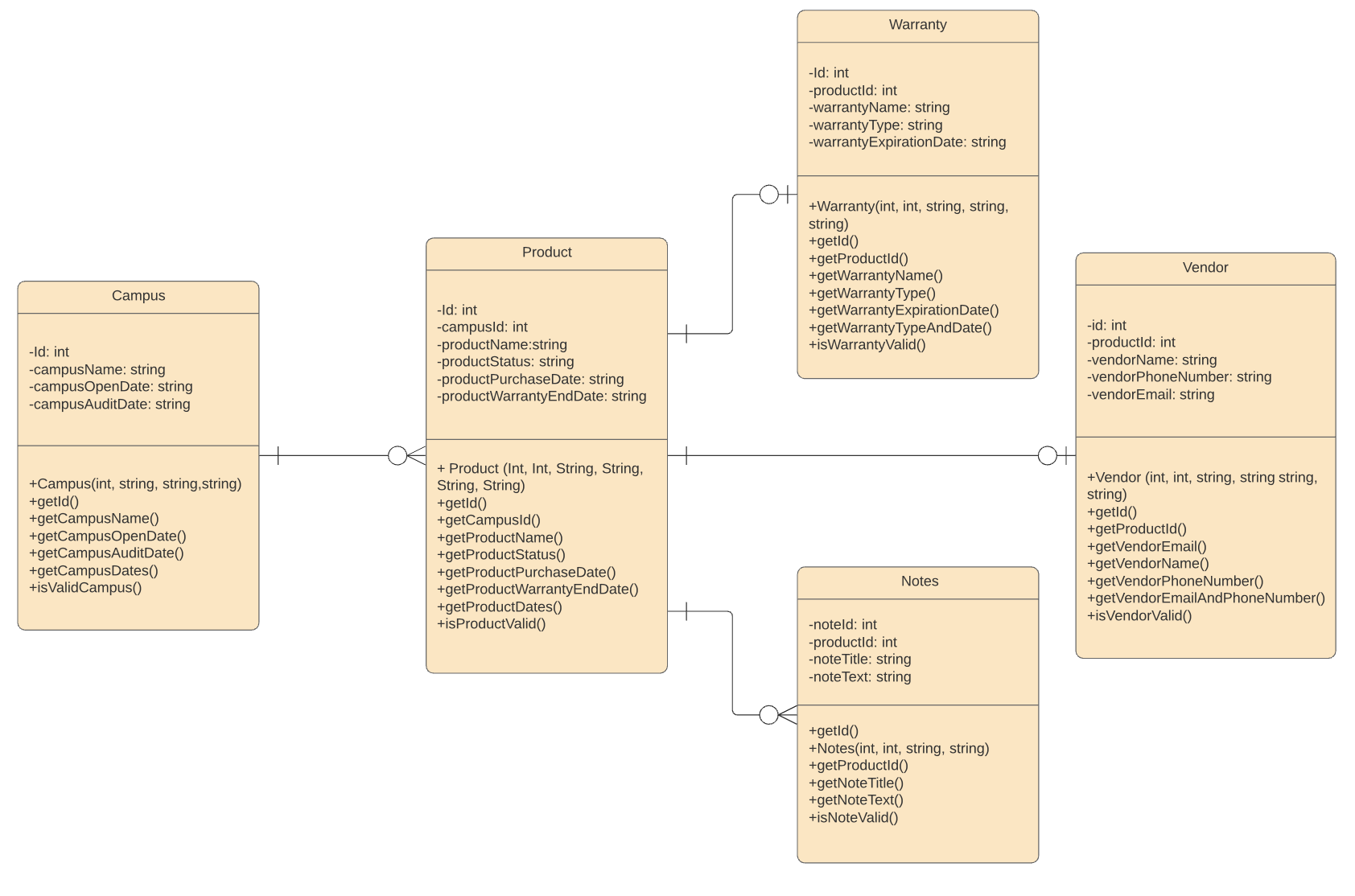
**Warranty Table**: The warranty table will contain all information related to any additional warranty or services that Pinnacle Technologies purchases beyond the standard manufacturer’s warranty. An additional warranty is optional on a product, and the company can only purchase one per product. This relationship will be represented by a zero-to-one relationship from the product table to the warranty table. A one-to-one relationship is formed in the other direction since a warranty needs to be assigned to a single product and cannot exist without a product.

**Vendor Table**: The vendor table represents the external company that is in charge of supporting a product. The vendor table will have a one-to-one relationship with the product (only a single vendor can be assigned to a product), and in the opposite direction the relationship will be a zero-to-one (not every product will need to have a vendor assigned).

**Note Table**: The note table will be created so that end users can add additional information to a product that was not captured in one of the other table options. A product does not need to have a note, but the end users will be able to create as many notes as they want, forming a zero-to-many relationship. A note must always be assigned to a single product, forming a one-to-one relationship in the other direction.

## Class Design

The classes within the Asset Management application will be designed around the five database tables and their schema. Each of the entity relationships created in the ERD will be translated into the class design. Through the use of Java-based Data Access Object (DAO) classes, the application will be able to convert the java objects defined by these classes into entities to be stored into the database. Data validation and referential integrity within the database will be enforced by java code. Each of the classes will contain a unique “is valid” method. This method will check to ensure that none of the required attributes are set to null and all of the attributes of the entity are in a proper format. A second layer of redundancy is introduced by the use of class constructors. The objects will be initialized via class constructors that require all attributes to be populated in order for the object to be created.

Additional class methods will be created such as getWarrantyTypeAndDate() and getVendorEmailAndPhoneNumber() to assist with the data population of the various adapters used in populating the applications Recycler Views.

