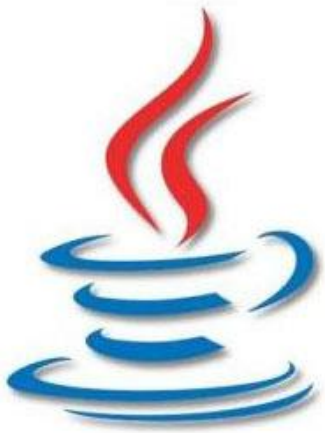
COUPON SYSTEM



Project characterization document

FIRST STEP:

Part 1 + Part 2

Presenting By:

Yanir Ifrah
ID: 046149076
Mail: yanir9@gmail.com
822.96 | 19.01.2017

Coupon System Project 822.96

Part 1 + Part 2

Table of Contents

Coupon System Project 822.96

Part 1 + Part 2

Coupon System Project 822.96

Part 1 + Part 2

Coupon System Project – part 1

INRTODUCTION:

The aim of this project is to create a website that holds a verity of coupons. The coupon management system allow companies to create a coupons as part of their Advertising campaigns. The main users of the system **is companies** that sales coupons, and **private customers** that buy the coupons. The third user of the system is 'ADMIN' (Administrator), which manage the system in the background. Each user have a verity of unique privileges he can perform on the system.

This part of the project focuses on the system core, and the communications with the date base, where the user's, customers and companies, and their information are storage. For example, the information on how many coupons every customer purchased, or what types of coupon each company created. All this information are storage in a specific tables in the Data base.

The system core will be divided in to three main parts:

(i) The creation of the database

(ii) Building a DAO (interface)/DBDAO (implement DAO) layers that allow to communicate with the date base in SQL format

(iii) Building a Façade classes, that serve as a user interface that enables each user to perform the relevant actions according to the system's permissions (Administrator, Customers and Companies)

Each part includes several small steps, that which together present the core of the system (create java beans, building a connection pool, coupon system singleton as the main interface of the system, Daily thread…)

Data Base:

The project uses the Apache Derby date base. As the project required, we create a local server on our local machine. All the information that accept in the system about the users (customer and companies) are storage in specific tables that we create on the date base.

Coupon System Project 822.96

Part 1 + Part 2

INSTRUCTIONS:

This is a coupon management system that allows companies to create coupons as part of their advertising and marketing campaigns. The system has registered customers, who can purchase restricted coupons in quantity and validity. Each customer can purchase one coupon of each type only. Access to the system is divided into three types of clients: companies, administrator and customers.
The System Coupon is obtained through the System Coupon class, which is responsible for initialize the communication layer against the database (connection pool and DAO layer.

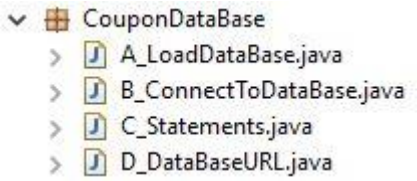
At this phase of the project, in order to test it and examine the system, you must run the Façade testsclasses, in order to examine the system's capabilities

Coupon System Project 822.96

Part 1 + Part 2

PROJECT PACKAGES:

Package CouponDateBase:



Classes:

A_LoadDataBase – load driverName = "org.apache.derby.jdbc.ClientDriver";

B_ConnectToDataBase – Test the connection to database.

C_Statment – The class **C_statment** used for delete and create all the tables in the Database.

D_DataBaseURL – used for creating a folder name 'files' that holds the connection URL to database**.**

Contains:

createTableAtDB (); - creating all the tables in the DB, as required in this specific project

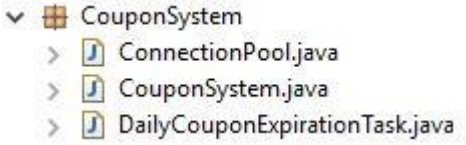
deleteTablesFromDB(); - delete all the tables from the DB

checkIfTablesExistInTheDB (); - Boolean method that check if the DB is exist. The rational of this method is to test only one table, and if it's exists, we assume that all tables are existed at the DB.

