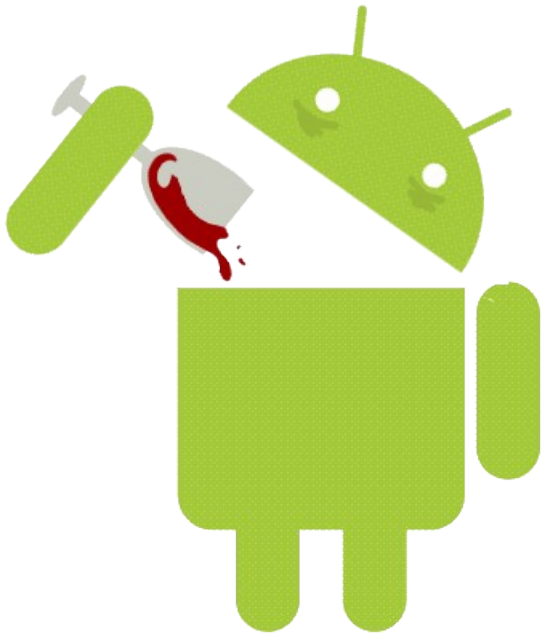
Design Use Cases



PROJECT CUI

SHARON CHUNG
SAM FORMATO
BRIAN TAI
NAY TUN
ARASH VATANPOOR
CHRIS WELLER
WILLIAM WERNER
ZHIHUI XIA
JESSE ANGELO YEH
ZHAOYANG ZENG

TABLE OF CONTENTS

Design Use Case #1 - “User Creates Account” [LA1]

Description:
This use case outlines the implementation details of creating a new personal account.

Desired Outcome:
The user will have created a new account, granting them access to the full functionality of the application.

User Goals:
The user wants to create an account within the wine application.

Dependency Use Cases:

N/A

Preconditions:

1. The user has not yet created an account.
2. The user’s input credentials do not already exist.

Postconditions:

1. The user’s account will be added to the database.
2. The user will have access to all functionality of the application.

Trigger:

1. From the login activity, the user specifies that he/she is a new user and clicks on “Create an

Account”.

Workflow:

1. SignUpActivity.java calls SignUpView to set the view.
2. SignUpView displays all fields for the user to fill out.
3. The user fills out all required fields.
4. SignUpManager calls commit() to ensure that all fields were filled out with valid information.
5. UserDAO.java calls getUserDAO() to ensure that the user does not already exist.
6. If the user profile can be created, UserDAO adds it to the user database using createUserOnServer().
7. The user is christened a new oenophile.

Alternate Paths:

1. UserManager.java determines that the user’s entered credentials already exist and calls login.java to present the user with an error message. The user is returned to the default login activity screen.

Options:

N/A

Description:
This design use case outlines the implementation details of how the user logs into the application.

Desired Outcome:
The user shall be able to login to the application and access their data.

User Goals:
The user wants to login to the application without losing his/her personal data.

Dependency Use Cases:

LA1

Preconditions:

The user shall have valid credentials and shall have a valid account in the User Database.

Postconditions:

1. The system shall redirect the user to their personalized selection screen.

Trigger:

The user shall enter their credentials and click on “Login”.

Workflow:

1. LoginActivity calls LoginView to set the view. Thread policy for all fields is set to StrictMode using setThreadPolicy().

2. The user enters his or her userID and password into the fields.
3. LoginController calls LoginManager to attempt to log in.
4. LoginManager calls its login() method to call UserDAO.java's getUserDAO() method to ensure that the user is in the database.
5. UserManager.java's getUserManager() method is called to set the current user to the user who has successfully logged in.
6. The user is redirected to the main activity screen.

Alternate Paths:

1. if the user's credentials are invalid and the user presses “Login”, login.java shall display an error message, and then returns to LoginActivity.xml.

Options:

N/A

Description:
This design use case outlines the implementation details of how the user logs out of the account.

Desired Outcome:
1. The user shall be logged out of the application with his or her data preserved.
2. The user shall be redirected to the login page.

User Goals:
The user wants to log out of the application without losing his or her personal data.

