

Student: Dragotanu Bogdan
Group: 30431

Table of Contents

.....	1
.....	1
Student:.....	1
Table of Contents.....	2
1. Requirements Analysis	3
1.1 Assignment Specification.....	3
1.2 Functional Requirements.....	3
1.3 Non-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 consumption date 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

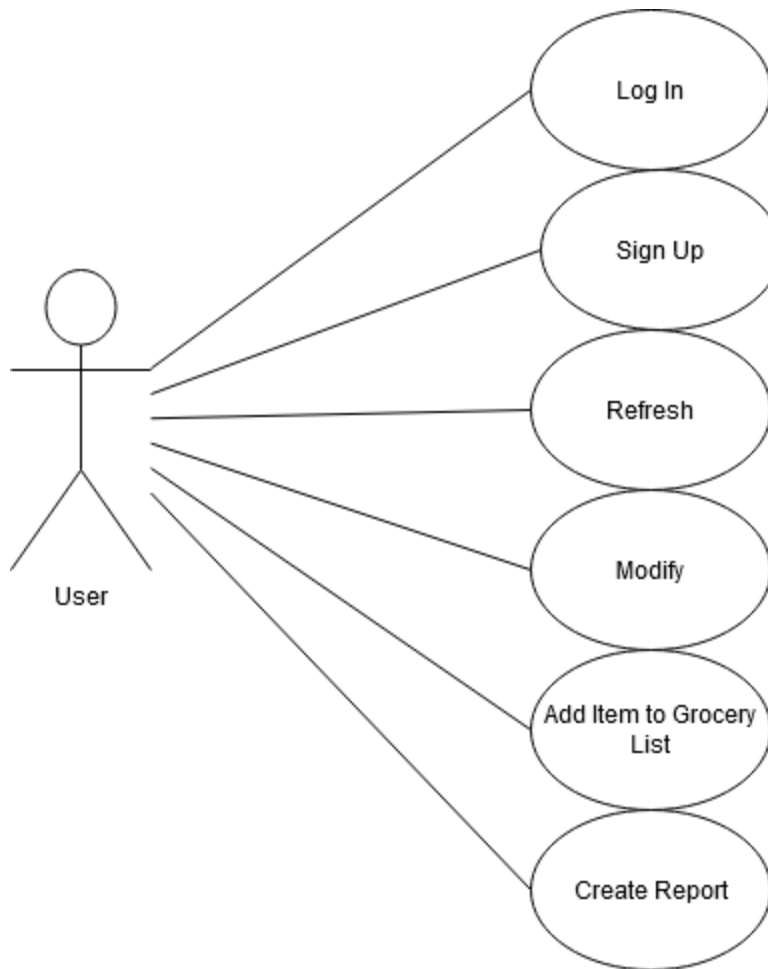
Level: user-goal

Primary actor: Registered User

Main success scenario: Creates report with the data from the grocery 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 a client server architecture.

The client will be made using a 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:

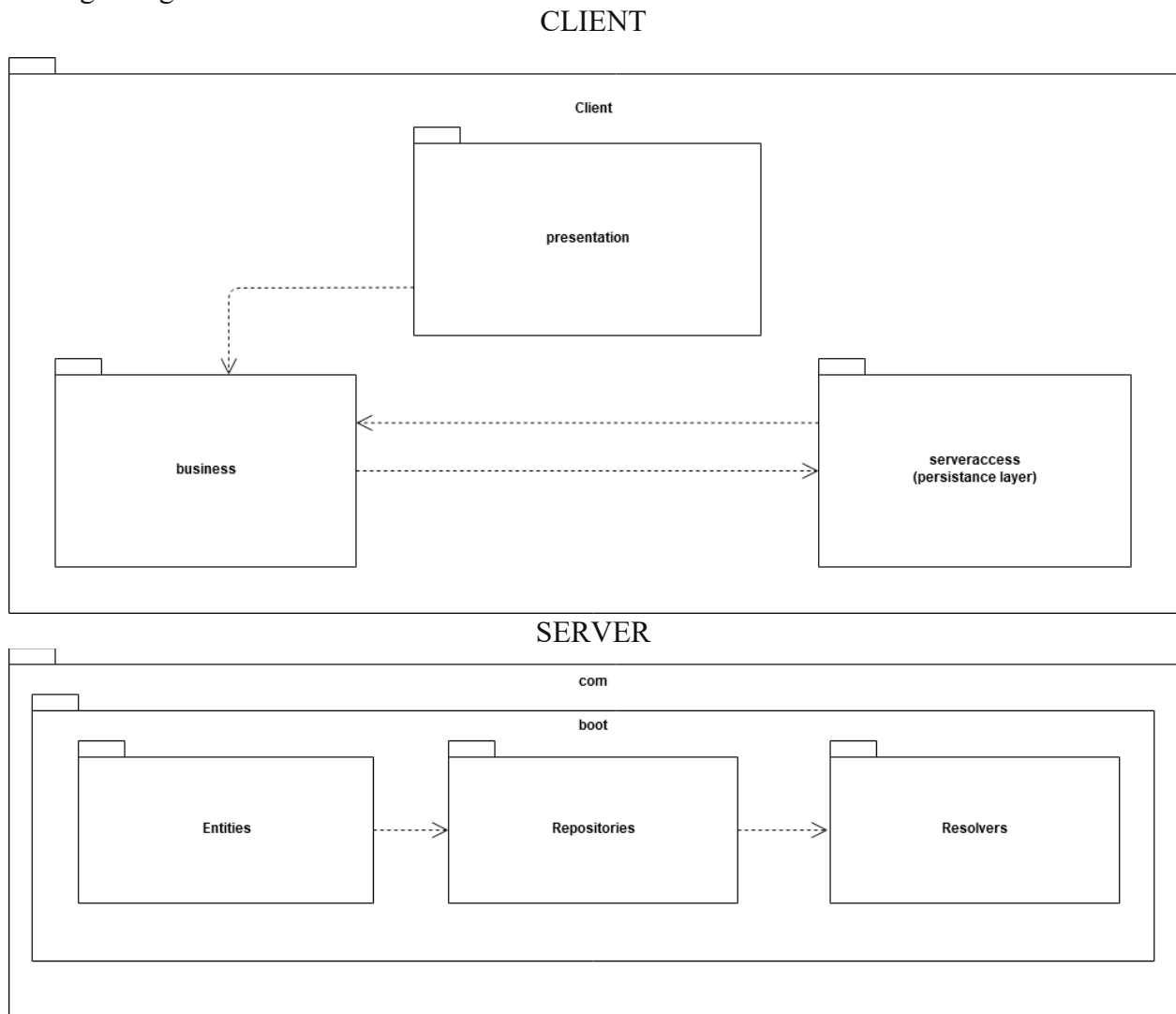
- Presentation → handles the user interface
- Business → handles requests
- Persistence (Data Access) → contains database tech
- Server

The layers are closed → requests move from layer to layer:

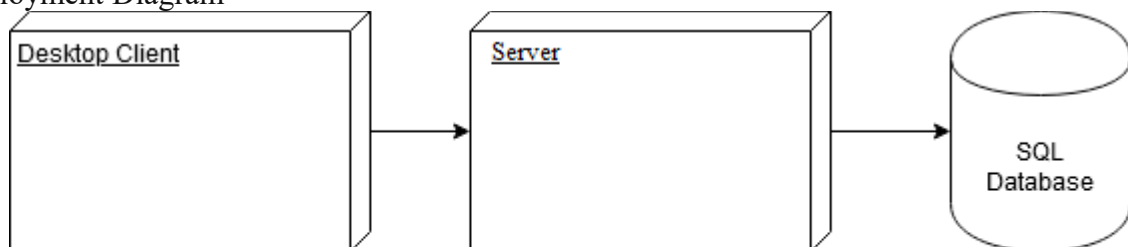
presentation → business → persistence → server

