ITN Self-Search

Requirements Documentation

Innovative Tech Nerds

CS-321

Intellectual Property

Table of Contents

Table of Figures

1.0 Project Description

Customer self help service that allows for better navigation towards products. Software runs on physical kiosks placed around the store. Kiosk allows customers to type the name of a product to reveal its location in the store. If the product name is unknown, the user can select related tags that best describe the desired item. If the product is not found through direct or tag descriptions, then the software will provide similar items. If product location is available, provide optimal route towards item via map visualization and given a guided description. Software also has administrative mode only accessible to employees. This mode allows for administrators to create store maps via a grid based system and import item databases.

2.0 Project Management

History

Our project was conducted over three phases. Phase I involved the planning of our software's use cases and functional requirements. Phase II involved the brainstorming of code (via CRC cards), and the creation of diagrams for both use cases and potential data classes. Phase III involved the actual implementation of code. This phase began with coding the data classes first, followed by the GUI classes, and ending with building of controller classes to connect our models and views. Phase III concluded with the revision of previous documentation.

Personnel

Kevin Patel - I am a Junior majoring in Computer Science with a math minor. I am currently working a paid Software Engineering Internship with SIEMENS. Prior to this class I have taken CS-103 Intro to Java which I felt helped me pick up concepts much easier in this course.

Harrison Matthews - I am a Junior majoring in Computer Science with a minor in mathematics. I took an intro to java class which helped me eaze into the programming language which was helpful for this class since it is faster paced.

Michael Kelly - I am a Sophomore majoring in Computer Science with a minor in Mathematics and Biology. I was a participant in Google's CSSI 2019 summer program. This opportunity provided me with the basics of coding fundamentals.

Bradley Mitchell - I am a Junior majoring in Computer Science with a minor in Mathematics.

Effort

Timeline:
January 22 - February 5: Phase 1
Number of Meetings: Three

February 5 - March 8: Phase 2
Number of Meetings: Seven

March 8 - April 13: Phase 3
Number of Meetings : Six

January 22 - April 13:
Total Number of Meetings: We took a total of 16 meetings during the course of this semester.

3.0 Use Cases

1. ***Customer search of direct product name and found***
   Precondition: Program on homescreen, User knows what they are looking for
   a. On the home screen, click direct search then type in product into search bar for direct item name, user types in product name
   b. System searches database for input name
   c. System brings up a window that has the item name, price, tags, and map location
   d. The user clicks on finished button that returns to the homescreen

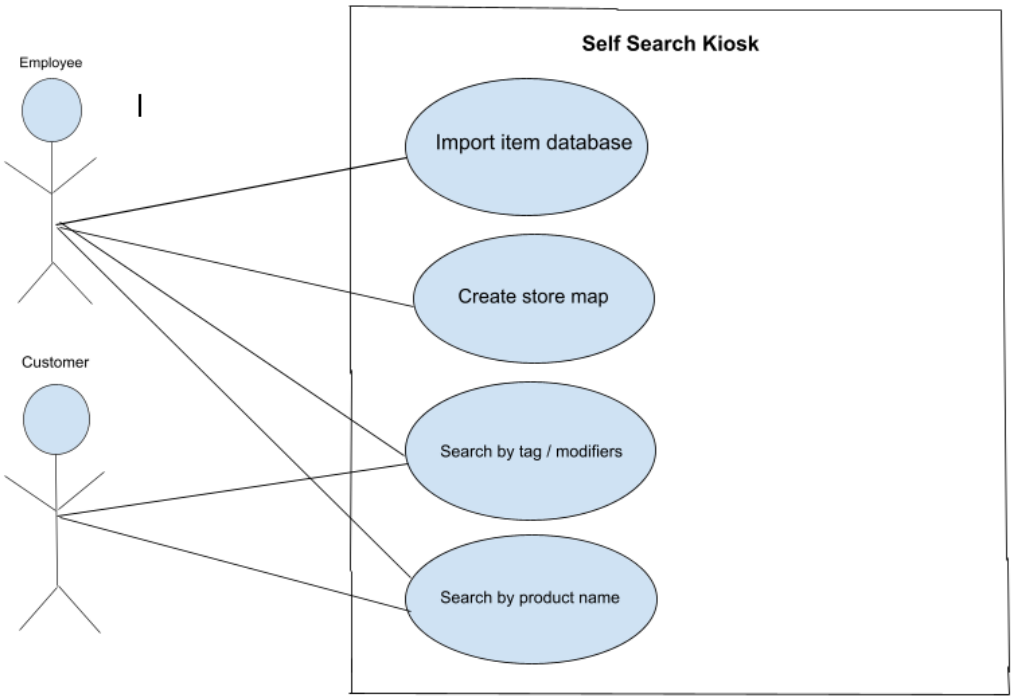
2. ***Customer search of direct product name and not found***
   Precondition: Program on homescreen, User knows what they are looking for
   a. On the home screen, click direct search then type in product into search bar for direct item name, user types in product name
   b. System searches database for input name
   c. Prompt user product is not found, returns to the homescreen

3. ***Customer search using tags/modifiers***
   Precondition: Program on homescreen, User knows what they are looking for
   a. On the home screen, user clicks on the tag search option
   b. Display menu of tags to select from (as many tags as user wants to add to hone search)
   c. System searches for items best related to the given tags
   d. User can select an item from a menu of related products or return to previous screen
   e. If an item is selected, system brings up a window that has the item name, price, tags, and map location
   f. The user clicks on the main menu button that returns to the homescreen

4. ***Administrator import of database***
   Precondition: Administrator must have password, database formatted for system reading
   a. User clicks on icon in top left corner of screen
   b. User is prompted to enter their password to log into the system as an administrator
   c. System menu comes up gives administrator options
   d. Selects import database option
   e. User is prompted to enter complete file path
   f. Provides administrator with view of database contents
   g. Asks administrator if they'd like to save database
   h. If user selects no, return to administrator menu
   i. If user selects yes, system overrides current database, returns to administrator menu

5. **Administrator Creation of Map**
   Precondition: Administrator must have password
   a. User clicks on icon in top left corner of screen
   b. User is prompted to enter their password to log into the system as an administrator
   c. System menu comes up gives administrator options
   d. Selects map creation option
   e. User is provided with an empty grid canvas
   f. User can select empty squares within canvas to plot and draw a map
   g. If user needs to delete a square, they can click plotted square again to do so
   h. User can select finish button to save changes
   i. System returns to administrator screen

Use Case Diagram:



4.0 Functional Requirements

1. ***Allow customer to direct search for an item***
    a. A customer can walk up to the kiosk and he/she is greeted by a main menu with three options: direct search option, tag search, and an administrative mode. Direct search allows the customer to type the name of the product they are looking for. If found then the item will come up with the tags, price, and map location.

2. ***Allow customer to search for an item via descriptive tags***
    a. A customer can walk up to the kiosk and he/she is greeted by a main menu with three options: direct search option, tag search, and an administrative mode. The system will display a menu of tags. Tag search allows for users to search for an item by allowing the user to include as many tags as needed. Based on the inputted tags, a menu of relevant products is displayed. The user can choose an item from the menu or return to the home screen. If the user selects an item, the system opens a window in which it gives the item name, tags, price, and map location. User then is able to select the finished button to go back to the home screen.