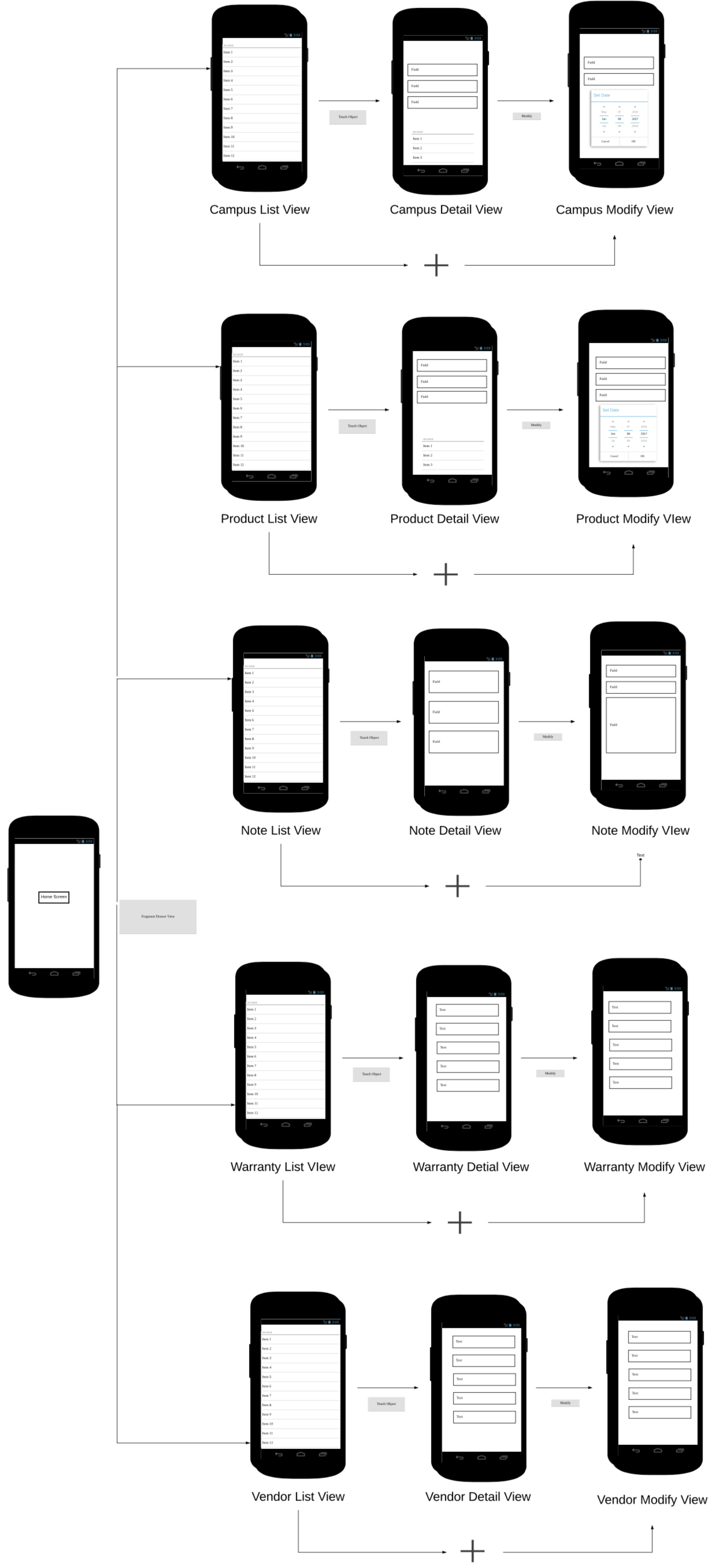
## Architecture & UI Design

The user interface of the asset management application will focus more on utility then design. When opening the application the user will be brought to the home screen. The home screen is one of the six view fragments that make up the navigation of the application. To navigate around the application, the user will need to click the hamburger icon along the top left corner of the screen, this will open a drawer that will let the user choose which of the entities from the database they would like to view. Each of the five entities will have three associated views : the *List View*, the *Detail View*, and the *Modify View.* Each of these three views, however, serves the same purpose across the five entities. More information on these purposes can be found below.

**The List View**: The *List View* contains a complete list of every object of the current entity type that exists within the database. From the list view the user can create a new entity by clicking the plus button along the bottom right corner, which will bring the user to the *Modify View*. A user can see more details about an object that already exists by simply clicking on the object.

**The Detail View**: The *Detail View* will show all of the attributes of the selected object from the *List View*. If the user would like to modify some of the details of the object, then they will click the edit button on the screen, which will bring them to the *Modify View*. They may navigate back to the *List View* at any time by clicking the back arrow along the top left corner of the screen.

**The Modify View**: The *Modify View* is a multi purpose view that will be used for the creation, the modification, or the deletion of an object depending on which view the user navigated from to reach the *Modify View*. For the creation and modification of a object, the user will need to ensure all attributes are filled out and then select the “save" option. If the user would like to delete an object, then they will need to select the “delete” option. The delete option will only appear on the screen if the user has traveled to the *Modify View* from the *Detail View,* and the delete option will not appear if the user navigated directly to the *Modify View* from the *List View.*

Asset Management Application Mobile Framework and UI

# Application Testing

## Test Plan Purpose

Within the Asset Management Application are Java classes that are used to represent the database entities. These modules will preform tests on the user inputed data to validate whether all required fields have been filled out and if valid data has been entered into each the fields. Built within each of these Java classes is a custom method that is used to validate various aspects of the class, such as if any of the fields are empty, if start dates are prior to end dates, and so on. These validation methods will be used every time one of the objects is created by the application. Android Studio has a built-in framework called JUnit4 that will be used to write repeatable unit tests. The unit tests created for this application are designed to verify that the validation checks within each of the classes will work correctly. If any of these checks are not designed properly, and therefore the checks do not catch the issues they are intended to identify, then this situation can cause cascading problems for the database. The unit test for this example is designed to check the four attributes that will be populated when creating a “Campus” object.

## Test Plan Overview

The *isValidCampus()* method within the Campus class is designed to check if all three of passed values do not equal “empty.” Next the method will attempt to parse the passed date fields and convert them to a SimpleDateFormat. If SimpleDateFormat is successful, then the method compares the two dates to ensure that the start date takes place prior to the second date (the end date). If all of the criteria of the *isValidCampus()* method are met, then the method will return true. If one of the conditions of the *isValidCampus()* method is not met, then the *isValidCampus()* method will return false. The *isValidCampus()* method is invoked every time a new Campus object is created or modified by the Asset Management Application.