Dependency Use Cases:

LA1

Preconditions:

1. The user must already be logged into his or her account.

Postconditions:

1. The user shall be logged out of his or her account.
2. The system shall redirect the user to the login screen.

Trigger:

1. The user shall click on the “Log Out” button in the side menu.

Workflow:

1. A new intent is created for the Log In activity
2. The Log In activity is started and the previous activity is finished.
3. The BAC count is set to 0.

Alternate Paths:

N/A

Options:

N/A

Design Use Case #4 - “Redirect to Default Home Screen” [LA4]

Description:
This design use case outlines the implementation details of the process in which the system will take the user to their default home screen.

Desired Outcome:
The user shall be shown their default home screen.

User Goals:
The user wants to access home screen.

Dependency Use Cases:

LA2

Preconditions:
1. The user must already be logged into his account.

Postconditions:
1. The user has logged into the system with a valid credentials.

Trigger:

1. The user shall press login, or choose not to login.

Workflow:
1. loginController creates a new intent for the main activity.
2. The Main activity is started.

Alternate Paths:

1. If the user chooses not to login, they will be redirected to the default home screen (CAW).

Options:

N/A

Description:
This design use case outlines the implementation details of how at any part of the application it is possible to navigate to other components of application such as Choose-A-Wine or Wine Cellar.

Desired Outcome:
The user has successfully left the previous activity (system), and has entered a new activity.

User Goals:
The user wants to go and use another component of the application.

Dependency Use Cases:

LA1

Preconditions:

1. The sideNavigation library is imported into each activity

Postconditions:
1. The sideNavigation menu has appeared on the left side.

Trigger:
1. From any activity, the user swipes the screen to the right
2. From the following list of activities, the user selects the desired one:
   a. WineCellar
   b. TopWines
   d. Current BAC

e. Clear BAC
f. Clear Wine
g. User Profile
h. Log Out

Workflow:

1. Any activity in the application is listening to the right long swipe gesture on the screen.
2. Upon each swipe-right gesture, the method getLength() form android Gesture API is called.
3. If the length of swipe is long enough, a call to sideNavigation.java class is made. Else, the gesture is ignored.
4. The View is updated with the menu appearing on the right side inside the same activity that the user is on.
5. Upon clicking on each entry view, onClick is invoked which runs the desired activity.
6. The View is updated with its respective activity.

Alternate Paths:

This functionality is available in all application activities.

Options:

1. Menu Button

a. There is a Menu View Button on each activity that the user is on.
b. The Menu onClick method which listens to this button makes a call to sideNavigation.java
c. The View is updated with the menu appearing on the left side.
d. Upon clicking each entry view, The respective onClick listener runs the desired activity.
e. The View is updated with the respective activity.

Description:
This design use case outlines the implementation details of how the user can search for specific wine. The search dynamically updates with wines the user has previously drank, or the user can search the online database for all wines matching the search.

Desired Outcome:
The typed wine name in the search box has found the wine desired by the user.

User Goals:
The user wants to search for a specific wine by name.

Dependency Use Cases:

LA1

Preconditions:

1. The user-desired wine exists in the online database or the local database.

Postconditions:

1. The desired wine result has been returned.

Trigger:

1. From the mainActivity view the user types the desired wine name in the search field and clicks on the wine.
2. From the mainActivity view the user types the desired wine name in the search field and clicks

on “search”

Workflow:

1. mainActivity.java's onCreate() method creates the search view for the user to enter text into.
2. Once the user has entered at least three letters into the search, onQueryTextChange() calls wineManager to attempt to find previously drank wines by the passed text.
3. WineManager.java calls DAO.java's getWineByName() method to query the database for wines.
4. DAO searches the local wine database and returns any valid wines.
5. OnQueryTextChange() updates the view to display cards representing each wine by repeated calls to the
   constructor of displaySearchCard.java to display the wines.
6. Each card calls setOnClickListener() to begin listening for a click.
7. If a card is clicked by the user, onClick() creates a new intent to display the wine profile.

