Student:Dragoteanu Bogdan
Group:30431

Table of Contents

....1

....1

Student:..1

Table of Contents..2

1. Requirements Analysis 3
    1.1Assignment Specification..3
    1.2Functional Requirements..3
    1.3Non-functional Requirements..3

2. Use-Case Model..3

3. System Architectural Design..3

4. UML Sequence Diagrams..3

5. Class Design..3

6. Data Model 3

7. System Testing..4

8. Bibliography..4

1. Requirements Analysis

1. Assignment Specification

The assignment is an application that helps users manage food waste.

Authenticated users can input grocery lists and see reports of how much food is wasted weekly and monthly. An item in the grocery list has the following data:

Name,quantity, calorie value, purchase date, expiration date and consumption date

The system allows the users to track the goals and minimize waste by reminding them if the waste levels are too high based on ideal burn down rates.

The system gives the users options to donate excess food to various local food charities and soup kitchens.

If an item expiration is close to the current date the user is notified.

2. Functional Requirements

- Search → The user is able to find data about the grocery item by searching explicitly for it
- Reports → The system will generate a weekly and monthly report for the user
- Modify → The user is able to change a grocery item's information
- AddItem → Insert a new item

2.1 Non-functional Requirements

- Security → The system shall ensure that data is protected from unauthorized access
  → System data may be accessed only by users authenticated by means of a
username and password
- Portability → The application works on both Windows and Linux machines
- Extendibility → Features can be further enriched

2. Use-Case Model

Use case: Log In

Level: user-goal

Primary actor: Registered User

Main success scenario: User log into the system

Extensions: On fail it notifies the user that something is wrong

Use case: Sign Up

Level: user-goal

Primary actor: User

Main success scenario:

Extensions:

Use case: Search

Level: user-goal

Primary actor: Registered User

Main success scenario: The system finds the item in the list and shows its data to the user

Extensions: On fail it notifies the user that the item doesn't exist or that they may have misspelled its name

Use case: Refresh

Level: user-goal

Primary actor: Registered User

Main success scenario: Refreshes the Grocery List

Extensions: -

Use case: Modify

Level: user-goal

Primary actor: Registered User

Main success scenario: Changes the data of a selected item in the List

Extensions: If some parameters are bad then the user is notified

Use case: Add Item to Grocery List

Level: user-goal

Primary actor: Registered User

Main success scenario: Adds the data of a new item to the List

Extensions: If some parameters are bad then the user is notified

Use case: Create Report

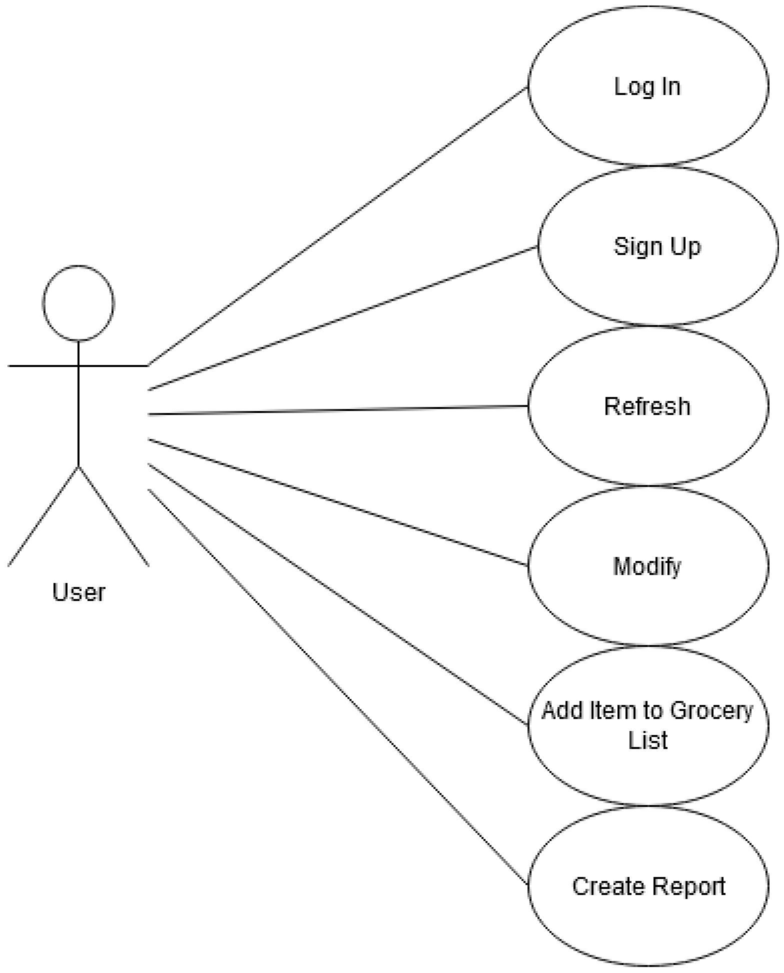
Level: user-goal

Primary actor: Registered User

Main success scenario: Creates report with the data from thegrocery list and shows it to the user.

