Wine Information Network

Design Use Cases

Authors

Shannon Harris – Project Manager

Eliah Overbey – Senior Systems Analyst

Michael Nguyen - Software Development Lead

Andrew Hwang - Database Specialist

Brian La - Quality Assurance Lead

David Smith - Software Architect

Rex West - Subject Matter Expert

Derek Chan – Algorithm Specialist

Kyle Gullicksen - Algorithm Specialist

Edward Wang – User Interface Specialist

Compiled by: Michael Nguyen

Compilation Date: 6/8/2013

Last Updated: 6/8/2013

Table of Contents

Use Case Information Tables

	Use Case Groupings
ISD	Design use case for Inventory System
RSD	Design use case for Recommender System
IRSD	Design use case for both Inventory and Recommender Systems
FPD	Design use case is for the Front Pages, implemented before the user chooses to access their inventory or recommendations
IWPD	Design use case for Specific Wine Pages that can be accessed from both the inventory system and the recommender system

	Priority
1	Highest Priority – Vital to Application
2	Medium Priority – Desirable, but not essential
3	Low Priority – Implement time permitting

	Progress Status
Planning	Use case is being considered for project.
Designing	Details of use case are being determined.
Developing	Use case is being programmed.
Testing	Use case is being tested to ensure it is working as designed.
Completed	Use case is done. There is no more work to do.
Pending	Development of use case is on hold.

	ISD-1: Add a Wine
Description:	This Use Case outlines the implementation details of a User adding a Bottle of Wine to their Inventory.
Desired Outcome:	The outcome of this Use Case is to add a Bottle of Wine to a User's Inventory.
User Goals:	The User wants to be able to add a specific Bottle of Wine to their Inventory, wherein: 1. Entering information about the Bottle of Wine (i.e. Wine Name, Wine Type, Varietal, Vintage, Quantity). 2. Entering information about the User's interpretation of the Bottle of Wine (i.e. Rating, Recommendation, Notes, Tasting Attributes). 3. Choosing a Storage Location for that Bottle of Wine, 4. Auto-completing information about the wine by selecting a matching wine in the database.
Primary Actor:	User of Application
Dependency Use Cases:	User Login – UC17
Requirements:	HP-2 Login to System HP-3 Create Account
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The User shall have created an Account. 2. The User shall be authenticated via that Account with the Application. 3. The Account shall have at least one Storage Location, and at least one Storage Unit within that Storage Location.
Post-conditions:	1. The User's Inventory shall reflect the addition of the Bottle of Wine.
Trigger:	1. From inventoryView.html the user fills the wine form and selects the "Add Wine" button
Workflow:	1. eventHandler.js posts the input form to /Inventory/addWine in Routes.py 2. Routes.py retrieves the infomation and verifies it 3. Routes.py calls inputAddWineUser() in InventoryIOController.py 4. InventoryIOController.py checks the input for problems and finds none 5. InventoryIOController.py calls addWineUser() in InventoryController.py 6. InventoryController.py calls dbAddWineUser() in InventoryModelAccess.py 7. InventoryModelAccess.py executes the SQL command to add an entry

	to the inventory based on the input 8. InventoryModelAccess.py sends a confirmation message to InventoryController.py if the entry was properly added 9. InventoryController.py sends the message to InventoryIOController.py 10. InventoryIOController.py sends the response to Routes.py 11. Routes.py renders inventoryView.html
Alternate Paths:	1. Routes.py, InventoryIOController.py, InventoryController.py, or InventoryModelAccess.py find an error 2. An error response is passed up the chain to inventoryView.html 3. inventoryView.html refreshes the page

	ISD-2: Add a Storage Location
Description:	This Use Case outlines the implementation details of a User adding an Inventory.
Desired Outcome:	The outcome of this Use Case is to add an Inventory Location to a User's profile.
User Goals:	The user wants to be able to add a new inventory based on location.
Primary Actor:	User of Application
Dependency Use Cases:	User Login – UC17
Requirements:	HP-2 Login to System HP-3 Create Account
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The User shall have created an Account. 2. The User shall be authenticated via that Account with the Application.
Post-conditions:	1. The User shall have a new Inventory based on a location with which he/she can add wines.
Trigger:	1. From inventoryView.html the user fills the inventory form and selects the "Add Inventory" button.
Workflow:	1. eventHandler.js posts the input form to /Inventory/addInventory in Routes.py 2. Routes.py retrieves the information and verifies it 3. Routes.py calls inputAddInventory() in InventoryIOController.py 4. InventoryIOController.py checks for problems and finds none

	5. InventoryIOController.py calls addInventory() in InventoryController.py. 6. InventoryController.py calls dbAddInventory in InventoryModelAccess.py. 7. InventoryModelAccess.py executes the SQL command to give the user a new inventory ID number. 8. InventoryModelAccess.py sends a confirmation message to InventoryController.py if the inventory was properly created. 9. InventoryController.py sends the message to InventoryIOController.py. 10. InventoryIOController.py sends the response to Routes.py. 11. Routes.py renders inventoryView.html
Alternate Paths:	1. Routes.py, InventoryIOController.py, InventoryController.py, or InventoryModelAccess.py find an error 2. An error response is passed up the chain to inventoryView.html 3. inventoryView.html refreshes the page

	ISD-3: Edit a Storage Location
Description:	This Use Case outlines the implementations details of a User editing an Inventory.
Desired Outcome:	The outcome of this Use Case is to edit a User's Inventory.
User Goals:	The User wants to be able to change the details of an Inventory (Name, Info, etc)
Primary Actor:	User of Application
Dependency Use Cases:	Add a Storage Location / Storage Unit – UC2 User Login – UC17
Requirements:	Misc-1: Modify Storage Attributes
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The User shall have created an Account. 2. The User shall be authenticated via that Account with the Application. 3. The Account shall have at least Inventory Location.
Post-conditions:	1. The User's Inventory shall reflect the changes to the Inventory.
Trigger:	1. From inventoryView.html the user shall select the edit icon attached to an Inventory Location.
Workflow:	1. eventHandler.js reveals a form model for editing Inventory