The tests for object creation will run utilizing the JUnit4 Framework and will be executed from within the Android Studio Integrated Developer Environment. The framework has two methods that will be used to verify the *isValidCampus()* method results: *assertTrue()* and *assertFalse()*. These two methods will compare the true and false values returned by the method *isValidCampus().* If an *assertTrue()* is used and the *isValidCampus()* method returns true, then the test will pass. The inverse is true for the *assertFalse()* method.The pass and fail results of the test will be printed to the IDEs EventLog video window of Android Studio.

## Test Plan

The *isValidCampus()* method uses three different criteria to check if a campus object should be created. As a whole, the method checks to see if the any of the passed values are empty, if the dates passed into the object are parseable to a simple date format, and if the start date is prior to the end date. In order to check these three criteria a total of five unit tests will need to be created. The first check that is made by the *isValidCampus()* method is if any of the three user inputs in criteria have empty values. To test this we will use three different unit tests, each test will pass an empty value into one of the three variables. The objects created by these tests should return fails when the *isValidCampus()* method is called. By comparing the results of the *isValidCampus()* method with *assertFalse()* JUnit4 method, and *isValidCampus()* test should pass.

The second test method that will be created will check to see if the values passed in by the date can be parsed into a simple date format. By passing in a string of randomized characters to one of the date variables, the is *isValidCampus()* method should fail to parse the dates and will return a false value. By comparing the results of the *isValidCampus()* method with *assertFalse()* JUnit4 method, the *isValidCampus()* test should pass

The final unit test will check to see if the start date that is passed is chronologically before the end date that is passed. To perform this test the function will pass a date into the end date that takes place prior to the past start date. As with the previous two tests, the *isValidCampus()* method should return a false result, and when compared to the *assertFalse()* JUnit4 method, the test should pass.

One final test will be created as a control and will be designed around the *isValidCampus()* method returning true. All of the variables passed into the unit test will meet the criteria of the *isValidCampus()* method. The method should return true, and when compared to the *assertTrue()* JUnit4 method for method the test will pass.

## Test Code Example

The following code is an example of the unit test preformed on the application. The unit test checks each of the three values passed in via user input to see if any of them are empty. If any of the values are empty the creation of the object will fail, and the test will be considered a success.

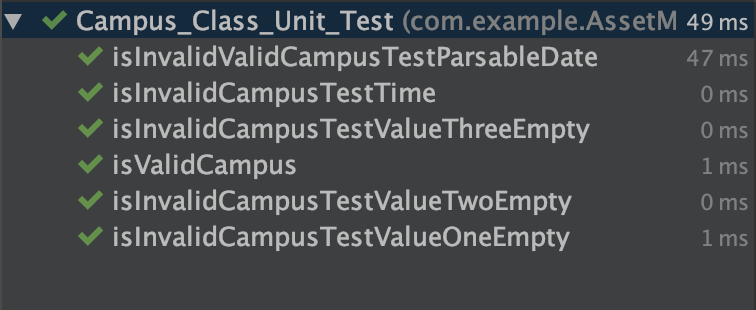
The Unit Test file is located within the application source code under:

test > java > com > example > AssetManagmentSystem > modules > Campus\_Class\_Unit\_Test

## Unit Test Pass/Fail Conditions

Each unit test developed for the Campus class is designed to produce a passing result. A unit tests is indicated by *isInvalid* within their name and are designed to check if the Campus object failed to be created. If the Campus object failed to be created, then the value will return false by the *isValidCampus()* method. The JUnit4 testing will check this false value by checking the return by *isValidCampus()* with the frameworks *assertFalse()* method. Unit tests that have *isValid* are designed to test if the Campus object was successfully created and will return true by the *isValidCampus()* method. As with the *isInvalid* tests, the *isValid* will use the JUint4 *assertTrue()* method.

## Unit Test Results

The output from the Android Studio EventLog shows the result from the *assertTrue()* and *assertFalse()* statements of the unit tests. Each of the different tests performed are listed below the Campus\_Class\_Unit\_Test method. The green check mark to the left of each of the unit tests indicates the test was completed successfully. For the Campus\_Class\_Unit\_Test, all tests designed to fail did, and the control test that was designed to pass did so as well. A of total six tests passed successfully out of the total six tests.

# Maintenance Guide

The following guide will demonstrate how to load the source code of the sset Management Application into the Android Studio software development kit. The source code of the Asset Management Application is loaded into the Android Studio software development kit so that and engineer may perform routine maintenance on the application or update the application.