Type depends on the chosen one (Weekly/Monthly).

Extensions: If the list is empty the user is notified that nothing can be shown.



3. System Architectural Design

The system will be made using the layered architecture.

3.1 Architectural Pattern Description

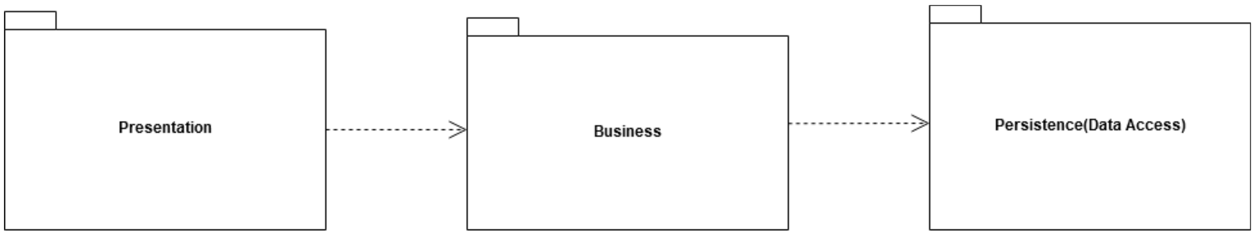
Components in a layered architecture are organized into horizontal layers, each performing a specific role within the application. In our case we have four layers:

- Presentation → handles the user interface
- Business → handles requests
- Persistence (Data Access) → contains database tech
- Database

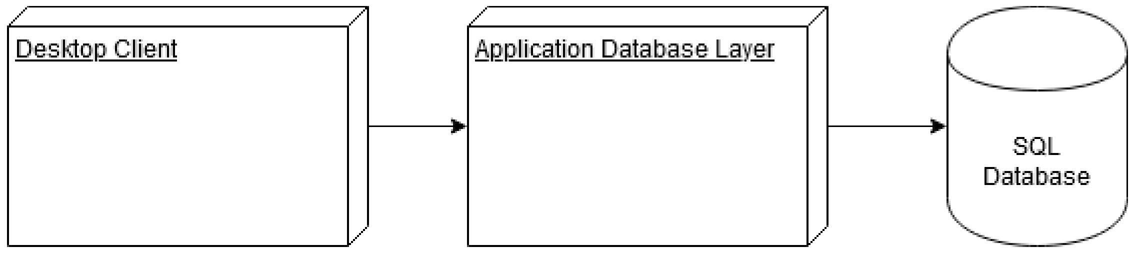
The layers are closed → requests move from layer to layer:
presentation → business → persistence → database