3. ***Administrator importing database***
    a. An employee approaches the kiosk and clicks the administrator icon in the top left of the screen. The system prompts the employee to enter their password to log into the system as an administrator. The system provides an administrator menu that has three options: map creation, database import, and return to home page. The admin selects the database import option. User provides the system with the complete file path of the database when prompted. The user is allowed to preview the contents of the new database before it is imported. If database contents are satisfactory, the employee can confirm database import and will then be returned to the administrator menu. When the user saves the database, if there was a previous database, it will be overwritten. If the user chooses not to save, they are returned to the administrator menu.

4. ***Administrator map creation***
    a. An employee approaches the kiosk and clicks the administrator icon in the top left of the screen. The system prompts the employee to enter their password to log into the system as an administrator. The system provides an administrator menu that has three options: map creation, database import, and return to home page. The admin selects the map creation option. System provides the user with an empty grid canvas. The user can pick from empty squares on the grid to plot and draw a map. To delete a square, the user must click the plotted square again to do so. When the user is done creating the map, they can select the finish button. System will ask the user whether or not they'd like to save changes. Saving a map will override previous map saves. System will then return to the administrator screen.

5. ***Database reading errors***
    a. If the database file is not found, incorrectly formatted, or contains bad data, the admin is prompted with an error message: “File was not found”. After the notification, the user is prompted to input another filepath (no changes to the current database unless a file is found).

6. ***Incorrect administrator password***
    a. If an unauthorized user attempts to log into the administrative mode with an incorrect password it will not let you into the admin menu.

Future modifications and extensions:

Taking a look at the project now that we have completed it based on how vision. There could be a few modifications that would help the user experience when using our program.

1. Making user input non case sensitive
   Right now if you were to search ‘apple’ and the apple is indicated in the database with a capital letter then the program would not be able to find the item.
2. Including a inventory system
   Including an inventory system is something that would be beneficial because we could then tell the customer if the item was sold out or not.This would benefit the customer because they would not have to waste their time to see for themselves. The database would subtract items that were sold and add items that were shipped in.

Summary:

The software will be idle at the home screen and respond accordingly to user input. For example, there will be three options at the home screen: Name Search, Tag Search, and Administration functions. The administrator functions can only be accessed if you have the correct key. The program will respond to user input and produce the following scenes as shown in the Use Cases we have written.

Associated Tests:

For each Use Case we will test that specific instance step by step and make sure that there are no issues with that first. Then we will intentionally try and break our code by doing strange things. For example, for the first Use Case Customer Search of direct product name and found we actually pressed the search buttons many times and found that there was a memory leak and the program crashed. After we have tried to break our code on this specific block of code we will then integrate it with the rest of the program to be sure that it works well with our other functions that are not needed for that specific use case.So, to breakdown our procedure we run the use case step by step and see if we get to a logical conclusion. Then we intentionally try to break the code.Then we will integrate it with the rest of our functions and try to run the use case again.

User Interface Requirements

UI Layout:

- **Administrator home screen:**
    - ***Map Creation***
        - Grid based map system
        - Finish button
            - Save button (Save map)
            - Load button (Load saved map)
    - ***Database Import***
        - Window prompting user for file path-name
            - If incorrect, prompted with an error message: “File can’t be read”
            - Ok button (Return to administrator home screen)
        - Window of database contents (preview)
            - Finish button
                - Yes button (Save database)
                - No button (Don’t save database)
    - ***Return Home Screen***
- **Home screen menu**
    - ***Direct Search***
        - Window with a search bar
        - If found, window containing item info (name, price, tags, map)
            - Finish button (return to home screen)
        - If not found, window prompt asking if they’d like to tag search
            - No button (return to home)
            - Yes button (system enters **Tag-Search**)
    - ***Tag-Search***
        - Window with menu of tags to choose from
            - Back button (return to home screen, cancels search)
            - Selectable item names
                - Window containing item info (name, price, tags, map)
                    - Finish button (return to home screen)
    - ***Administrator Icon***
        - Window prompting user for administrator password
            - If incorrect password is entered, prompted with an error message: “Incorrect password”
        - If correct password is entered, open administrator home screen

Required GUI

The main page will consist of four buttons and a search bar: Name Filter,Tag Filter, Search Button, administrator icon and a search bar.

1. *Direct Search* will pop up a window with a search bar that allows the user to input a name of whatever they want to search for.
    a. If found, the system will provide a window for the item information. Tags, price, and map location. Also there will be a finish button that takes you back to the main menu.
2. *Tag Search* will contain a window with a menu of common tags to choose from and a back button that returns you to the main menu.
    a. If an item is selected by clicking on it, the system brings up a window that has the items name, tags, price, map location, and a finished button that will return the user back to the home screen.
3. Administrator icon will open up a window asking for the administrator password.
    a. If an incorrect password is entered, the system will prompt the user with an error message “Incorrect password”.
    b. If the correct password is entered, the system opens the administrator menu. The admin menu has three buttons *map creation*, *Import database*, *return to home screen.*
        i. *Map creation* will pop up a window that contains an empty grid. If a user selects an empty grid space, it will be filled in as blue. If a user wants to deselect a grid space, clicking on it again will make it white/empty. Map creation also contains a finished button and once you click on it, a prompt will ask the user if they want to save their creation (yes), or if they would like to return to the main menu without saving (no).
        ii. *Import database* will pop up a window that prompts the user for a file path. If the file is not found/corrupted, a window with the error message “File can’t be read” appears. The window prompt has an *Ok button* that returns to the administrator menu when clicked. If the file is found, a window pops up containing a preview of the database contents. After verifying contents, the user can click the *finish button.* Window prompt appears asking if the user would like to submit changes. If the *yes button* is clicked, the imported database will override current system contents and return administrator menu. If the *no button* is clicked, then the import is canceled and the user is returned to the administrator menu.

Required Inputs For System:

- Employee password (for admin mode, inputted via keyboard)
- Database file (for system to read in items)
- Name of product (for direct search use case, inputted via keyboard)

Outputs produced by System:

- Product information (name, price, tags, location in store)
- List of all tags in database (for use with tag search)
- List of all items in database (administrator database preview)
- Map of the store
- Error Messages (Database reading errors, incorrect administrator password)

5.0 Design

Model:

Product:
One of the most important data models in our program is product, as the entire program revolves around the product class. This will contain all of the information related to a product. It will contain the name, price, tags, and the location of where it is located. As it is read in from a file it will feed into the database which is one of our other important data models.

Database:
The database is responsible for storing products and admin passwords. It also stores the amount of products in the database and provides products for tag/name filtered searches. It stores the maps from our map class into the database.

Coordinate:
The coordinate class is responsible for saving the coordinate values for a product.

GridTile:
This class represents the individual tiles in the map editor. Contains data about which type of tile it is (Blank, Shelf, Path, etc).

View:

Home Screen View:
The Home Screen will display a JPanel with 4 buttons and a search bar. An admin button that will take the user to the User View to enter a password. A search bar that the user can enter products into. A search button that allows the user to search the database. A name filter toggle button that when toggled allows the user to click the search button and search for a product. The tag menu button that when clicked displays a list of tags that the user can select.

Admin Screen View:
The admin Screen will show a Panel containing 3 buttons. It will have a button for database import which will allow the user to import the database. It will have a map editor button which will take them to the map editor GUI. It also will have a home button which takes the user back to the home screen.

ProductGUI:
The product display view will display a JPanel with the name, price, tags, and a home screen button. The purpose of it is to display the contents of a product for the user.