Alternate Paths:

1. mainActivity.java's onCreate() method creates the search view for the user to enter text into.
2. At any point the user can click the “search” button to call wineManager's downloadWineByName() to search
   the online database.
3. downloadWineByName() calls the DAO's downloadWineByName() method to return all wines from the online
   database which match the search text.
4. mainActivity.java then calls displaySearchActivity.java to display the results of the search.
5. displaySearchActivity.java calls displaySearchController, which calls the putWineInfoOnCards() method of

displaySearchManager.
6. putWineInfoOnCards() sets the cards through repeated calls to the constructor of displaySearchCard.java
7. DisplaySearchView is called to set the list view of cards
8. Each card calls setOnClickListener() to begin listening for a click.
9. If a card is clicked by the user, onClick() creates a new intent to display the wine profile.

Options:

1. The user shall be able to search specifically by name, vintage, producer, or varietal.

Description:
This design use case outlines the implementation details of how the wine intake is inputted to the system.

Desired Outcome:
The wine intake a correctly recorded and maintained in the system.

User Goals:
The user wants to add the amount of wine that he/she has consumed, to be able to keep the account of his or her number of drinks and BAC level.

Dependency Use Cases:

LA1

CAW1

CAW3

Preconditions:

1. The user has searched for a wine.
2. The user has selected a specific wine.

Postconditions:

1. The input has correctly been stored.

Trigger:

1. From the wineInfoPage view, the user clicks on the “Drink+” button.

Workflow:

1. The User has selected a wine and is viewing the wineInfoPage view.

2. The user clicks the “Drink+” button.

3. The “Drink+” button is listening to the click, and upon the user click, the onClick method calls BAC.java to add the new drink.

4. Before the drink is added, the user's settings are checked, and the BAC calls setGlassSize() and either setConservative() or unsetConservative() to set the BAC formula.

5. BAC.java updates the user's BAC.

6. The system toasts the user the updated BAC.

Alternate Paths:

A standAlone BAC activity is started, if the user wanted to input the Total Wine Intake that has been consumed at once.

Options:

1. The User shall add counts of his or her drinks at once in the BAC Stand Alone system.

Design Use Case #8 - “Change Glass Size” [CAW3]

Description:
This design use case outlines the implementation details of how the the glass size is changed to capture the different modes that wine is consumed.

Desired Outcome:
The glass size (sip size, regular 5 oz. size, or full (> 5oz size) has been updated.

User Goals:
The user wants to change the glass size to better reflect his/her way of drinking wine.

Dependency Use Cases:

LA1

CAW1

Preconditions:

1. The user has requested the BAC system to be activated.
2. The user has selected a specific wine.

Postconditions:

1.The glass size is correctly updated.

Trigger:

1. From the main_activity view the user swipes left to bring up the sliding menu.

2. The user clicks on the User Settings system, and changes the glass size..

Workflow:

1. In mainActivity.java, a new intent is initialized with the UserSettingsActivity.class.
2. The created intent is passed into the startActivity() method, to start the UserSettings Activity.
3. User Settings.java displays buttons with three different glass sizes.
4. Upon selecting the size, the onClick() method calls BAC.java to set the drink size to the clicked one.
5. The variable drink_size in BAC.java is updated.
6. Further calculation of BAC takes this value into account.
7. The view is updated with the new glass size.

Alternate Paths:

N/A

Options:

1. The User shall add counts of his or her drinks at once in the BAC Stand Alone system.

Description:
This design use case outlines the implementation details of a selected wine profile.

Desired Outcome:
The profile of selected wine is display on the view.

User Goals:
The user wants to see a specific wine profile that contains the image, some information about the wine, and tasting notes.

Dependency Use Cases:

LA1

CAW1

Preconditions:

1. The user has done a successful wine search.
2. The user has clicked on a specific wine.

Postconditions:

1. The screen shows the profile of selected wine.

Trigger:

1. The user shall long click a specific wine from the list of search results.

Workflow:

1. WineInfoActivity.java's onCreate() method calls WineInfoView.java to set the view.
2. WineInfoView calls WineInfoController.java, which calls the get...() methods of WineInfoManager to populate the view with the information.
3. WineInfoController.java calls the initmViewPager() method to display the default WineOverviewFragment beneath the basic information.
4. WineOverViewFragment's onCreate() method calls WineInfoManager's getDetailedWine() method to obtain the wine overview description.
5. The fragment is then displayed beneath the basic information.
6. The user can swipe the fragment right or left to move to a different fragment and view other information.