3.2 Diagrams

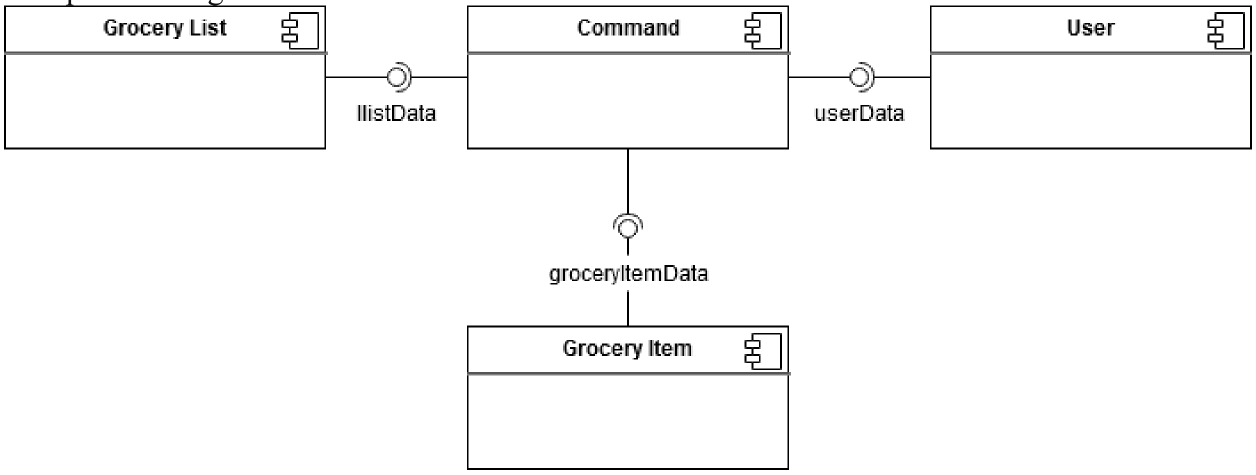
Package Diagram



Deployment Diagram



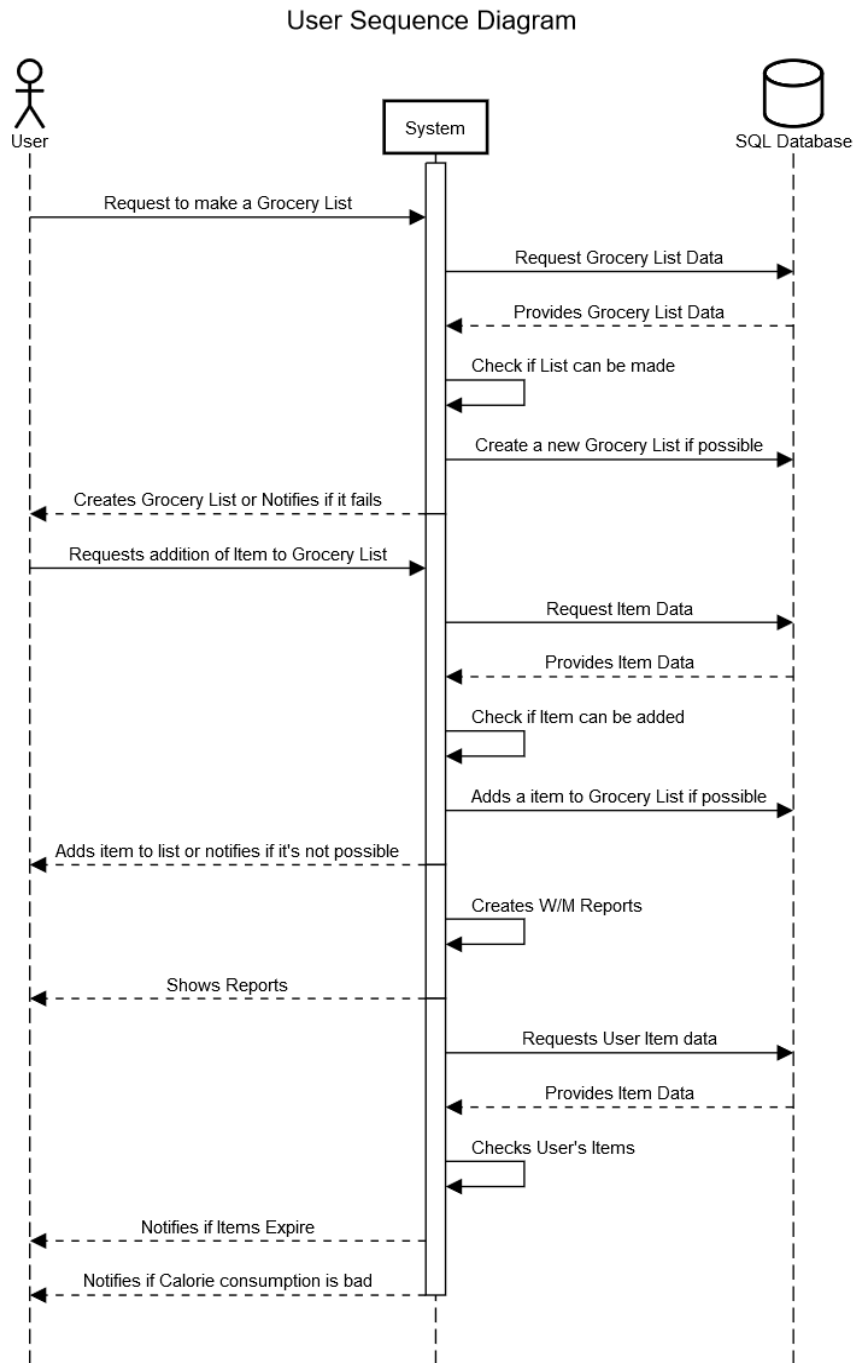
Component Diagram



Patterns used:

- layered → used to structure the program into groups of subtasks
  → each layer provides services to the next higher layer
- Dependency Injection pattern - Used to make the main class independent of its functions
- ORM (Object Relational Mapping) – Used to hide database work from user
- Maps a class using an xml file based on which it knows what to look for in the database and retrieve to it

4. UML Sequence Diagrams


User Sequence Diagram
User
System
SQL Database
Request to make a Grocery List
Request Grocery List Data
Provides Grocery List Data
Check if List can be made
Create a new Grocery List if possible
Creates Grocery List or Notifies if it fails
Requests addition of Item to Grocery List
Request Item Data
Provides Item Data
Check if Item can be added
Adds a item to Grocery List if possible
Adds item to list or notifies if it's not possible
Creates W/M Reports
Shows Reports
Requests User Item data
Provides Item Data
Checks User's Items
Notifies if Items Expire
Notifies if Calorie consumption is bad