Return True- if the tables is exist | False- if the tables is not exist

resetTables (); - called the deleteTablesFromDB(); and

Package CouponSystem



Classes:

ConnectionPool: (See appendix A) **- handles** all connections of the application with the database

CouponSystem - A singleton that function as the top layer of the application, starting the ConnectionPool instance and closing it

DailyCouponExpirationTask: (See appendix D) - a daily thread that delete expired coupons from the Date base

Coupon System Project 822.96

Part 1 + Part 2

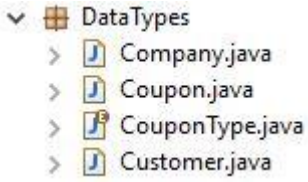
Package DAO.interface: (See appendix B)

A DAO layer that contains all the method that communicate with the Date base. This layer inherit the method to the DBDAO layer**.**

Classes: CompanyDAO | CouponDAO | CustomerDAO | JoinedTableDAO

JoinedTableDAO class– an interface that contained all the methods that related to the joined table in the Date base (Company_Coupon, Customer_Coupon)

Package DateTypes:



Classes: Company | Coupon | Customer – representing three types of object that can be insert to the Date base

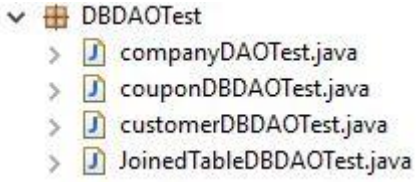
CoupoType class – enum CouponType class that define a different categories of coupons

Package DBDAO: (See appendix B)

A DBDAO layer that are implement the DAO.interface methods, which enable to contact with the Coupon Date base easily and safely. All methods contains the sql Query for communicate with the Date base effectively.

(* for more information see appendix A)

Package DBDAOTest:

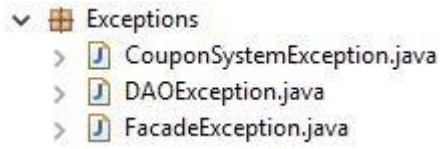


Testing for all DBDAO layer methods, and their operation on the data Base by calling all the methods. This test created for the programmer to examine all methods and communication with the date base and the program capabilities

Coupon System Project 822.96

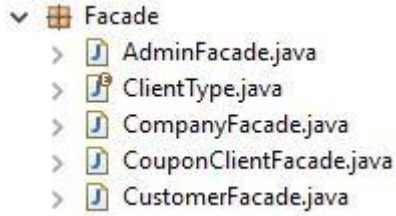
Part 1 + Part 2

Package Exceptions:



An Exceptions classes that handled all the Exception in the program

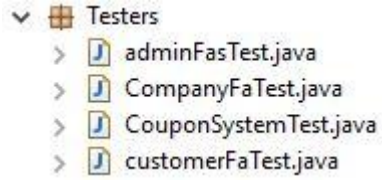
Package Facade: (See appendix C)



Classes that represent the client layer of the system. All Façade of each user contains all the methods that can be executed by the type of user, used the DAO layer In order to allow the relevant operations

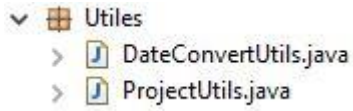
ClientType class – enum ClientType class that define a different types of coupons

Package Testers:



Testing for all Facade user's operations. All classes are contain a main method and can be run and test the program. This classes and created for testing and for the programmer only

Package Utiles: (See appendix E)



DateConvertUtils:

An easy way to use java.util.Date and convert it to java.sql.Date by using the methods in that class.