	2. The user fills the form and selects the “Submit” button. 3. eventHandler.js posts to /Inventory/editInventory in Routes.py 4. Routes.py retrieves the information and verifies it 5. Routes.py calls inputEditInventory() in InventoryIOController.py. 6. InventoryIOController.py checks for problems and finds none 7. InventoryIOController.py calls editInventory() in InventoryController.py 8. InventoryController.py calls dbEditInventory() in InventoryModelAccess.py 9. InventoryModelAccess.py changes the database according to the changes 10. InventoryModelAccess.py sends a confirmation message to InventoryController.py 11. InventoryController.py sends the message to InventoryIOController.py 12. InventoryIOController.py sends the message to Routes.py 13. Routes.py renders inventoryView.html
Alternate Paths:	1. Routes.py, InventoryIOController.py, InventoryController.py, or InventoryModelAccess.py find an error 2. An error response is passed up the chain to inventoryView.html 3. inventoryView.html refreshes the page

	ISD-4: Editing Entries
Description:	This Use Case outlines the implementation details of the User editing an entry in their Inventory.
Desired Outcome:	Edited parts of the entry will be saved to the database, while unedited parts remain unchanged.
User Goals:	To edit desired parts of an entry in their database, including the default picture
Primary Actor:	User of Application
Dependency Use Cases:	Add a Wine – UC 1 User Login – UC 17
Requirements:	Misc-1: Modify Storage Attributes Misc-2: Modify Wine Attributes
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The User is logged in. 2. The User has entries to edit. 3. Any detail included when adding a wine can be edited.

Post-conditions:	1. The only parts of the entry that are modified are those specified by the User.
Trigger:	1. From inventoryView.html the user shall select the edit icon attached to a wine.
Workflow:	1. eventHandler.js reveals the edit wine form 2. The user fills the form and selects the "Submit" button. 3. eventHandler.js posts the form to /Inventory/editEntry in Routes.py 4. Routes.py retrieves the information and verifies it 5. Routes.py calls inputEditEntryUser() in InventoryIOController.py. 6. InventoryIOController.py checks for problems and finds none 7. InventoryIOController.py calls editEntryUser() in InventoryController.py 8. InventoryController.py calls editEntryUser() in InventoryModelAccess.py 9. InventoryModelAccess.py changes the database according to the changes 10. InventoryModelAccess.py sends a response message to InventoryController.py 11. InventoryController.py sends the message to InventoryIOController.py 12. InventoryIOController.py sends the message to Routes.py 13. Routes.py renders inventoryView.html
Alternate Paths:	1. Routes.py, InventoryIOController.py, InventoryController.py, or InventoryModelAccess.py find an error 2. An error response is passed up the chain to inventoryView.html 3. inventoryView.html refreshes the page

	ISD-5: Importing/Exporting Inventory
Description:	This Use Case outlines the implementation details of the User importing/exporting an Inventory from an external source like XML or CSV.
Desired Outcome:	The User can obtain a copy of their Inventory that can be accessed offline, or can upload their Inventory
User Goals:	To obtain an offline copy of their Inventory or upload a copy of their Inventory to the service
Primary Actor:	User of Application
Dependency Use Cases:	User Log In – UC 16
Requirements:	HP-2: Login to System HP-3: Create Account S/R-1: Store Wine Bottle/Name

	S/R-2: Store Wine Attributes S/R-3: Store Storage Location
Details:	Priority Level: 3 Status: Pending
Pre-conditions:	1. User is logged in. 2. User has entries in his/her Inventory OR 3. User has a copy of his/her Inventory to be uploaded.
Post-conditions:	1. An offline copy of the Inventory is made available for the User to view (as an XML, CSV, or text file) OR 2. The uploaded copy is integrated into the User's Inventory in the service.
Trigger:	1. The "Download inventory" button is selected from inventoryView.html OR 2. The "Upload an Inventory" button is selected from inventoryView.html
Workflow:	1. eventHandler.js posts to /Inventory/downloadXML in Routes.py 2. Routes.py calls inputDownloadXML() in InventoryIOController.py 3. InventoryIOController.py calls downloadXML() in InventoryController.py 4. InventoryController.py calls dbDownloadXML() in InventoryModelAccess.py 5. InventoryModelAccess.py pulls all wines and wine info from the inventory and formats in XML form 6. InventoryModelAccess.py sends the results to InventoryController.py 7. InventoryController.py sends the results to InventoryIOController.py 8. InventoryIOController.py sends the results to Routes.py 9. Routes.py downloads the XML file to the user's computer.
Alternate Paths:	1. The user is prompted with a windows explorer to select the XML file to upload. 2. Routes.py responds to the input and calls inputUploadXML() in InventoryIOController.py 3. InventoryIOController.py calls uploadXML() in InventoryController.py 4. InventoryController.py calls dbUploadXML() in InventoryModelAccess.py 5. InventoryModelAccess.py uploads all the wines to the database from the XML file. 6. InventoryModelAccess.py sends the results to InventoryController.py 7. InventoryController.py sends the results to InventoryIOController.py 8. InventoryIOController.py sends the results to Routes.py

	9. Routes.py renders inventoryView.html

	ISD-6: Removing Entries
Description:	This Use Case outlines the implementation details of a User removing a particular wine in the Inventory.
Desired Outcome:	Entries specified by the User are removed from the User's Inventory.
User Goals:	To remove wines they no longer have in their inventory.
Primary Actor:	User of Application
Dependency Use Cases:	User Log In – UC 16
Requirements:	HP-2: Login to System HP-3: Create Account S/R-1: Store Wine Bottle/Name S/R-2: Store Wine Attributes S/R-3: Store Storage Location Di-1: Display Wine Attributes
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The User is logged in. 2. The User has entries in his/her Inventory. 3. The User has entries he/she wishes to remove from the Inventory.
Post-conditions:	1. The specified entries are removed from the Inventory. 2. Other items remain unchanged in the Inventory. 3. The specified entries are not removed from the general database.
Trigger:	1. The user clicks on the delete icon attached to a wine in inventoryView.html
Workflow:	1. eventHandler.js reveals a confirmation model 2. The user clicks the accept button 3. eventHandler.js posts to /Inventory/deleteWine in Routes.py 4. Routes.py retrieves information about the wine 5. Routes.py calls inputDeleteWineUser() in InventoryIOController.py 6. InventoryIOController.py calls deleteWineUser() in InventoryController.py 7. InventoryController.py calls dbDeleteWineUser() in InventoryModelAccess.py 8. InventoryModelAccess.py deletes the wine from the inventory