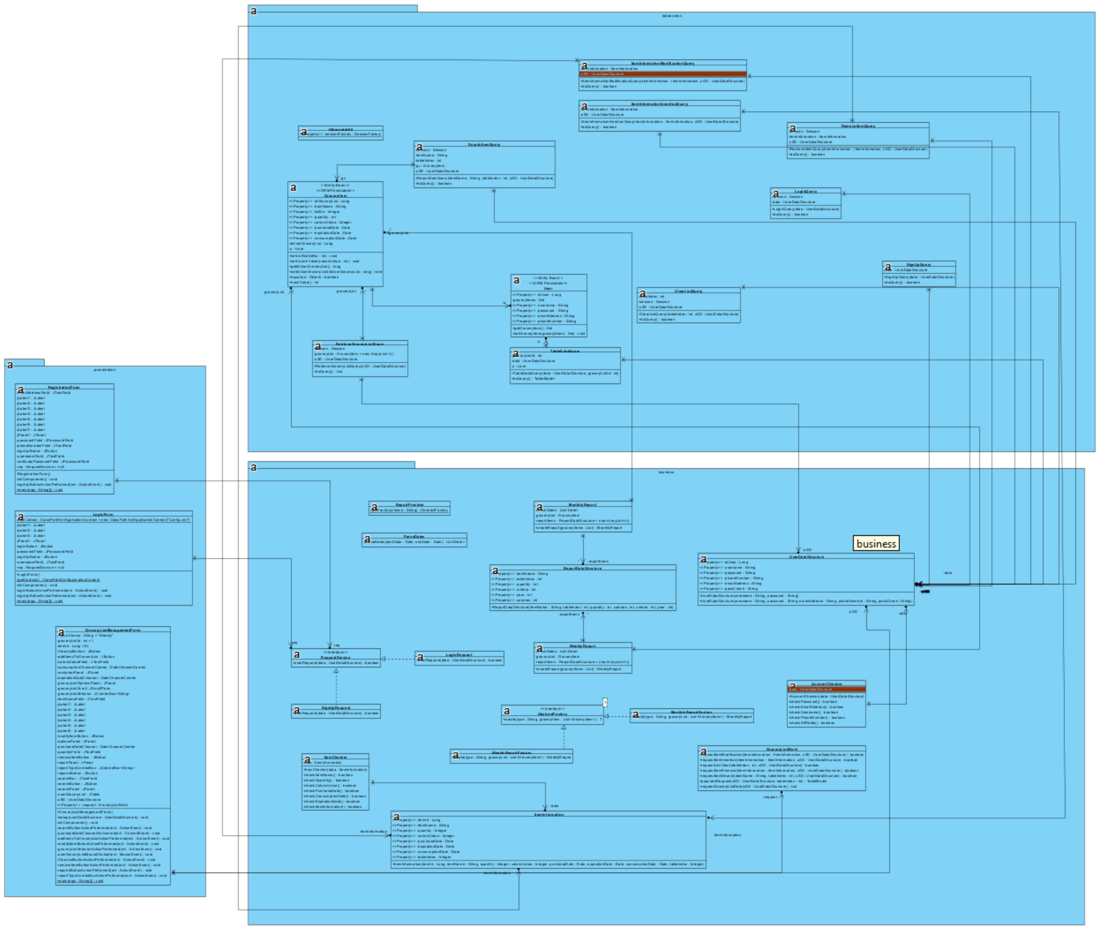
5. Class Design

5.1 Design Patterns Description

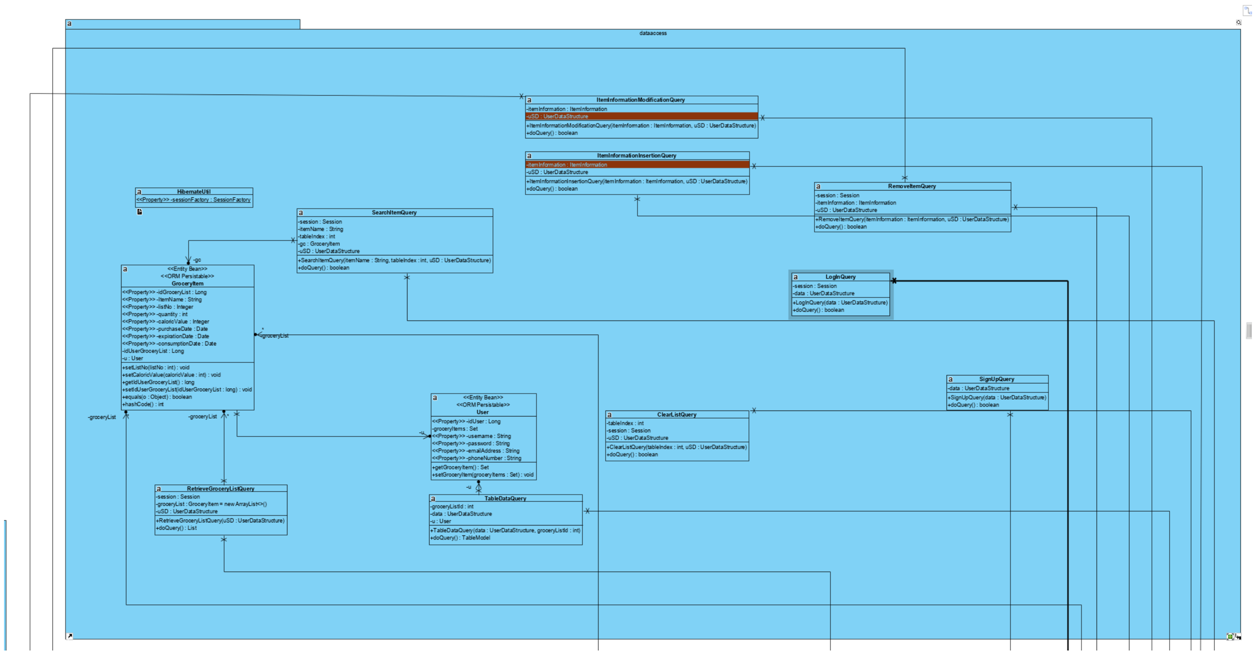
- Abstract Factory → Used in the creation of Weekly/Monthly Reports