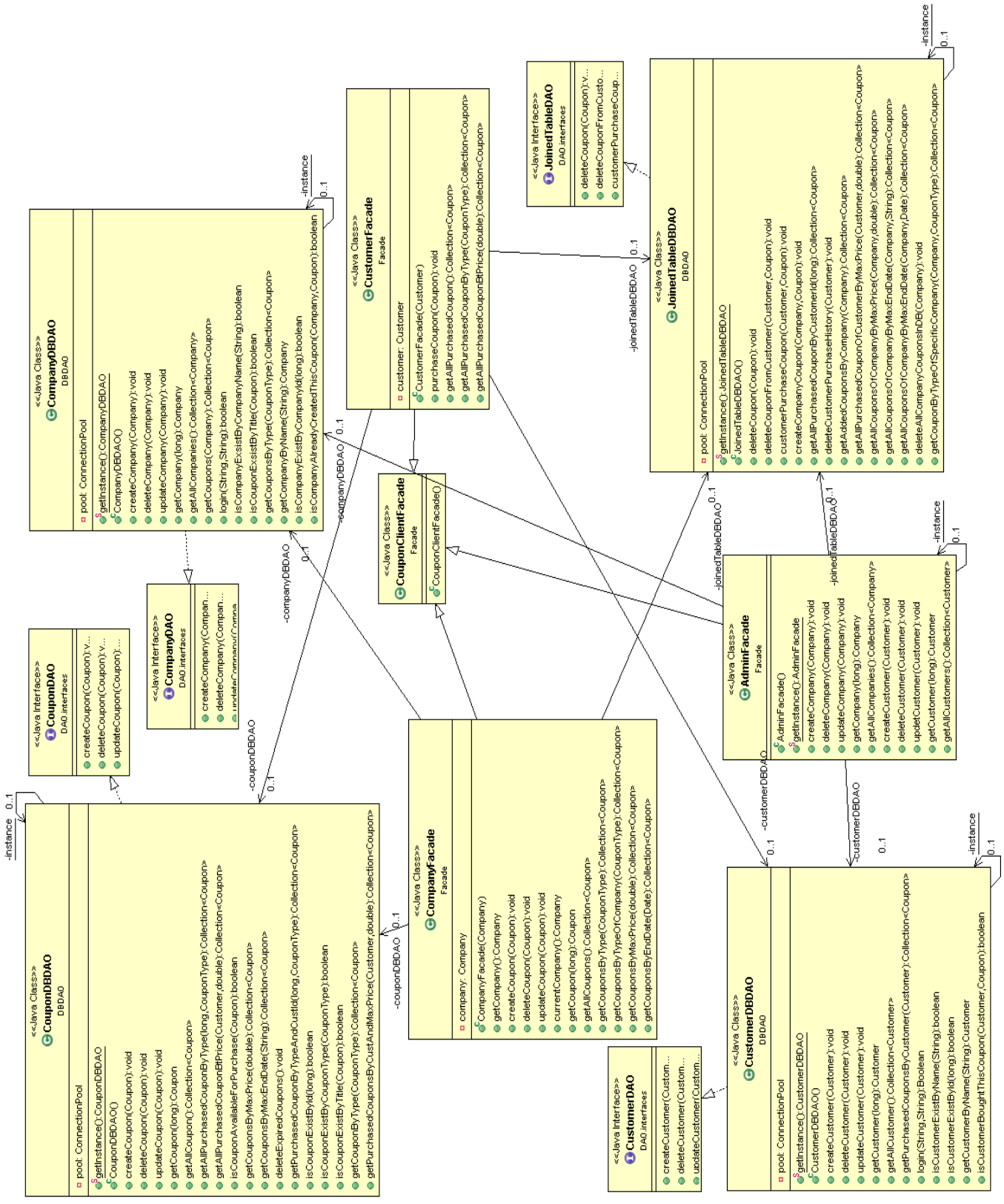
ProjectUtils:

We choose to create a class ProjectUtils that extract all coupon attribute (id, title, Start Date, End Date, amount, type, massage, price and image from the result set.

Coupon System Project 822.96

Part 1 + Part 2

UML:



Coupon System Project 822.96

Part 1 + Part 2

Appendix A

Connection Pool:

Connection Pool (singleton) - The connection pool function, is to have a set of 10 open connections and allow each method from the DBDAO to take a connection to connect to the Database and return it to the pool immediately after the method and its operations have been completed.

HashSet as a collections co connection:

We Create a HashSet that holds all connections because HashSet doesn't maintain any order, the elements would be returned in any random order.
HashSet doesn't allow duplicates, and allows null values and it's easy to iterate over the HashSet

Connection pool methods:

- **CreateConnection ()** method is reading the Database URL from a file stored in the folder 'files' in the project package. This way we ensure safety reading without worrying about whether the URL will change by mistake.
- **Get Connection ()** and **return Connection ()** methods is a synchronized methods In order to avoid the possibility of attempting to take or return connection from and to the connection pool. For that we use the wait () and the notify () method.
- **Close All Connections ()** method is for closing all connections when the application is shut down.

Coupon System Project 822.96

Part 1 + Part 2

Appendix B

Building a DAO layer:

In order to manage access to the Database. This is an interface layer that contains the methods required to support DBDAO classes that implement this methods. These are the Database access classes, which constitute a layer of isolation that allows convenient work from JAVA against the required sql and operations against the Database.