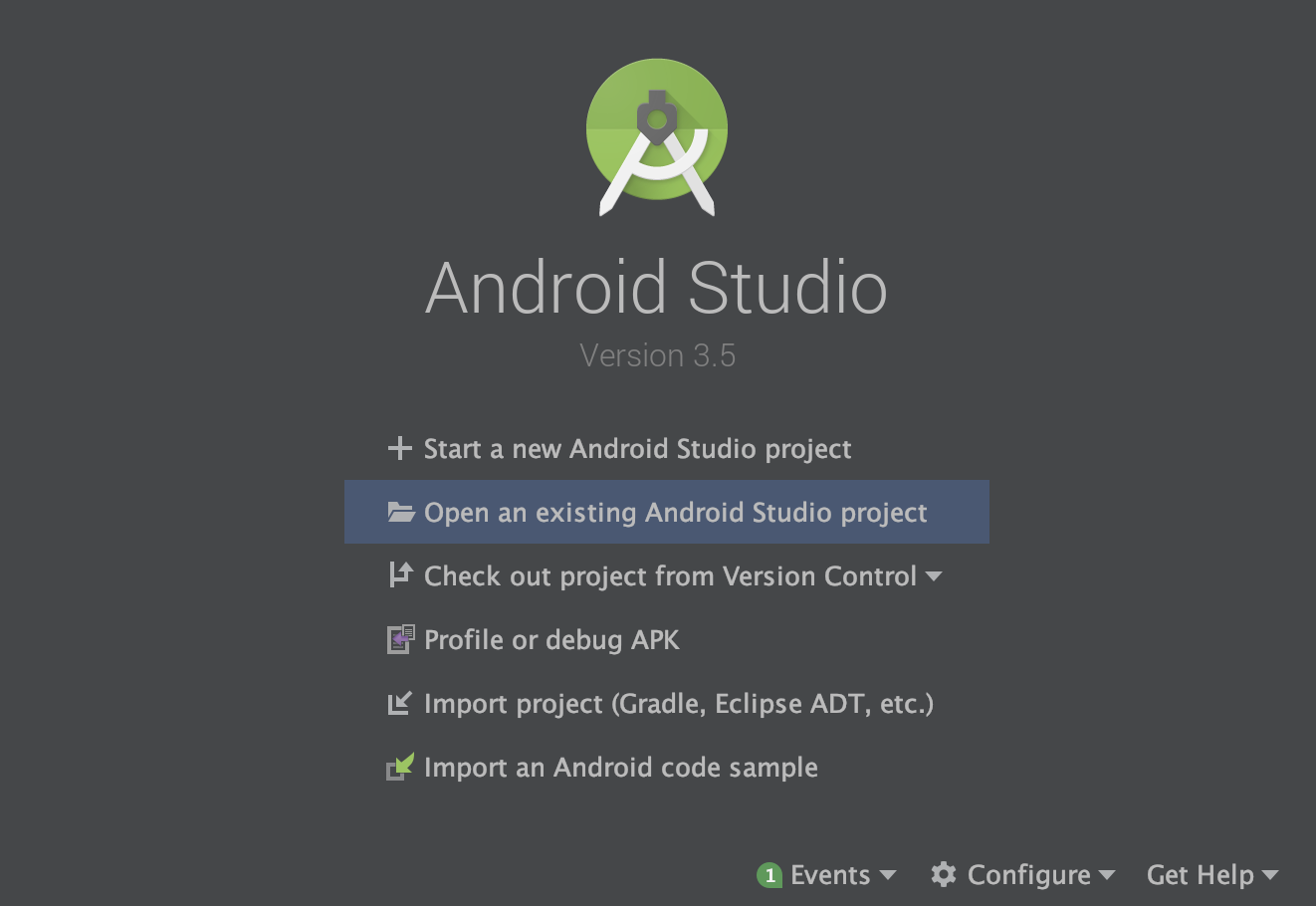
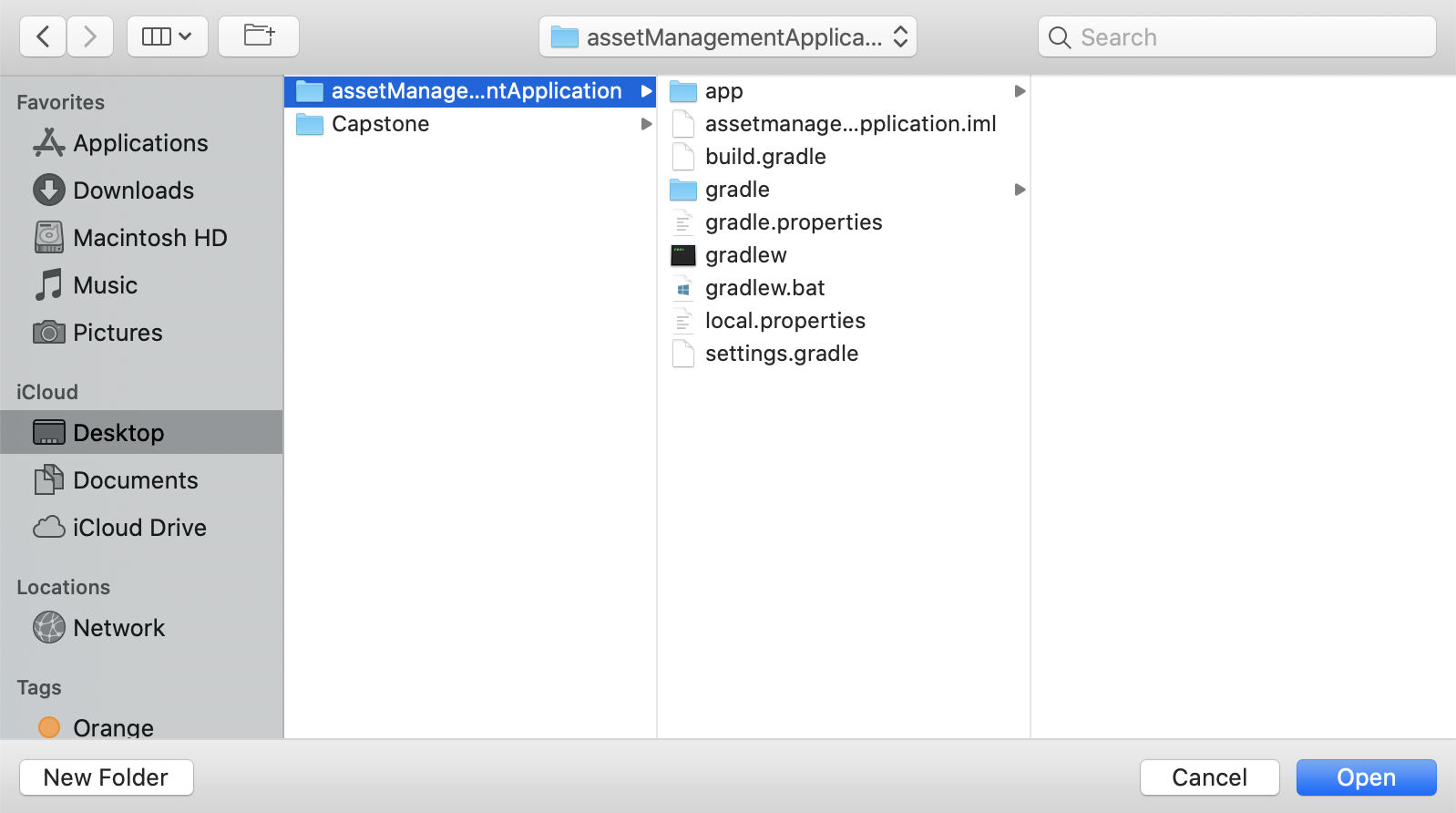
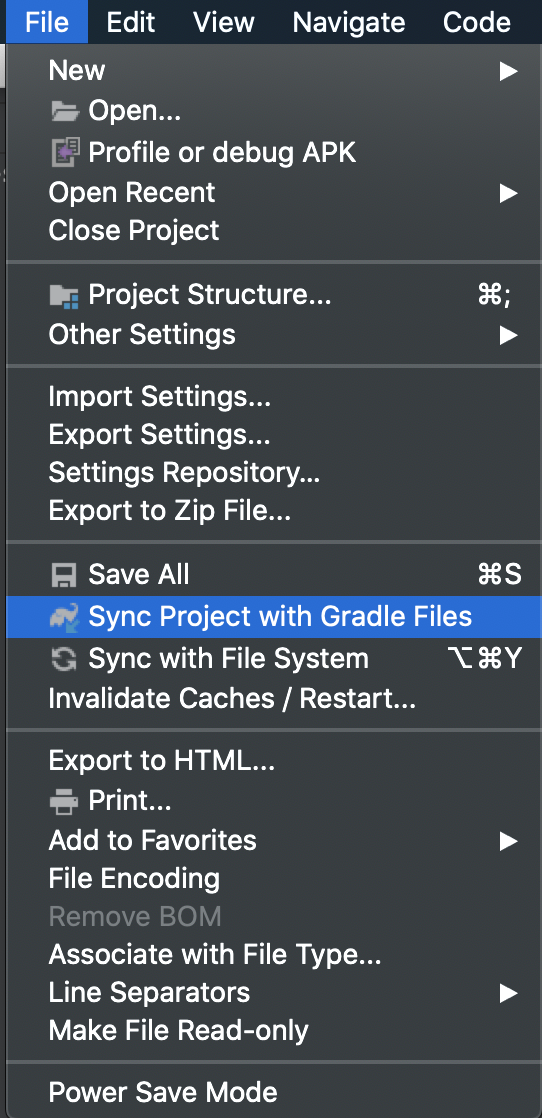
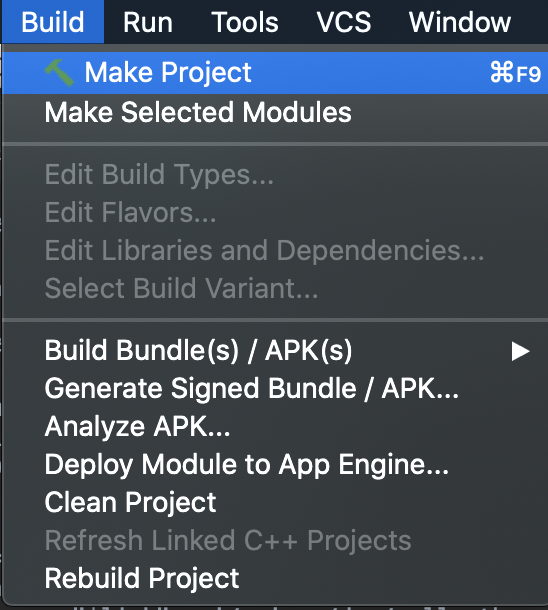
## Maintenance Prerequisites