Alternate Paths:

N/A

Options:

N/A

Description:
This design use case outlines the implementation details of viewing a specific wine's related notes.

Desired Outcome:
The tasting notes about the current wine is displayed.

User Goals:
The user wants to see tasting notes regarding a specific wine that he or she has selected.

Dependency Use Cases:

LA1

CAW1

CAW4

Preconditions:

1. The user has done a successful wine search.
2. The user has clicked on a specific wine.
3. The user is on a specific wine profile.

Postconditions:

1. The screen shows the wine's notes.

Trigger:

1. The user shall swipe to the notes fragment.

Workflow:

1. WineInfoController.java calls the initmViewPager() method to display the WineNotesFragment beneath the

basic information.

2. WineNotesFragment's onCreate() method calls WineInfoManager's getDetailedWine() method to obtain the

wine's tasting notes.

3. The fragment is then displayed beneath the basic information.

4. The user can swipe the fragment right or left to move to a different fragment and view other information.

Alternate Paths:

N/A

Options:

N/A

Design Use Case #11 : “Edit Wine Notes”

Description:
This design use case outlines the implementation details of edit the wine notes.

Status: Deferred

Desired Outcome:
The system should update the user’s Notes for the wine.

User Goals:
The user wants to edit his or her own wine notes.

Dependency Use Cases:

CAW5

Preconditions:

1. The user shall choose a wine.
2. The system shall already in the view wine profile page (WineInfo.xml).

Postconditions:

1. The system shall show all modified notes in the notes area.

Trigger:

1. The user should double click on the wine notes area to call setUserComments function in the

userManager.java.

Workflow:

1. The userManager.java call the function in the DAO.java
2. DAO.java allowed user to add string, and modify the the info in the database
3. After update the new notes, the user click the OK button and the DAO.java call the userManager.java to display the wine notes

Alternate Paths:

N/A

Options:

N/A

Design Use Case #12 : “View Wine Information”[CAW7]

Description:
This design use case outlines the implementation details for the user to view a wine information.

Desired Outcome:
The system shall display a simple description of the specific wine.

User Goals:
The user wants to view the wine profile.

Dependency Use Cases:

CAW1

CAW4

Preconditions:

1. The user has searched for a wine.
2. The user has selected a wine and is on the wine's profile page.

Postconditions:
1. The system has displayed the wines information.

Trigger:

The user swipes to the basic information fragment.

Workflow

1. WineInfoController.java calls the initmViewPager() method to display the WineOverviewFragment beneath the

basic information.

2. WineOverviewFragment's onCreate() method calls WineInfoManager's getDetailedWine() method to obtain the

wine's basic overview information.

3. The fragment is then displayed beneath the basic information.

4. The user can swipe the fragment right or left to move to a different fragment and view other information.

Alternate Paths:

N/A

Options:

N/A

Design Use Case #13 : “Display Tagging Info” [CAW7]

Description:
This design use case outlines the implementation details for the user to see tagging info pulled from a wine database that responds with the wine.

Status: Deferred

Desired Outcome:
The system shall display tags (i.e. white wine, red wine) that correspond with the selected wine.

User Goals:
The user want to see the tag of the wine.

Dependency Use Cases:

CAW1

CAW7

Preconditions:

User selects a specific wine in the Choose-A-Wine System.

Postconditions:

User is able to see tags pulled from a wine database that are associated with the selected wine from the Choose-A-Wine System.

Trigger:

N/A

Workflow:

1. WineInfo.java calls WineManager.java to call DAO.java to obtain the tagging information.
2. DAO.java calls WineDAO.java to pair the wine with the database and get the tags information.
3. WineInfo.java displays the tags for the selected wine.

Alternate Paths:

N/A

Options:

N/A

Use Case #14: “Create new Tags” [CAW9]

Description:
This use case outlines the implementation details of create new tags from the wine profile screen.

Status: Deferred

Desired Outcome:
The system shall add new tag to database, and display the user’s new tag

User Goals:
The user wants to be able to create new tags for wines.