DBDAO Classes (implement DAO.interface):

CompanyDBDAO methods:

Method:	Description:
createCompany(Company)	As required by the project
deleteCompany(Company)	As required by the project
updateCompany(Company)	As required by the project
getCompany(long id): Company	As required by the project
getAllCompanies():Collection<Company>	As required by the project
getCoupons(Company): Collection<Coupon>	As required by the project
login (compName, password)	As required by the project
Added methods	
isCompanyExsistByCompanyName (name): Boolean	Check if the company exist in DB by the compName. return true or false if the company is already exists in the database
isCouponExsistByTitle(coupon): Boolean	Check if the coupon exist in DB by the coupon title. return true or false if the title is already exists in the database
getCouponsByType(CouponType): Collection<Coupon>	return a collection of coupon by a specific type
isCompanyExistByCompanyId(long id)	Check if the company exist by a given company id.
getCompanyByName (name):Company	Return a new company from the database.
isCompanyAlreadyCreatedThisCoupon(Company, Coupon): boolean	checking if the company already created a specific coupon

CouponDBDAO methods:

Method:	Description:
createCoupon(Coupon)	As required by the project
deleteCoupon(Coupon)	As required by the project
updateCoupon(Coupon)	As required by the project
getCoupon(long id): Coupon	As required by the project
getAllCoupon():Collection<Coupon>	As required by the project
getCouponByType(type): collection <Coupon>	As required by the project
getCouponsByMaxPrice(maxPrice): Collection<Coupon>	As required by the project
getCouponsByMaxEndDate(maxEndDate): Collection<Coupon>	As required by the project
Added methods	
isCouponAvailableForPurchase(Coupon coupon) boolean	Checking if coupon is available to purchase
deleteExpiredCoupons()	delete all expired coupons from the database

getAllPurchasedCouponByType(customerId, couponType): Collection<Coupon>	Get all the coupons that bought by the specific customer by Type. return a collection of Purchased coupon
getAllPurchasedCouponByPrice(price): Collection<Coupon>	Get all the coupons that bought by the specific customer by price. return a collection of Purchased coupon
isCouponExistById(long id) boolean	Checking if coupon is exist by a specific coupon id
isCouponExistByTitle(Coupon coupon) boolean	Checking if coupon is exist by a specific coupon title
getCouponByTypeOfSpecificCompany	Getting all coupons matching a specific coupon type of a specific company from DB.
getPurchasedCouponByTypeAndCustId(long customerId, CouponType type)	Getting all coupons of a specific customer, with a specific coupon type
isCouponExistByCouponType(CouponType couponType): boolean	Checking if coupon is exist by a specific type of coupon
getPurchasedCouponsByCustAndMaxPrice(customer, maxPrice): Collection<Coupon>	Getting all the purchase coupons of a specific customer, limited by a specific price

CustomerDBDAO methods:

Method:	**Description:**
createCustomer(Customer)	As required by the project
deleteCustomer(Customer)	As required by the project
updateCustomer(Customer)	As required by the project
getCustomer(id): Customer	As required by the project
getAllCustomer():Collection <Customer>	As required by the project
Collection<Coupon> getPurchasedCouponsByCustomer(Customer customer)	As required by the project
login(custName, password):boolean	As required by the project
Added methods	
isCustomerExsistByName (customer):boolean	return true or false if the customer is already exists or not in the database by the customer name
isCustomerExsistById(id): boolean	return true or false if the customer is already exists or not in the database by the customer id
getCustomerByName(String custName)	getting a specific customer by name from the Customer table
isCustomerBoughtThisCoupon(customer, coupon) :boolean	return true or false if the customer is already Bought a specific coupon or

Coupon System Project 822.96

Part 1 + Part 2

In order to manage the join table and take action on them, we created a JoinedTableDBDAO class that contain all the method's that handle the join table.

JoinedTableDBDAO methods:

Method:	Description:
deleteCoupon(Coupon coupon)	delete coupon from **Company_Coupon** and **Customer_Coupon** tables
deleteCouponFromCustomer(customer, coupon)	delete a specific customer and specific coupon in the Customer_Coupon table in the DB
deleteCustomerPurchaseHistory(customer)	delete customer and all of its coupon purchase history from **Customer_Coupon** table (delete a row in the **Customer_Coupon** table
CustomerPurchaseCoupon (Customer, Coupon)	Add new customer and coupon **for Customer_Coupon** table.
createCompanyCoupon (Company, Coupon)	Add new Company and coupon for **Company_Coupon** table.
getAllPurchasedCouponByCustomerId (Custid,): Collection<Coupon>	getting a collections of all the purchased coupon of a Specific customer by the customer id
getAddedCouponsByCompany(Company company): Collection<Coupon>	getting a collections of all the coupons that added by a specific company by the company ID
getAllPurchasedCouponOfCustomerByMaxPrice (Customer, double maxPrice)	getting a collections of all the coupons that Purchased by a specific Customer by the Customer ID and max Price
deleteAllCompanyCouponsInDB (Company company)	Delete all the company coupons from the Database. Used in admin façade to delete a company and all of here coupons
getCouponByTypeOfSpecificCompany(Company, type): Collection<Coupon>	Getting all the coupons of a specific company, with a specific coupon type
getAllCouponsOfCompanyByMaxPrice(Company, double maxPrice)	getting all coupons from DB limited by a maximum price (maxPrice)
getAllCouponsOfCompanyByMaxEndDate(Company company, **Date** maxEndDate)	getting all coupons from DB limited by a specific coupon End date
getAllCouponsOfCompanyByMaxEndDate(Company, **String** maxEndDate)	getting all coupons from DB limited by a specific coupon End date

Coupon System Project 822.96

Part 1 + Part 2

Appendix C

Facades:

Building facade that representing the classes and having access for ADMIN, COMPANY, and CUSTOMER. These types (using enum ClientType class) are the system's clients. These are classes that inherit from the Interface class CouponClientFacade. These layers use the DAO classes we created that contain the methods that link to the Database. All these classes are contains the business logic of the system.

Admin Façade methods:

Method:	Operation:
createCompany(Company)	Use the **isCompanyExsistByCompanyName** method for checking if the company name is already exist
deleteCompany(Company)	*use **isCompanyExistByCompanyId** method for checking if the company id is exist *use **deleteAllCompanyCoupons** method for delete all company coupon from the Company_Coupon join table and delete all coupons of this company from the Customer_Coupon join table
updateCompany(Company)	*used **isCompanyExsistByCompanyName** method to check id the company exist by name Update only the Password and Email (According to the sql question and project request)
getCompany(long id) : Company	get the specific company detail from the DB ***used isCompanyExistByCompanyId** method to check id the company exist by id
getAllCompanies() : Collection<Company>	Getting a collections of all companies from the Database. return a collection rather if is empty or full with companies details
createCustomer (Customer)	Creating a new customer in Customer table in the Database. Using **isCustomerExsistByName** method to check if the customer already exist
deleteCustomer (Customer)	delete a customer in Customer table in the Database *use **isCustomerExsistById** method for checking if the customer exist by ID *use **joinedTableDBDAO.deleteCustomer** method for deleting the customer from the Customer_Coupon table (all purchase history of the deleting customer.
updateCustomer (Customer)	updating customer by name (update only the password as required) *Use the **isCustomerExsistById** method for checking if the customer exist by id.
getCustomer (id): Customer	Getting all the customer detail in the DB.
getAllCustomers : Collection<Customer>	getting a collections of all customers from the DB

Coupon System Project 822.96

Part 1 + Part 2

Company Façade (throw Façade Exception)

Method:	Operation:
createCoupon(Coupon)	create the new coupon in for the specific company *Use the **isCouponExsistByTitle** method for checking if the coupon is already exist *Use **couponDBDAO.createCoupon** method to create a new coupon in the Coupon table *Use **joinedTableDBDAO.createCompany_Coupon** method to add new row for Company_Coupon table.
deleteCoupon(Coupon)	Delete a coupon owns by the specific company *use **couponDBDAO.deleteCoupon** method for deleting a coupon from coupon table of the DB by the coupon id. *use **joinedTableDBDAO.deleteCoupon** method for delete the coupon from the Company_Coupon join table and Customer_Coupon join tables
updateCoupon(Coupon)	*used **couponDBDAO.updateCoupon** update the specific coupon price and End date
getCoupon (long id): Coupon	*used **joinedTableDBDAO .getAddedCouponsByCompany** method for getting coupon of this company
currentCompany()	Get all the specific company details *used **companyDBDAO.getCompany**
getAllCoupons() : Collection<Coupon>	Returns a collection of all the coupons of the specific company *used **joinedTableDBDAO .getAddedCouponsByCompany** method for getting all coupons of this company
getCouponsByType(type)	Returns a collection of all the coupons of the specific type of coupon
getCouponsByMaxPrice(Company, maxPrice)	Get a collection of all coupons of the current company, which their price is not more than a specific maxPrice
getCouponsByEndDate(Company, maxEndDate)	Get a connection of All coupons of the current company, which their expiration date is before a specific date