Map Editor:
The map editor will allow the user to select squares from a 2d matrix to create the layout of the store map. The squares will be colored when a product has been placed in that specific location and they will be grey if nothing currently occupies the space.

Database Display:
Displays the contents of the database.

ImportView:
This is the GUI that allows you to enter the file path to import the database.

User View:
This is the GUI that appears when you are entering the password for admin access.

Tag Menu:
This GUI appears when you click on Tag Menu. It shows you all of the tags that are in your database.

ProductSelection:
This GUI appears when you select tags. It shows you the products that are related to the tags the user has entered.

ProductLocator:
This class is responsible for drawing a path from an input product to a kiosk on a store map.

Control:

Dynamic Main:
This class is responsible for cycling between panels within our software. Dynamic Main itself is a JFrame, so it is both a view and a controller.

Database Import:
Database import will read in a json file with data.

Tag Menu Controller:
Controls the tag menu screen and controls the buttons for search and back. The user will click on tags by CTRL left clicking to select multiple and add these tags to an Array List of input tags.

Name Filter:
The name filter will take in the item name entered by the user and compare it with the item names in the database to find the relevant product. If not found by name, then the user has the option to find by tags or return to the homescreen.

Tag Filter:
The Tag Filter allows the user to select from a menu of descriptive tags in order to find a product. The user will select tags with a JCheckBox. After the user makes their selections, they are displayed on the product display screen.

ActiveDatabase:
This class is responsible for updating a database object (database in this case acts as a node would in an abstract structure).

ImportController:
This class is responsible for receiving data from an input json file before software runs.

homeController:
This class is responsible for controlling the buttons in the homeView panel.

userController:
This class is responsible for controlling the textbox/button functionality of the userView panel.

MapTemplate:
This class is a controller that is responsible for loading and saving maps.

Class Diagrams

Data Classes

DatabaseImport
Attributes - filePath : String - boolean : imported
Methods - wasImportred() : boolean + DatabaseImport(inputPath : String) + importDatabase(databaseProducts : ArrayList<Product>, storeTags : ArrayList<Product>, databasePasswords : ArrayList<Product>, kioskCoordinate : ArrayList<Product>, mapData : ArrayList<Product>) : void

Product
Attributes - productName : String - productPrice : Double - productTags : ArrayList<String> - productTagNumber : int - productLocation : Coordinate
Methods + Product(inputName : String, inputPrice : double) : void + Product() : void + clone() : Product + setProductName(inputName : String): void + getProductName() : String + setProductPrice(inputPrice : Double) : void + getProductPrice() : Double + setProductTags(inputTags : ArrayList<String>): void + setProductTag(inputTag: String) : void + getProductTags() : ArrayList<String> + setProductLocation(inputLocation : Coordinate) + getProductLocation() : Coordinate + printProductInfo() : void + setProductTagNumber() : void + getProductTagNumber() : int

Database
Attributes - productCatalogue : ArrayList<Products> - adminPasswords : ArrayList<String> - storeMap : ArrayList<String> - storeTags : ArrayList<String> - kioskLocation : ArrayList<Coordinate> - storeMap : ArrayList<String>
Methods + addProduct(inputProduct : Product) : void + addPassword(inputPassword : String) : void + setStoreMap(inputMap : String) : void + addKioskLocation(inputCoordinate: Coordinate) : void + displayDatabase() : void + getProductCounter() : int + getStoreMap() : String + getStoreMapArrayList() : ArrayList<String> + getProductCounter() : int + getPasswordCounter() : int + getStoreTagCounter() : int + getProductCatalogue() : ArrayList<Product> + getPasswords() : ArrayList<String> + getStoreTags() : ArrayList<String> + getKioskLocation() : Coordinate + getKioskLocationArrayList() : ArrayList<Coordinate> + Database () : void + Database (defaultPassword: String, importCall : DatabaseImport) : void + validProductName() : boolean + clone() : Database

ActiveDatabase
Attributes - activeDatabase : ArrayList<Database>
Methods + ActiveDatabase() + ActiveDatabase(inputDatabase : Database) + ActiveDatabase(defaultPassword : String) + getDatabase() : Database + getPasswords() : ArrayList<String> + getProducts() : ArrayList<Products> + getStoreTags() : ArrayList<String> + getKioskLocation() : ArrayList<Coordinate> + getStoreMap() : String + getStoreMapArrayList() : ArrayList<String> + setStoreMap(inputMap : String) : void + updateDatabase(newDatabase : Database) : void + displayActiveDatabase() : void + clone() : ActiveDatabase

Coordinate
Attributes - xCoordinate : int - yCoordinate : int
Methods + Constructor() + Constructor(inputX : int, inputY : int) + getX() : int + getY() : int + setX(inputX : int) : void + setY(inputY : int) : void

MapTemplate

Attributes

\# mapSizeX : int = 12
\# mapSizeY : int = 12
\# gridArray : GridTile[][]
\# mapSaveListener : ArrayList<ChangeListener>
\# mapLoadListener : ArrayList<ChangeListener>

Methods

\+ Constructor()
\+ saveMapData() : String
\+ addMapSaveListener(newListener : ChangeListener) : void
\+ loadMapData(mapData : String) : void
\+ addMapLoadListener(newListener : ChangeListener) : void
\# registerTile(x : int, y : int) : void

GridTile

Attributes

\- gridState : int
\- tileColors : Color[]
\- owner : ChangeListener

Methods

\+ Constructor()
\+ registerOwner(tileOwner : ChangeListener) : void
\+ getGridState() : int
\+ setGridState(newState : int) : void
\+ paintComponent(graphics : Graphics) : void
– updateTooltip() : void

View Classes

ProductGUI
Attributes - productName : String - displayedProduct : Product - databaseContents : String - databasePreview : JTextArea - Mainlistener : ArrayList<ChangeListener>
Methods + constructor(inputProduct : Product, kioskCoordinate : Coordinate, storeMap : String) + addMainListener(newListener : ChangeListener) : void

homeView
Attributes - userText: JTextField - nameFilterToggle : JToggleButton - tagMenuButton : JButton - adminButton : JButton - ITN JLabel
Methods - createFrame() : void - addComponents(Container Pane) : addComponents + homeView() : constructor + getUserText(): JTextField + getnameFilterToggle(): JToggleButton + getTagMenuButton(): JButton + getSearchButton() : JButton + getAdminButton() : JButton

importView
Attributes - importingDatabase : DatabaseImport - newDatabase : Database - filePath : String - prompt : String - listeners : ArrayList<ChangeListeners> - backListeners : ArrayList<ChangeListeners>
Methods - ImportView() + addChangeListener() : void + addBackListener() : void + clearNewImport() : void + getNewImport() : Database

AdminView
Attributes - databaseImportButton : JButton - mapEditorButton : JButton - homeButton : JButton - adminScreen: JLabel - mainListener: ArrayList<ChangeListener> - mapListener: ArrayList<ChangeListener> - importListener: ArrayList<ChangeListener>
Methods + adminView(): Constructor - createPanel(): void + addMainListener(ChangeListener newListener): void + addMapListener(ChangeListener newListener): void + addImportListener(ChangeListener newListener): void

MapEditor
Attributes - blankTile : JRadioButton - shelfTile : JRadioButton - aisleTile : JRadioButton - wallTile : JRadioButton
Methods + Constructor() + getTileTypeSelection() : int # registerTile(x : int, y : int) : void + addBackListener(newListener : ChangeListener) : void