**Step 1**: Before beginning the engineer must have Android Studio installed on their computer. The system requirements for Windows, MacOs, and Linux; a download link for each operating system; and a walk through on the Android Studio installation can be found on the developer’s website: <https://developer.android.com/studio>

Please note that the Java development kit must be installed in order to run Android Studio. If the application does not prompt you to download the Java software development kit, then the Java software development kit can be accessed via the web: https://www.oracle.com/technetwork/java/javase/downloads/index.html

**Step 2:** Before performing any maintenance on the application or running any tests, it is advisable to configure a emulation device within Android Studio. After completing the installation of Android Studio, navigate to this link and configure a phone emulator: https://developer.android.com/studio/run/emulator

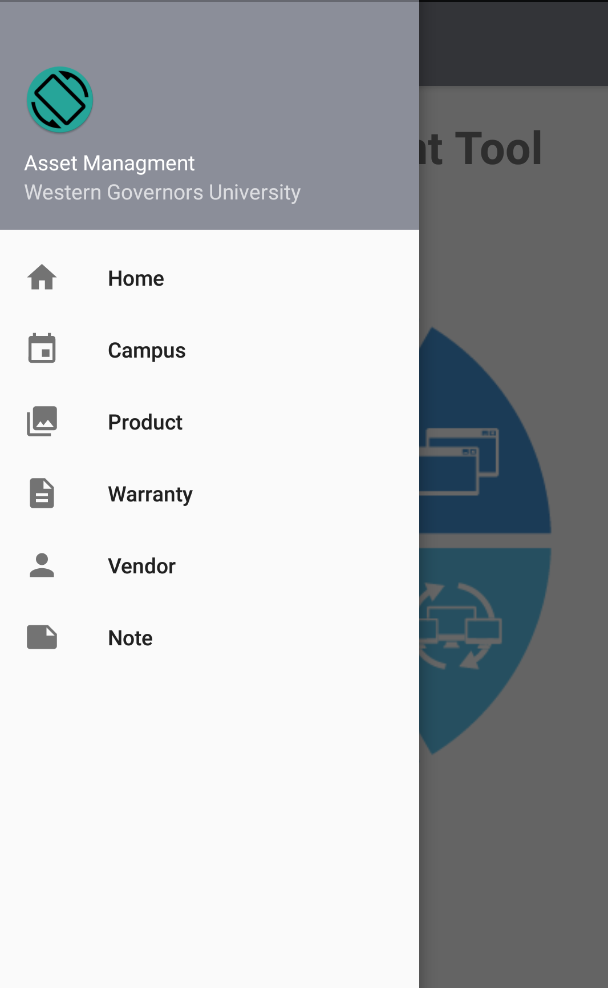
## Opening the Application Code

1. Launch the Android Studio software development kit.
2. Once the application has launched you will be on the following home screen. Select “Open an existing android studio project”.
3. Navigation to the asset management code base will vary depending on which operating system you are using. In this example the source code is located on the desktop of a computer running MacOS.
4. Select “open” to import the project into Android Studio. This may take time to load as the software development kit is importing all of the code and settings.
5. Once the project is finished thinking sync the Gradle files. To do this select the file drop-down menu, and then select "Sync Project with Gradle Files.”
6. After the grade files have synced, navigate to the builds drop down menu and select “Make Project.”  
     
     
     
     
     
     
     
     
   If all of the steps have been followed up into this point and an emulator has been installed, then the application in the simulated environment can be run and edits can be made to the code as needed.