Customer Façade (throw Façade Exception)

Method:	Operation:
purchaseCoupon(Coupon coupon)	Check if the customer already buy this coupon *Use the **getAllPurchasedCouponByCustomer** for the specific customer *Use the **getAmount**() to check if there is coupon left to buy *use **createCustomer_Coupon** to create a coupon in customer id in the customer coupon table * use **setAmount**() and **updateCoupon** to set and update the new amount of the coupon
getAllPurchasedCoupon()	Get a collection of all the purchase coupon of the specific customer.
getAllPurchasedCouponByType(CouponType type)	Get a collection of all coupons bought by this customer with a specific type of coupon. *Use **getAllPurchasedCouponByType**
getAllPurchasedCouponBtPrice(double price)	Get a collection of all coupons bought by this customer with a specific price *Use **getAllPurchasedCouponBtPrice(price)**

Coupon System Project 822.96

Part 1 + Part 2

Appendix D

Daily thread:

The daily thread is responsible to delete all the expire coupon in the DB.

Class **DailyCouponExpirationTask** implements Runnable, is a thread that delete all expired coupons in the Database. The thread is starting by calling the run method in CouponSystem class (after we get an instance of CouponSystem).
The thread is removing to the blocked pool after he was interrupted by the **run ()** method for 24 hours.
The thread is shutting down and stopped only by calling the shutdown () method from the class CouponSystem.
The thread use the **deleteExpiredCoupons ()** method in **CouponDBDAO** class to delete all the coupons that have expired date from the coupon and Company coupon table in the DB.

Appendix E

Utils:

ProjectUtils:

We choose to create a class ProjectUtils that extract all coupon attribute (id, title, Start Date, End Date, amount, type, massage, price and image from the result set.

The method **extractCouponFromResultSet (ResultSet resultSet)** return a coupon with all the attribute in it. This way it's easier to create and get a coupon in the DBDAE layer by calling the **extractCouponFromResultSet** method.

DateConvertUtils:

An easy way to use java.util.Date and convert it to java.sql.Date by using the methods in that class.

Methods:

```
Java.sql.Date convertToSql (Date date): convert string to
sql.date format
return – dateToSqlDate (date): the string date in sql format as
an easy way to use the sql string in the methods in the DBDAO
classes that used date parameter

Date stringToDate (String dateInFormat): Translate the string
input into java.util.Date.
return dateFormater.parse (dateInFormat)-a format of
SinmpleDateFormat as "dd/mm/yyyy"

Java.sql.Date dateToSqlDate (Date date) - method that
converting the Java.util.date into Java.sql.date format
```

Coupon System Project 822.96

Part 1 + Part 2

Coupon System Project – part 2

INRTODUCTION:

The second part of the project is about building an infrastructure for connecting customers via the Internet.
The customers supported in the system are: 'Administrator', 'Company' and 'Customer'.
For each one of them we build a web service that which is based on REST technology integration with the system. This part will include setting up a website using the single page application (S.P.A), which enables login for each system user: administrator, company or customer, and performing all the actions supported by the system.

Part 2 of the project is divided into two main parts:

- Build network services for each client supported by the system
- Build HTML front pages and connect them to the system using JavaScript and $http Ajax request. All web pages are connected via Angular module 'CouponSystem', which allows to perform operations by calling to the controllers. Each controller, for any action given by the system permission for each user, makes a call to a service in which there is an AJAX request to the web service.

INSTRUCTIONS:

In the first part of the project we created a database with tables containing all the information received in the system. In this project we already created a tables with existing information such as: customers, companies, coupons and main admin how manage the system.
Client connection info:

Type: **Admin:**
User: admin | password: 1234

Type: **Company:**
User: test1 | password: 12345678
User: test2 | password: 12345678
….
User: test9 | password: 12345678
User: test10 | password: 12345678

Type: **Customer:**
User: test1 | password: 12345678
User: test2 | password: 12345678
….
User: test9 | password: 12345678
User: test10 | password: 12345678

Coupon System Project 822.96

Part 1 + Part 2

BUILDING A WEB SERVICE:

For the communication with the database, which we established in the first part of the project, we created three web services AdminService, CompanyService and CustomerService. Each of them has the relevant methods, under the system permissions for each user.

