

פרויקט מלווה מערכת ניהול קופונים

פרוייקט מלווה

מערכת ניהול קופונים

קורס 822

הסבה לתכנות JAVA לסביבות Client,
Big Data-ו Enterprise

תוכן עניינים

3 כללי
5 שלב 1
15 שלב 2
20 שלב 3

במסגרת הקורס משולב פרוייקט מלווה המפורט במסמך.

הפרוייקט מחולק ל-3 שלבים:

1. בניית ליבת המערכת – בשלב זה יוקם 'המוח' של המערכת. ליבת המערכת תהיה אחראית על קליטת נתונים, אכסונום ב-Data Base, וניהולם.

2. שלב ה-WEB - בשלב זה תיחשף המערכת לאינטרנט. אופן בניית מודול זה יתמוך במגוון של לקוחות. במסגרת שלב זה יבנה אתר אינטרנט אך תהיה אפשרות לבנות גם לקוחות מובייל ו-PC.

3. שילוב יכולות Enterprise - שלב זה יתווסף לליבת המערכת אך יעשה שימוש בתשתיות שרת מתקדמות. התוספת לליבת המערכת תתרכז בתיעוד הכנסות המערכת. תיעוד זה יעשה ע"י בניית Microservice תוך שימוש בטכנולוגיות JAVA עדכניות לפיתוח צד-שרת כגון SpringData, SpringBoot ועוד.

חלה חובת הגשה כל השלבים

תיאור המערכת

מערכת ניהול קופונים מאפשרת לחברות (Companies) לייצר קופונים כחלק מקמפיינים פרסומיים ושיווקיים שהן מקיימות.

למערכת יש גם לקוחות רשומים (Customers). הלקוחות יכולים לרכוש קופונים (תיעוד הכנסות מתווסף בשלב השלישי של הפרוייקט). קופונים מוגבלים בכמות ובתוקף. לקוח מוגבל לקופון אחד מכל סוג.

המערכת מתעדת את הקופונים שנרכשו ע"י כל לקוח.

הכנסות המערכת הן מרכישת קופונים ע"י לקוחות רשומים ומיצירת ועדכון קופונים חדשים ע"י חברות.

גישה למערכת מתחלקת ל- 3 סוגי לקוחות

1. Administrator – מנהל רשימת חברות ולקוחות

2. Company – ניהול רשימת קופונים המשוייכים לחברה

3. Customer – רכישת קופונים

הנחיות כלליות

- עבודה על הפרוייקט יכולה להתבצע באופן עצמאי או בקבוצות של עד 3 מתכנתים.
- מועדי הנחיה והגשה – של שלב בפרוייקט מוגש עד לתאריך הצגת השלב הבא. מועדי הצגות הפרוייקט מופיעות בתוכנית הקורס. לוח זמנים זה קשיח, ולשינוי נדרש אישור מרכזת הקורס או מהגורמים המקצועיים. בכל מקרה של בקשה לשינוי בלוח הזמנים, יש לקחת בחשבון הורדת ציון.
- יש להציג את הפרוייקט עובד ומותקן.
- יש להגיש את קבצי המקור – ע"י שליחת קבצים באינטרנט.
- יש להשתמש בתיעוד, רצוי לייצר Java Docs.
- יש להגיש, בכל שלב, מסמך טקסט מפורט לגבי ההתקנה, שמות משתמשים וסיסמאות, נתוני התחברות ל-Data Base והנחיות למשתמש.
- במידה והתווספו יכולות מעבר לדרישות אלו נעשה שימוש ב-API מעבר לנלמד בכיתה – הדבר מבורך ויש לציין רשימת תוספות אלו עם ההגשה על מנת שיתייחסו לכך בבדיקה.
- אין להגיש פרוייקט אשר מדפיס stack trace במקרים של Exceptions. יש לייצר Exceptions מותאמים למערכת ולהשתמש בהודעות ברורות וקריאות למשתמש בכל מקרה של שגיאה

שלב 1

בניית ליבת המערכת

Java-SE, Threads, JDBC

בניית ליבת המערכת

תיאור:

בשלב זה יוגדר מסד הנתונים לאכסון ושליפת מידע אודות לקוחות, חברות וקופונים. מעל מסד הנתונים תוקם שכבת בידוד שתאפשר עבודה נוחה מ-Java אל מול ה-SQL הנדרש בפעולות מול DB.

כמו כן, יוקמו שירותי תשתית בסיסיים כגון Thread ו-ConnectionPool יומי המתחזק את המערכת ומנקה אותה מקופונים שפג תוקפם.

יוגדרו 3 Entry Points למערכת עבור כל אחד מסוגי לקוחותיה (אדמיניסטרטור, חברה או לקוח) אשר יתחברו בביצוע Login.

שלבים:

- א. הגדרת Data Base ובניית טבלאות
- ב. בניית Connection-pool ושכבת אובייקטים לניהול הגישה אל ה-Data Base – DAO (Data Access Objects)
- ג. יצירת מחלקות הגישה למערכת בהתאם לתפקידים הנתמכים בה (אדמיניסטרטור, חברה ולקוח)
- ד. הגדרת תת-תהליך (Thread) יומי למחיקת קופונים שפג תוקפם מהמערכת
- ה. בניית Singleton אשר מאפשר כניסה למנויים וביצוע פעולות בהתאם לזהותו
- ו. הכנה של Test.main() - לקוח שמדגים את כלל יכולות המערכת

דגשים:

- חשוב לתעד בגוף הקוד
- חשוב להשתמש בשמות מחלקות, מתודות ומשתנים משמעותיים
- חובה לכתוב באופן יעיל, ללא העתקת קוד

- יש להשתמש ב- Exceptions שהוגדרו עבור המערכת (למשל: CouponExistsException). בהגשת הפרוייקט Test.main() יעשה שימוש ב-try-catch והדפסת ה-message למסך. אין להדפיס stack trace.
- מומלץ לעבוד עם Derby DB – אין תועלת בעבודה עם מסדי נתונים המצריכים התקנות גדולות וניהול מורכב
- חובה להשתמש בחבילות בכל מחלקה. את המחלקות יש למקם בחבילות בעלות שמות משמעותיים המעידים על תפקידן.