Dependency Use Cases:

CAW3

Preconditions:

1. The user has logged in to the system with a valid credentials.
2. The user is at a wine's wine profile.

Postconditions:

1. The system shall display the new wine tag.

Trigger:

1. The user shall click “create new tag”, fill in the info, then click “submit” button.

Workflow:

1. DisplaySearchActivity.java processes the search input and calls wine.java.
2. wine.java returns with basicWineInfo.xml activity and calls tags.java.
3. basicWineInfo.xml activity sends submitted data to tags.java in the form of parameters.
4. tags.java processes tag input and assigns to specified wine in wine.java.
5. DAO.java allows the user to add strings and will modify the information in the database.
6. After updating with new notes, the user clicks the OK button and DAO.java will call userManager.java to display the wine notes.

Alternate Paths:

N/A

Options:

N/A

Design Use Case #15: “Display popular tags” [CAW10]

Description:
This design use case outlines the implementation details of how the user access popular tags inputted by the users for a specific wine.

Status: Deferred

Desired Outcome:
The most popular tags for a specific wine presented to the user.

User Goals:
The user wants to view the most popular category tags of selected wine in the database.

Dependency Use Cases:

CAW2

Preconditions:

1. User must choose a specific wine from the Choose-A-Wine screen.

Postconditions:

1. User shall be able to see popular tags inputted by the user that are associated with the selected wine from Choose-A-Wine System

Trigger:

1. The user chooses a specific wine.

Workflow:

1. Tags.class calls DAO.class to query the most common adjective for the selected wine.
2. Tags.class returns those adjectives to basicWineInfo.xml.
3. basicWineInfo.xml displays those adjectives as tags.

Alternate Paths:

N/A

Options:

N/A

Use Case #16: “Pair Wine to Food” [WP1]

Description:
This use case outlines the implementation details of pairing foods based on wine.

Desired Outcome:
The user will be shown appropriate pairings of foods with their selected wine.

User Goals:
The user wants to pair foods based on selected wine.

Dependency Use Cases:

N/A

Preconditions:

1. The user has searched for a wine.
2. The user is on the wine's profile page.

Postconditions:

1. The system shall display the food pairings for the specific wines.

Trigger:.

1. The user swipes to the food pairings fragment.

Workflow:

1. WineInfoController.java calls the initmViewPager() method to display the WineFoodFragment beneath the

basic information.

2. WineFoodFragment's onCreate() method calls WineInfoManager's getDetailedWine() method to obtain the

wine's food pairing information.

3. A staggeredGridView is created to display the food pairings and an image for each food.

4. The user can swipe the fragment right or left to move to a different fragment and view other information.

Alternate Paths:

N/A

Options:

N/A

Use Case #17: “Pair Food to Wine” [WP2]

Description:
This use case outlines the implementation details of pairing wines based on foods.

Status: Deferred

Desired Outcome:
The user will be shown appropriate pairings of wines with their selected foods.

User Goals:
The user wants to pair wines based on selected foods.

Dependency Use Cases:

N/A

Preconditions:

1. The user shall have valid food name information and shall have a valid account in User Database.

Postconditions:

1. The system shall return wine information on the screen.

Trigger:

1. The user shall enter food's name and click submit.

2. The user shall click on the option to “Pair Foods”.

Workflow:

1. pair.xml is presented to the user.
2. wineManager.java processes the search input and calls wine.java to find matching food pairs with selected input.
3. pair.xml is presented to the user with corresponding foods to wine.

Alternate Paths:

1. wineManager.java determines that search input for foods name is invalid and returns an error message.

Options:

N/A

Use Case #18: “Return from Pairing” [WP3]

Description:
This use case outlines the implementation details of return the pairing from database after user submit his choice.

Status: Deferred

Desired Outcome:
The application will return and display the appropriate pairing of wine and foods to the user.

User Goals:
The user shall be able to retrieve data from database.

Dependency Use Cases:

N/A

Preconditions:

1. The user shall have valid food name information and shall have a valid account in User Database.

Postconditions:

1. The system shall return wine information on the screen.

Trigger:

1. The user shall enter food's name and click submit.

2. The user shall click on the option to “Pair Foods”.

Workflow:

1. DisplaySearchActivity.java processes the search input and calls wine.java.
2. wine.java returns basic wine info and calls pair.java to find matching food pairs with selected input.
3. pair.xml is presented to the user with corresponding foods to wine.