As part of the process of conveying sensitive information on the network, we have created 'beans' which can pass through the network and return without sensitive information. To do this, we created conversion methods for each of the 'beans'.
We use the conversion methods in the web service

Each of the web services contains a getFacade () method that get the current façade from the session from the login servlet.

Login servlet:

I created a login page that contains three inputs, a user type selection, a user name and a password. Each user type, after a successful login, is redirecting to its home page, which contains all the relevant actions and information about it.

The login form on the login page activates the login servlet, which checks the user by type. The servlet takes the parameters from the request and places the facade on the session. In addition, the servlet returns with the response three cookies that I have created, containing the user's parameters, in order to identify future client re-entry to the system. We use the servlet to put the cookies on the client browser.

Logout servlet:

The logout servlet created for the clients to log out from the system. Ones the client is logging out, we set the cookies age to 0 in order to end the cookies life time, and then we end the session by session.invalidate() method.

Logger (java.util.logging.Logger):

In this project we use a logger object to log messages for the system.
The logging messages are forwarded to registered Handler objects, which forward the messages to the console and a file 'MyLogger' in the WebService.CouponWebService package.
The Logger is defined at the Info level in order to receive all the messages from the Info level and higher - setLevel (Level.INFO);

Coupon System Project 822.96

Part 1 + Part 2

Home page:

Each user in the system has his own S.P.A.
For each user in the system, client, company or administrator, I have created a home page that contains all the relevant methods according to the system permissions.
Each user can perform the actions relating to him, and receive the data and answers from the database, which relate to his actions.
Admin home page:
http://localhost:8080/CouponWebService/admin/admin.html#/home
Company home page:
http://localhost:8080/CouponWebService/company/company.html#/home
Customer home page:
http://localhost:8080/CouponWebService/customer/customer.html#/home

Html5 video element:

The HTML5 <video> element specifies a standard way to embed a video in a web page.
I inserted a video song in the system coupon Deal to allow you to control the video, play, paus, stop, play in full scream and download the song to the user's machine.

Coupon System Project 822.96

Part 1 + Part 2

My directives:

In my project I created two different directives:
The first directive is the AngularJS directive – 'myDirective', who is responsible for showing an html template from 'Footer.html' page into the admin's main single page and set to display as an element. <my-directive></my-directive> row 155 in admin.html

The second directive is the An AngularJS directive - 'StarRating', which is responsible for displaying the HTML element within the page, and it's defined within the template area in the directive. In the directive there is a controller the share data in the scope who is responsible for passing the parameter and displaying it in the ranking

```
<div class="container" ng-controller="appController">
   <div>
      <div class="alert alert-success" style="float: left;">
        <span class="label label-info">Star Rating Us...</span>
          <div star-rating rating="starRating1" read-only="false"
            max-rating="10" click="click1(param)"mouse-
            hover="mouseHover1(param)" mouse-leave="mouseLeave1(param)">
          </div>
          <div>
            <span class="label label-primary">Star Rating:
            {{starRating1}}</span> <span class="label label-primary">New
            Rating: {{hoverRating1}}</span>
          </div>
      </div>
   </div>
</div>
```

Carousel element (css3):

Within the project there are 2 elements of a carousel.

The first, on the file **footer.html**, has a slide carousel, that shown on the administrator page by the directive - MyDirective. This carousel takes the css design from the **/style/Carousel.css** file

The second one is a picture carousel. When you move with the mouse over the image, its open a slider showing information about the coupon. This carousel is showing in the company and customer pages. This carousel takes the css design from the **/style/Carousel.css** file

I decided to display this on the site in part of advertising coupons to a customer who connects to the system.

The function pf the carousel is within the customer and company single page, on a script tag :

```
<script type="text/javascript">
	//carousel image function
	$(function () {
		$('.carousel').each (function () {
			var $cfs = $(this);
			$cfs.carouFredSel ({
				direction : 'up',
				circular : false,
				infinite : false,
				align : false,
				width : 275,
				height : 250,
				items : 1,
				auto : false,
				scroll : {
					queue : 'last'
				}
			});
			$cfs.hover(function() {
				$cfs.trigger('next');
			}, function() {
				$cfs.trigger('prev');
			});
		});
	});
</script>
```