ProductLocator
Attributes - distanceMap : int[][]
Methods + Constructor() + pathfinder(kioskLocation : Coordinate, productLocation : Coordinate) : boolean – generateDistanceMap(productLocation : Coordinate) : void – drawKiosk(kioskLocation : Coordinate) : void – drawProductLocation(productLocation : Coordinate) : void # registerTile(x : int, y : int) : void

ProductSelection

Attributes

- listPanel : JPanel
- buttonPanel : JPanel
- scrollPane: JScrollPane
- productList: ArrayList<Product>
- Select: JButton
- Cancel: JButton
- tagsSelected: JTextArea
- productNames: ArrayList<String>
- database: Database
- tagController: tagMenuController
- p : Product
- cancelListener: ArrayList <ChangeListener>
- selectListener: ArrayList <selectListener>

Methods

+ productSelection(ArrayList<Product>: Database: tagMenuController): Constructor
- getProductNames() : void
- createPanel() : void
- prepareProduct() : void
- updateCancel() : void
+ addCancelListener: void
+ addSelectListener: void
+ getP : Product
+ getTagsSelected : JTextArea

userView

Attributes

- password : JLabel
- info : JLabel
- userPassword : JPasswordField
- enterButton : JButton
- backButton : JButton

Methods

+ constructor()
+ getEnterButton() : JButton
+ getBackButton() : JButton
+ setUserPassword(userPassword : JPasswordField) : void
+ getUserPassword() : JTextField
– createPanel() : void

Tag Menu
Attributes - uniqueTags : ArrayList<String> - Search : JButton - Cancel : Jbutton - listOfButtons : JList - Tag : JLabel - instructionMultiple : JLable - ButtonPanel : JPanel - listPanel : JPanel - scrollPane : JScrollPane - D : Database
Methods + constructor(Database data) + getAvailableTags(inputDatabase : Database) : void + createPanel() : void + getListOfButtons() : JList + getSearch() : JButton + getCancel() : JButton + getUniqueTags() : ArrayList<String>

DatabaseDisplay
Attributes - databasePreview : JTextArea - databaseContents: String
Methods + DatabaseDisplay(Database) : Constructor

Control Classes

homeController
Attributes - View : homeView - Namefilter : NameFilter - Tagfilter: TagFilter - database: Database - retrievedProduct: Product - nameSearchListener : ArrayList<ChangeListener> - tagMenuListener : ArrayList<ChangeListener> - adminListener : ArrayList<ChangeListener>
Methods + homeController(homeView: NameFilter: TagFilter: Database) : constructor + initController() : void - callTagMenu() : void - search() : void - adminView() : void + getRetrievedProduct() : Product + addNameSearchListener(ChangeListener) : void + addTagSearchListener(ChangeListener) : void + addAdminListener(ChangeListener) : void + refreshDatabase(Database) : void

UserController
Attributes - passwordScanner : Scanner - inputPassword : String - correctPassword : ArrayList<String> - adminMode : boolean - defaultPassword : String - uView : userView - d : Database - adminLoginListener : ArrayList<ChangeListener> - backListener : ArrayList<ChangeListener>
Methods + Constructor(v : userView, data : Database) + verifyPassword(inputPassword : String, correctPassword ArrayList<String>) : boolean + initController() : void + addAdminLoginListener(newListener : ChangeListener) : void + addBackListener(newListener : ChangeListener) : void + updateBackListener() : void

Import Controller
Attributes - newDatabase : Database - importWindow : ImportView - database Panel : DatabaseDisplay - Importable : boolean + listeners : ArrayList<ChangeListener> + mainListener : ArrayList<ChangeListener> - programStart : boolean
Methods + ImportController(firstImport : boolean) : Constructor + addChangeListener(newListener : ChangeListener) : void + addFirstImportListener(newListener : ChangerListener) : void + clearNewImport() : void + overrideDatabase() : Database + canImport() : boolean + firstImport() : boolean

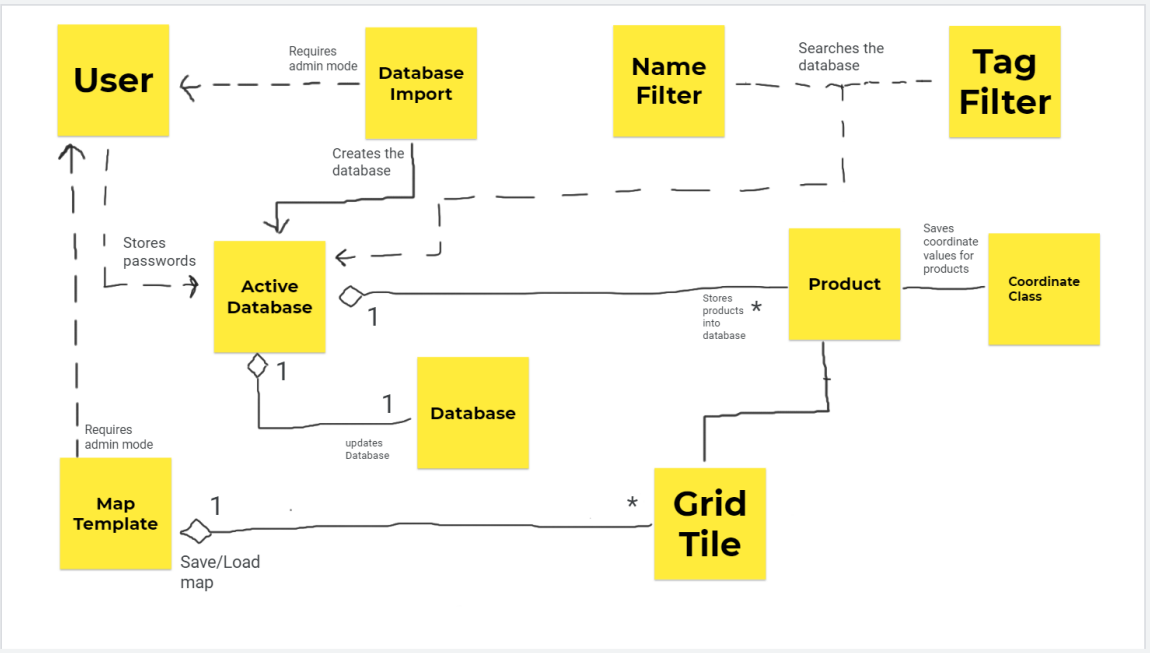
Tag Filter
Attributes
Methods + retrieveByTags(inputTags : ArrayList<String>, d : Database) : ArrayList<Product> + checkDuplicate(matchingProducts : ArrayList<Product>, product : Product) : boolean

Name Filter
Attributes
Methods + retrieveByName(inputName : String, d : Database) : Product

tagMenuController
Attributes - Menu : tagMenu - Database : Database - Tagfilter : TagFilter - inputTags : ArrayList<String> - retrievedProducts : ArrayList<Product> - searchListener : ArrayList<ChangeListener> - backListener: ArrayList<ChangeListener>
Methods + tagMenuController(tagMenu: Database: TagFilter) : Constructor + initController() : void - cancel() : void - search() : void + addBackListener(ChangeListener) : void + addSearchListener(ChangeListener) : void + getRetrievedProducts() : ArrayList<Product> + getInputTags() : ArrayList <String>

