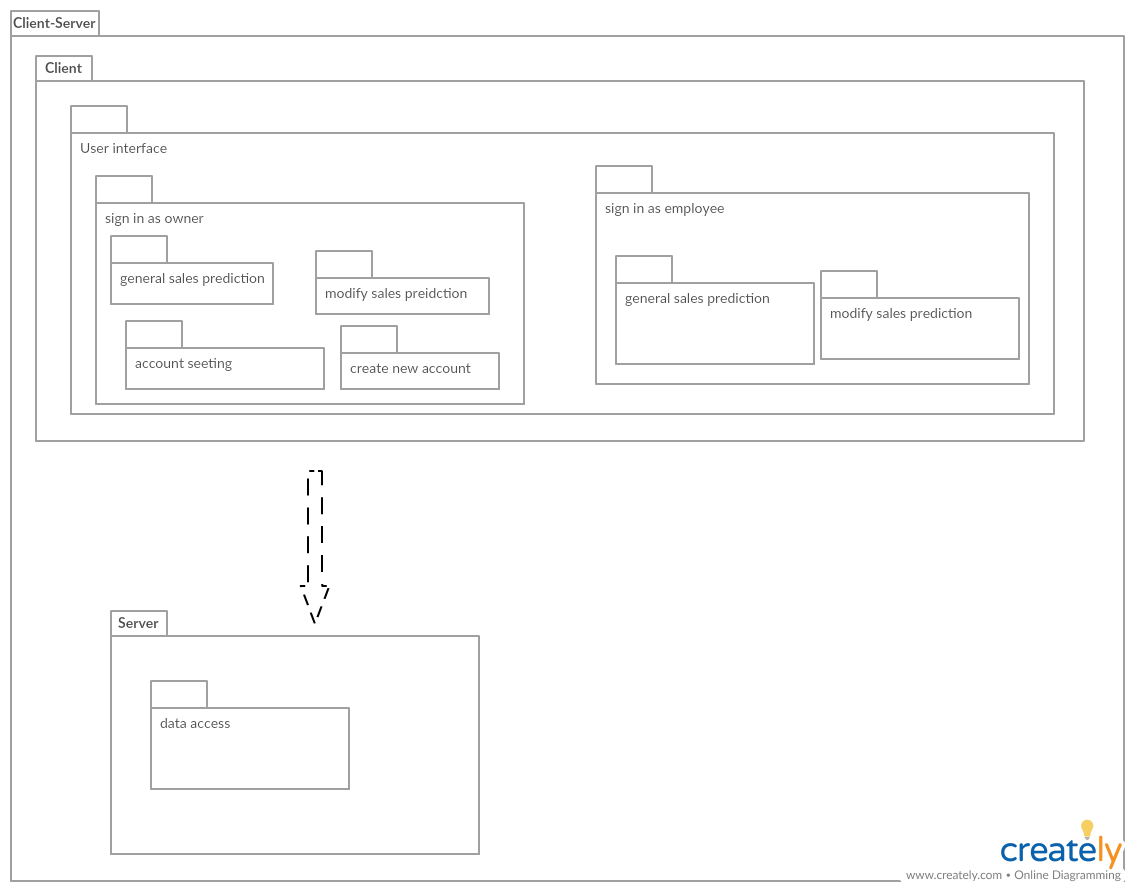
Software architecture



The architecture pattern we chose to implement is a Client-Server design. As a client, you will have the ability to login as an account with either manager or employee authorization. Due to the different authorization levels, a user may have restricted access to certain data. A user with an employee authorization level will be brought to a subtly different home screen with fewer options.

As a user with manager authorization, there will be two more options on the home screen, *Account Settings* and *Create New Account*.

When any user selects the *Generate Sales Prediction* option, a new window will pop up allowing the user to input a certain date range and, according to the sales data previously inputted in the database, predict the average gross sales for that future date.

When any user selects the *Input Sales Data* option, a window pops up prompting the user to select a date and is given the option to input a value for sales data. The user can then submit this data by pressing the *Upload* button.

A user with manager authorization can choose *Account Settings*. After the user selects the *Load* button, the usernames, passwords and associated positions of those accounts from the database are then loaded into the table on this screen. The user can then edit an account’s credentials and position after inputting new data and then clicking *Update*.

A user with manager authorization can click *Create New Account* and will then be prompted to enter an account’s username, password and choose a role. This new user’s information will then be submitted into the account database.

Design pattern:

**Creational pattern:** Inabstract factory, it provides an interface for creating families of related of dependent objects, but we do not have any interface to use. In builder design pattern, we also need to have an interface to build a complex object using simple objects, but in our project, we did not use either. In factory method, define an interface for creating an object, but let subclasses decide which class to instantiate and we do use either. In prototype pattern, we need to have an abstract class, but we do not any abstract class we do not use that pattern either. In **Singleton pattern**, it involves only one class which is responsible to instantiate itself. In our application, we have a branch of static classes and single object can be used by all other classes.

**public** String getUser() {  
        **return user**.get();  
    }  
  
    **public** String getPass() {  
        **return pass**.get();  
    }  
  
    **public** String getPos() {  
        **return pos**.get();  
    }

*//updates an entire month for the new data input***public static boolean** MonthUpdate(String date) {  
   String month = *WhatMonth*(date);  
   String array[] = date.split(**"/"**);  
   System.***out***.println(month);  
   String days[] = {**"mon"**, **"tue"**, **"wed"**, **"thu"**, **"fri"**, **"sat"**, **"sun"**};  
   ArrayList<dailyavg> hold = **new** ArrayList<dailyavg>();  
   hold = DbManager.*filllist*();  
   *//String months[] = { "01", "02", "03", "04", "05", "06", "07", "08", "09", "10", "11", "12" };* **float** total = 0;  
   **int** count = 0;  
   **float** checker = 0;  
  
   **for** (**int** z = 1; z <= 5; z++)  
      **for** (**int** y = 0; y <= 6; y++) {  
         **float** avg = DbManager.*GetAvg*(month, z + days[y]);  
         String query = **"SELECT** *\** **FROM "** + month + **" WHERE DayOfMonth = '"** + z + days[y] + **"'"**;  
         *//System.out.println(z+days[y]);* **float** high = 0;  
         **try** {  
            avg = DbManager.*GetAvg*(month, z + days[y]);  
            ResultSet rs = *st*.executeQuery(query);  
            *//System.out.println("Hello");* **if** (rs.next())  
               checker = rs.getFloat(**"AvgGrossSales"**);  
            rs.close();  
            **if** (checker > 0)  
               avg = checker;  
         } **catch** (SQLException e) {  
            System.***out***.println(e);  
            *//continue;* **return false**;  
  
         }  
         query = **new** String();  
         query = **"SELECT** *\** **FROM dailyinformation WHERE Date LIKE '"** + array[0] + **"%' AND DayOfMonth = '"** + z + days[y] + **"'"**;  
         System.***out***.println(array[0]);  
         **try** {  
            ResultSet rs = *st*.executeQuery(query);  
            **if** (rs.next()) {  
               high = rs.getFloat(**"GrossSales"**);  
               DbManager.*UpdateOneDay*(month, high, z + days[y], avg);  
               System.***out***.println(high);  
            }  
            rs.close();  
         } **catch** (SQLException e) {  
            System.***out***.println(e);  
         }  
  
  
         *//System.out.println(temp.grosssales);* DbManager.*UpdateOneDay*(month, high, z + days[y], avg);  
  
      }  
   **return true**;  
}  
*//****todo: update every year***