א. הגדרת Data Base ובניית טבלאות

1. Company – טבלת חברות

- ID – LONG, PK
- COMP_NAME – STRING
- PASSWORD – STRING
- EMAIL – STRING

2. Customer – טבלת לקוחות

- ID – LONG, PK
- CUST_NAME - STRING
- PASSWORD – STRING

3. Coupon – טבלת קופונים

- ID – LONG, PK
- TITLE - STRING (תיאור קצר של הקופון - כותרת)
- START_DATE – DATE (תאריך יצירת הקופון במערכת)
- END_DATE – DATE (תאריך תפוגה)
- AMOUNT – INTEGER (כמות קופונים במלאי)
- TYPE – ENUM (STRING) (קטגוריה – מזון, חשמל, פנאי, נפש....)
- MESSAGE – STRING (מסר מפורט יותר, המלל של הקופון)
- PRICE – DOUBLE (מחיר הקופון ללקוח)
- IMAGE – STRING (קישור \ מיקום תמונה רלוונטית)

4. Customer_Coupon – טבלת לקוחות-קופונים – זוהי טבלת JOIN אשר ה-ID של לקוח וה-ID של קופון משמשים כ-PK. כך מתאפשר תיעוד של רכישת הקופונים ע"י הלקוחות ובנוסף נמנעת רכישת קופונים זהים ע"י אותו לקוח. (בהמשך ישנה דוגמת SQL להגדרת טבלת JOIN)

- CUST_ID – LONG
- COUPON_ID - LONG

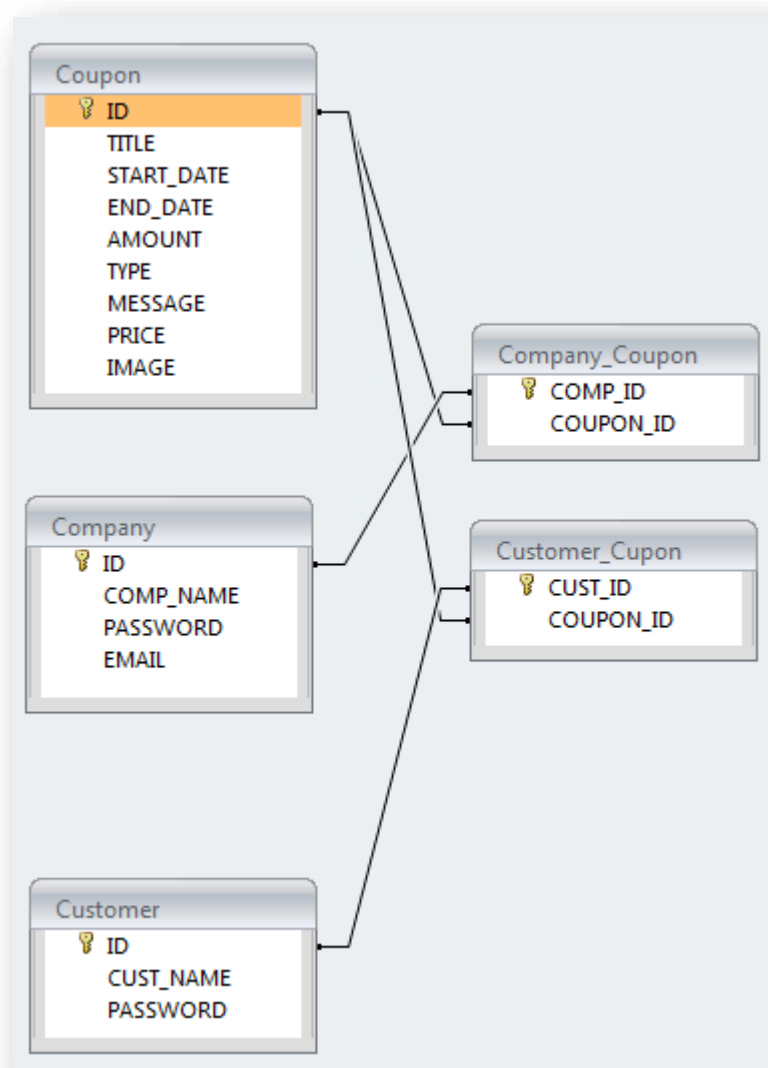
2. Company_Coupon – טבלת חברות-קופונים – גם זו טבלת JOIN כאשר ה-ID של חברה וה-ID של קופון משמשים כ-PK. כך מתאפשר שיוך קופונים לחברה ובנוסף נמנעת יצירת קופונים זהים ע"י אותה חברה. (בהמשך ישנה דוגמת SQL להגדרת טבלת JOIN)

- COMP_ID – LONG
- COUPON_ID - LONG

דוגמת SQL לטבלת JOIN :

```
CREATE TABLE Join_Table (  
    tableA_ID NUMERIC,  
    tableB_ID NUMERIC,  
    PRIMARY KEY (tableA_ID, tableB_ID)  
);
```

סכימת טבלאות המערכת והיחסים ביניהן:



ב. בניית Connection-pool ושכבת DAO

1. **ConnectionPool** – Singleton בעל מספר קבוע של Connections הנמצאים ב-Set Collection.

מכיל את המתודות הבאות:

`getConnection()` , `returnConnection(Connection)`, `closeAllConnections()`

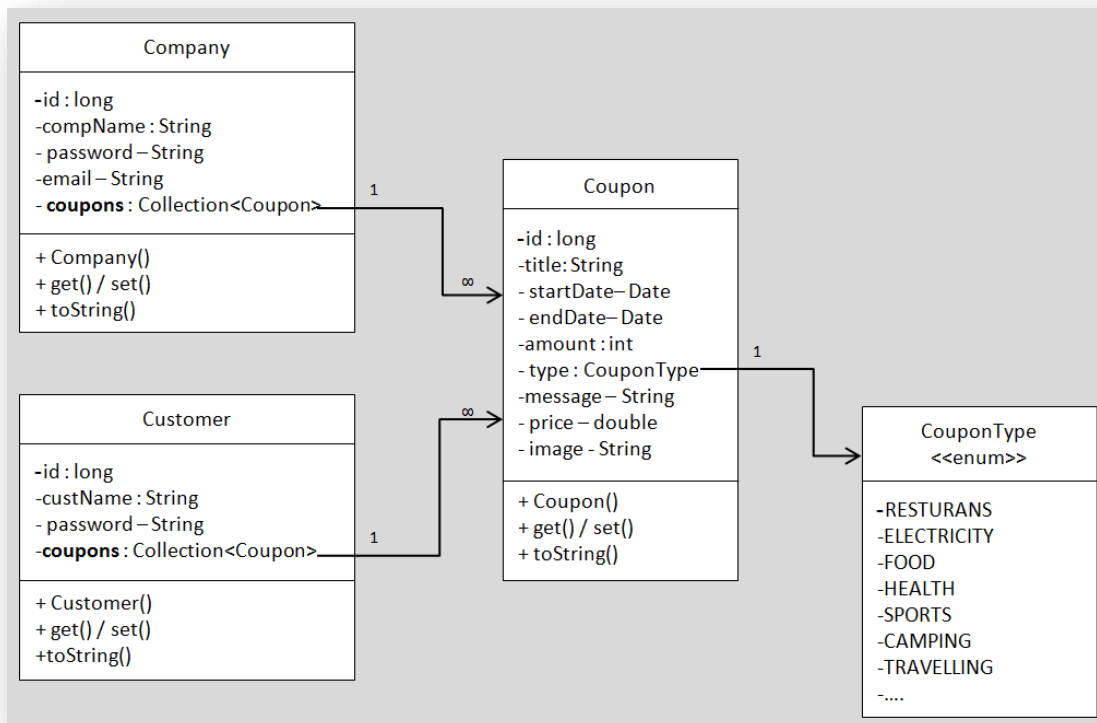
המתודה `getConnection()` מבצעת נעילה במידה ואין Connections פנויים