Coupon System Project 822.96

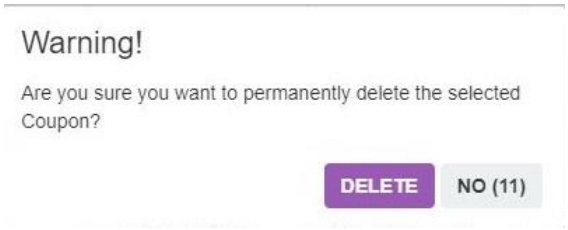
Part 1 + Part 2

Angular Confirm:

As part of a system that displays and sells products (in our case - coupons) there must be a way to conduct a dialogue with the client in order to confirm to him the success or failure of his actions on the system.
One way to do this is by Angular Confirm.
In most controllers I injected **$ngConfirm** in order to conduct the dialogue with the client.
Example:

```
Function deleteCouponCtrlCtor (mainCompanyService,$ngConfirm,$state,$scope)
```

In this function, in order to delete a specific coupon, the client will get a conformation popup, In order to warn him that he is about to delete the coupon. Pressing the delete button, will called the deleteCoupon function in the controller.



Example of conformation popup

Bootstrap

In my app I used Bootstrap. a free and open-source front-end web framework for designing websites and web applications.
This template allows the site to be responsive and adapt itself to any screen where it will be displayed.

Coupon System Project 822.96

Part 1 + Part 2

Validation:

Sending data is not enough — we also need to make sure that the data the users fill out in forms is in the correct format we need to process it successfully, and that it won't break our applications. We also want to help our users to fill out our forms correctly and don't get frustrated when trying to use our apps. Form validation helps us achieve these goals

For validation, I use the required property.
Each field from all application forms includes the required property. In some cases, I used the **ng-submit="onSubmit (theForm.$valid)** method to validate the form by calling the method and checking the STATE status

Validation for dates:

In some controllers, validation was required on dates. For example, in the createCouponCtrl, we must to ensure that the coupon's start date and end date have been entered correctly.
For this purpose, I have written several functions that validate the 2 parameters of the start date and end date of the coupon

convert: function(d):

Converts the date in d to a date-object. The input can be: a date object: returned without modification an array: Interpreted as [year, month, day]. NOTE: month is 0-11. a number : Interpreted as number of milliseconds since 1 Jan 1970 (a timestamp) a string : Any format supported by the javascript engine, like "YYYY/MM/DD","MM/DD/YYYY", "Jan 31 2009" etc. an object : Interpreted as an
Object with year, month and date attributes. **NOTE** month is 0-11.

```
convert: function(d) {
      return (
      d.constructor === Date ? d :
      d.constructor === Array ? new Date(d[0],d[1],d[2]):
      d.constructor === Number ? new Date(d) :
      d.constructor === String ? new Date(d) :
      typeof d === "object" ? new Date(d.year,d.month,d.date):NaN
      );
},
```

compare: function (a,b):

Compare two dates (could be of any type supported by the convert function above) and returns:
-1 : if a < b, 0 : if a = b, 1 : if a > b and NaN : if a or b is an illegal date
NOTE: The code inside isFinite does an assignment (=).

```
compare: function(a,b) {
        return (
             isFinite(a=this.convert(a).valueOf()) &&
             isFinite(b=this.convert(b).valueOf()) ?
             (a>b)-(a<b) : NaN
               );
          }
```

$watch on the start date and end date and respond to the client,based on the start and end date of the coupon. Create A date as today date for comparing to startDate,using the compare method, check if the startDate is Earlier then today. The system will not allow you to select a date that is earlier than today, and it will prevent entering the such a date and reset the start date field in the form Html

```
$scope.$watch('[c.newCoupon.startDate, c.newCoupon.endDate]', () =>
{
// watch the code in the createCouponCtrl...
}
```

Additional use of these methods is also done in an updateCouponCtrl. Validate the coupon end date, compared to today's date.

Coupon System Project 822.96

Part 1 + Part 2

Fusioncharts:

Fusioncharts is a library that allows the programmer to capture information on the site using graphs, charts, maps and more. In order to view the graphs on the site, we need to import the Fusioncharts.charts library.

```
<script type="text/javascript" src="angular/Fusioncharts.js"></script>

<script type="text/javascript"
src="angular/Fusioncharts.charts.js"></script>


<script type="text/javascript" src="angular/Angular-
fusioncharts.min.js"></script>
```

I insert a graph on the admin.html that is supposed to show the number of existing companies, the number of customers and coupons in the system. Unfortunately, there was not enough time to complete the graph and bring in the data, so I put in the default data, just to display it as part of the site design. (we talked about it in class)