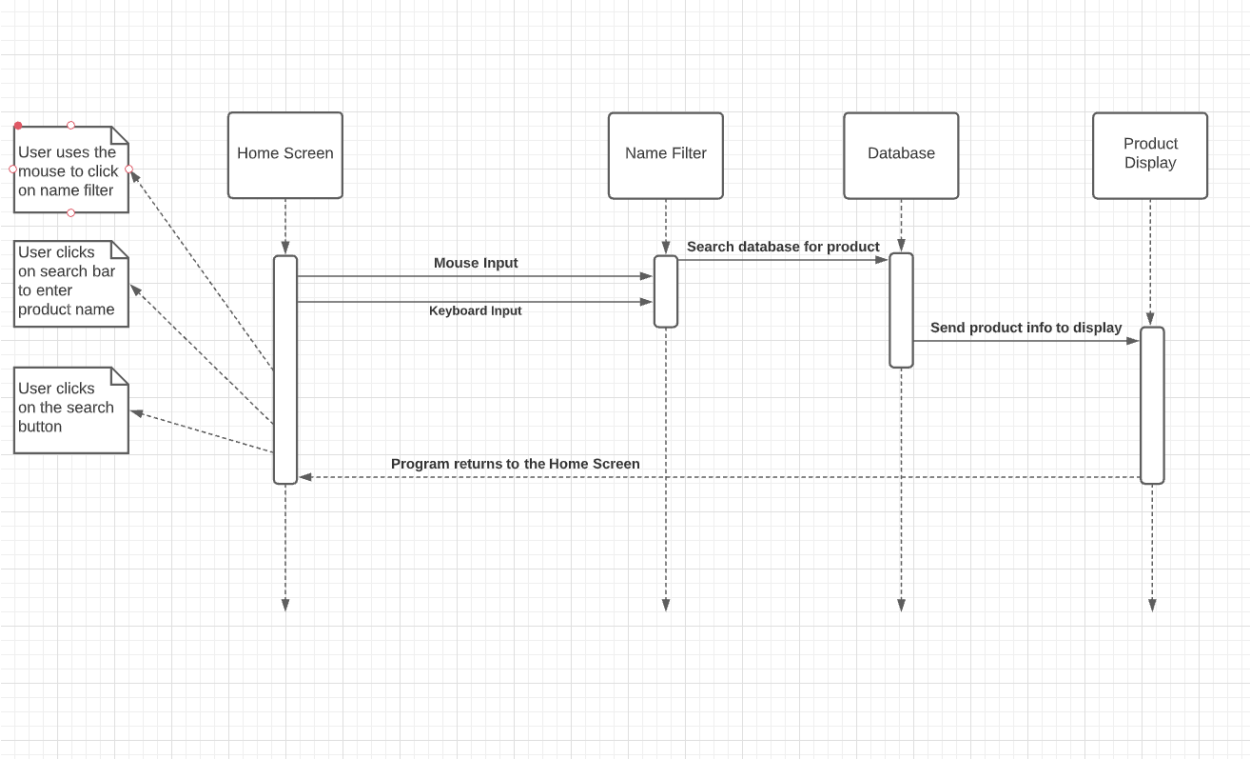
DynamicMain
Attributes - testDatabase : ActiveDatabase - importListeners : ArrayList<ChangeListener>
Methods + dynamicMain(inputActiveDatabase : ActiveDatavase) : Constructor + getProductSize() : int + addImportedListener(newListener : ChangeListener) : void + getTestDatabase() : ActiveDatabase