3.2 Diagrams

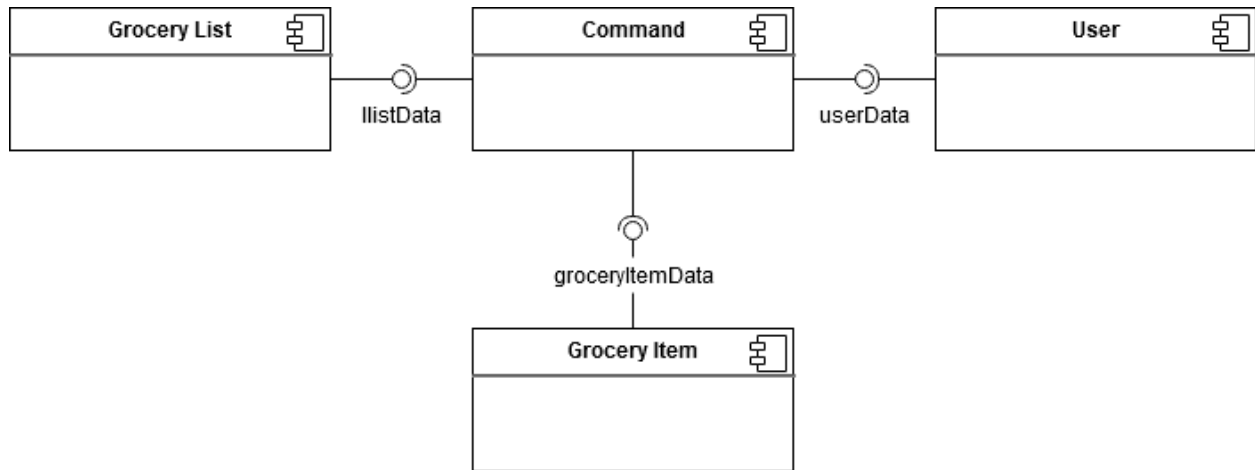
Package Diagram



Deployment Diagram



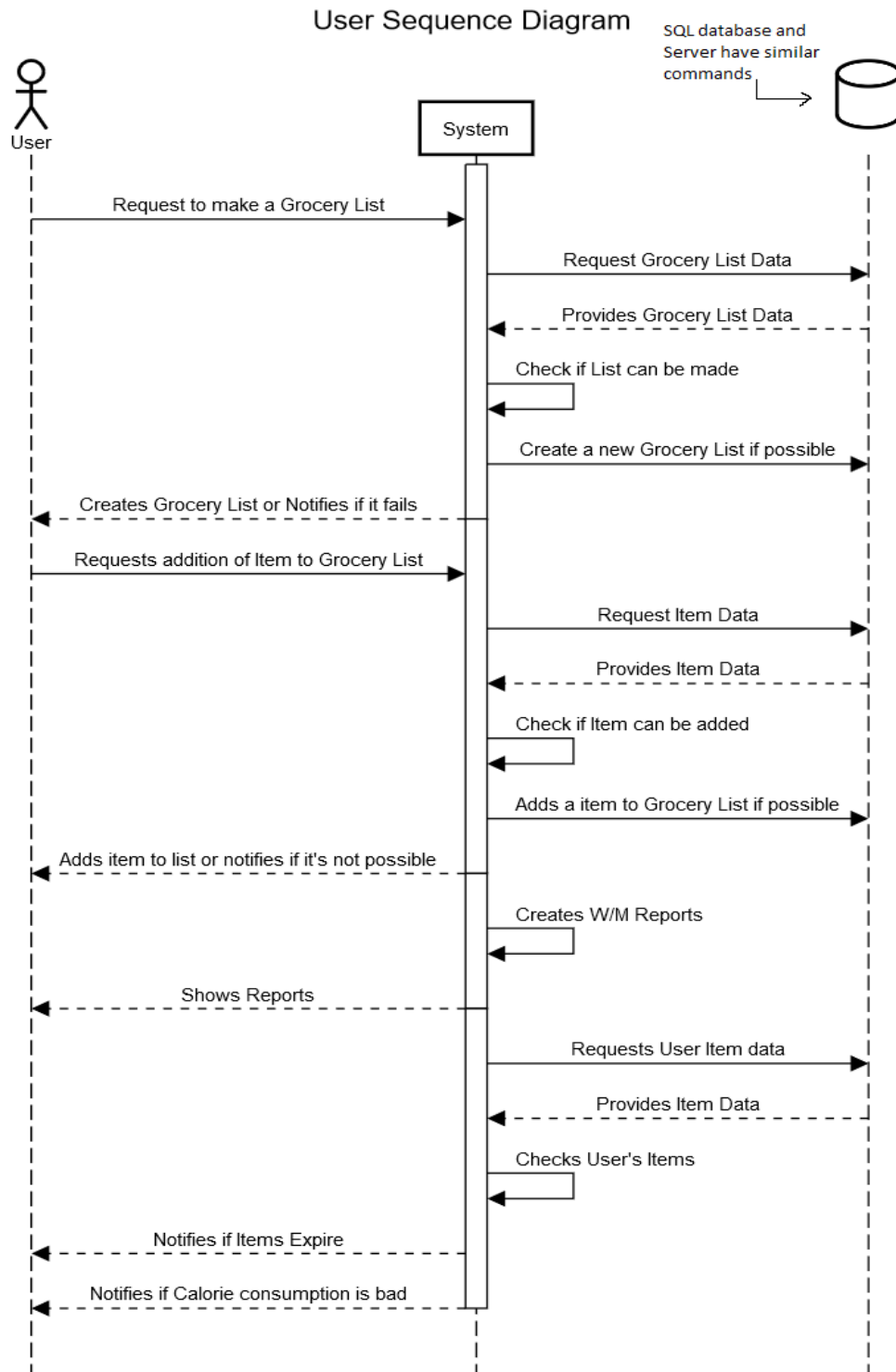
Component Diagram



Patterns used:

- layered → used to structure the program into groups of subtasks
→ each layer provides services to the next higher layer
- Server will be using and ORM and dependency injection that come from the libraries used:
 - Spring
 - Graphql
- Observer → Used to notify the user if today is the final day set for certain item:
 - Notified with a list that contains the name and list of the item

4. UML Sequence Diagrams



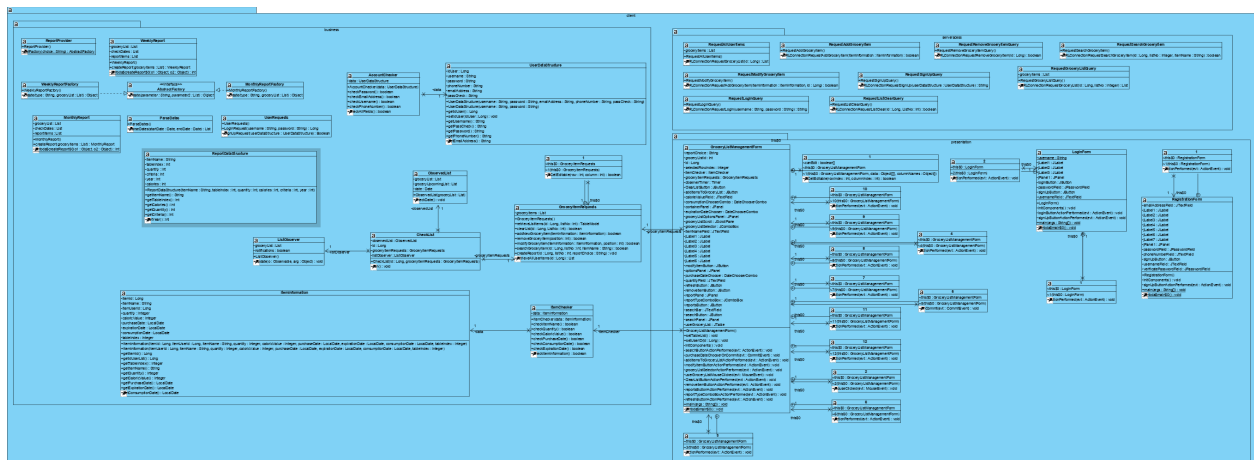
5. Class Design

5.1 Design Patterns Description

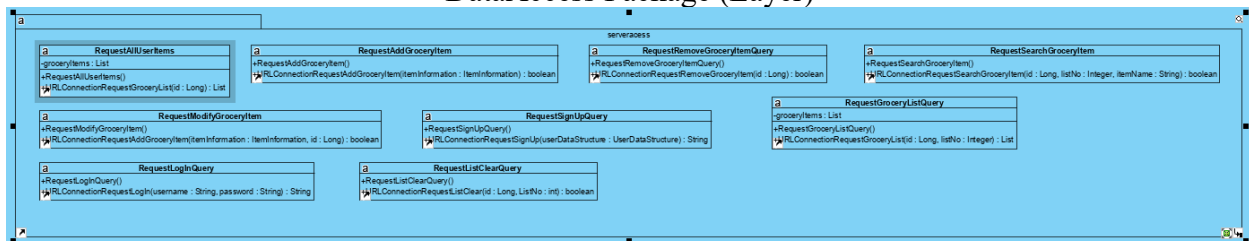
- Abstract Factory → Used in the creation of Weekly/Monthly Reports
 - Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
- Layered → used to structure the program into groups of subtasks
 - each layer provides services to the next higher layer