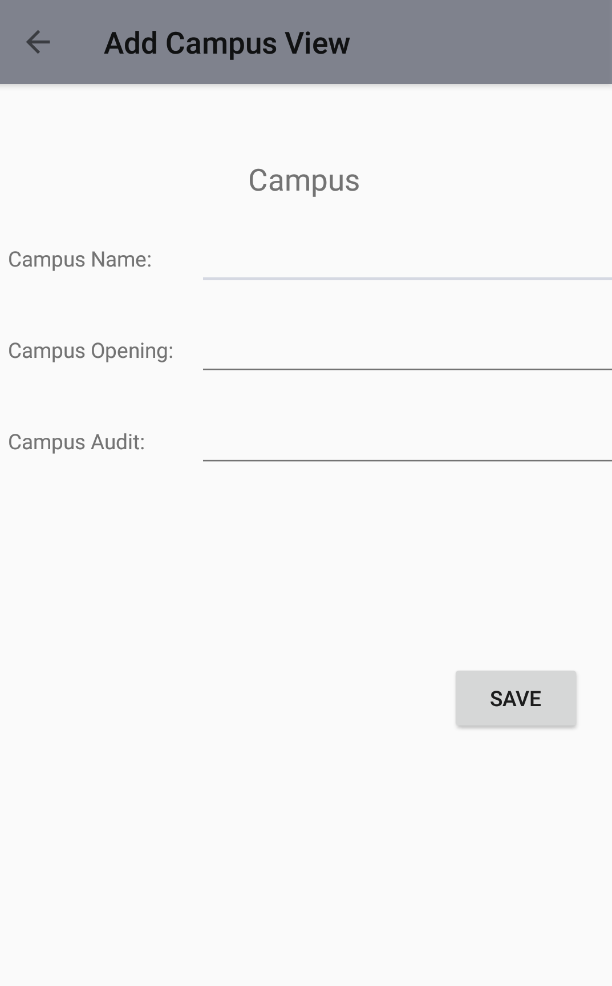
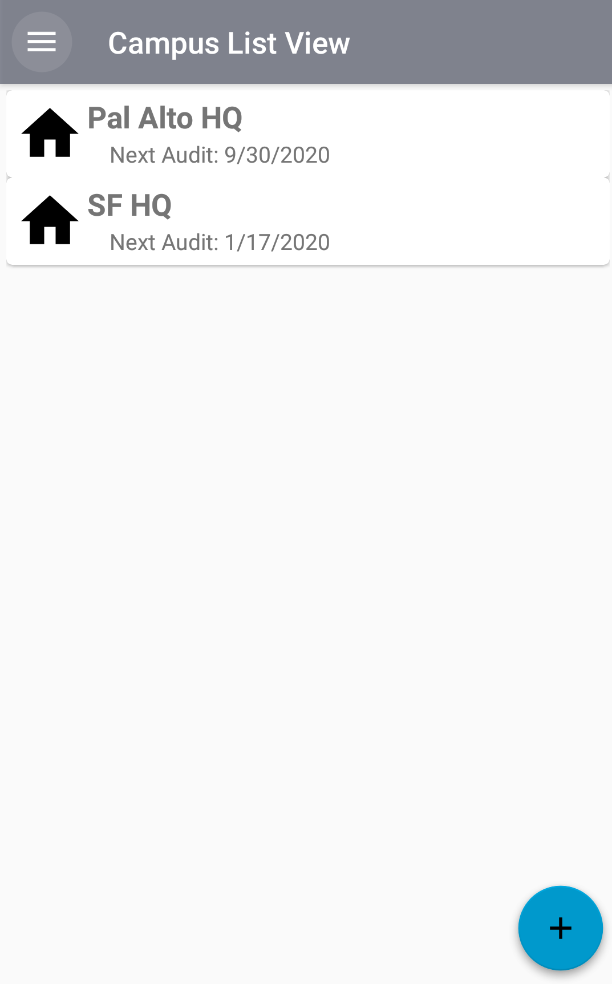
# End User Guide

## Navigating the Asset Management Application

Upon opening the Asset Management Application the end user will be brought into the Home View. This view is the default page that is loaded every single time the end user opens the application. In order to view information related to the different entities within the database, the end user will navigate within the application using the hamburger menu. This is accomplished by selecting the hamburger menu along the top left corner of the application. Selecting the hamburger menu will open a drawer that allows the end user to quickly navigate between the various objects that can be added to the database.



## Create a New Entity

Within the Asset Management Application, to create a new entity from any of the five objects, the user will select the blue “plus” icon along the bottom right-hand corner of the application. This will bring the user into the Create View for whatever entity they are working with. If for any reason the user needs to leave this view, they can select the “back” arrow along the top left-hand corner of the screen.

Once the user has filled out all of the required information, they then select the “save” button along the bottom right corner to save the entity.