המתודה `returnConnection(..)` מחזירה Connection ומשחררת נעילה לממתינים (wait-notify)

המתודה האחרונה סוגרת את כל ה-Connections ומשמשת בסגירה מוחלטת של המערכת

2. Java Beans – הגדרת אובייקטים להעברת מידע מהאפליקציה ל-DAO אשר מתרגם אותם ל-SQL ו-JDBC. לכל טבלה עיקרית יש אובייקט משלה ואילו את טבלאות ה-JOIN ניתן לייצג באמצעות Collections. Customer מחזיק אוסף של Coupons וכמוהו גם Company.

להלן סכימת המחלקות המשמשות כ-Java Beans:



3. בניית DAO

בשלב זה מדובר בפעולות גולמיות המכונות C.R.U.D (Create, Read, Update, Delete). הלוגיקה המתבקשת בתוכנית לא באה לידי ביטוי כאן.

המטרה היא ששכבת האפליקציה והלקוחות תוכלנה לייצר Java Beans ולשלוח אותם ל-DAO על מנת שתהפוך אותם לפעולות SQL באמצעות JDBC.

כמו כן, בקשה לקבלת נתונים אמורה להתבצע בפועל ב-SQL – אבל שכבת ה-DAO תדע לתרגם את התוצאה ל-Java Bean או ל-Collection.

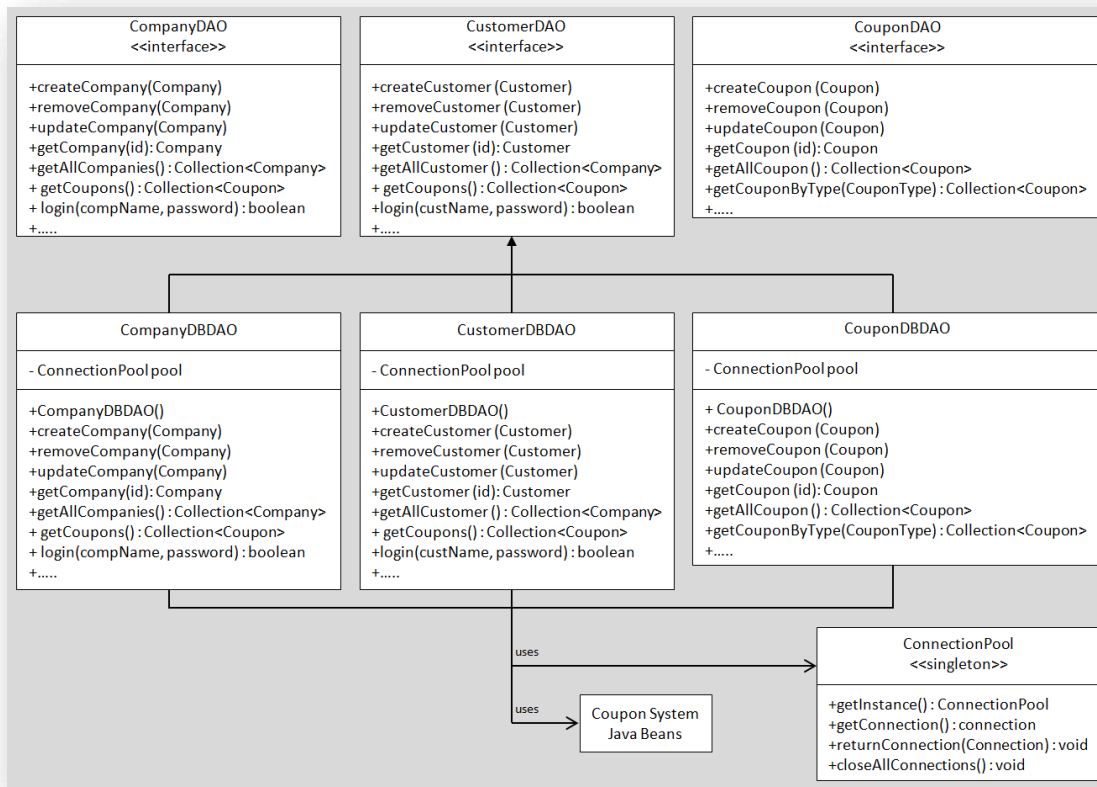
מטעמי Code design אנו משאירים אופציות שונות למימוש שכבת אכסון הנתונים. בפרוייקט שלנו המימוש היחידי יהיה אכסון ב-DB באמצעות JDBC אולם עיצוב שכבת ה-DAO יאפשר לממש אותה בעתיד גם

למקורות אכסון אחרים כגון קבצים או מחשבים מרוחקים. את זאת נשיג באמצעות הגדרת שכבת ממשקים ולאחר מכן נממש אותה ב-JDBC:

מחלקות אלו יוגדרו באופן הבא:

- שכבת Interfaces המגדירה פעולות כלליות לכל טבלה

- שכבת מחלקות מממשות אשר נעזרות ב-ConnectionPool (כל מתודה צורכת Connection בתחילתה ומחזירה אותו ל- pool בסיימה). המחלקות האלה עושות שימוש ב-JDBC על מנת לתרגם אובייקטים ל-SQL ולהפוך תוצאות QUERY ל-Collection.



חשוב! יש לטפל בשגיאות ולא "לקבור" אותן. לשם כך רצוי לייצר Exceptions ספציפיים למערכת הקופונים אשר יכילו מסרים מתאימים יותר למשתמש ("שם משתמש קיים במערכת" עדיף על "SQLException: Duplicate Key...")

ג. יצירת מחלקות הגישה למערכת בהתאם לתפקידים הנתמכים בה (אדמיניסטרטור, לקוח וחברה)

מחלקות אלו מייצגות את שכבת ה- Clients של המערכת. בשלבי הסיום של בניית הליבה יהיה Singleton שייצג את המערכת ובאמצעותו לקוחות יבצעו Login. תוצאת פעולת Login מוצלחת תהיה קבלת מופע של Client.

