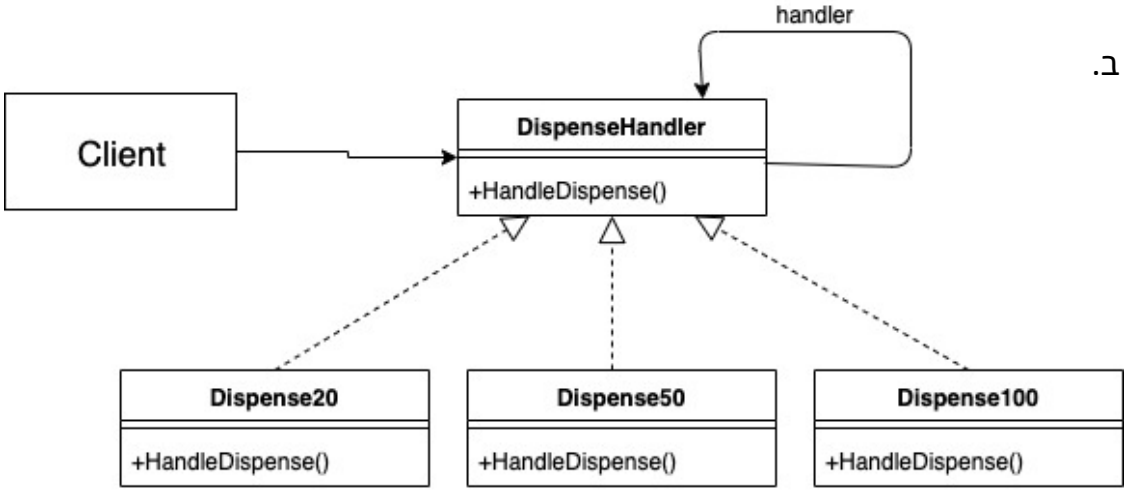
עבודה 2 –יסודות הנדסת תוכנה

שאלה 2

א.
נניח שאנחנו צריכים לממש את מנגון משיכת הכסף בכספומט.
המשתמש מזין כמה כסף הוא רוצה להוציא, והכסף מוחזר בשטרות של 100,50,20. נניח שיש לנו מתקן עבור שטרות של 100, מתקן עבור שטרות של 50 ומתקן עבור שטרות של 20, וכמובן נניח שאנחנו רוצים להחזיר שטרות לפי הסדר הבא : 100 , 50 ו 20.
לכן שרשרת האחריות שלנו תהיה : החזר שטרות של 100 -> החזר שטרות של 50 -> החזר שטרות של 20, כך שכל חולייה בשרשרת תנסה לקחת כמה שיותר שטרות מהסוג שלה, ואם היא לא מצליחה היא **תעביר את האחריות לחולייה הבאה**.(נניח כי סכום הכסף המבוקש הוא תמיד כפולה של 10). כמובן שבמימוש קונקרטי כל חוליה כזאת תממש את הממשק Handler והפונקציית ()handle שלה היא זו שתבצע את הלוגיקה שמפורטת הנ"ל.

ב.



תפקיד ה Handler - DispenseHandler שמהווה את הממשק אותו ממשים שאר מנפקי הכסף. המתודה ()HandleDispense מחליטה כמה שטרות לקחת מאותו סוג ואז קוראת למנפק הבא.
תפקיד ה ConcreteHandler – הם ה Dispense20, Dispense50, Dispense100 שממשים את הממשק.
תפקיד ה Client – המשתמש אשר רוצה למשוך כסף ומזין את הסכום המבוקש.

ג.
דמיון :
- בשתי המימושים אנו מקבלים מבנה דומה, של מעין רשימה מקושרת, כאשר כל אובייקט מכיל את האובייקט הבא בשרשרת שלו יש לקרוא.

- שני תבניות העיצוב בוחרות בקומפוזיציה כפתרון **במקום הורשה** כדי לספק פתרון שהוא יותר גמיש בזמן ריצה.

שוני :

- ב Chain Of Responsibility אנו עוצרים את הקריאות ל successor ברגע שהגענו לחולייה אשר יכולה להתמודד עם הקלט, לעומת זאת ב Decorator אנו ממשיכים את הקריאות עד להגעת למחלקת האב.
- ב Chain Of Responsibility אנו משתמשים כאשר אנחנו רוצים להתמודד עם הודעות שונות, ואילו ב Decorator אנו משתמשים כאשר אנו רוצים להוסיף **פונקציונליות או מידע חדש**.

ד.