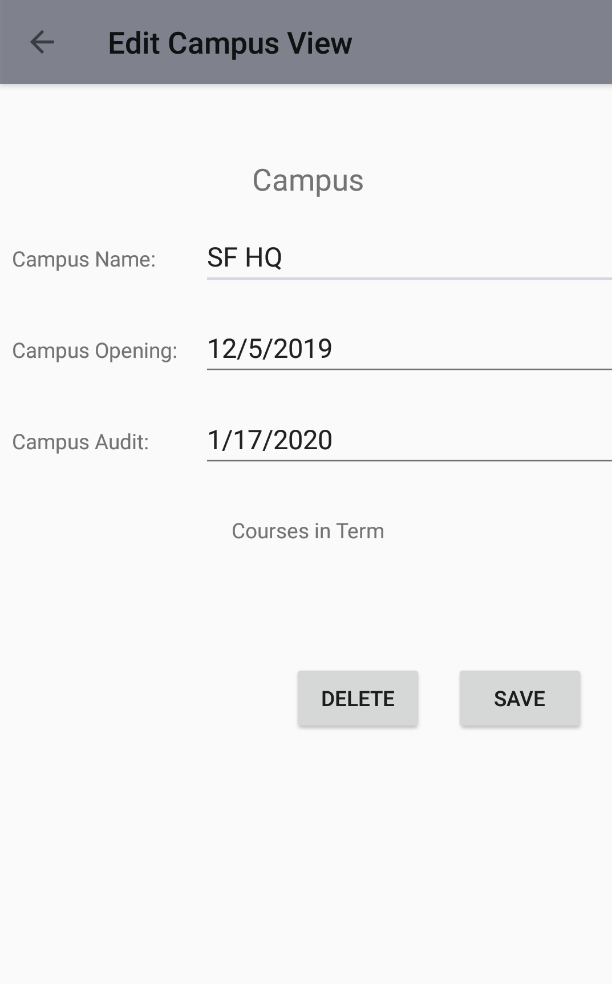
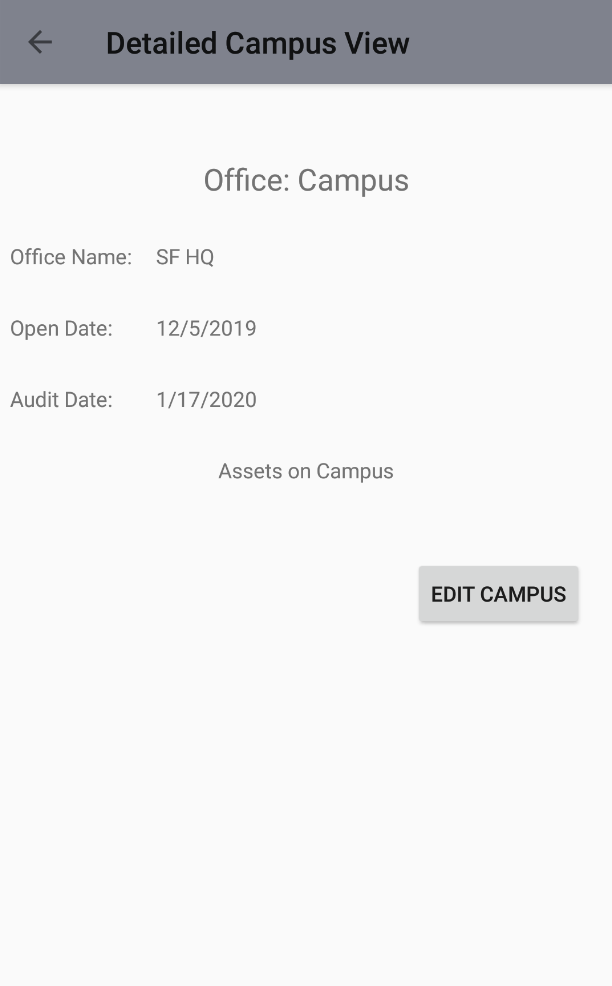
## Viewing Entity Details

In order to modify an entity that has already been created, the user will navigate to the list view of the object type they would like to modify. The user will then select the row with the entity they would like to modify, and that selection action will bring them to the detail entity view. All of the fields in the detail view will be populated with information from the entity the user selected. If at anytime the user would like to leave the detail view and go back, they need to select the “back” arrow along the top left-hand corner of the application.

## 

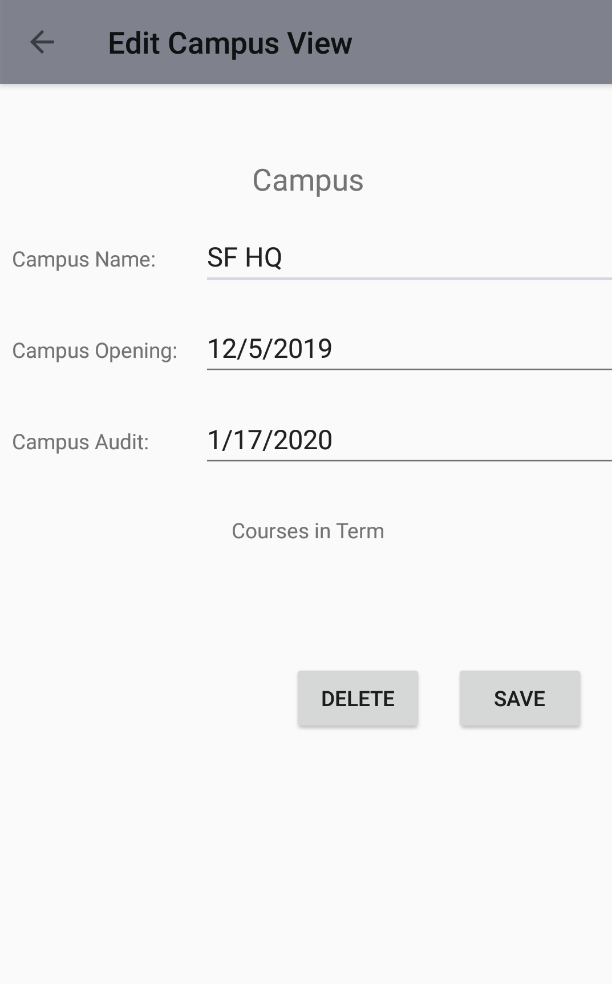
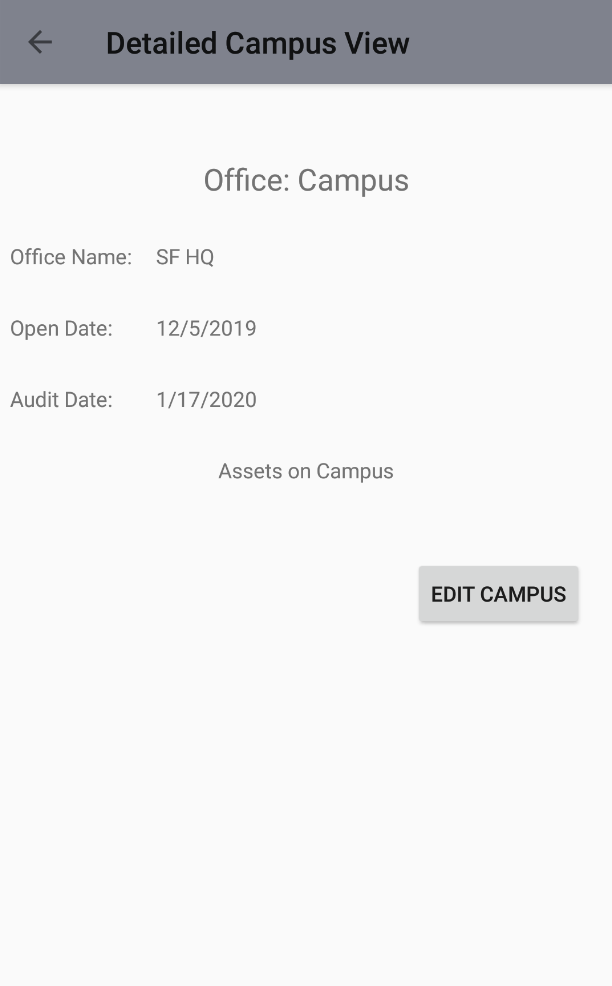
## Modifying an Entity