Class Relationship Diagram

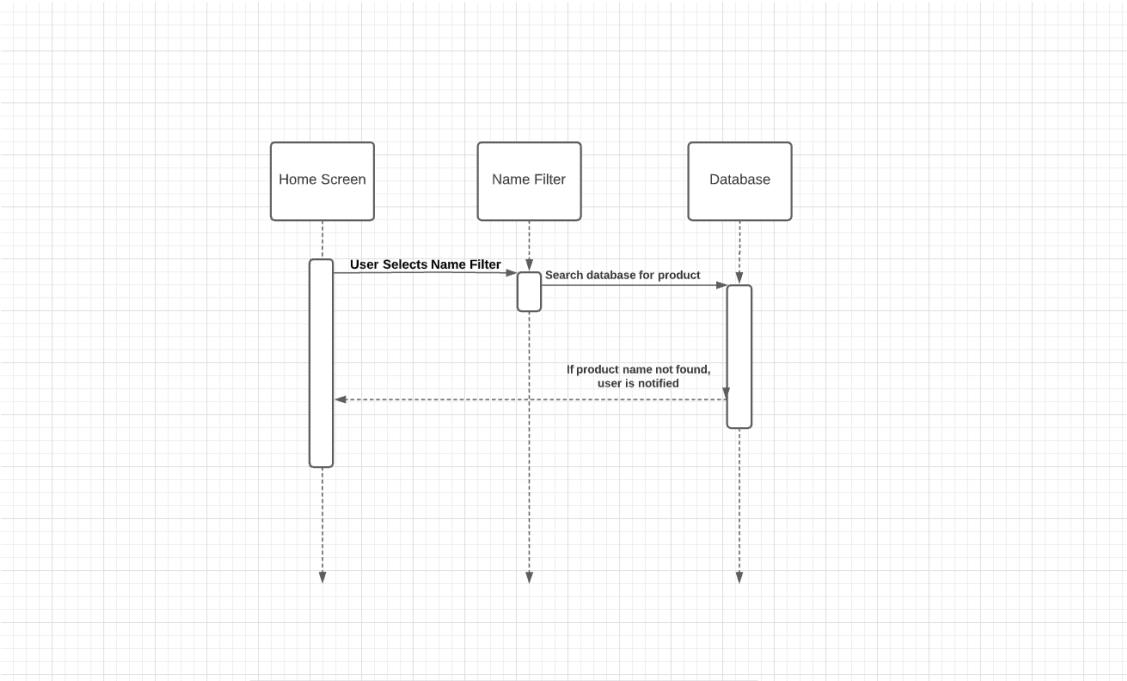


Class Sequence Diagrams

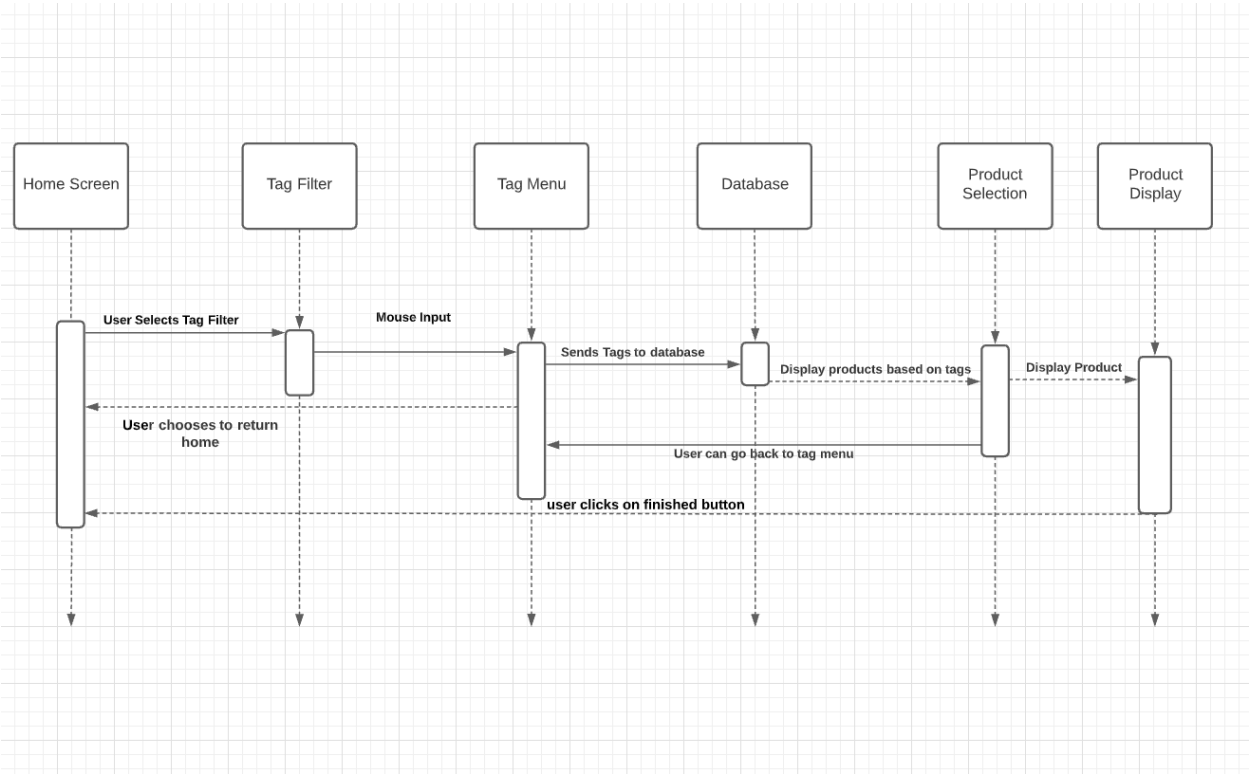
Case 1:



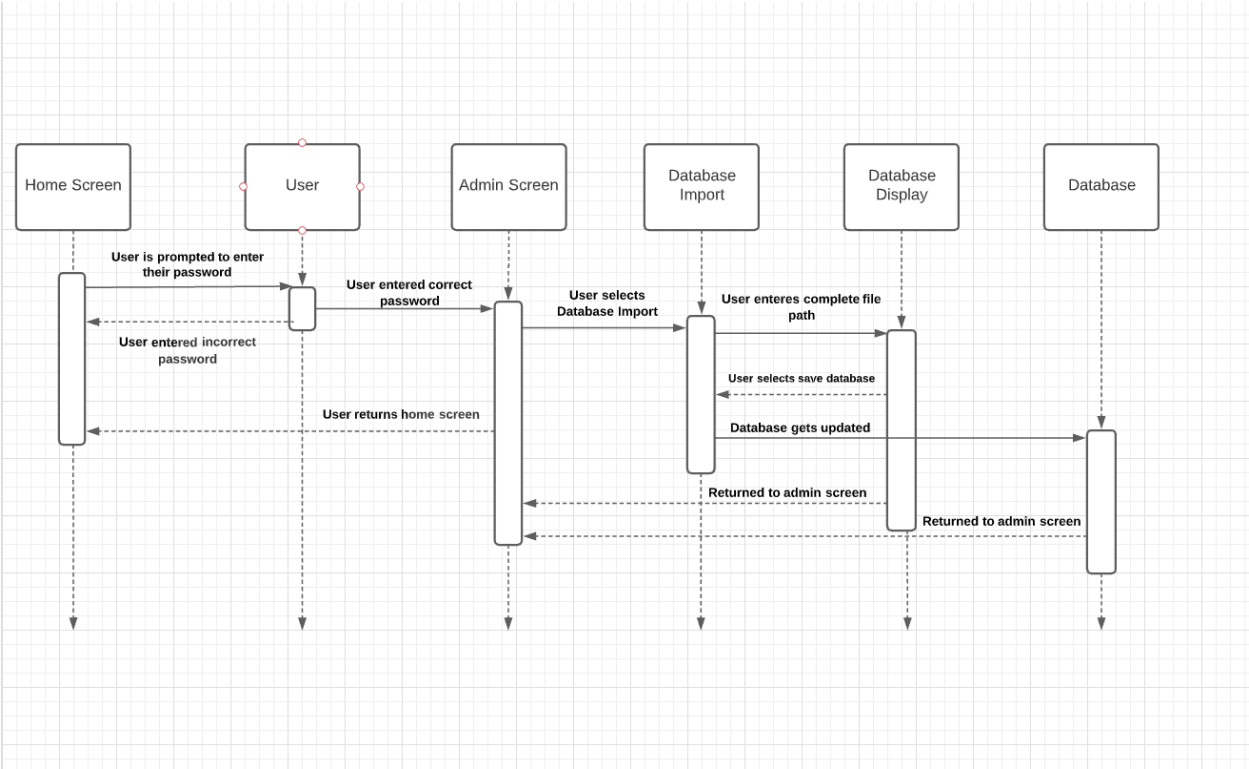
Case 2:


Home Screen
Name Filter
Database
User Selects Name Filter
Search database for product
If product name not found,
user is notified

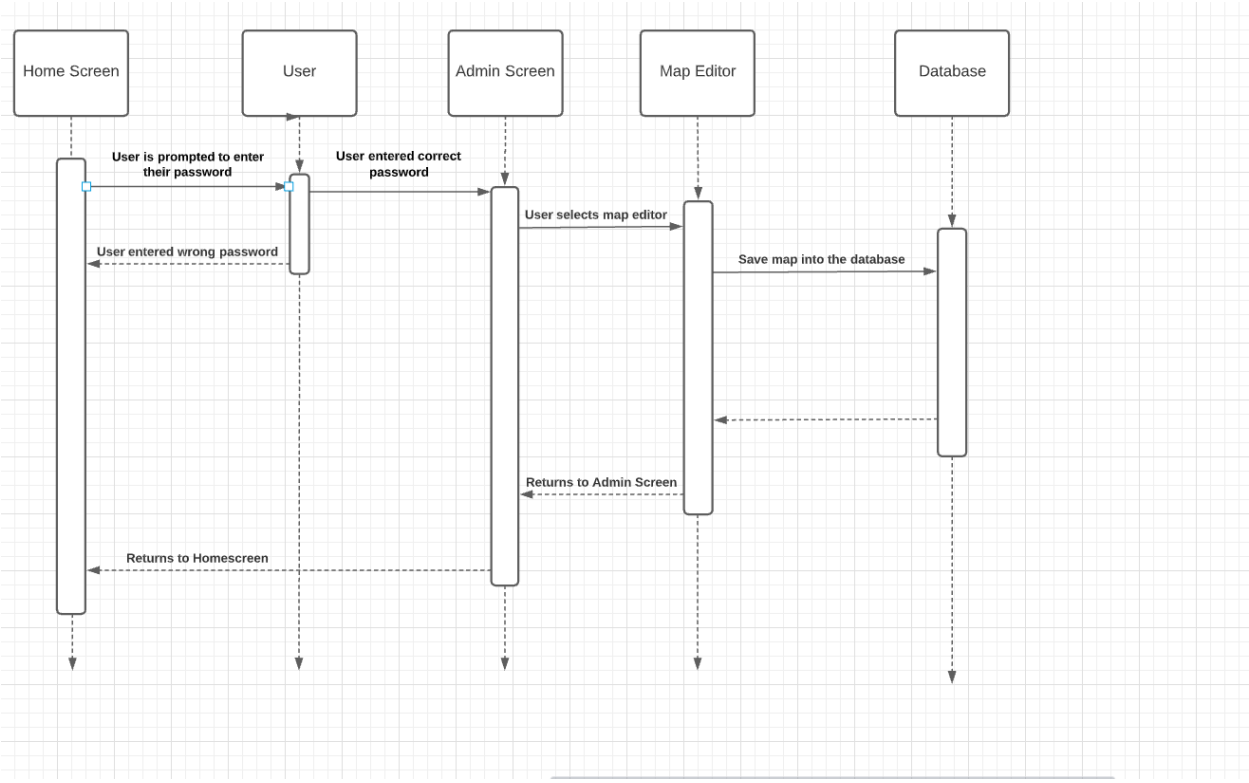
Case 3:


Home Screen
Tag Filter
Tag Menu
Database
Product Selection
Product Display
User Selects Tag Filter
Mouse Input
Sends Tags to database
Display products based on tags
Display Product
User chooses to return home
User can go back to tag menu
user clicks on finished button

Case 4:


Home Screen
User
Admin Screen
Database Import
Database Display
Database
User is prompted to enter their password
User entered correct password
User selects Database Import
User enteres complete file path
User entered incorrect password
User selects save database
User returns home screen
Database gets updated
Returned to admin screen
Returned to admin screen

Case 5:


Home Screen
User
Admin Screen
Map Editor
Database
User is prompted to enter their password
User entered correct password
User selects map editor
User entered wrong password
Save map into the database
Returns to Admin Screen
Returns to Homescreen

6.0 Implementation

Packages and classes:

Search Package:

NameFilter Class: The namefilter class is a concrete class that has a dependency type fship with the database package. The javadocs state that the class is used to search for a product in the database using the product's exact name. This class fills the design specification above regarding the idea that you need a way to determine what final product you are going to display to the user.

API: The API of the NameFilter is very simple and straightforward, it has a public function called retrieveByName it has two parameters a String and the Database its return type is a Product.

Product GUI Class: The ProductGUI class is a concrete class that has a dependency type relationship with the database package similar to NameFilter. It also inherits JPanel from the java library. In the javadocs it states that the class manages the GUI used to display product information. It is the final window that the user will see when using our program. This class fills the design specification addressed above regarding a window that houses all of the product components into one.

API: The API of ProductGUI is simple and straightforward; it has a single function that takes in a product and displays that product information.

ProductSelection Class: The ProductSelection class is a concrete class that has a dependency type relationship with the tagMenuController class. It also inherits JPanel from the java library. In the javadoc it states that it is the GUI that provides a list of relevant products based on the tags chosen by the user. This class gives the option of a product to select which is a necessary part of our design.

API: The API of productSelection is based on displaying the panel that we see and two functions that relate to buttons being selected on screen.

TagFilter Class: The tagFilter class is a concrete class that has a dependency type relationship with the tagMenuController and the database. The javadoc states that the class is used to search for a product based on tags in the database. This class generates the list for the productSelection list based on the tags entered by the user.

API: The API of tagFilter is a public function that searches for products based on tags entered. It takes in an ArrayList <String> of inputTags and the database. It also has a private function that is called checkDuplicates and this function is responsible for checking to make sure there are no duplicates that go into the final product selection.

tagMenu Class: The tagMenu class is a concrete class that has a dependency type relationship with the database as it must populate the view with tags within the database. It also inherits JPanel from the java library. The javadoc states that the class is used as a view class. It displays all of the tags into a list and displays them on the panel. This class is an integral part of the design as it shows the user which tags they can select when searching by tags.

API: The tagMenu class has a constructor that takes in the database and calls another private method within the class that creates the panel that the user sees. It has a couple of public get buttons. For example this menu has 2 buttons search, and cancel so it has gets for both of those. Also has a get for the JList used for the tags, and a ArrayList<String> get for the Uniquetags that is used to populate a JList.

tagMenuController Class: The tagMenuController class is a concrete class that has a dependency with the tagMenu class as it works with it to supply functionality to the buttons that are shown by the tagMenu view. The javadoc states that the class controls our buttons in tagMenu GUI. This class is responsible for controlling the tagMenu panel so that the buttons have functionality and it updates DynamicMain.

API: This class has a public constructor that takes in tagMenu, Database, and Tag Filter. It initializes all of these class variables and then it has a public function initController that just has listeners that listens for buttons pressed by the user. If a button is pressed it goes to its respective function which is private.

Database Package:

ActiveDatabase: The ActiveDatabase class is a concrete class that the software uses to store and retrieve data from. It contains one Database object that acts as a node within the structure. The javadocs state that this class is responsible for managing the database being used by the software.

API: The API of ActiveDatabase allows for the Database node stored within to be updated and accessed.

Database: The Database class is a concrete class that the software uses as a node within the ActiveDatabase class. This node is what holds the information being accessed and stored by ActiveDatabase. The javadocs state that this class is responsible for holding data regarding product info, passwords, and the store map.

API: The API of Database allows for its parent structure, ActiveDatabase, to access and manipulate data stored within (gets, sets, and a display function that provides a panel showing data contents).

DatabaseDisplay: The DatabaseDisplay class is a concrete class that inherits JPanel and serves to display the data contents within a Database. The javadocs state that this class is responsible for outputting the contents within a Database.

API: The API of DatabaseDisplay consists only of its constructor. This constructor creates the layout of the panel and adds the Database data into a viewable textbox.

DatabaseImport: The DatabaseImport class is a concrete class that extracts data from an input JSON file and stores it into a Database. This class depends on ActiveDatabase/Database to store data. The javadocs state that this class is responsible for parsing data from JSON files into a Database object.

API: The API consists of a constructor, a method to initiate importation, and a method to notify external classes if import was successful.

ImportController: The ImportController class is a concrete class that is currently responsible for ensuring that a file has been read prior to program start. The javadocs state that this class is used to mediate the importation process.

API: The API allows for an external class to initiate an import and access the new data.

ImportView: A concrete class that inherits Jpanel and is responsible for taking a filePath as user input and switches to the corresponding databaseDisplay Panel. According to the javadocs this class is responsible for creating a panel that searches for an import file.

API: The API consists of a constructor in charge of accepting a filePath and looking up filePath. It also has 2 Action Listeners and a clear and getNewImport function.

Product: A concrete class that implements cloneable and has a dependency relationship to database import and the database is dependent on the product class. This class is in charge of setting each product's name, tags, price, and location. According to the javadocs this class sets the product name and price as well as create an ArrayList for product tags and a new coordinate for the product location.

API: This class contains 2 constructors, one that accepts no parameters and one that accepts a product name and price. There are also "sets" and "gets" for each product attribute.

Core Package:

Main: A concrete class that is used to start the software. Main is responsible for calling ImportController and initializing the ActiveDatabase that is stored in DynamicMain.

DynamicMain: A concrete class that extends JFrame. DynamicMain is used to create the softwares main GUI frame, control which panel is currently displayed, and store the ActiveDatabase being used by the software.

API: DynamicMain has a public method to get the ActiveDatabase stored within DynamicMain.

AdminView: The AdminView class is a concrete class that has a dependency relationship to the userController class. With correct credentials, AdminView provides access to functions such as Map Editor, Database Import, and the Home Button. The JavaDocs states this class is for the GUI Implementation of AdminView.