המחלקות היורשות מ-Client מייצגות את 3 התפקידים במערכת:

- AdminFacade
- CompanyFacade
- CustomerFacade

כל אחת מהמחלקות מאפשרת את הפעולות הרלוונטיות לה (כפי שיפורט בהמשך) ולשם כך עושות שימוש בשכבת ה-DAO

בחלק זה יש לממש את הלוגיקה העסקית של המערכת – למשל: במחלקה CustomerFacade תהיה מתודה המאפשרת רכישת קופון – purchaseCoupon(Coupon). מעבר לכך שדרוש עדכון כמות הקופונים שנותרה – יש לבדוק אם בכלל נותרו קופונים ולוודא שהלקוח לא רכש בעבר קופון זהה...

AdminFacade – תפקידים ומגבלות

- כניסה למערכת (login) – במקרה זה אין צורך לבדוק מול ה-DB. שם המשתמש והסיסמא יהיו תמיד:

username – admin
password - 1234

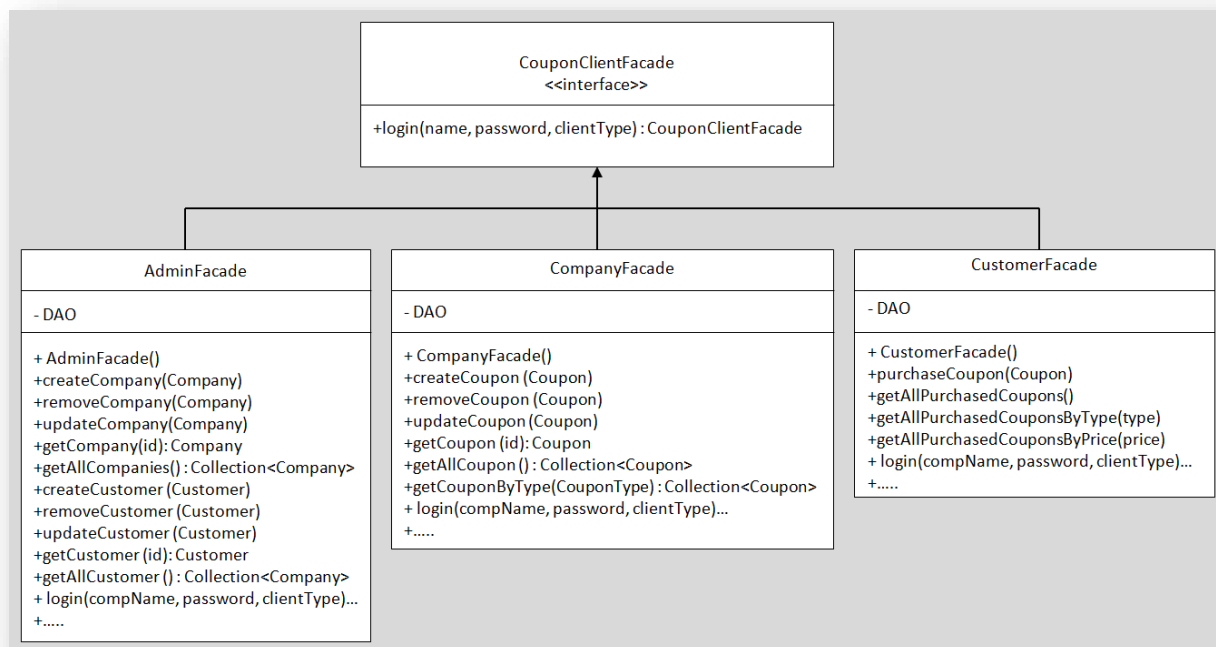
- הוספת חברה – לא ניתן להוסיף חברות עם שם זהה לקיים
- מחיקת חברה – חייבת להתבצע במקרה זה גם מחיקת הקופונים של אותה חברה וגם קופונים שנרכשו ע"י לקוחות
- עדכון פרטי חברה, למעט שם החברה
- צפייה ברשימת כלל החברות
- צפייה בפרטי חברה ספציפית
- הוספת לקוח – לא ניתן להוסיף לקוח עם שם זהה לקיים
- מחיקת לקוח – חייבת להתבצע במקרה זה גם מחיקת הסטורית רכישות הקופונים של אותו לקוח
- עדכון פרטי לקוח – למעט שם הלקוח
- צפייה ברשימת כל הלקוחות
- צפייה בפרטי לקוח ספציפי

CompanyFacade – תפקידים ומגבלות

- הוספת קופון – לא ניתן להוסיף קופון עם כותרת זהה לקיים
- מחיקת קופון – חייבת להתבצע במקרה זה גם מחיקת הקופונים שנרכשו ע"י לקוחות
- עדכון קופון – תאריך סיום, מחיר
- צפייה בפרטי החברה ספציפית
- צפייה ברשימת כלל הקופונים של החברה
- צפייה בקופונים של החברה לפי:
 - סוג קופון
 - עד מחיר מסויים
 - עד לתאריך מסויים

CustomerFacade – תפקידים ומגבלות

- רכישת קופון – לא ניתן לרכוש קופון יותר מפעם אחת, או אם אזל מהמלאי או אם פג תוקפו
- צפייה בהסטוריית רכישות הלקוח
- צפייה בהסטוריית רכישות הלקוח לפי:
 - סוג קופון
 - עד מחיר מסויים

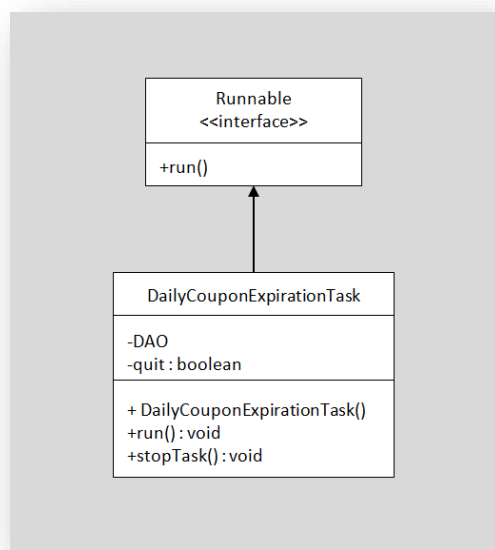


חשוב ! חלק מהפעולות המפורטות, בעיקר הוספה ומחיקה, משפיעות גם על טבלאות ה-JOIN.

ד. הגדרת תת-תהליך (Thread) יומי למחיקת קופונים שפג תוקפם מהמערכת

בשלב זה תוגדר Task (Runnable) אשר יבצע בדיקה יומית של הקופונים במערכת. במידה ונמצאו קופונים שתאריך הסיום שלהם הגיע – הם ימחקו הן מטבלת Coupon והן מטבלאות ה-JOIN.