Alternate Paths:

1. DisplaySearchActivity determines that search input for wine name is invalid and returns an error message.

Options:

N/A

Design Use Case #19 - “Access Build-A-Wine System” [BAW1]

Description:
This design use case outlines the implementation details of building a wine (search for a wine).

1. Display the most popular wines initially.
2. Change the results dynamically.
3. Display the best matches.

Status: Deferred

Desired Outcome:
1. The most popular wines in the database is displayed on the view.
2. The search results change dynamically.
3. Display the best matches.

User Goals:
The user wants to search for a wine using category.

Dependency Use Cases:

LA1

Preconditions:

The category types exist in the database.

Postconditions:

1. The wines most relevant to user’s input has been returned.

Trigger:

1. The user shall click on the Build-A-Wine button on side menu.

Workflow:

1. androidManifest.xml is presented to the user.
2. In mainActivity.java, a new intent is initialized with buildAWineActivity.class
3. buildAWine.class processes the user’s input tags.
4. buildAWine.class calls wineManager.class to query for the best matches in local database. Else, getWineByName's request is sent as a URL to the online database.
5. If fetched results are obtained locally, then they are formatted and are placed in an array list of Wine type, an instance of class Wine.java.
6. The JSON file is passed to the parseXML in the activity to be stripped and create an instance of class wine.java.
7. All related attributes of the wine object is set.
8. The View is updated dynamically with the ArrayList<wine> that was filled in the last steps.

Alternate Paths:

N/A

Options:

1. The user can return to the Build A Wine page.

Design Use Case #20 - “Access Search Guide” [BAW2]

Description:
This design use case outlines the implementation details of how the user access the search guide for the “Build-A-Wine” System.

Status: Deferred

Desired Outcome:
The category tags of the wine display on screen.

User Goals:
The user wants to search for a wine using the category tags.

Dependency Use Cases:

LA1

BAW1

Preconditions:

The wine associated to the categories exist in the database.

Postconditions:

1. Next possible wine category tags has been displayed.

Trigger:

1. The user shall click on the build-A-Wine button on side menu.
2. The user shall click on one of the tags displayed on screen.

Workflow:

1. androidManifest.xml is presented to the user.
2. In mainActivity.java, a new intent is initialized with buildAWineActivity.class.
3. buildAWine.class displays the most broadest category tags on screen.
4. buildAWine.class processes the user’s input tags.
5. buildAWine.class displays next possible category tags on screen.
6. buildAWine.class calls wineManager.class to query for the best matches in local database. Else, getWineByName's request is sent as a URL to the online database.
7. If fetched results are obtained locally, then they are formatted and are placed in an array list of Wine type, an instance of class Wine.java
8. The JSON file is passed to the parseXML in the activity to be stripped and create an instance of class wine.java.
9. All related attributes of the wine object is set.

Alternate Paths:

N/A

Options:

1. The user can return to the Build A Wine page.

Design Use Case #21 - “Access Randomize Button” [BAW3]

Description:
This design use case outlines the implementation details of how the user access the randomize button in the Side Menu

Desired Outcome:
A randomly selected wine from database is displayed on screen.

User Goals:
The user wants to picked a wine from database without any input.

Dependency Use Cases:

LA1

BAW1

Preconditions:

The user has accessed the Side Menu

Postconditions:

1. A randomly selected wine is displayed on screen.

Trigger:

The user shall click on the Random Button on side menu.

Workflow:

1. Any activity in the application is listening to the right long swipe gesture on the screen.

2. Upon each swipe-right gesture, the method getLength() form android Gesture API is calledIf the length of

swipe is long enough, a call to sideNavigation.java class is made. Else, the gesture is ignored.

3. The View is updated with the menu appearing on the right side inside the same activity that the user is on.

4. When the “Randomize” button is pressed, WineManager.java uses a random number generator to

to select a random wine based on WineID.