	9. InventoryModelAccess.py sends the results to InventoryController.py 10. InventoryController.py sends the results to InventoryIOController.py 11. InventoryIOController.py sends the results to Routes.py 12. Routes.py renders inventoryView.html
Alternate Paths:	None

	ISD-7: Viewing Statistics
Description:	This Use Case outlines the implementation details of the User's ability to view detailed statistics about his/her Inventory or specific wines in his/her Inventory.
Desired Outcome:	Information about the inventory or the specific wine the user wants to see is displayed.
User Goals:	To view various statistics about the inventory (% of wines that are red, white, dry, sweet, etc.) or of a particular wine (vintage, winery, price, etc.)
Primary Actor:	User of Application
Dependency Use Cases:	User Log In - UC16
Requirements:	HP-2: Login to System HP-3: Create Account PP-1: View Profile S/R-1: Store Wine Bottle/Name S/R-2: Store Wine Attributes S/R-3: Store Storage Location
Details:	Priority Level: 2 Status: Completed
Pre-conditions:	1. The User is logged in. 2. The User has wines in his/her Inventory.
Post-conditions:	1. No entries are modified
Trigger:	1. The user loads the User Profile model, activating the /User route in Routes.py
Workflow:	1. Routes.py calls inputViewStats() in InventoryIOController.py for all Locations owned by the user 2. InventoryIOController.py calls viewStats() in InventoryController.py 3. InventoryController.py calls various data retrieval methods in InventoryModelAccess.py 4. InventoryModelAccess.py gathers needed wine info from the

	inventory database 5. InventoryModelAccess.py sends the results to InventoryController.py 6. InventoryController.py process the wines and forms statistics 7. InventoryController.py sends the results to InventoryIOController.py 8. InventoryIOController.py sends the results to Routes.py 9. Routes.py renders the Profile model
Alternate Paths:	None

	ISD-8: Filter Inventory
Description:	This Use Case outlines the implementation details of the ability for the User to filter their wine Inventory by selecting from various attributes of wine (i.e. dryness, sweetness, date added, location).
Desired Outcome:	The user shall be able to filter their wines in their inventory by certain attributes, such as dryness, sweetness, date added, location, etc.
User Goals:	The user wants to filter their wines by attributes.
Primary Actor:	User of Application
Dependency Use Cases:	User Login - UC16 Add to Inventory – UC1
Requirements:	S/S-1: Filter Wines S/S-2: Filter Wines by Attribute
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The User is logged in. 2. There exist items in the Inventory.
Post-conditions:	1. The system shall filter the wines in the specified attribute requested. 2. The system shall display the wines specified.
Trigger:	1. The User clicks on the inventory button to view inventoryView.html. 2. The User chooses which attribute to filter.
Workflow:	1. eventHandler.js uses jquery to hide wines without the selected attribute 2. Isotopes.js rearranges the web page to present the remaining wines
Alternate Paths:	

	ISD-9: Archive/History of Inventory
Description:	This Use Case outlines the implementation details of the ability for the User to view their archive/history of their Inventory, so they can see the transactions that have been made in their Inventory, like added and removed wines.
Desired Outcome:	The User shall be able to view their archive/history of added or removed wines from their Inventory.
User Goals:	The User wants to view their archive/history of their wine Inventory.
Primary Actor:	User of Application
Dependency Use Cases:	User Login - UC16 Add to Inventory – UC1
Requirements:	S/S-3: Wine History - Everything S/S-4: Wine History - Recently Added S/S-5: Wine History - Removed
Details:	Priority Level: 2 Status: Completed
Pre-conditions:	1. The User is logged in. 2. There have been transactions with the Inventory.
Post-conditions:	1. The system shall show the archive/history to the User.
Trigger:	1. The User opens the User Profile model, activating the /User route in Routes.py
Workflow:	1. ISD-7 occurs, and Routes.py receives Histories as part of the returned statistics 2. Routes.py sorts the histories by time 3. Routes.py renders the Profile model
Alternate Paths:	

	ISD-10: Search Inventory
Description:	This Use Case outlines the implementation details of the ability for the User to search from either the entire wine database or their wine Inventory for wines by name or attributes.
Desired Outcome:	The User shall be able to search for wines by name, red wine, white wine, etc. The User also wants to search for wines that are in their inventory.
User Goals:	The User wants to find the wines they are looking for.
Primary Actor:	User of Application
Dependency Use Cases:	User Login - UC16 Add to Inventory – UC1

Requirements:	S/S-6: Search Database for Wine - Name S/S-7: Search Database for Wine – Attri. S/S-8: Search Inventory for Wine – Name S/S-9: Search Inventory for Wine – Attri.
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The User is logged in. 2. The form content is valid. 3. There exist items in the Inventory if the User is searching from their Inventory.
Post-conditions:	1. The system shall search the wines in either the database or user Inventory. 2. The system shall display the wine(s) specified.
Trigger:	1. The User clicks on the inventory button to view inventory in inventoryView.html. 2. The User starts filling out the search form.
Workflow:	1. eventHandler.js searches the currently displayed wines for those having information matching the search expression 2. eventHandler.js hides all other wines 3. Isotopes.js rearranges the web page
Alternate Paths:	