To modify the attributes of an entity, the user starts by navigating to the detail view of the entity they would like to modify. From the detail view, the user selects the “edit” button along the bottom right-hand corner of the screen. This will bring up the entity modify view. From here the user can change any of the attributes of the entity. Once the user is done editing the attributes, they will select the “save” button along the bottom right corner to update the entity. If at anytime the user would like to leave the modify view and go back, then the user will need to select the “back” arrow along the top left-hand corner of the application.



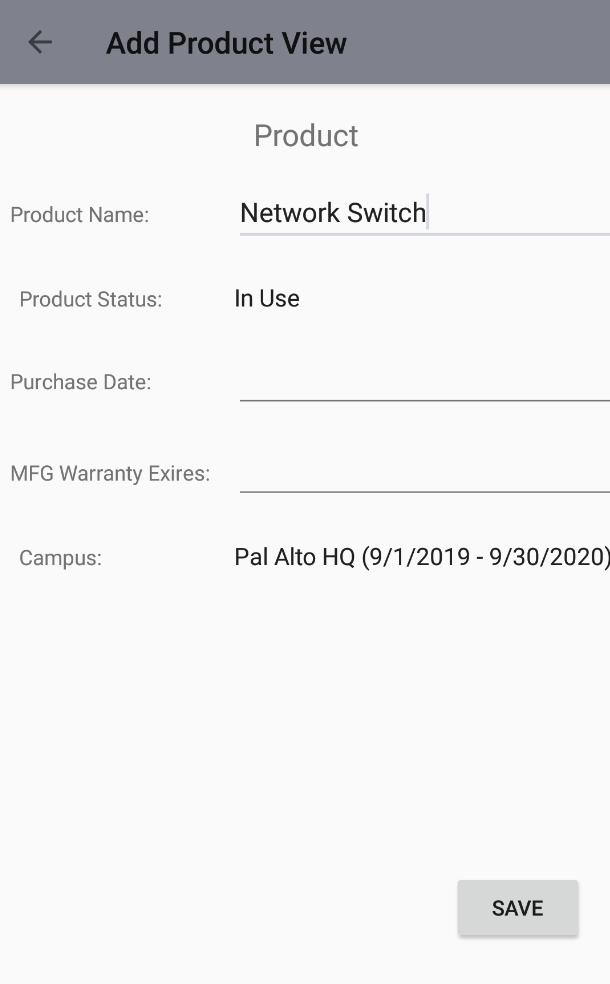
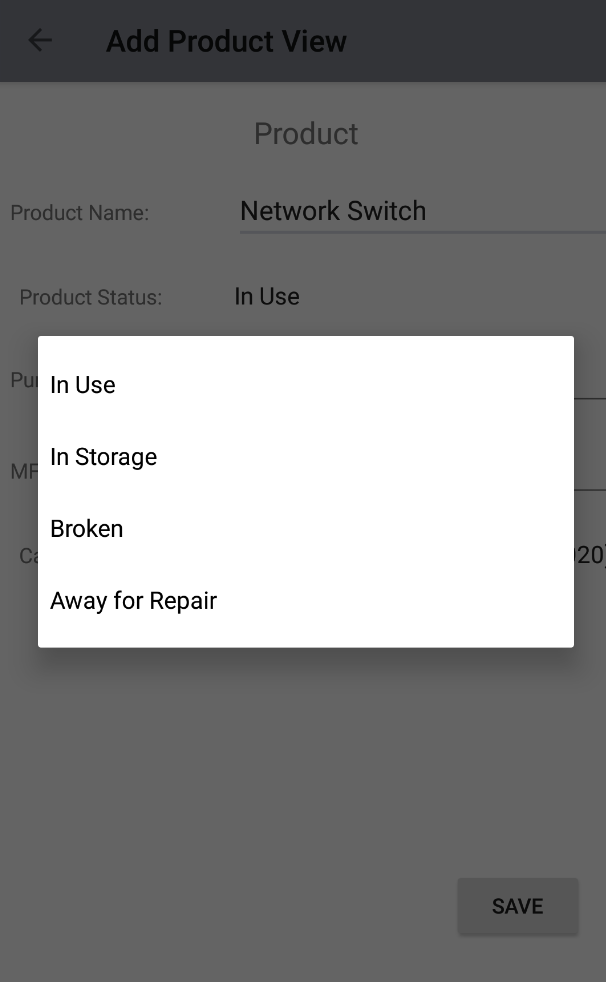
## Deleting an Entity

To delete an entity, the user will start by navigating to the detail view of the entity they would like to modify. From the detail view the user will select the “edit” button along the bottom right-hand corner of the screen. This will bring up the entity modify view. From here the user selects the “delete” button along the bottom right corner to delete the entity. Please note that once an entity is deleted, it cannot be recovered. If at anytime the user would like to leave the modify view and go back, then they need to select the “back” arrow along the top left-hand corner of the application.



## Working with Pre-Defined Attributes

Certain attributes within the Asset Management Application must utilize a set of predefined values. In situations like this, the application will prompt the user with the options from the set of predefined values to be entered in the fields. In these situations, the user selects the value that needs to be inserted and then the application will populate the field with the selected value. These values can be changed at any time by simply modifying the object and selecting the attribute that needs to be modified.



## Linking Entities to One Another

Certain entities, like vendors and warranties, must be linked to a product and cannot be created without the linkage in place. In creating one of these entities, the user will be able to select the product the entity is to be linked to from a list of products that are already in the database. The user will select the product attribute, then the user will be presented with a list of products to choose from and select the appropriate product. If for any reason the wrong product is selected, the selection can be changed at any time by simply modifying the entity.

