Coupons Project with Spring Boot Framework

This system allows companies to create coupons that can be purchased by customers.

The system has 3 client types: administrator, company and customer:

Each client type can login to the system, but each can perform different operations.

(for example - administrator can view all customers registered to the system, and customer can purchase a coupon)

Using Spring Boot Framework

With Spring Boot we can tell Spring to create a bean that is managed by the spring context(pool of all Spring beans).

We only need to specify the @Component annotation above the class name, or @Repository/@Service(both contain @Component).

Spring also knows to inject beans to other beans(by using @Autowired or constructor).

Coupon Expiration Daily Job Thread

This system is capable of deleting expired coupons from DB every 24 hours.

With the help of @Scheduled annotation above method signature in CouponExpirationDailyJob class, we tell Spring to run this method asynchronously every 24 hours after each method execution.

the job must run on parallel with the execution of the program, so users can still use the application while the job removes all expired coupons at the same time.

Repositories

With the help of JPA (Java Persistence API) we can use all CRUD methods in order to query the DB with simplicity and without much code.

All that is required is to put @Repository above the interface declaration and to extend JpaRepository interface.

---

installation requirements

- Eclipse IDE Version: 2020-12
- JDK-11.0.11
- MySQL Server 8.0.25
- MySQL Workbench 8.0.25 (MySQL Server GUI)
- Connector/J 8.0.25 (in MySQL Installer)
- MySQL connector J - (via maven dependencies in pom.xml)

MySQL login details:

- username: root
- password: Ofiri0007!

Maven Dependencies

This project depends on several dependencies which are defined in pom.xml (configuration file) :

- spring-boot-starter (the framework we are using)
- spring-boot-starter-data-jpa (which is using Hibernate for Object-Relational Mapping: mapping from object to table record and conversely)
- mysql-connector-java (API for connecting our Java application to MySQL DBMS specifically)
- jaxb-api (used by Hibernate to facilitate the ORM technique)
- lombok - library used to build classes with less code by using special annotations above our Java Beans/entities classes.
- spring-aop - library for creating aop aspects

SQL Statements for Tables Creation

Categories Table

CREATE TABLE `categories` (`id` int NOT NULL AUTO_INCREMENT, `name` varchar(60) NOT NULL, PRIMARY KEY (`id`))

- id and category name cannot be null. category name can have 60 characters max.

Companies Table

CREATE TABLE `companies` (`id` int NOT NULL AUTO_INCREMENT, `name` varchar(100) NOT NULL, `email` varchar(150) NOT NULL, `password` varchar(200) NOT NULL, PRIMARY KEY (`id`))

- all fields in companies table cannot be null. name should be up to 100 characters, 150 characters for email, and 200 for password(in case of encryption)

ALTER TABLE `coupon_system` . `companies` ADD UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE;

- email field has 'unique constraint' so company cannot update its email to an existing one in DB.

Coupons Table

CREATE TABLE `coupons` (`id` int NOT NULL AUTO_INCREMENT, `company_id` int NOT NULL, `category_id` int NOT NULL, `title` varchar(200) DEFAULT NULL, `description` varchar(800) DEFAULT NULL, `start_date` date NOT NULL, `end_date` date NOT NULL, `amount` int DEFAULT NULL, `price` double DEFAULT NULL, `image` varchar(700) DEFAULT NULL, PRIMARY KEY (`id`), KEY `company_id_idx` (`company_id`), KEY `category_id_idx` (`category_id`), CONSTRAINT `category_id_fk` FOREIGN KEY (`category_id`) REFERENCES `categories` (`id`), CONSTRAINT `company_id_fk` FOREIGN KEY (`company_id`) REFERENCES `companies` (`id`) ON DELETE CASCADE)

- company id and category id cannot be null, and should point to existing PKs. title and description may be null in case the company doesn't want to enter title/description for the coupon.

both category id and company id are foreign keys:

- category id is defined as 'RESTRICT' for both delete and update.(because we don't want to delete a category from categories, or update category id from categories table).
- company id is defined as 'RESTRICT' for update because company id cannot be updated. company id is defined as 'CASCADE' for delete because if a company is deleted - all its coupons should be removed as well.

Customers Table

CREATE TABLE `customers` (`id` INT NOT NULL AUTO_INCREMENT, `first_name` VARCHAR(100) NOT NULL, `last_name` VARCHAR(100) NOT NULL, `email` VARCHAR(150) NOT NULL, `password` VARCHAR(200) NOT NULL, PRIMARY KEY (`id`));

- all fields in customer table cannot be null. first name and last name can have no more than 100 characters. email can be no more than 150 characters, and password can be up to 200 characters in case of password encryption.

ALTER TABLE `coupon_system` . `customers` ADD UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE; - email field has 'unique constraint' so customer cannot update its email to an existing one in DB.

Customers_VS_Coupons Table

CREATE TABLE `customers_vs_coupons` (`customer_id` int NOT NULL, `coupon_id` int NOT NULL, PRIMARY KEY (`customer_id` , `coupon_id`), KEY `coupon_id_fk_idx` (`coupon_id`), CONSTRAINT `coupon_id_fk` FOREIGN KEY (`coupon_id`) REFERENCES `coupons` (`id`) ON DELETE CASCADE, CONSTRAINT `customer_id_fk` FOREIGN KEY (`customer_id`) REFERENCES `customers` (`id`) ON DELETE CASCADE)

both customer id and coupon id are PKs and FKs: - customer id is defined as 'RESTRICT' for update, because customer id cannot be updated. customer id is defined as 'CASCADE' for delete, because if a customer is removed - all his coupons purchases will be removed as well.

- coupon id is defined as 'RESTRICT' for update, because coupon id cannot be updated. coupon id is defined as 'CASCADE' for delete, because if a coupon is removed - all its purchases by customers will be removed as well.

Changes to instructions

@RequiredArgsConstructor

- this annotation was added as a shortcut - instead of adding a constructor(for Spring to inject the object dependencies). it is preferable to use constructor for object dependencies instead of using autowired annotation. By using constructor we can make the field immutable (final). this cannot be done with autowired. also, by using constructor Spring throws an error in case of circular dependency, but with autowired we will not get informed about this issue.

In CustomerService, method -public void purchaseCoupon(int couponId)

- instead of getting Coupon object as an argument - this method can get coupon id and make a purchase

In Coupon class, toString()

- instead of returning the Category enum(with capital letters), the enum is converted to String with small letters except for the first letter.

(Utils.convertEnumToString(Enum myEnum))

In Company class, toString() and Customer class, toString()

- Utils.getCouponsAsStr(coupons) is used for a better String representation of the list of coupons.

Log Files

- there are 2 log files: one for documenting all exceptions that have been thrown(exception_log.txt) and another for documenting all successful operations(operations_log.txt)

In LoginManager - login(String email, String password, ClientType clientType)

- in case of login fail (because of bad email or password)- instead of returning null as requested, BadLoginException is thrown to the user with a detailed and clear message. (instead of handling NullPointerException in main method and throwing this exception from main)