5.2 UML Class Diagram

CLIENT



DataAccess Package (Layer)



[illegible]

```

classDiagram
    class GroceryListManagementForm {
        reportChoice : String
        reportListId : int
        id : Long
        selectedRowIndex : Integer
        itemChecker : ItemChecker
        groceryItemRequests : GroceryItemRequests
        observeTime : Timer
        clearButton : Button
        addItemsToGroceryList : Button
        cancelValueField : JTextField
        containerChooserCombo : DateChooserCombo
        containerPanel : JPanel
        expirationDateChooser : DateChooserCombo
        groceryListOptionPanel : JPanel
        groceryListEditor : JScrollPane
        groceryListSelector : JComboBox
        itemNameField : JTextField
        label1 : JLabel
        label2 : JLabel
        label3 : JLabel
        label4 : JLabel
        label5 : JLabel
        label6 : JLabel
        modifyItemButton : Button
        optionPanel : JPanel
        purchaseDateChooser : DateChooserCombo
        quantityField : JTextField
        refreshButton : Button
        removeItemButton : Button
        reportPanel : JPanel
        reportTypeComboBox : JComboBox
        reportsButton : JButton
        searchBar : JTextField
        searchButton : JButton
        searchPanel : JPanel
        userGroceryList : JTable
        ~GroceryListManagementForm()
        resetTableList() void
        resetUserDid : Long() void
        initComponents() void
        searchButtonActionPerformed(evt : ActionEvent) void
        purchaseDateChooserOnCommitEvt : CommEvt() void
        addItemsToGroceryListActionPerformed(evt : ActionEvent) void
        modifyItemButtonActionPerformed(evt : ActionEvent) void
        groceryListSelectorActionPerformed(evt : ActionEvent) void
        userGroceryListHasClicked : MouseEvent() void
        clearListButtonActionPerformed(evt : ActionEvent) void
        removeItemButtonActionPerformed(evt : ActionEvent) void
        reportsButtonActionPerformed(evt : ActionEvent) void
        reportTypeComboBoxActionPerformed(evt : ActionEvent) void
        refreshButtonActionPerformed(evt : ActionEvent) void
        mainPanelScroll() void
    }

    class LoginForm {
        username : String
        label1 : JLabel
        label2 : JLabel
        label3 : JLabel
        jPasswordField1 : JPasswordField
        signUpButton : JButton
        passwordField : JPasswordField
        usernameField : JTextField
        ~LoginForm()
        initComponents() void
        signUpButtonActionPerformed(evt : ActionEvent) void