מימוש של תוכנה לחברה המכינת פיצות, כך שיש סוגי פיצות שונות וסוגי תוספות שונות. נשתמש בתבנית העיצוב Decorator על מנת להוסיף או להסיר פונקצינאליות שונה לכל פיצה, נעשה זאת כך שתהיה לנו פיצה בסיסית (לדוגמא רק עם גבינה) ועליה נוסיף עוד סוגים/תוספות.

דוגמא זאת לא מתאימה ל Chain Of Responsibility מכיוון שכל חוליה בשרשרת היא קריטית (מהווה תוספת לפיצה) ומוסיפה מידע חדש וחשוב, לעומת זאת Chain Of Responsibility נועדה על מנת להתמודד עם הודעות וברגע שהגענו לחוליה בשרשרת אשר יודעת להתמודד (handle) עם ההודעה, היא מטפלת בה ועוצרת את התהליך.

<u>שאלה 3</u>

א.

נבחר בתבנית העיצוב Observer. היא מתאימה משום שיש לנו תלות של אחד לרבים בין הפרוייקט לסטודנטים שנרשמו אליו, כך שכאשר הפרוייקט משנה אתה מצבו, כל הסטודנטים הנ"ל צריכים להתעדכן על כך באופן אוטומטי.

שימוש בתבנית זו מספק לנו את האפשרות לבצע את ההתראות האלה, תוך כדי שימוש באבסטרקציה כך שהפרויקט לא צריך להכיר את האובייקטים התלויים בו.

כמו כן, בחרנו בגישת ה push : ה subject שולח ל observers שלו את פרטי המידע שהשתנו.

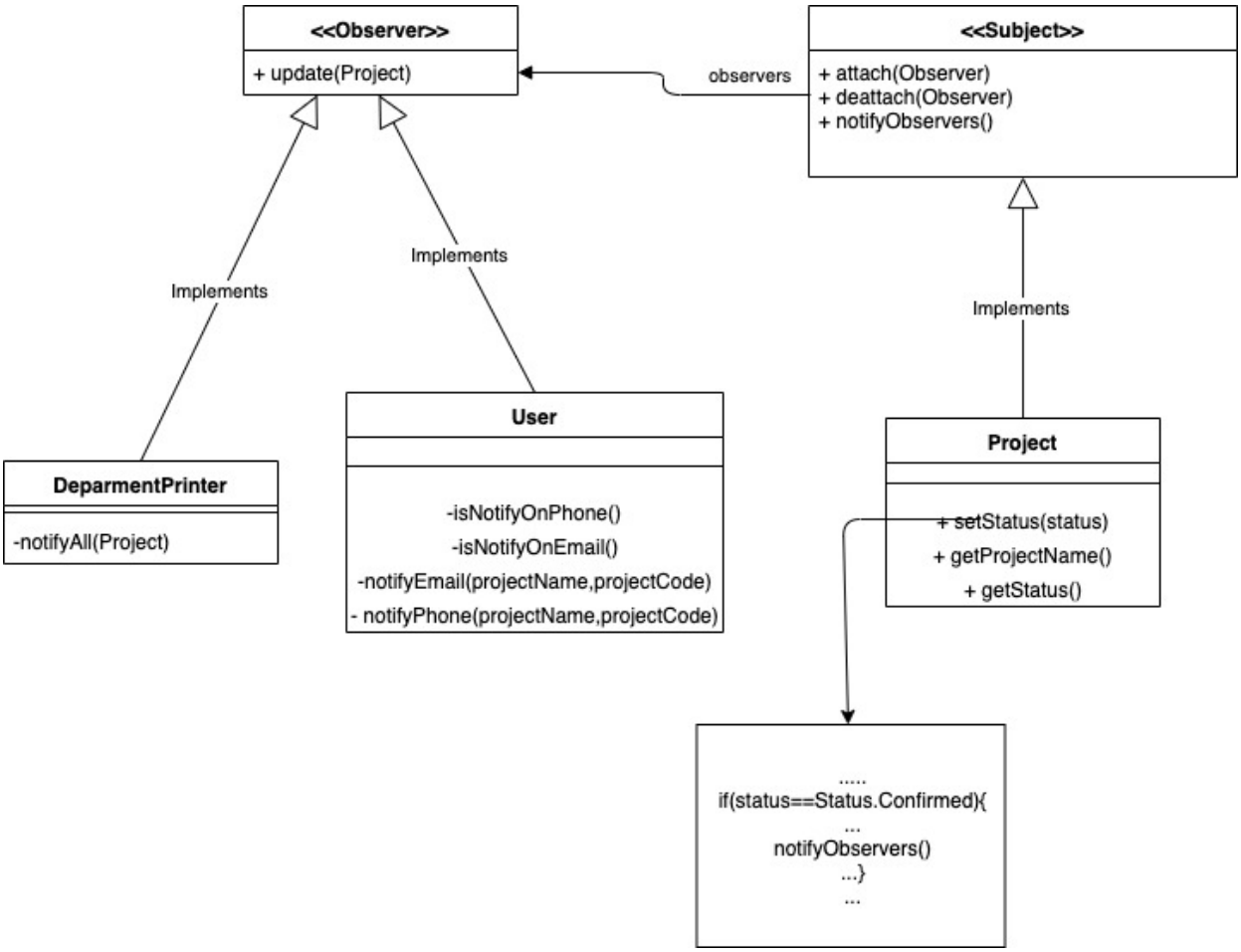
ב.