5. WineManager.java calls DAO.java, which calls WineDAO.java to query the database for the wine.

6. The WineInfo Activity is updated to display the info of the randomly selected wine.

Alternate Paths:

N/A

Options:

N/A

Design Use Case #22 : “Add Wine Notes”

Description:
This design use case outlines the implementation details of adding tasting notes.

Desired Outcome:
The system shall allow the user to add a tasting note for the selected wine.

User Goals:
The user wants to add his or her own wine notes.

Dependency Use Cases:

CAW5

Preconditions:

1. The user shall choose a wine.
2. The system shall already in the view wine profile page (WineInfo.xml).
3. The user has swiped to the tasting notes fragment.

Postconditions:

1. The system shall show all modified notes in the notes area.

Trigger:

1. The user should double-click on the wine notes area to call setUser Comments function in userManager.java.

Workflow:

1. WineNotesFragment.java creates an editText comment box using the editText constructor for the user to input text into.
2. The comment box creates an onEditorActionListener.
3. The listener waits for the user to input text into the box.
4. When the user enters text, an onClick listener is created for the “OK” button beneath the text field.
5. When the user is finished entering text and clicks the “OK” button, the onClick() method is called.
6. OnClick() calls onEditorAction() to store the new comment.
7. the onEditorAction() method of WineNotesFragment.java is called.
8. OnEditorAction() calls the setComment() method of userManager.java.
9. SetComment() adds the comment to the user's comments in the user database.
10. OnEditorAction() calls updateComments(), which refreshes the wine's list of comments, displaying the new comment.

Alternate Paths:

N/A

Options:

N/A

Description:
This use case outlines the implementation details of viewing all wines the user has drank in the past.

Desired Outcome:
The user will be able to see visual representations of each wine that they have drank in the past. By clicking on the representation, the user will be taken to the profile page of the wine.

User Goals:
The user wants to see a history of all past wines that they have drank, and wants to be able to view information on
all these wines.

Dependency Use Cases:

DH1

Preconditions:

The user has drank at least one wine.

Postconditions:

The system displays the representation of all past wines drank.

Trigger:

1. The user shall select the “Wine Cellar” button from the swipe menu.

Workflow:

1. WineCellar.java's onCreate() starts the activity.
2. WineCellar.java calls WineCellarController.java to fetch the user's drank wines.
3. WineCellarController.java calls getWines() and getWinesImages() to obtain the user's drank wines and the images of each wine.
4. WineCellarController.java calls WineCellarManager.java's setUpandStartAct() method to display the information.
5. setUpandStartAct() calls WineCellarView.java to set up the view to display the representation of each wine as a clickable object.
6. WineCellarView.java calls setGridView() to set the StaggeredGridView.
7. If a wine's button is clicked, WineCellarManager's onItemClick() method passes the information to wineProfile to display the profile of the selected wine.

Alternate Paths:

N/A

Options:

N/A

Design Use Case #24 - “Enable Wine Session” [DS1]

Description:
This use case outlines the implementation details of the user enabling a new wine session.

Status: Deferred

Desired Outcome:
The user will be able to begin a new wine drinking session.

User Goals:
The user wants start a new wine drinking session to monitor the length of the session and the amount of wine consumed (in mL), as well as record their wine notes.

Dependency Use Cases:

N/A

Preconditions:

N/A

Postconditions:

1. The system shall display the details of the current wine session details to the user.

Trigger:

1. The user shall enable a wine session.

Workflow:

1. WineSession.java is called to enable a new wine drinking session.
2. WineSession.java calls WineInfo.java to query and fetch information regarding the wine the user is drinking.
3. WineSession.java processes this information and appropriate stores it into variables, such as wineName, country, varietal, alcoholContent, et cetera.
4. A new activity is created to display a timer for the user to monitor session length.
5. This activity will also display information specific to the wine the user is currently drinking.

Alternate Paths:

1. If, in SettingsActivity, “Dynamic BAC” is enabled, the background of SplashScreen will become a live background to reflect the user's current BAC.

Options:

1. In SettingsActivity, Dynamic BAC animations can either be enabled or disabled.
2. In SettingsActivity, the flow of the Dynamic BAC animations can be reversed.