        loginButtonActionPerformed(evt : ActionEvent) void
        mainPanelScroll() void
    }

    class RegistrationForm {
        emailAddressField : JTextField
        label1 : JLabel
        label2 : JLabel
        label3 : JLabel
        label4 : JLabel
        label5 : JLabel
        label6 : JLabel
        label7 : JLabel
        jPasswordField1 : JPasswordField
        signUpButton : JButton
        usernameField : JTextField
        verifyPasswordField : JPasswordField
        ~RegistrationForm()
        initComponents() void
        signUpButtonActionPerformed(evt : ActionEvent) void
        mainPanelScroll() void
    }

    class ItemChecker {
        ~ItemChecker()
    }

    class GroceryItemRequests {
        ~GroceryItemRequests()
    }

    class DateChooserCombo {
        ~DateChooserCombo()
    }

    class JScrollPane {
        ~JScrollPane()
    }

    class JComboBox {
        ~JComboBox()
    }

    class JTextField {
        ~JTextField()
    }

    class JLabel {
        ~JLabel()
    }

    class JButton {
        ~JButton()
    }

    class JTable {
        ~JTable()
    }

    class MouseEvent {
        ~MouseEvent()
    }

    class CommEvt {
        ~CommEvt()
    }

    GroceryListManagementForm --|> ItemChecker
    GroceryListManagementForm --|> GroceryItemRequests
    GroceryListManagementForm --|> DateChooserCombo
    GroceryListManagementForm --|> JScrollPane
    GroceryListManagementForm --|> JComboBox
    GroceryListManagementForm --|> JTextField
    GroceryListManagementForm --|> JLabel
    GroceryListManagementForm --|> JButton
    GroceryListManagementForm --|> JTable
    GroceryListManagementForm --|> MouseEvent
    GroceryListManagementForm --|> CommEvt

    LoginForm --|> ItemChecker
    LoginForm --|> GroceryItemRequests
    LoginForm --|> DateChooserCombo
    LoginForm --|> JScrollPane
    LoginForm --|> JComboBox
    LoginForm --|> JTextField
    LoginForm --|> JLabel
    LoginForm --|> JButton
    LoginForm --|> JTable
    LoginForm --|> MouseEvent
    LoginForm --|> CommEvt

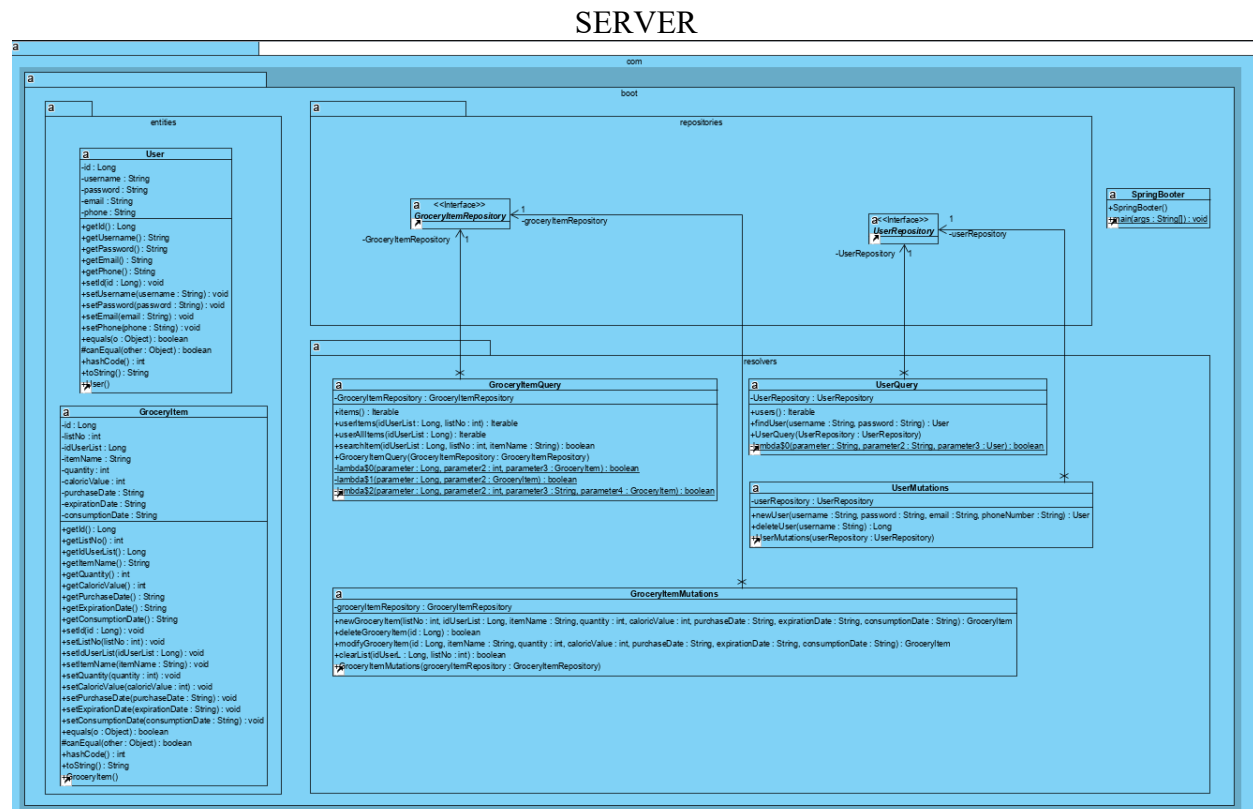
    RegistrationForm --|> ItemChecker
    RegistrationForm --|> GroceryItemRequests
    RegistrationForm --|> DateChooserCombo
    RegistrationForm --|> JScrollPane
    RegistrationForm --|> JComboBox
    RegistrationForm --|> JTextField
    RegistrationForm --|> JLabel
    RegistrationForm --|> JButton
    RegistrationForm --|> JTable
    RegistrationForm --|> MouseEvent
    RegistrationForm --|> CommEvt
  