	ISD-11: Wine Wishlist
Description:	This use case outlines the implementation details of a user adding wines to a wishlist of wines they would like to have.
Desired Outcome:	The user shall be able to view the wines on their wishlist and add more wines according to the procedure in ISD-1.
User Goals:	The user wants to keep track of wines they would like to have
Primary Actor:	User of Application
Dependency Use Cases:	None
Requirements:	TBD
Details:	Priority Level: 2 Status: Pending
Pre-conditions:	1. The user is logged in. 2. A wine wishlist exists
Post-conditions:	1. The system shall display the wines in the user's wishlist 2. The system shall display the wine's name, picture, and vintage

Trigger:	1. From inventoryView.html the user clicks on their wishlist
Workflow:	1. eventHandler.js posts to /Inventory/wishlist in Routes.py 2. Routes.py calls inputGetWishlist() in InventoryIOController.py 3. InventoryIOController.py calls getWishlist() in InventoryController.py 4. InventoryController.py calls dbGetWishlist() in InventoryModelAccess.py 5. InventoryModelAccess.py retrieves the user's wishlist 6. InventoryModelAccess.py sends the wines to InventoryController.py 7. InventoryController.py sends the wines to InventoryIOController.py 8. InventoryIOController.py sends the wines to Routes.py 9. Routes.py sends the wines in JSON form to eventHandler.js 10. eventHandler.js updates the DOM with the wines
Alternate Paths:	

	ISD-12: Inventory Security
Description:	This use case outlines the implementation details of a user choosing their level of privacy..
Desired Outcome:	The user shall be able to edit the security setting to select their level of privacy.
User Goals:	The user wants to have a security setting to limit who can view their inventory.
Primary Actor:	User of Application
Dependency Use Cases:	FP-2: Login Account
Requirements:	PP-4: Privacy Settings - Update Di-6: Display Privacy Settings
Details:	Priority Level: 3 Status: Pending
Pre-conditions:	1. The user is logged in.
Post-conditions:	1. The user shall have their inventory privacy setting set.
Trigger:	1. From inventoryView.html the user clicks on the edit security button
Workflow:	1. eventHandler.js reveals the Security Modal 2. The user selects their level of privacy 3. eventHandler.js posts to /Inventory/security in Routes.py 4. Routes.py calls inputSetSecurity() in InventoryIOController.py 5. InventoryIOController.py calls setSecurity() in InventoryController.py

	6. InventoryController.py calls dbSetSecurity() in InventoryModelAccess.py 7. InventoryModelAccess.py modifies the user's security setting 8. InventoryModelAccess.py returns the results to InventoryController.py 9. InventoryController.py sends the results to InventoryIOController.py 10. InventoryIOController.py sends the results to Routes.py 11. Routes.py renders inventoryView.html
Alternate Paths:	

	ISD-13: Rate and Review
Description:	This Use Case outlines the implementation details of the user's ability to rate and review a wine.
Desired Outcome:	The user shall have submitted a rating and review of the wine that the system now stores.
User Goals:	The user wants to rate and review a wine.
Primary Actor:	User of Application
Dependency Use Cases:	User Login – UC17
Requirements:	Di-2: Display User Ratings Di-3: Display User Reviews
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user is logged into the application
Post-conditions:	1. The system shall store the rating of the wine 2. The system shall store the review of that wine 3. The system shall make the rating and review available for others to see. 4. The system shall take the rating into account for recommendations(if applicable)
Trigger:	1. The user shall edit the form in a Wine Info modal and submit their rating and review.
Workflow:	1. ISD-4 occurs, saving the user's review and rating
Alternate Paths:	None

	ISD-14: Remove an Inventory Location
Description:	This Use Case outlines the implementation details of the user's ability to remove a location from their inventory.
Desired Outcome:	The user shall have removed an inventory location.
User Goals:	The user wants to remove an inventory location.
Primary Actor:	User of Application
Dependency Use Cases:	User Login – UC17
Requirements:	FP-2: Login Account IS-2: Add a Storage Location
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user is logged into the application. 2. The user is viewing the inventory page.
Post-conditions:	1. The location is removed
Trigger:	1. The user shall click the remove button attached to the Location icon.
Workflow:	1. eventHandler.js shows the Confirmation Modal 2. The user confirms 3. eventHandler.js posts to /Inventory/deleteInventory in Routes.py 4. Routes.py calls inputDeleteInventory() in InventoryIOController.py 5. InventoryIOController.py calls deleteInventory() in InventoryController.py 6. InventoryController.py calls dbDeleteInventory() in InventoryModelAccess.py 7. InventoryModelAccess.py deletes the Storage Location 8. InventoryModelAccess.py returns the results to InventoryController.py 9. InventoryController.py sends the results to InventoryIOController.py 10. InventoryIOController.py sends the results to Routes.py 11. Routes.py sends to results to eventHandler.js in JSON form 12. eventHandler.js updates the DOM to show the changes
Alternate Paths:	None

	RSD-1: View Recommendations
Description:	This Use Case outlines the implementation details of a recommendation system for wines the user may want to try, using what wines the User has seeded shelves with.
Desired Outcome:	The User shall be able to view recommended wines.
User Goals:	The User wants to discover new wines or decide what wine he or she should try.
Primary Actor:	The System
Dependency Use Cases:	RS-5: Create a Shelf
Requirements:	RS-1: Create Shelf RS-2: Export Shelf Seed RS-3: Create Shelf – Preexisting Seed RS-14: Generate Recommendations
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user is logged in. 2. The user has seeded a channel with a wine.
Post-conditions:	1. The system displays a list of wines as recommendations.
Trigger:	1. The user clicks the Recommend More button in recommenderView.html
Workflow:	1. eventHandler.js posts to /Recommend/getRecommendations in Routes.py 2. Routes.py calls inputRecommend() in RecommendIOController.py. 3. RecommendIOController.py calls recommend() in RecommendController.py 4. RecommendController.py calls dbGetCSOs() in RecommenderModelAccess.py to find all wine clusters. 5. RecommenderModelAccess.py returns wine clusters for each request from RecommendController.py 6. RecommendController.py determines the wine cluster closest to the seeds. 7. RecommendController.py calls dbGetWinesInCluster() in RecommenderModelAccess.py. 8. RecommenderModelAccess.py returns wines in the cluster. 9. RecommendController.py finds the wines closest to the seeds in the cluster. 10. RecommendController.py sends a list of wines to