יש לממש אפשרות להפסיק את ה-Thread ע"י קריאה למתודה מתאימה.



בפועל, ה-Thread יופעל עם עליית המערכת כחלק מתהליך טעינת ה-DAO

ה. בניית CouponSystem Singleton אשר מאפשר כניסה למנויים וביצוע פעולות בהתאם לזהותו

מחלקה זו תשמש כבסיס לחיבור מרכיבי מערכת הקופונים ומתן אפשרות ללקוחות השונים להתחבר ולהשתמש בה.

עם יצירת המופע תתבצענה הפעולות הבאות:

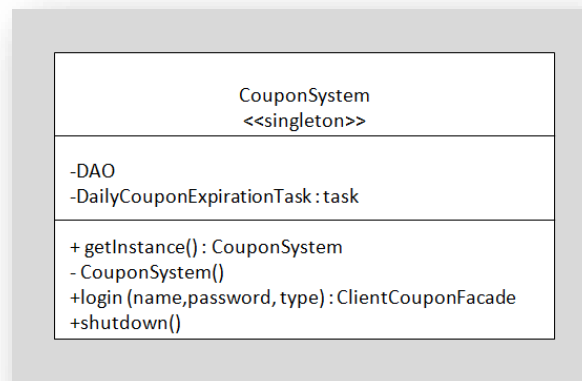
- טעינת DAO
- יצירת ואתחול DailyCouponExpirationTask

מתודות נוספות:

- login(name, password, clientType) : CouponClientFacade
- shutdown() : void

המתודה המבצעת Login מקבלת clientType שיכול להיות enum או String. באמצעותו תדע המערכת מה עליה להחזיר : AdminFacade, CompanyFacade, CustomerFacade

המתודה המבצעת shutdown – תסגור את ה-ConnectionPool ותעצור את ה-DailyTask.



ו. הכנה של Test.main() - לקוח שמדגים את כלל יכולות המערכת

חלק זה נועד לאפשר בדיקה של המערכת. המטרה היא לייצר את ה-Singleton העיקרי – CouponSystem ולבצע Login לכל אחד מסוגי הלקוחות על מנת לבצע "תצוגת יכולות" של כל אחד מהם.

אין צורך בקלט מהמשתמש – הערכים עימם עובדים יכולים להיות Hard-coded. הלקוחות האמיתיים יגיעו למערכת באמצעות אתר האינטרנט שיוקם בשלב הבא.

כל פעולה שעשויה לזרוק Exception צריכה להיות מטופלת ב-try-catch והמסרים של ה-Exceptions יודפסו למסך.

שלב 2

בניית תשתית לחיבור לקוחות באמצעות האינטרנט

JAX-RS, Angular 4

בניית תשתית לחיבור לקוחות באמצעות האינטרנט

תיאור:

בשלב זה יחשפו כל אחד מסוגי הלקוחות כ-Service.

כלומר, יהיו 3 שירותי רשת (web - services) אשר יאפשרו אינטגרציה באמצעות REST עם המערכת.

בשלב זה יבנה אתר אינטרנט בשיטת Single Page Application המאפשר ביצוע Login כאדנימיסטרטור, חברה או לקוח וביצוע כלל הפעולות הנתמכות במערכת.

שלבים:

א. הגדרת Web Services עבור כ"א מהלקוחות הנתמכים במערכת

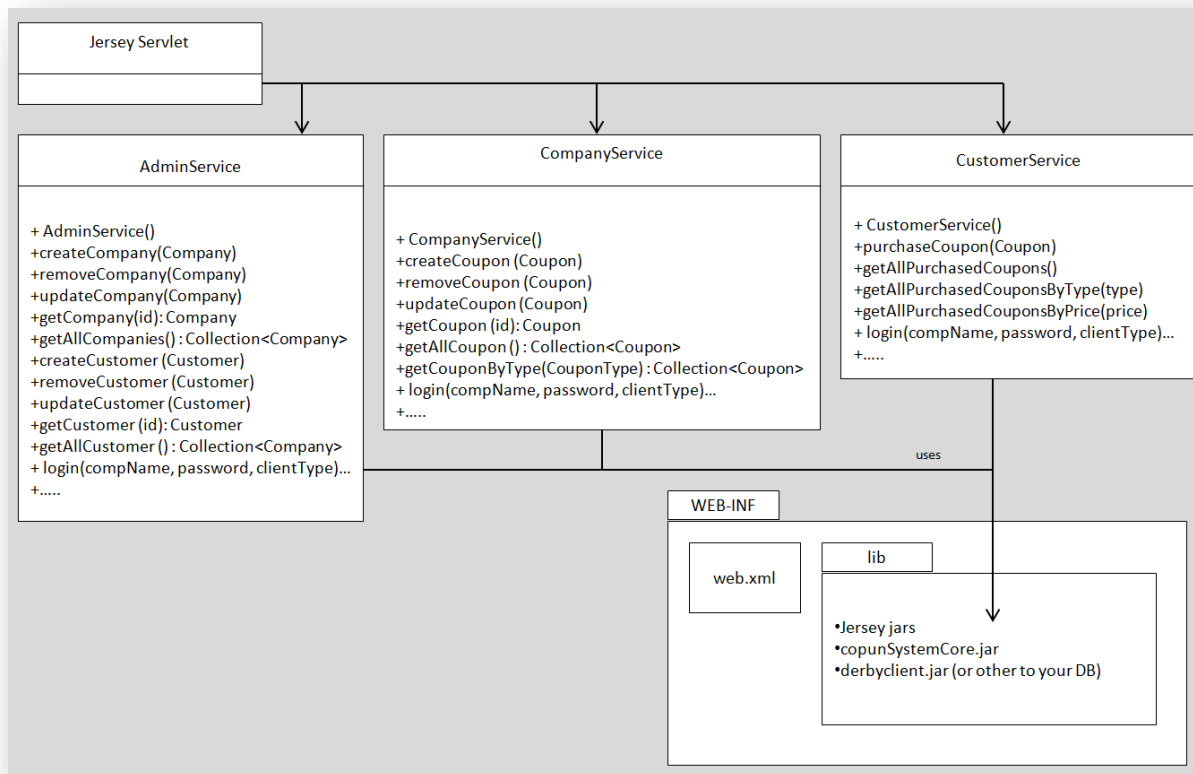
ב. בניית דפי HTML וחיבורם עם המערכת באמצעות Jscript ו-AJAX.