Description:
This use case outlines the implementation details of accessibility of user profile on the application. The user will be able to view their basic profile information.

Desired Outcome:
The user shall be able to access their user profile and view their data.

User Goals:
The user wants to have access to view their user profile and data.

Dependency Use Cases:

N/A

Preconditions:

1. The user shall have valid credentials and shall have a valid account in User Database.
2. The user shall be logged in.

Postconditions:

1. The System displays the user's profile information.

Trigger:

1. The user swipes to the right to access the slide menu.

2. The user clicks the “User Profile” button in the menu.

Workflow:

1. DrawerItemController.java sets the view for the user's profile information.
2. DrawerItemController sets and displays the “Cancel” and “OK” buttons.
3. DrawerItemController creates an alertDialog box which calls UserManager() to obtain the user's profile information.
4. The alertDialog box displays the user's profile information.
5. The Dialog box sets onClick listeners for the “Cancel” and “OK” buttons.
6. When either button is clicked, the onClick() method is called to exit the display.

Alternate Paths:

N/A

Options:

N/A

Use Case #26: “Edit User Profile” [SET2]

Description:
This user case outlines the implementation details of editing the user profile on the application. The user will be able to choose to edit details pertaining to their profile such as view restrictions, their history of wine notes, basic profile information, user ratings, and photos.

Status: Deferred

Desired Outcome:
The user shall be able to access their user profile and modify their data.

User Goals:
The user wants to have access to view their user profile and data.

Dependency Use Cases:

N/A

Preconditions:

1. The user shall have valid credentials and shall have a valid account in the User Database.
2. The user shall be logged in.

Postconditions:

1. The system shall redirect the user to their editable profile page.

Trigger:

1. The user shall click on “Edit Profile” button on the “View Profile” screen.

Workflow:

1. androidManifest.xml presents a menu to the user.
2. androidManifest.xml directs user to the userProfile screen.
3. userManager.java loads the user’s details into the corresponding fields (Username, Password, Email, profile photo).
4. myWineHistory.java loads the user’s details into the corresponding fields (wine intake history, wine notes).
5. userProfile.xml redirects user to editUserProfile.xml.
6. editUserProfile.xml sends submitted data to userManager.java in the form of parameters.
7. userManager.java confirms submitted data and commit the changes.

Alternate Paths:

N/A

Options:

N/A

Description:
This use case outlines the implementation details of changing various settings of the app. The user will be able to edit general settings of the application, such as modifying view restrictions, options on the main menu, and changing the background.

Desired Outcome:
The user shall be able to change various settings of the application.

User Goals:
The user wants to have access to view their user profile and data.

Dependency Use Cases:

N/A

Preconditions:

1. The user has clicked on “Settings” on the swiped menu.

Postconditions:

1. The system has updated the user's settings.

Trigger:

1. The user shall click on the “Settings” button in the swipe menu.

Workflow:

1. The system creates an OnPreferenceChangeListener for each settings preference.
2. When the user selects a setting to edit, OnPreferenceChange() is called to edit the preference.
3. The user inputs the new information for the selected setting.
4. OnPreferenceChange() calls bindPreferenceSummaryToValue(), which calls the preference manager to save the new value for the setting.
5. Once the new setting has been saved, the system redirects the user to their previous page.

Alternate Paths:

N/A

Options:

N/A

Description:
This use case outlines the implementation details of the Fizzy Bubbly Bubble Pop Game.

Status: Implemented

Desired Outcome:
The user will have fun playing the Fizzy Bubbly Bubble Pop Game.

User Goals:
The user wants to have fun with some sort of unorthodox, hidden app functionality.

Dependency Use Cases:

N/A

Preconditions:

1. The user has opened the swipe menu.

Postconditions:

1. The user will have fun.

Trigger:

1. The user has selected the “Don't Press Me” button in the menu.

Workflow:

1. The onCreate method of the game activity creates the game objects by calling the constructor of AndroObjectView.java
2. The onCreateMethod creates the Linear Layout for the view.
3. Each time the screen is clicked, the activity creates a new AndroObject.
4. The objects slowly scroll across the screen, and the user can add as many as he or she desires.

Alternate Paths:

N/A

Options:

N/A