תפקידי המשתתפים :

ConcreteObserver – User and DeparmentPrinter

ConcreteSubject- Project

התרשים מצורף למטה, וכפי שניתן לראות באותו, בעת שינוי מצבו של ה Subject במתודה setStatus אנו מודיעים על כך ל Observers.



<u>על מנת להשתמש ברכיבים הנ"ל :</u>

שתי מתודות רלוונטיות ב ProjectManagment :

registerObserverToProject – אשר מקבלת שם משתמש, מספר פרויקט אליו הסטודנט רוצה להירשם ושני דגלים עבור קבלת ההתראות בטלפון או במייל

setProjectStatus – על מנת לשנות מצב פרויקט למאושר, כדי לגרום לנוטפיקציות.

<u>שאלה 4</u>

א.

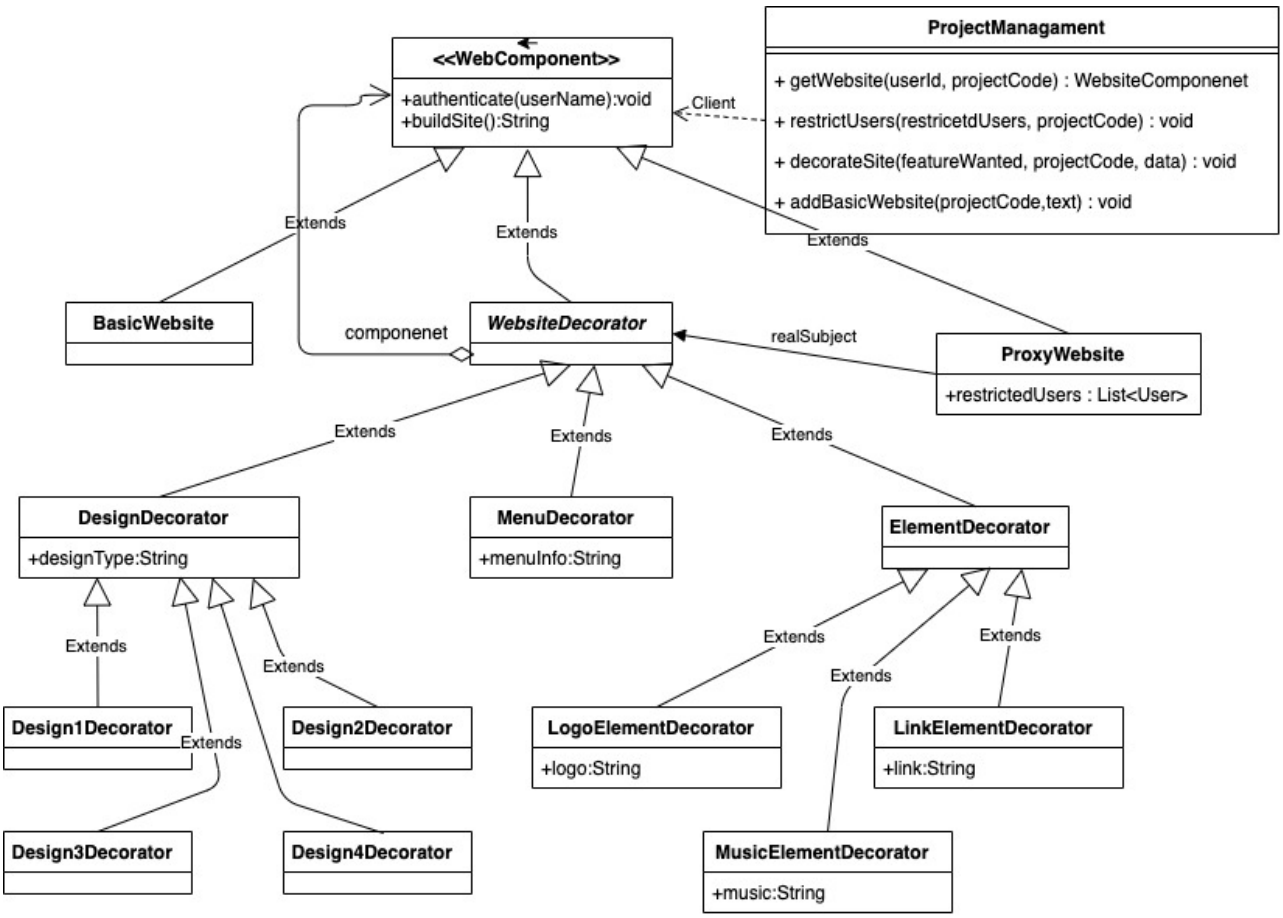
נבחר בתבניות העיצוב Decorator ו- Proxy.