  → Provides an interface for creating families of related or dependent objects without specifying their concrete classes.

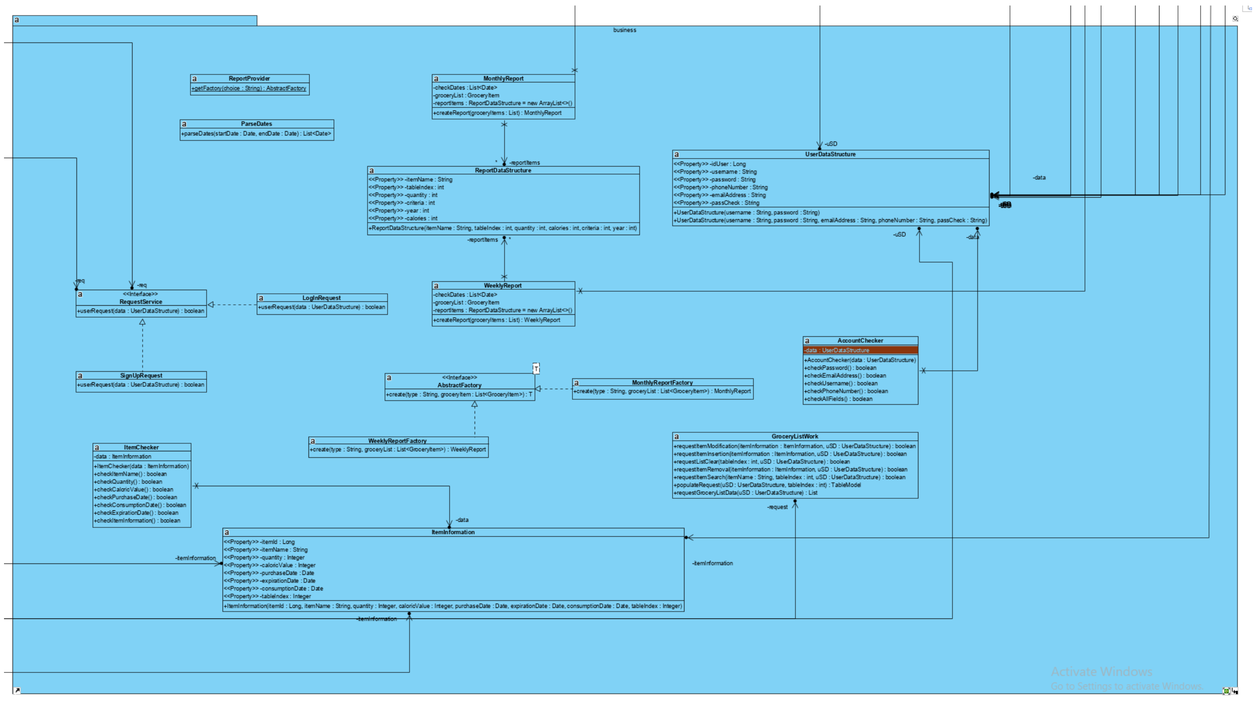
5.2 UML Class Diagram



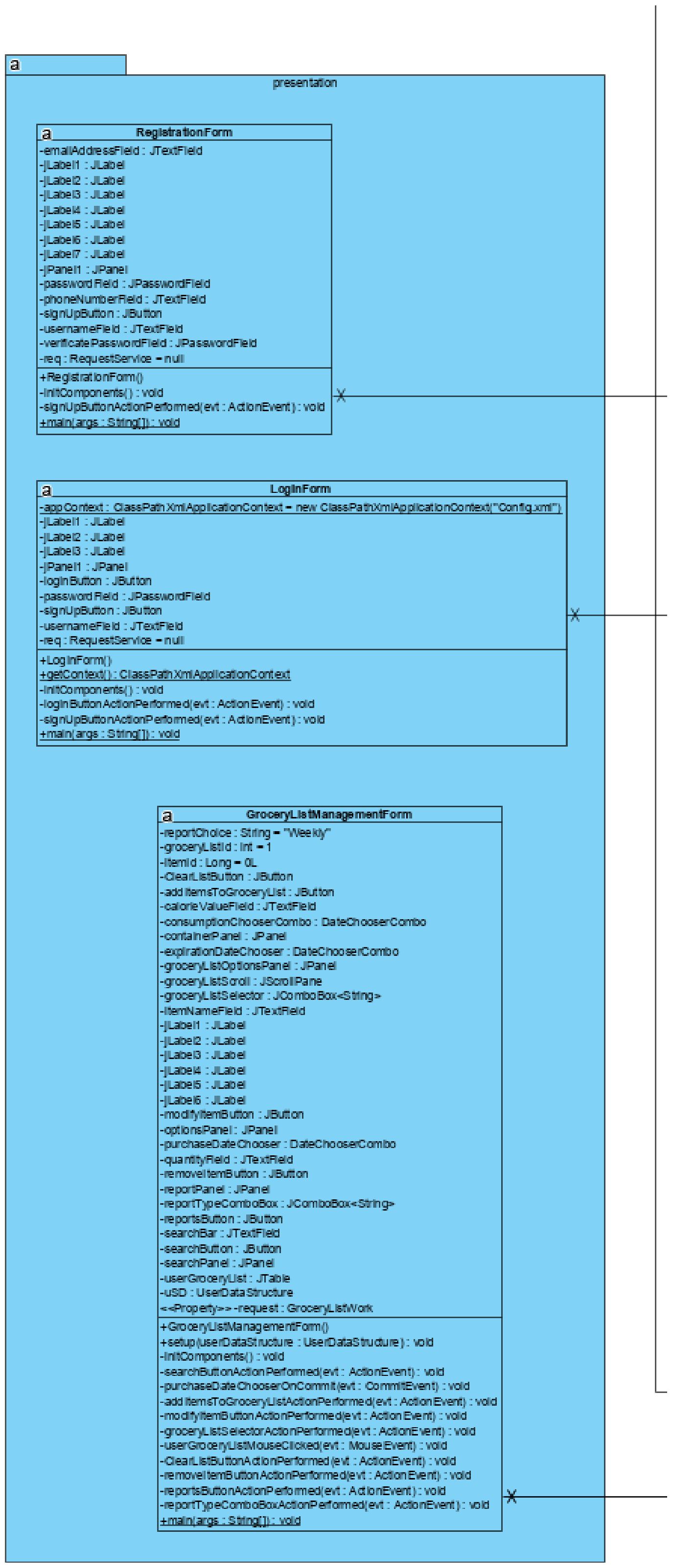
DataAccess Package (Layer)


dataaccess
ItemInformationModificationQuery
-itemInformation : ItemInformation
-uSD : UserDataStructure
+ItemInformationModificationQuery(itemInformation : ItemInformation, uSD : UserDataStructure)
+doQuery() : boolean
ItemInformationInsertionQuery
-itemInformation : ItemInformation
-uSD : UserDataStructure
+ItemInformationInsertionQuery(itemInformation : ItemInformation, uSD : UserDataStructure)
+doQuery() : boolean
RemoveItemQuery
-session : Session
-itemInformation : ItemInformation
-uSD : UserDataStructure
+RemoveItemQuery(itemInformation : ItemInformation, uSD : UserDataStructure)
+doQuery() : boolean
HibernateUtil
<<Property>> -sessionFactory : SessionFactory
SearchItemQuery
-session : Session
-itemName : String
-tableIndex : int
-gc : GroceryItem
-uSD : UserDataStructure
+SearchItemQuery(itemName : String, tableIndex : int, uSD : UserDataStructure)
+doQuery() : boolean
-gc
<<Entity Bean>>
<<ORM Persistable>>
GroceryItem
<<Property>> -idGroceryList : Long
<<Property>> -ItemName : String
<<Property>> -listNo : Integer
<<Property>> -quantity : int
<<Property>> -caloricValue : Integer