דגשים:

- יש להשתמש ב-Tomcat
- אין להשתמש ב-JSP. דפי ה-HTML חייבים להיות 100% טכנולוגיות לקוח (CSS, DHTML,...)
- מעבר הנתונים הגולמיים בין הלקוח לשרת יהיה באמצעות JSON ו-Plain Text. לא באמצעות XML.
- חובה לכתוב באופן יעיל, ללא העתקת קוד
- Exceptions שהוגדרו עבור המערכת יועברו כ-Message ללקוח על מנת שיוצגו במידת בצורך
- ניתן לבנות לקוח ב-JAVA תוך שימוש ב-Jersey Client jars

- חובה לממש Filter לבדיקת Login ולוידוא שה-Session עדיין פעיל. בכל מקרה ואין Login או Session פעיל – יש לדרוש ביצוע Login מחדש
- הממשק צריך להיות ברור ונוח לשימוש. ניתן לעבוד עם כל סיפריית UI ב-Jscript וב-JQUERY אך לא להשקיע יותר מדי בעיצוב – רק אם הזמן מאפשר זאת.
- חובה לתמוך באפשרות של Logout ע"י המשתמש
- מצורף בסוף פירוט שלב זה נספח המפרט את אופן העבודה עם JSON ב-JSCRIPT לטובת בניית הממשק באתר והצגת הנתונים בתוכו

להלן פירוט רכיבי ה-WEB ו-REST :



א. הגדרת Web Services עבור כ"א מהלקוחות הנתמכים במערכת

1. בניית מחלקות AdminService, CompanyService, CustomerService. כל אחת מאפשרת שליחת וקבלת נתונים ל-Facade הרלוונטי באמצעות REST Based Web-Services.

2. עם ביצוע Login (יהיה דף יעודי 'login.html' לכל הלקוחות או לכל סוג של לקוח) ייוצר Session ובתוכו יאוחסן ה-Facade הרלוונטי ללקוח. (השתמש ב-@Context על מנת להזריק HttpServletRequest ל-Service ומשם תהיה גישה ליצירת וניהול ה-Session)

שים לב! ה-Browser יודע להצמיד את ה-Session Cookie לכל בקשה עתידית מרגע היווצרות ה-Session כך שבעבודה עם Jscript תהיה תמיכה אוטומטית וכאשר שולחים AJAX request היא תגיע לשרת עם ה-Session Cookie. לא כך הוא הדבר כאשר בונים לקוח בסביבות אחרות כגון PC ומובייל (Android). במקרים אלו יש צורך לקחת את ה-Session Cookie מה-Response ולהעמיס אותה על ה-Request שבאה אחריה...אחרת יאבד הקשר עם ה-Session.

בשלב זה נבנה אתר האינטרנט הנצפה ע"י לקוחות מתוך הדפדפן. ההערה הנ"ל רלוונטית למקרה בו יוחלט עצמאית לספק גם סוגי לקוחות נוספים – אך אין דרישה כזו בפרוייקט הנוכחי.

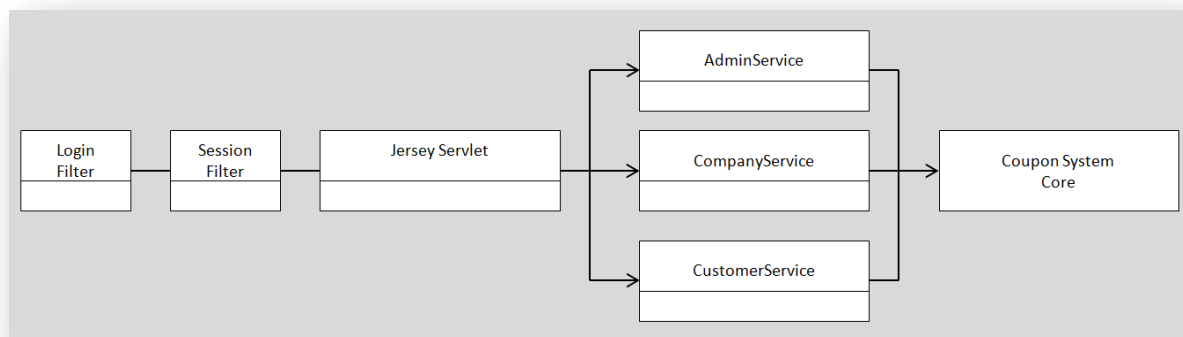
3. לאחר מכן – יוכל הלקוח להפעיל את הפונקציות הרלוונטיות לו מתוך Jscript העובדת מול ה-Service – ומשם לליבת המערכת עצמה

4. כל פעולה שאמורה להתבצע לאחר login ועם קיומו של Session תעבור דרך Filters. מכיוון שכל פעילות ה-REST מתבצעת דרך Jersey Servlet, יש למקם את ה-Filters עבורו.

- LoginFilter – לבדיקה אם הלקוח המפעיל מתודה ביצע כניסה מסודרת למערכת
- SessionFilter – SESSION – שפג תוקף ה-SESSION

ניתן לייצר Filter אחד שעושה את כל הבדיקות.

את הפילטרים יש להצמיד ל-Jersey Servlet באמצעות web.xml.



ב. בניית ממשק משתמש בטכנולוגיית Angular 4

חלק זה מייצר מסכים עבור הלקוחות השונים של המערכת. המימוש צריך להיות לפי Single Page Application – כלומר דפי HTML בודדים המאפשרים מכלול של פעולות מול השרת וזאת באמצעות פונקציות Jscript ו-AJAX.

ניתן לייצר דף נפרד לביצוע Login אולם מרגע כניסה למערכת העבודה צריכה להיות מבוססת על דף יחיד. את הנתונים ניתן להציג באופן דינאמי תוך שימוש ב-DIV, SPAN

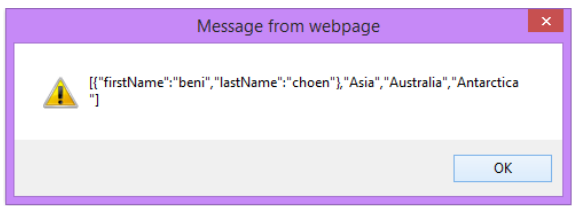
ניתן לשלב כל טכנולוגיית לקוח כגון JQUERY, CSS על מנת להעשיר את הUI ולהקל על האינטגרציה

תפריט הפעולות ישתנה בהתאם לסוג הלקוח (אדמיניסטרטור, חברה או לקוח) – ניתן לבצע זאת ע"י 3 דפי HTML נפרדים או בדף יחיד אשר מקבל מידע אודות סוג הלקוח עם טעינתו ומציג את התפריט הרלוונטי.