API: In AdminView we have a createPanel which creates the GUI implementation of adminView. We have a constructor which we used for testing purposes to let us know when we enter this function. We also have 3 Listeners for addMain, addMap, and addImport.

HomeController: A concrete class that has a dependency relationship to HomeView. HomeController is in charge of changing which view is displayed within HomeView depending on the user's input. The JavaDocs states this is a class that controls the homeview when a user clicks on a button.

API: The HomeController contains a constructor which will initialize variables for each of the 3 views. We have a initController which simply switches to whichever view is clicked. We then have 3 functions for tagMenu, Search, and AdminView which will only execute if the respective button is pressed. We have a function to return the retrieved product for productGUI and 3 action listeners for each view. Then there is the final refresh function to update the database in the main menu.

HomeView: A concrete class that inherits Jpanel and is responsible for displaying the homescreen. This class has a dependency relationship to homeController. According to the javadocs this is a class that implements our GUI for the home screen.

API: This function contains a constructor which simply calls createFrame. The CreateFrame function simply calls addComponents (which is a container for our panels) with the settings associated with the frame. We also have 5 functions which are “gets” for userText, NameFilterToggle, tagMenuButton, searchButton, Adminbutton.

UserController: A concrete class that has a dependency relationship to UserView. This class is in charge of comparing passwords stored in our database to a user input and allows them access to the AdminView if their password is valid. According to the javadocs, this class compares the input password to the arrayList of correct passwords and returns true if they match.

API: This function has a constructor which sets the userView and database variables. We have a verifyPassword function which compares the user input to an arrayList of correct passwords and returns true if it matches. It also contains 3 listeners for addAdminLogin, addBack, and updateBack.

UserView: A concrete class that extends JPanel. It is used to create the GUI used when entering an administrator password.
API: This function has a constructor which simply calls createPanel which will create a JPanel with the correct options. We have 2 buttons for “back” and “enter” and a JTextField which will contain the user's password. We also have a “set” and “get” for the user password.

Map Package:

Coordinate: A concrete class that implements the Cloneable interface. It is used to store a pair of integer values representing coordinates of a location on the 2D map grid as a single object.

API: The Coordinate class contains public methods providing the ability to get and set the values of the integers stored in the coordinate object.

MapTemplate: An abstract class that extends JPanel and uses the Observer design pattern. It is used to provide common methods used by the MapEditor and ProductLocator classes, such as creating a 2D map grid, saving map data, and loading map data.

API: The MapTemplate class contains four public methods. The saveMapData and loadMapData methods provide the ability to save map data to and load map data from a String. The addMapSaveListener and addMapLoadListener methods allow other classes to be notified when a map panel attempts to save/load a map.

GridTile: A concrete class that extends JButton and uses the Observer design pattern. A GridTile stores an integer value representing the state of the tile and is able to change its appearance based on the current state of the tile.

API: The GridTile class contains public methods providing the ability to get and set the state of the tile. It also contains a method to register the map panel that the GridTile is a part of, allowing the panel to be notified when the GridTile is clicked.

MapEditor: A concrete class that extends MapTemplate and uses the Observer design pattern. It handles the GUI used for the map editor screen.

API: The MapEditor class contains a public method to get the current selection of the radio buttons used to select which type of tile to place when creating a map.

ProductLocator: A concrete class that extends MapTemplate. It handles the copy of the map used on the productGUI panel and is able to generate the shortest path between two points on the map.

API: The ProductLocator class contains a public method to generate the shortest path between two points on a map.

Utility classes and packages:

JSON Simple: Our project uses an open source library called JSON Simple. This utility provided us with JSON reading/writing capabilities that allowed us to store and retrieve data within our program.

Tested Functionality:

All use cases have been tested using a JSON file holding proper data. On the homescreen, the search filter and tag filter work as intended. The administer login window also functions as expected. The admin menu features also work as intended, including the map editor and database import. Upon first testing the transitions between panels, we detected a memory leak that occurred when adding new JPanels into a cardset in the main frame (DynamicMain class). This was remedied by deleting each panel from the cardset after the user was done viewing them.

Untested Functionality:

We did not test if the .JSON file was correctly formatted for the database. When we were viewing functions that we should test this was one of the last on the list because we felt that for this project we could make sure the .JSON file would be good for the demonstration. We were mainly focused on making sure the program ran through all of the Use Cases and making sure those work with no issues, and unfortunately we did not get around to this because of the time constraints. Additionally, we assumed the users of this software are already aware of the file formatting requirements.

7.0 Discussion

One of the main lessons that our team learned from this project was how to create GUIs and their associated controllers. Prior to the project, all members were proficient with coding data oriented classes. This project gave us valuable experience on the inner workings behind GUI code. One of the main challenges that we faced was cycling between view classes. At first, we try calling multiple frames at once and updating data through static main. This proved to be inefficient, as the static main function would not allow variables to be updated once the code was updated. To work around this, we decided to create a class called ‘DynamicMain’. This class served as the main frame for our software. Within DynamicMain, we created a card set that cycled through our view classes (we had to convert them from JFrames to Jpanels). Since DynamicMain was an active frame, all variables could be updated in real time (hence the “dynamic” part of the name).

Another issue we ran into was importing data into the program. We opted to use JSON files as a medium of importation. What we weren’t aware of, however, was that JSON reading/writing capabilities was not native to the Java language. Instead, we had to include an open source JSON library to our project called JSON-Simple. After a day of reviewing the library’s documentation, we were able to fully implement logic for importing data into the program.

One area that could have been improved in our code is the ImportController class logic. Currently, the importController class is called only once in the program when it first runs. This class is essential to the program, as it ensures that a file is read into the software before use. Our original goal was to also call this function within the DynamicMain class when an admin wanted to import a file after the program is already in use. Unfortunately, due to how the ChangeListeners and EventListeners function in our code, we could not directly call the class inside DynamicMain. Instead, we opted to copy some of the logic in the ImportController class and paste it within DynamicMain. If we had more time for this project, we would like to have liked to rework the class slightly for better compatibility with DynamicMain.

Appendix A: CRC Cards

Database Import

Responsibilities	Class Associations
• Read a specific file type and parse data for item attributes.	***Verify Login***

User

Responsibilities

- Compare password entered by user to check if it is a match.
- If the user password is correct they will have administrator permissions.
- If an unauthorized user attempts to log into the administrative mode with an incorrect password too many times, the program will be “soft locked”.

Class Associations

Database

Tag Filter

Responsibilities

- Compare tag entered by user with every items’ tag in the database and find relevant products.

Class Associations

Database

Name Filter

Responsibilities	Class Associations
• Compare item name entered by user with every item name in the database and find relevant products.	***Database***

Database

Responsibilities	Class Associations
• Stores products.	**Import Database**
	Products
• Store admin passwords.	
• Search for a product based on name.	
• Search for a product based on tags.	
• Store the amount of products in the database.	
• Provides products for searches.	

Product

Responsibilities	Class Associations
• Store the product name in a variable.	**Database** **Coordinate Class**
• Store the product price in a variable.	
• Store the product tags in a variable.	
• Store the product location in a variable.	

Map Editor

Responsibilities	Class Associations
• Allow user to select squares from a grid in order to create a store map.	***Verify Login*** ***Coordinate Class***
• Allow users to load and save maps.	

Coordinate Class

Responsibilities	Class Associations
• Allow user to save coordinate values into one object.	

Map Class

Responsibilities	Class Associations
• Allows user to save map for later use in finding item location.	***Coordinate Class***