```

The diagram illustrates the relationships between four main components: **GroceryListManagementForm**, **LoginForm**, **RegistrationForm**, and **ItemChecker**. Each component is a class that inherits from **ItemChecker** and implements the **GroceryItemRequests** interface. The **GroceryListManagementForm** class is the most complex, containing numerous attributes (e.g., `reportChoice`, `reportListId`, `id`, `selectedRowIndex`, `itemChecker`, `groceryItemRequests`, `observeTime`, `clearButton`, `addItemsToGroceryList`, `cancelValueField`, `containerChooserCombo`, `containerPanel`, `expirationDateChooser`, `groceryListOptionPanel`, `groceryListEditor`, `groceryListSelector`, `itemNameField`, `label1` through `label6`, `modifyItemButton`, `optionPanel`, `purchaseDateChooser`, `quantityField`, `refreshButton`, `removeItemButton`, `reportPanel`, `reportTypeComboBox`, `reportsButton`, `searchBar`, `searchButton`, `searchPanel`, `userGroceryList`) and methods (e.g., `resetTableList()`, `resetUserDid()`, `initComponents()`, `searchButtonActionPerformed()`, `purchaseDateChooserOnCommitEvt()`, `addItemsToGroceryListActionPerformed()`, `modifyItemButtonActionPerformed()`, `groceryListSelectorActionPerformed()`, `userGroceryListHasClicked()`, `clearListButtonActionPerformed()`, `removeItemButtonActionPerformed()`, `reportsButtonActionPerformed()`, `reportTypeComboBoxActionPerformed()`, `refreshButtonActionPerformed()`, `mainPanelScroll()`). The **LoginForm** and **RegistrationForm** classes are simpler, with attributes like `username`, `label1` through `label3`, `jPasswordField1`, `signUpButton`, `passwordField`, `usernameField`, and methods like `initComponents()`, `signUpButtonActionPerformed()`, `loginButtonActionPerformed()`, and `mainPanelScroll()`. The **ItemChecker** class is the base class for all three, with a single method `~ItemChecker()`. The **GroceryItemRequests** interface is implemented by all three classes, with methods like `~GroceryItemRequests()`, `resetTableList()`, `resetUserDid()`, `initComponents()`, `searchButtonActionPerformed()`, `purchaseDateChooserOnCommitEvt()`, `addItemsToGroceryListActionPerformed()`, `modifyItemButtonActionPerformed()`, `groceryListSelectorActionPerformed()`, `userGroceryListHasClicked()`, `clearListButtonActionPerformed()`, `removeItemButtonActionPerformed()`, `reportsButtonActionPerformed()`, `reportTypeComboBoxActionPerformed()`, `refreshButtonActionPerformed()`, and `mainPanelScroll()`.

As it can be seen from the 4 images the structure of the program is divided 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 .

Observer Pattern is used to notify the user



The server is divided into 3 main parts:

- entities – Contains the structures of the items (Contains the columns of the sql table)
- repositories – Contains the list of items based on the entity data an id type
- resolvers - Contains the functions that work with the database
 - Mutation → add new items, modify data, delete data etc.
 - Query → Retrieve data using certain filter parameters

The server knows how to connect to the database based on the parameters in the application.properties fil. Connection to the database is established by the spring framework.

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-world 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
- The functions that were implemented in the server were tested using the graphiql library
 - using the link: <http://localhost:8080/graphiql> you are sent to an interface
 - in the interface graphql queries can be tested

8. Bibliography

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

- Email checking
- Abstract factory (?)
- Parsing dates

[2] <https://www.baeldung.com> → Used for Explanation of:

- Spring
- HTTPConnection Explanation

[3] <https://github.com> → Used for:

- GraphQL Documentation
- HTTPConnection Example Explanation