הנחיות כלליות לגבי הממשק:

- צריך להיות ברור ונקי.
- יש צורך ליידע את הלקוח בסיום פעולות.
- ווידוא קלט במילוי טפסים יתבצע ב-Jscript
- במצב של בחירה מרובה – יש להזין לשדה את הנתונים הקיימים במערכת ובכך למנוע טעויות לקוח (למשל – לבחירת לקוח מסויים כדאי לייצר List ובה רשימת הלקוחות הקיימים כפי שנטענו מה-DB. עדיף מאשר האדמיניסטרטור יקליד שם או ID של לקוח באופן חופשי...)
- לגבי Image של קופון – ישמרו בקבצים תחת תיקיה חיצונית (למשל: C://coupons/images) אבל ניתן יהיה לטעון אותם ולהציגם באתר כחלק מפרטי הקופון מומלץ להגדיר את מיקום התמונות ב-web.xml כ-Context Initial Parameter ולהפוך אותו לזמין עבור כל Service ע"י אכסונו ב-ServletContext.
- על מנת להזריק את ה-ServletContext ל-Services יש להשתמש ב-@Context בדיוק כפי שמוזרק ה-HttpServletRequest.

נספח – כיצד עובדים עם JSON מתוך JSCRIPT נכתב ע"י – חיים גלבע

<pre>{ "employees": [{ "firstName": "John", "lastName": "Doe" }, { "firstName": "Anna", "lastName": "Smith" }, { "firstName": "Peter", "lastName": "Jones" }] }</pre>	<p>JSON מבנה נתונים</p>
<pre>var text = '{ "employees1" : [' + '{ "firstName": "John", "lastName": "Doe" },' + '{ "firstName": "Anna", "lastName": "Smith" },' + '{ "firstName": "Peter", "lastName": "Jones" }]}'; var obj = JSON.parse(text); alert(obj.employees1[0].firstName);</pre>	<p>JSON הכנת אובייקט</p>
<pre>var contact = new Object(); contact.firstname = "Jesper"; contact.surname = "Aaberg"; contact.phone = ["555", "555-010"]; alert(JSON.stringify(contact));</pre> 	<p>JSON מאובייקט ל</p>
<pre>var continents = new Array(); continents[0] = new Object(); continents[0].firstName = "beni"; continents[0].lastName = "choen"; continents[1] = "Asia"; continents[2] = "Australia"; continents[3] = "Antarctica"; alert(JSON.stringify(continents));</pre> 	<p>JSON ממערך ל-</p>

שלב 3

בניית MICROSERVICE לתיעוד הכנסות המערכת

SpringBoot, SpringMVC, SpringData, Maven

תיעוד הכנסות המערכת

תיאור:

בשלב זה, פעולות הנעשות ע"י חברות ולקוחות באתר וכרוכות בתשלום עבור שירות – יתועדו במערכת. התיעוד יתבצע תוך מימוש Microservice – בשילוב הטכנולוגיות והכלים הבאים:

- SpringBoot
- SpringMVC+Embedded Tomcat
- SpringData
- Maven

תיאור ה-Flow:

- לקוח מבצע פעולה שיש לה עלות כלשהי (יפורט בהמשך)
- ה-Service מייצר מופע של Income המתאר את פרטי הפעולה, הלקוח והסכום
- ה-Service מעביר את ה-Income לשכבת DAO מבוססת (JPA) Hibernate על מנת לשמור את הנתונים

שלבים:

- א. הקמת פרוייקט Maven: SpringBoot הכולל SpringData, SpringMVC. יש לכלול גם את את DataBase Driver (אפשרי להשתמש ב-MYSQL ע"י הוספת MySQL Connector בהגדרת פרוייקט SpringBoot)
- ב. הגדרת משתנים ליצירת חיבור ל-DB ב-resources\application.properties
- ג. הגדרת Entity Bean (@Entity) בשם Income שתשמש כהודעת תשלום ותישמר ב-DB
- ד. הגדרת DAO (רכיב מסוג @Registry) לביצוע שמירת ושליפ נתוני הכנסות (יפורט בהמשך)

- ה. בניית Service (@Controller) המאפשר להפעיל את ה-DAO באמצעות REST
ו. הוספת Business Delegate לשכבת ה-Web (שלב ב) על מנת לדווח על הכנסה עקב ביצוע פעולת לקוח ותמיכה בהפקת דוחות כספיים (יפורט בהמשך) ולמעשה להתחבר לשירות החדש
ז. עדכון ה-Services השונים (שלב ב) כך שייצרו מופעי Income בכל פעם שמתבצעת פעולה הכרוכה בתשלום וישלחו אותה לאחסון באמצעות ה-Business Delegate
ח. הוספת יכולות צפייה בדו"חות כספיים רלוונטיים עבור ה-Administrator וה-Company

פירוט הרכיבים והפעולות

א. Entity Bean בשם Income שתשמש כהודעת תשלום ותישמר ב-DB

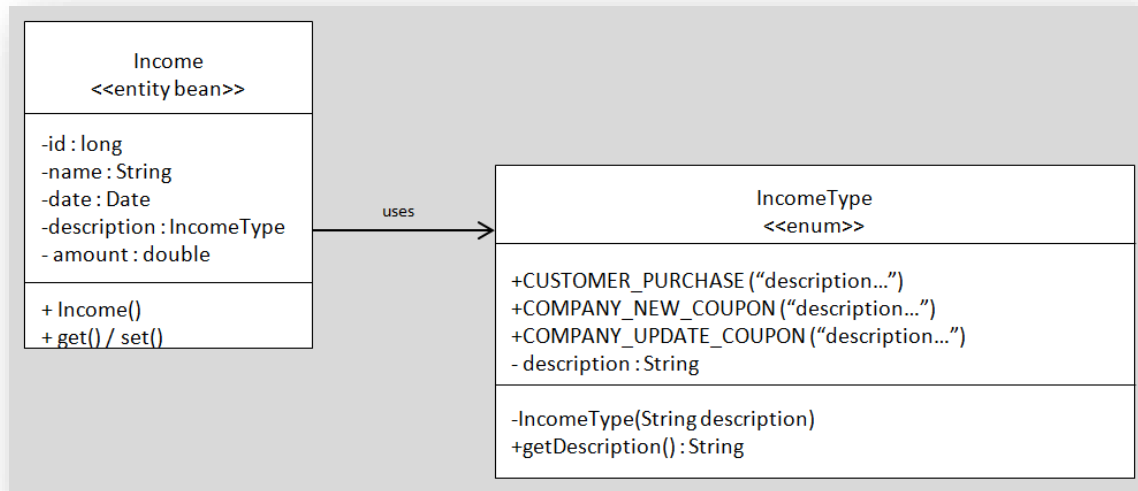
מבנה רשומת נתוני תשלום יהיה מורכב מ-

- PK – id
- name – שם מבצע הפעולה (שם חברה או לקוח)
- date – תאריך ביצוע הפעולה
- description – תיאור הפעולה (יפורט בהמשך)
- amount – סכום

להלן הפעולות בגינן יגבה תשלום:

סכום	תיאור הפעולה
מחיר הקופון	לקוח – רכישת קופון
תשלום חד פעמי של 100	חברה – מייצרת קופון חדש
10	חברה – עדכון פרטי קופון

מומלץ לייצר enum אשר יספק description לכל פעולה



ב. הגדרת DAO אשר מקבל Income ושומר אותו ב-DB באמצעות SpringData Template

רכיבים [IncomeServiceTemplate & IncomeServiceDAO] אלו מאפשר גישה ל-DB לשם שמירת Income.