<<Property>> -purchaseDate : Date
<<Property>> -expirationDate : Date
<<Property>> -consumptionDate : Date
-idUserGroceryList : Long
-u : User
+setListNo(listNo : int) : void
+setCaloricValue(caloricValue : int) : void
+getIdUserGroceryList() : long
+setIdUserGroceryList(idUserGroceryList : long) : void
+equals(o : Object) : boolean
+hashCode() : int
-groceryList
LogInQuery
-session : Session
-data : UserDataStructure
+LogInQuery(data : UserDataStructure)
+doQuery() : boolean
SignUpQuery
-data : UserDataStructure
+SignUpQuery(data : UserDataStructure)
+doQuery() : boolean
<<Entity Bean>>
<<ORM Persistable>>
User
<<Property>> -idUser : Long
-groceryItems : Set
<<Property>> -username : String
<<Property>> -password : String
<<Property>> -emailAddress : String
<<Property>> -phoneNumber : String
+getGroceryItem() : Set
+setGroceryItem(groceryItems : Set) : void
-u
ClearListQuery
-tableIndex : int
-session : Session
-uSD : UserDataStructure
+ClearListQuery(tableIndex : int, uSD : UserDataStructure)
+doQuery() : boolean
RetrieveGroceryListQuery
-session : Session
-groceryList : GroceryItem = new ArrayList<>()
-uSD : UserDataStructure
+RetrieveGroceryListQuery(uSD : UserDataStructure)
+doQuery() : List
TableDataQuery
-groceryListId : int
-data : UserDataStructure
-u : User
+TableDataQuery(data : UserDataStructure, groceryListId : int)
+doQuery() : TableModel