	RecommendIOController.py 11. RecommendIOController.py passes the response to Routes.py 12. Routes.py returns the list in JSON form to eventHandler.js 13. eventHandler.js updates the DOM with the wines
Alternate Paths:	

	RSD-2: Shuffle Shelves
Description:	This Use Case outlines the implementation details of a user being shown wine that was recommended to them from one of their shelves. The shelf is chosen at random.
Desired Outcome:	The user is shown a random wine that was selected based on their preferences and history.
User Goals:	To be shown a wine the recommender thinks they will like with no preference for what shelf the wine is recommended from.
Primary Actor:	User of Application
Dependency Use Cases:	User Login – UC17
Requirements:	SR-32: Generate Recommendations RS-12: Recommender to Inventory S/S-3: Wine History – Everything S/R-5L: Store User Reviews S/R-4L: Store User Ratings
Details:	Priority Level: 2 Status: Pending
Pre-conditions:	1. The user is logged in 2. The user is on the shelves page
Post-conditions:	1. The user sees a bottle of wine from a random shelf and info about that wine
Trigger:	1. The user clicks the “Shuffle” button in recommendationView.html
Workflow:	1. eventHandler.js posts to /Recommend/shuffle in Routes.py 2. Routes.py calls inputShuffle() in RecommendIOController.py 3. RecommendIOController.py calls shuffle() in RecommendController.py 4. RecommendController.py calls dbShuffle() in RecommenderModelAccess.py 5. RecommenderModelAccess.py gathers needed wine info from the database 6. RecommenderModelAccess.py sends the results to

	RecommendController.py 7. RecommendController.py sends the results to RecommendIOController.py 8. RecommendIOController.py sends the results to Routes.py 9. Routes.py sends the results to eventHandler.js in JSON form 10. eventHandler.js modifies the DOM to show the wine
Alternate Paths:	None

	RSD-3: Buy Wine
Description:	This use case outlines the implementation details of a user buying a wine
Desired Outcome:	The user shall be able to buy a particular wine.
User Goals:	The user wants to be redirected to a page where they can buy a wine they saw on our app.
Primary Actor:	User of Application
Dependency Use Cases:	Account Login - UC17 Recommendations - UC13
Requirements:	
Details:	Priority Level: 2 Status: Pending
Pre-conditions:	None
Post-conditions:	1. The system shall open a new tab, directing the user to a google shopping search of the wine they clicked on.
Trigger:	1. From any webpage on which a wine is listed (Everything except the setting page) the user selects the “Buy Now” option
Workflow:	1. eventHandler.js posts to /Wine/buy in Routes.py 2. Routes.py retrieves information about the wine 3. Routes.py redirects the user to page of shopping search results for the wine
Alternate Paths:	None

	RSD-4: Create a Shelf
Description:	This use case outlines the implementation details of user’s ability to create a shelf based off of one or more wines.
Desired Outcome:	A shelf is made based off of the wine(s) the user selects and added to the user’s shelf list

User Goals:	The user wants to create a shelf based off of a wine, more than one wine, or a default shelf.
Primary Actor:	User of Application
Dependency Use Cases:	User Login – UC17
Requirements:	HP-2, RS-1, RS-2, RS-3, RS-9
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user is logged in
Post-conditions:	1. The system shall create a shelf 2. The system shall add the shelf to the user's shelf list 3. The system shall display recommendations from this shelf
Trigger:	1. The user shall select a "Create Shelf" button from the shelf list on recommendView.html
Workflow:	1. eventHandler.js reveals the Add Shelf Modal 2. The user fills in the form and clicks submit 3. eventHandler.js posts to /Recommend/createShelf in Routes.py 4. Routes.py retrieves the information and verifies it 5. Routes.py calls inputCreateChannel in RecommendIOController.py. 6. RecommendIOController.py checks the information for problems and finds none 7. RecommendIOController.py calls createChannel() in RecommendController.py 8. RecommendController.py calls dbCreateChannel() in RecommenderModelAccess.py. 9. RecommenderModelAccess.py injects the new shelf into the Database and returns a response. 10. RecommendController.py receives the response and sends it to RecommendIOController.py. 11. RecommendIOController.py receives the response and sends it to Routes.py. 12. Routes.py renders recommendView.html
Alternate Paths:	

	RSD-5: Remove a Shelf
Description:	This Use Case outlines the implementation details of a User removing a shelf from their recommendation page.
Desired Outcome:	The selected shelf shall be removed from the user's recommendation page.
User Goals:	The user shall desire to remove a shelf from their recommendation page.
Primary Actor:	User of Application
Dependency Use Cases:	User Login – UC16 Create a Shelf – UC20
Requirements:	HP-2, RS-4, RS-5, RS-6, RS-7, RS-8, RS-9
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user is logged in to their account 2. The user has at least two shelves 3. The user has selected a shelf to delete
Post-conditions:	1. The selected shelf shall be absent from the user's recommendation page. 2. The selected shelf shall be removed from the Database.
Trigger:	1. The user shall click the "Remove" button of a particular shelf on the page generated by recommendView.html.
Workflow:	1. eventHandler.js displays a Remove Channel Confirmation Modal 2. The User clicks on the Remove button. 3. The Remove button, on click, posts the Channel information to /Recommend/delete in Routes.py 4. Routes.py calls inputRemoveChannel() in RecommendIOChannel.py. 5. RecommendIOChannel.py calls removeChannel() in RecommendController.py 6. RecommendController.py calls dbRemoveChannel() in recommenderModelAccess.py 7. recommenderModelAccess.py removes the shelf from the Database and returns a response. 8. RecommendController.py receives the response and sends it to RecommendIOController.py. 9. RecommendIOController.py receives the response and sends it to Routes.py 10. Routes.py renders recommendView.html

Alternate Paths:	