בתבנית העיצוב Decorator נבחר משום שאנו רוצים לבנות אתר עם אפשרות של הרחבת הפיצ'רים שמופיעים בו, מעבר לאתר הבסיסי. תבנית העיצוב Decorator מאפשר לנו לבצע את ההרחבות האלה בזמן ריצה, בלי שימוש בפתרון לא גמיש כמו ירושה, אלא באמצעות קומפוזיציה.

כל הרחבה לאתר תהיה Decorator אשר מכיל בתוכו קומפוננטה אחרת של האתר, ומכיוון שלכל הרכיבים באתר יש אותו ממשק, הדבר מאפשר לנו לבטל את התלות בין הרכיבים השונים.
כמו כן נתבקשנו להגביל אפשרות גישה של משתמשים מסוימים לרכיבים מסוימים באתר, לשם כך בחרנו בתבנית העיצוב Proxy אפשר מאפשרת לנו לדמות רכיב מסוים באתר כשלמעשה היא מגבילה את הגישה לאותו רכיב.

ב.
ניצור ממשק של WebCompnonet המכיל שתי מתודות : authenticate(userID) אשר מאפשרת לאתר לזהות את המשתמש שניגש אליו, ו -()buildSite אשר מחזירה מחרוזת שמתארת את האתר. את הממשק הזה יממשו שלושה רכיבים עיקריים :
אתר בסיסי (BasicWebsite)– אתר אשר מכיל רק טקסט.
רכיב הפרוקסי – מימוש רכיב ה Proxy המאפשר להגביל את הגישה לקומפוננטה מסוימת המרכיבה את האתר.
דקורטור – רכיב המאפשר הוספת מאפיינים שונים לאתר.
בסוף סעיף ג' אתאר flow של כיצד להשתמש ברכיבים הנ"ל .

ג.



תקפידי המשתתפים :

עבור Decorator :
Component (WebsiteCompnonet) - הגדרת ממשק לרכיבים אשר רוצים להרחיב את האתר.
ConcreteComponent (BasicWebsite) - אובייקט אשר אליו יצורפו הרחבות שונות.
Decorator (WebsiteDecorator) - מממש את הממשק של שאר הרכיבים באתר, ומכיל הפניה לקומפוננטה נוספת.
ConcreteDecorator - ההרחבות עצמן לאתר, לדוגמא : MenuDecorator, DesignDecorator ועוד.

עבור Proxy :
Subject (WebComponent) – ממשק משותף אשר מאפשר ל Proxy "להתחזות" לרכיב המקורי.
Proxy(ProxyWebsite) – מכיל בתוכו הפנייה לאתר אמיתי, ורשימה של משתמשים אשר אין להם לגישה לאתר האמיתי.
RealSubject (WebsiteDecorator and BasicWesbite) – האובייקטים האמיתיים אותם מכיל ה Proxy.

תיאור עבודה עם הרכיבים הנ"ל :
בתחילה סטודנט אשר רוצה ליצור אתר לפרויקט יקרא למתודה addBasicWebsite שתיצור אתר בסיסי המקושר לפרויקט.
בכל פעם שהסטודנט רוצה להרחיב את האתר עם אפשרות חדשה, הוא יקרא למתודה decorateSite , אשר תקבל מהסטודנט מידע לגבי איזה רכיב הוא רוצה להוסיף לאתר, ומידע לגבי הרכיב עצמו. המתודה תעטוף את האתר הנוכחי ב WebsiteDecorator המתאים לדרישות המשתמש.
בכל פעם שהסטודנט רוצה להגביל רכיב מסוים באתר למשתמשים ספציפיים אחרים, הוא יקרא למתודה restrictUsers אשר מקבלת רשימה של מזהיי משתמש אותם יש לחסום.
המתודה תעטוף את האתר הנוכחי של הפרויקט ב ProxyWebsite אשר יודע לבצע את החסימה הדרושה.
כאשר משתמש מסוים רוצה גישה לאתר הוא יקרא למתודה getWebsite עם שם משתמש, המתודה תבצע אוטנתיקציה באתר למשתמש (לפיו ה Proxy ידע להגביל את המשתמש), ותחזיר את האתר הדרוש.