Business Package (Layer)


business
ReportProvider
+getFactory(choice : String) : AbstractFactory
ParseDates
+parseDates(startDate : Date, endDate : Date) : List<Date>
MonthlyReport
-checkDates : List<Date>
-groceryList : GroceryItem
-reportItems : ReportDataStructure = new ArrayList<>()
+createReport(groceryItems : List) : MonthlyReport
-reportItems
ReportDataStructure
<<Property>> -itemName : String
<<Property>> -tableIndex : int
<<Property>> -quantity : int
<<Property>> -criteria : int
<<Property>> -year : int
<<Property>> -calories : int
+ReportDataStructure(itemName : String, tableIndex : int, quantity : int, calories : int, criteria : int, year : int)
-uSD
UserDataStructure
<<Property>> -idUser : Long
<<Property>> -username : String
<<Property>> -password : String
<<Property>> -phoneNumber : String
<<Property>> -emailAddress : String
<<Property>> -passCheck : String
+UserDataStructure(username : String, password : String)
+UserDataStructure(username : String, password : String, emailAddress : String, phoneNumber : String, passCheck : String)
-data
-req
<<Interface>>
RequestService
+userRequest(data : UserDataStructure) : boolean
LogInRequest
+userRequest(data : UserDataStructure) : boolean
WeeklyReport
-checkDates : List<Date>
-groceryList : GroceryItem
-reportItems : ReportDataStructure = new ArrayList<>()
+createReport(groceryItems : List) : WeeklyReport
AccountChecker
-data : UserDataStructure
+AccountChecker(data : UserDataStructure)
+checkPassword() : boolean
+checkEmailAddress() : boolean
+checkUsername() : boolean
+checkPhoneNumber() : boolean
+checkAllFields() : boolean
SignUpRequest
+userRequest(data : UserDataStructure) : boolean
<<Interface>>
AbstractFactory
+create(type : String, groceryItem : List<GroceryItem>) : T
T
MonthlyReportFactory
+create(type : String, groceryList : List<GroceryItem>) : MonthlyReport
WeeklyReportFactory
+create(type : String, groceryList : List<GroceryItem>) : WeeklyReport
GroceryListWork
+requestItemModification(itemInformation : ItemInformation, uSD : UserDataStructure) : boolean
+requestItemInsertion(itemInformation : ItemInformation, uSD : UserDataStructure) : boolean
+requestListClear(tableIndex : int, uSD : UserDataStructure) : boolean
+requestItemRemoval(itemInformation : ItemInformation, uSD : UserDataStructure) : boolean
+requestItemSearch(itemName : String, tableIndex : int, uSD : UserDataStructure) : boolean
+populateRequest(uSD : UserDataStructure, tableIndex : int) : TableModel
+requestGroceryListData(uSD : UserDataStructure) : List
-request
ItemChecker
-data : ItemInformation
+ItemChecker(data : ItemInformation)
+checkItemName() : boolean
+checkQuantity() : boolean
+checkCaloricValue() : boolean
+checkPurchaseDate() : boolean
+checkConsumptionDate() : boolean
+checkExpirationDate() : boolean
+checkItemInformation() : boolean
ItemInformation
<<Property>> -itemId : Long
<<Property>> -itemName : String
<<Property>> -quantity : Integer
<<Property>> -caloricValue : Integer
<<Property>> -purchaseDate : Date
<<Property>> -expirationDate : Date
<<Property>> -consumptionDate : Date
<<Property>> -tableIndex : Integer
+ItemInformation(itemId : Long, itemName : String, quantity : Integer, caloricValue : Integer, purchaseDate : Date, expirationDate : Date, consumptionDate : Date, tableIndex : Integer)
-itemInformation