	RSD-6: Navigate Shelves
Description:	This Use Case outlines the implementation details the user navigating to and between their existing shelves.
Desired Outcome:	The user shall be presented with the wines recommended by the shelf of their choice.
User Goals:	The user shall desire to view the recommendations from some shelf.
Primary Actor:	User of Application
Dependency Use Cases:	User Login – UC16 Create a Shelf – UC25
Requirements:	HP-2, RS-7, RS-8, RS-9, RS-10
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user is logged in to their account 2. The user has at least one shelf 3. The user elects to navigate to one of those shelf
Post-conditions:	1. The recommendView.html page will show wines recommended by that shelf
Trigger:	1. The User shall click on a shelf's Icon on the shelf list of the page generated by recommendView.html.
Workflow:	1. The channels' Icon, on click, calls the filter method in eventHandler.js 2. eventHandler.js reveals wines recommended by the selected shelf 3. Foundation.js rearranges the page to show the wines
Alternate Paths:	

	RSD-7: Recommend Random Wine
Description:	This Use Case outlines the implementation details of the User being recommended a random wine.
Desired Outcome:	The user is shown a random bottle of wine as a recommendation
User Goals:	The user wants to be shown wine at random
Primary Actor:	User of Application

Dependency Use Cases:	User Login – UC17
Requirements:	HP-2: Login to System HP-3: Create Account HP-5: Random Wine
Details:	Priority Level: 2 Status: Pending
Pre-conditions:	1. The user is logged in 2. The user is on the shelves page
Post-conditions:	1. The user sees a random bottle of wine and info about that wine
Trigger:	1. The user clicks the "Show Random Wine" button on recommendView.html
Workflow:	1. eventHandler.js posts to /Recommend/random in Routes.py 2. Routes.py calls inputRandomWine() in RecommendIOController.py 3. RecommendIOController.py calls randomWine() in RecommendController.py 4. RecommendController.py calls dbRandomWine() in RecommenderModelAccess.py 5. RecommenderModelAccess.py quaries the database for the pseudorandom wine 6. RecommenderModelAccess.py sends the results to RecommendController.py 7. RecommendController.py sends the results to RecommendIOController.py 8. RecommendIOController.py passes the response Routes.py 9. Routes.py sends the wine to eventHandler.js in JSON form 10. eventHandler.js updates the DOM to show the wine
Alternate Paths:	None

	RSD-8: Edit Shelf
Description:	This Use Case outlines the implementation details of a user being able to change the seed wines of a shelf.
Desired Outcome:	The user shall be able to change the recommendations they receive from an existing shelf
User Goals:	To be change the behavior of an existing shelf.

Primary Actor:	User of Application
Dependency Use Cases:	FP-2: Login Account RS-5: Create a Shelf
Requirements:	RS-15: Edit a Shelf
Details:	Priority Level: 2 Status: Completed
Pre-conditions:	1. The user is logged in 2. The user is viewing the recommender page.
Post-conditions:	1. The user shall see different kinds of recommendations on a shelf
Trigger:	2. The user clicks the Edit Shelf button in recommendationView.html
Workflow:	1. eventHandler.js reveals the Edit Shelf Modal 2. The user specifies what seeds to use 3. eventHandler.js posts to /Recocommend/editShelf in Routes.py 4. Routes.py calls inputRemoveSeeds() in RecommendIOController.py 5. RecommendIOController.py calls removeSeeds() in RecommendController.py 6. RecommendController.py calls dbRemoveSeeds() in RecommendModelAccess.py 7. RecommendModelAccess.py removes the old seeds 8. RecommendModelAccess.py sends the results to RecommendController.py 9. RecommendController.py sends the results to RecommendIOController.py 10. RecommendIOController.py sends the results to Routes.py 11. Routes.py calls inputAddSeeds() in RecommendIOController.py 12. RecommendIOController.py calls addSeeds() in RecommendController.py 13. RecommendController.py calls dbAddSeeds() in RecommendModelAccess.py 14. RecommendModelAccess.py adds the new seeds to the shelf 15. RecommendModelAccess.py returns the results to RecommendController.py 16. RecommendController.py sends the results to RecommendIOController.py 17. RecommendIOController.py sends the results to Routes.py 18. Routes.py sends the results to eventHandler.js in JSON form 19. eventHandler.js resets the page
Alternate Paths:	None

	IRSD-1: From Recommender to Inventory
Description:	This Use Case outlines the implementation details of allowing the user to place wines which were recommended to them directly into their user inventories.
Desired Outcome:	The recommended wine is added to the user's inventory.
User Goals:	The user may face the situation where they have been recommended an amazing sounding wine, but already have a house full to bursting with wine. Thus, the user can retain a note of the current wine for when they have drunk enough wine to order it.
Primary Actor:	User of Application
Dependency Use Cases:	Add Wine - UC1 Account Login - UC17 Recommendation - UC13
Requirements:	S/R-1: Store Wine Bottle/Name S/R-3: Store Storage Location HP-2: Login to System S/R-2: Store Wine Attributes S/R-3: Store Wine Location S/R-4: Store User Ratings S/R-5: Store User Reviews SR-32: Generate Recommendations
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user has an inventory 2. The user has an account 3. The user has been recommended a wine.
Post-conditions:	1. The described wine is placed into the user's inventory.
Trigger:	1. The user selects the 'Add to inventory' button on the recommendView.html page
Workflow:	1. eventHandler.js posts to /Recommend/addFromRecommendToInventory in Routes.py 2. Routes.py calls inputAddWineUserByID() in InventoryIOController.py 3. InventoryIOController.py calls addWineUserByID() in InventoryController.py 4. InventoryController.py calls dbAddWineUserByID() in InventoryModelAccess.py 5. InventoryModelAccess.py finds and adds the wine to the user's inventory 6. InventoryModelAccess.py sends the results to InventoryController.py 7. InventoryController.py sends the results to InventoryIOController.py 8. InventoryIOController.py sends the results to Routes.py 9. Routes.py renders recommendView.html