- IncomeServiceTemplate – ממשיך המאפשר את הפעולות הבאות:
 - storeIncome (Income) : void (בהמשך מפורט באילו מקרים המתודה מופעלת)
 - viewAllIncome() : Collection<Income> - מופעלת ע"י שירות אדמין
 - viewIncomeByCustomer(customerID) : Collection<Income> - מופעלת ע"י שירות אדמין
 - viewIncomeByCompany(companyID) : Collection<Income> - מופעלת ע"י שירות חברה
- IncomeServiceDAO - רכיב @Registry המפעיל את IncomeServiceTemplate ומכיל למעשה את אותן מתודות:
 - storeIncome (Income) : void
 - viewAllIncome() : Collection<Income>
 - viewIncomeByCustomer(customerID) : Collection<Income>
 - viewIncomeByCompany(companyID) : Collection<Income>

ג. בניית REST Service (@RestController)

תפקידו של השירות הוא לייצר ל-4 המתודות שב-DAO שכבת REST ובכך לאפשר לפנות אל השירות באמצעות HTTP.

לנוחיותכם – להלן רשימת Spring MVC - Annotations רלוונטיות בהשוואה ל-JAX-RS

SPRING-MVC	JAX-RS
@Controller / @RestController	@Path - class level
@RequestMapping(method=RequestMethod.GET)	@GET/POST/DELETE/PUT/HEAD
@RequestMapping(value="/somePath")	@Path – method level
@RequestMapping(produces="application/xml")	@Produces
@RequestMapping(consumes="application/xml")	@Consumes
@PathVariable	@PathParam
@RequestParam	@QueryParam

ד. הוספת Business Delegate לשכבת ה-Web (משלב 2) על מנת לדווח על הכנסה עקב ביצוע פעולת לקוח ותמיכה בהפקת דוחות כספיים

תפקידו של Business Delegate הוא לשמש כ-Proxy בין שכבת ה-Web ובין שכבת ה-Business.

מכיוון ששרות זה עושה שימוש בטכנולוגיית JAX-RS יש להריץ אותו על מופע אחד של TOMCAT (העושה שימוש ב-8080 POST) ולהשתמש ב-Jersey-Client-API על מנת להפעיל את ה-Microservice שנבנה בשלב זה ורץ על מופע אחר של TOMCAT העושה שימוש ב-PORT אחר (למשל 8888).

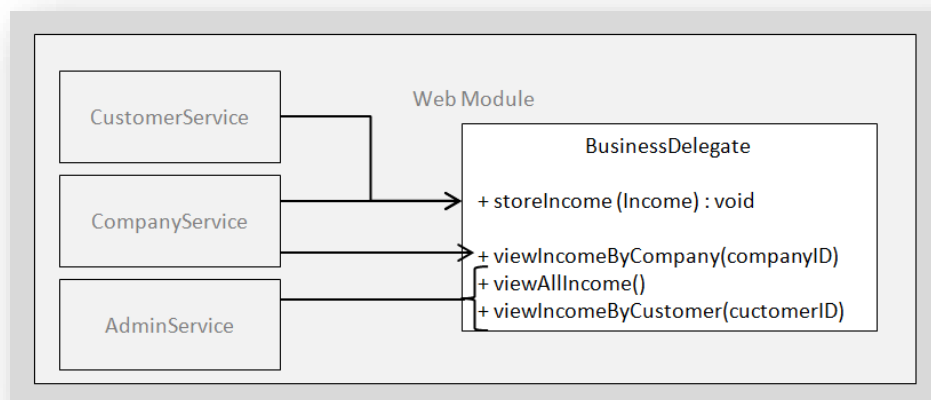
- על מנת לשנות את ה-Default port של Embedded Tomcat בפרוייקט הנוכחי יש להוסיף לקובץ – resources\application.properties את השורה הבאה:

server:port = 8888

כיוון ש-Business Delegate משרת את כלל הלקוחות יש להגדיר את המתודות שלו כ-synchronized.

להלן רשימת הפעולות המערבות Business Delegate

- CustomerService
 - מבצע storeIncome בכל פעם שלקוח רוכש קופון (ההכנסה = מחיר הקופון)
- CompanyService
 - מבצע storeIncome בכל פעם שנוצר קופון חדש (ההכנסה = 100)
 - מבצע storeIncome בכל פעם שמתעדכן קופון קיים (ההכנסה = 10)
 - מבצע viewIncomeByCompany על מנת לראות כמה כסף שילמה החברה בסה"כ
- AdminService
 - מבצע viewIncomeByCompany לצפייה בסך ההכנסות מחברה מסויימת
 - מבצע viewIncomeByCustomer לצפייה בסך ההכנסות מלקוח מסוים
 - מבצע viewAllIncome לצפייה בסך ההכנסות



ה. עדכון ה- Services השונים כך שיצרו מופעי Income בכל פעם שמתבצעת פעולה הכרוכה בתשלום וישלחו אותה לאחסון באמצעות ה-Business Delegate

כל Service רשאי להחזיק לעצמו מופע BusinessDelegate משלו.

ו. הוספת יכולות צפייה בדו"חות כספיים רלוונטיים עבור ה-Administrator וה-Company

יש לעדכן את ה-CustomerService, CompanyService, AdminService ואת המסכים (HTML) על מנת לתמוך בשליחת תיעוד תשלומים ובהצגת דוחות כספיים

- הודעה בדבר קליטת התשלום והסכום בכל פעם שלקוח רוכש קופון ובכל פעם שחברה מייצרת או מערדכנת קופון
- הצגת דו"ח עבור חברה (צפייה בכל התשלומים שביצעה)
- הצגת דו"חות עבור אדמיניסטרטור (צפייה בכלל ההכנסות, בהכנסות מחברה ובהכנסות מלקוח)

בהצלחה !