Presentation Package (Layer)


presentation
RegistrationForm
-emailAddressField : JTextField
-jLabel1 : JLabel
-jLabel2 : JLabel
-jLabel3 : JLabel
-jLabel4 : JLabel
-jLabel5 : JLabel
-jLabel6 : JLabel
-jLabel7 : JLabel
-jPanel1 : JPanel
-passwordField : JPasswordField
-phoneNumberField : JTextField
-signUpButton : JButton
-usernameField : JTextField
-verificatePasswordField : JPasswordField
-req : RequestService = null
+RegistrationForm()
-initComponents() : void
-signUpButtonActionPerformed(evt : ActionEvent) : void
+main(args : String[]) : void
LoginForm
-appContext : ClassPathXmlApplicationContext = new ClassPathXmlApplicationContext("Config.xml")
-jLabel1 : JLabel
-jLabel2 : JLabel
-jLabel3 : JLabel
-jPanel1 : JPanel
-loginButton : JButton
-passwordField : JPasswordField
-signUpButton : JButton
-usernameField : JTextField
-req : RequestService = null
+LoginForm()
+getContext() : ClassPathXmlApplicationContext
-initComponents() : void
-loginButtonActionPerformed(evt : ActionEvent) : void
-signUpButtonActionPerformed(evt : ActionEvent) : void
+main(args : String[]) : void
GroceryListManagementForm
-reportChoice : String = "Weekly"
-groceryListId : int = 1
-itemId : Long = 0L
-ClearListButton : JButton
-addItemsToGroceryList : JButton
-calorieValueField : JTextField
-consumptionChooserCombo : DateChooserCombo
-containerPanel : JPanel
-expirationDateChooser : DateChooserCombo
-groceryListOptionsPanel : JPanel
-groceryListScroll : JScrollPane
-groceryListSelector : JComboBox<String>
-itemNameField : JTextField
-jLabel1 : JLabel
-jLabel2 : JLabel
-jLabel3 : JLabel
-jLabel4 : JLabel
-jLabel5 : JLabel
-jLabel6 : JLabel
-modifyItemButton : JButton
-optionsPanel : JPanel
-purchaseDateChooser : DateChooserCombo
-quantityField : JTextField
-removeItemButton : JButton
-reportPanel : JPanel
-reportTypeComboBox : JComboBox<String>
-reportsButton : JButton
-searchBar : JTextField
-searchButton : JButton
-searchPanel : JPanel
-userGroceryList : JTable
-uSD : UserDataStructure
<<Property>> -request : GroceryListWork
+GroceryListManagementForm()
+setup(userDataStructure : UserDataStructure) : void
-initComponents() : void
-searchButtonActionPerformed(evt : ActionEvent) : void
-purchaseDateChooserOnCommit(evt : CommitEvent) : void
-addItemsToGroceryListActionPerformed(evt : ActionEvent) : void
-modifyItemButtonActionPerformed(evt : ActionEvent) : void
-groceryListSelectorActionPerformed(evt : ActionEvent) : void
-userGroceryListMouseClicked(evt : MouseEvent) : void
-ClearListButtonActionPerformed(evt : ActionEvent) : void
-removeItemButtonActionPerformed(evt : ActionEvent) : void
-reportsButtonActionPerformed(evt : ActionEvent) : void
-reportTypeComboBoxActionPerformed(evt : ActionEvent) : void
+main(args : String[]) : void

As it can be seen from the 4 images the structure of the program is devided into the three parts mentioned at the architectural pattern. This separates the data flow into 3 sections which makes it easier to troubleshoot and modify.
Abstract Factory is used to generate Factories to create reports based on the user's preference. This is a grouping of individual factories that have a common theme but no concrete classes. It separates the details of implementation the sets of objects WeeklyReport and Monthly Reports from their general usage and relies on object composition, as object creation is implemented in methods exposed in the factory interface .
Dependency Injection is used in the Log In Screen to create the Main Screen (GroceryListManagementForm) it knows form the xml file what the usage of the class is and injects the needed dependencies into it.
ORM (Object Relational Mapping) has three important classes that are used:

- Users
- GroceryItem
- HibernateUtils

User contains the database structure of the table that contains user data. The setters defined here are used to pass the data form the database to the class while the getters are used in order to retrieve the stored database data. Same for GroceryItem but in this case it retrieves data from the table that contains the grocery lists.
HibernateUtils is the base of the session creation in the dataaccess layer. By calling this class's function a session is created which is then used for the database communication.

6. Data Model

Relational Model for Database information:

- This model is based on first-order predicate logic and defines a table as an n-ary relation.
- Data is stored in tables called relations.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.

Entity-Relationship Model in DatabaseMS:

- Entity – is a real-word entity having attributes
  – attributes are defined by a domain
- Relationship – Mapped entities (in our case one to one (ex: PrimaryUserKey to PrimaryGroceryListKey))

7. System Testing

The testing method used was Unit Testing:

- Program was tested step by step through snippets of code
- Input data is verified by means of two classes AccountCheker and ItemChecker which look for bad inputs by seeing if they are in certain intervals (boundary analisys)
- Data Flow testing has been used in several areas to check if there's a proper flow (report making, retrieving Grocery List data). Here the data was checked to be:

→ properly defined in place

→ used properly (completely) for their usage

8. Bibliography

[1] www. stackoverflow.com → Used for explanations of:

- email checking
- abstract factory (?)
- parsing dates

[2] https://www.baeldung.com → Used for explanations of:

- Hibernate
- Some of Calendar's gimmicks
- Spring Container