Alternate Paths:	None

	IRS-2: Autocomplete
Description:	This Use Case outlines the implementation details of a User selecting wine names based off of a prepopulated field of common wines.
Desired Outcome:	The User shall be able to search through lists of wines using a common form element, automatically populating suggestions for wine names.
User Goals:	The User wants to quickly find what wine they're looking for.
Primary Actor:	User of Application
Dependency Use Cases:	User Login - UC16 Wine Search – UC10
Requirements:	Di-9: Autocomplete - Database Search Di-10: Autocomplete - Inventory Add S/S-6: Search Database for Wine - Name
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The User is logged in. 2. There exists items in the search query located in the database.
Post-conditions:	1. The system shall populate a dropdown with the relevant wine names. 2. The system shall update when the form element Is updated.
Trigger:	1a. From the Add Wine model in inventoryView.html the User starts filling in the name of the wine. 1b. From the Edit Shelf model in recommendView.html the User starts filling in the name of a wine
Workflow:	1. eventHandler.js posts to /Recommend/candidateWines in Routes.py 2. Routes.py calls inputGetCandidateWines() in RecommendIOController.py 3. RecommendIOController.py calls getCandidateWines() in RecommendController.py 4. RecommendController.py calls dbShelfWineCandidates() in RecommenderModelAccess.py 5. RecommenderModelAccess.py returns to RecommendController.py a list of matching wines 6. RecommendController.py sends the list to RecommendIOController.py 7. RecommendIOController.py sends the list to Routes.py 8. Routes.py returns the list in JSON form to eventHandler.js 9. eventHandler.js updates the page to show the wines
Alternate Paths:	

	FPD-1: Popular Wines
Description:	This Use Case outlines the implementation details of the ability for the system to keep track of popular wines for display on the front page.
Desired Outcome:	The homepage displays the most popular wines across the entire database.
User Goals:	To see what wines are widely possessed by other users
Primary Actor:	System
Dependency Use Cases:	None
Requirements:	None
Details:	Priority Level: 2 Status: Pending
Pre-conditions:	1. User must be visiting the site initially
Post-conditions:	1. System view displays the popular wines in the front page
Trigger:	1. User shall visit the home page indexView.html
Workflow:	1. The indexView.html posts to /Inventory in Routes.py 2. Routes.py calls inputGetPopularWines() in WineIOController.py. 3. WineIOController.py calls getPopularWines() in WineController.py 4. WineController.py calls dbGetPopularWines() in WineModelAccess.py 5. WineModelAccess.py pulls a query from the database to find the highest ranking wines 6. WineModelAccess.py returns the results to WineController.py 7. WineController.py sends the results to WineIOController.py 8. WineIOController.py sends the results to Routes.py 9. Routes.py sends the information in JSON form to eventHandler.js 10. eventHandler.js updates the DOM to show the wines
Alternate Paths:	

	FPD-2: Login Account
Description:	This Use Case outlines the implementation details of a User logging in to their account.
Desired Outcome:	The user shall have access to the recommendation system and inventory system after logging in.

User Goals:	The user shall desire to login to access full functionality.
Primary Actor:	User of Application
Dependency Use Cases:	User Login – UC16 Create a Shelf – UC20
Requirements:	HP-2, RS-4, RS-5, RS-6, RS-7, RS-8, RS-9
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user has a currently registered account.
Post-conditions:	1. The user can obtain access to their account.
Trigger:	1. The user selects a 'Sign in' button.
Workflow:	1. eventHandler.js shows a username, password, and remember me form. 2. The user enters in their information. 3. eventHandler.js posts to /User/login in Routes.py 4. Routes.py calls inputLogin() in UserIOController.py 5. UserIOController.py calls login() in UserController.py 6. UserController.py calls dbLogin() in UserModelAccess.py 7. UserModelAccess.py verifies the information 8. UserModelAccess.py sends the results to UserController.py 9. UserController.py sends the results to UserIOController.py 10. UserIOController.py sends the results to Routes.py 11. The controller creates a unique session id, storing a cookie on the user's computer.
Alternate Paths:	1. UserModelAccess.py finds that the information doesn't match 2. UserModelAccess.py sends an error message up the chain to Routes.py 3. Routes.py returns the message in JSON form 4. inventoryView.html updates to show the error

	FPD-3: Register an Account
Description:	This Use Case outlines the implementation details of the ability for the user to create a new user account in the system, using an email and password for logging in to the new account.
Desired Outcome:	The user shall be able to create a new account in the system.
User Goals:	The user wants to be able to create an account.
Primary Actor:	User of Application
Dependency Use Cases:	None
Requirements:	HP-3: Create Account

Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user is not logged in. 2. The user is viewing the front splash page, which has a login option
Post-conditions:	1. The user is logged into their new account and can access the rest of the system.
Trigger:	1. The user clicks the Register An Account button on indexView.html
Workflow:	1. eventHandler.js reveals the Register Account modal 2. The user fills in the form and submits 3. The eventHandler.js posts to /User/create in Routes.py 4. Routes.py retrieves the information and verifies it 5. Routes.py calls inputCreateUser in UserIOController.py 6. UserIOController.py checks for problems and finds none 7. UserIOController.py calls createUser() in UserController.py 8. UserController.py calls dbCreateUser() in UserModelAccess.py 9. UserModelAccess.py creates a new user in the database 10. UserModelAccess.py returns the results to UserController.py 11. UserController.py sends the results to UserIOController.py 12. UserIOController.py sends the results to Routes.py 13. Routes.py renders inventoryView.html
Alternate Paths:	

	FPD-4: Edit an Account
Description:	This Use Case outlines the implementation details of the ability for the user to modify their account information (eg. password, display name, location, birthday, profile image)
Desired Outcome:	The user shall be able to have their account information modified to the user's preferences.
User Goals:	The user wants to be able to edit their user account.
Primary Actor:	User of Application
Dependency Use Cases:	None
Requirements:	None
Details:	Priority Level: 1 Status: Completed

Pre-conditions:	1. The user is logged in.
Post-conditions:	1. The user has their account modified
Trigger:	1. The user requests to edit an account from the profile window
Workflow:	1. eventHandler.js reveals the Edit Account modal 2. The user fills in the form and submits 3. eventHandler.js posts to /User/edit in Routes.py 4. Routes.py retrieves the information and verifies it 5. Routes.py calls inputEditUser() in UserIOController.py 6. UserIOController.py checks for problems and finds none 7. UserIOController.py calls editUser() in UserController.py 8. UserController.py calls dbEditUser() in UserModelAccess.py 9. UserModelAccess.py edits the user's information in the database 10. UserModelAccess.py returns the results to UserController.py 11. UserController.py sends the results to UserIOController.py 12. UserIOController.py sends the results to Routes.py 13. Routes.py renders profileView.html
Alternate Paths:	1. Routes.py or UserIOController.py detect invalid information in the form 2. Routes.py renders profileView.html

	FPD-5: Delete an Account
Description:	This Use Case outlines the implementation details of the ability for the user to delete their account from the system.
Desired Outcome:	The user shall be able to have their account deleted.
User Goals:	The user wants to be able to delete their user account.
Primary Actor:	User of Application
Dependency Use Cases:	None
Requirements:	None
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user is logged in
Post-conditions:	1. The user has their account deleted and is redirected to the front splash page
Trigger:	1. The user clicks the Delete Account button in the Profile Modal

Workflow:	1. eventHandler.js reveals the Confirmation Modal 2. The user accepts 3. eventHandler.js posts to /User/delete in Routes.py 4. Routes.py calls inputDeleteUser() in UserIOController.py 5. UserIOController.py calls deleteUser() in UserController.py 6. UserController.py calls dbDeleteUser() in UserModelController.py 7. UserModelController.py deletes the user account from the database 8. UserModelController.py returns the results to UserController.py 9. UserController.py sends the results to UserIOController.py 10. UserIOController.py sends the results to Routes.py 11. Routes.py renders indexView.html
Alternate Paths:	

	FPD-6: Logout
Description:	This Use Case outlines the implementation details of the ability for the user to log out from their account.
Desired Outcome:	The user shall be able to log out.
User Goals:	The user wants to be able to log out..
Primary Actor:	User of Application
Dependency Use Cases:	None
Requirements:	None
Details:	Priority Level: 1 Status: Completed
Pre-conditions:	1. The user is logged in.
Post-conditions:	1. The user is logged out from the system and is redirected to the front splash page.
Trigger:	1. The user clicks to logout button from the main toolbar in base.html
Workflow:	1. eventHandler.js posts to /User/logout in Routes.py 2. Routes.py deletes the user's session cookie 3. Routes.py renders indexView.html
Alternate Paths:	

	IWPD-1: Detailed Wine Information
Description:	This Use Case outlines the implementation details of the User being given information about a wine on that wine's individual page
Desired Outcome:	The user is more knowledgeable about the wine they are viewing, influencing their decision to try that wine.
User Goals:	The user will obtain the following information about the wine: • Food pairings • Optimal glass shape • Temperature the wine should be served at • Whether the wine should be aged
Primary Actor:	The user of the application
Dependency Use Cases:	User Login – UC17
Requirements:	TBD
Details:	Priority Level: 2 Status: Completed
Pre-conditions:	1. The user is logged in 2. The user is on the Recommend or Inventory pages
Post-conditions:	None
Trigger:	1. The user clicks a wine icon
Workflow:	1. eventHandler.js reveals the Wine Info modal
Alternate Paths:	

	IWPD-2: Report System Abuse
Description:	This Use Case outlines the implementation of the ability for the user to report system abuse of other users (inappropriate names/pictures)
Desired Outcome:	The user shall be able to flag other submitted content for moderation by an administrator.
User Goals:	The user wants to maintain a high quality system application for his or her personal use.
Primary Actor:	User of Application
Dependency Use Cases:	User Login - UC16

Requirements:	Misc-4: Recognize System Abuse Misc-5: Remove System Abuse
Details:	Priority Level: 2 Status: Pending
Pre-conditions:	1. The user is logged in. 2. The user is viewing a specific wine page.
Post-conditions:	1. The system shall send a report to the administrator notifying them of a possible system abuse.
Trigger:	1a. The user selects the “View Wine Info” button from recommendView.html 1b. The user selects the “View Wine Info” button from inventoryView.html 2a. The user selects the “Report” Icon on the View Wine Info modal within recommendView.html. 2b. The user selects the “Report” Icon on the View Wine Info modal within inventoryView.html. 3. The user specifies the reason to report the wine. 4. The user selects the “Send” button.
Workflow:	1. eventHandler.js posts to /Wine/report in Routes.py 2. Routes.py retrieves the post information and verifies it 3. Routes.py calls inputReport() in WineIOController.py 4. WineIOController.py calls report() in WineController.py 5. WineController.py calls dbReport() in WineModelAccess.py 6. WineModelAccess.py executes the SQL command to log the report corresponding to the wine 7. WineModelAccess.py sends a confirmation message to WineController.py 8. WineController.py sends the message to WineIOController.py 9. WineIOController.py sends the message to Routes.py 10. Routes.py sends the message to eventHandler.js in JSON form 11. eventHandler.js updates the DOM to display the message
Alternate Paths:	None

	IWPD-3: View Average Price
Description:	This Use Case outlines the implementation details of the ability for a user to view the average price of a wine.
Desired Outcome:	The outcome of this use case is to display the average price of a wine.
User Goals:	The user wants to view the average price of a particular wine.
Primary Actor:	User of Application
Dependency Use Cases:	User Login – UC17

Requirements:	Di-8: Display Average Price Misc-3: Calculate Average Price
Details:	Priority Level: 2 Status: Pending
Pre-conditions:	1. The user shall be logged into their account
Post-conditions:	1. The system shall display the average price of a particular wine
Trigger:	1a. The user selects the “View Wine Info” button from recommendView.html 1b. The user selects the “View Wine Info” button from inventoryView.html
Workflow:	1. IWPD-1 occurs, revealing the average price as part of the information modal
Alternate Paths:	None