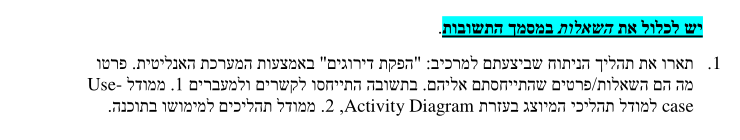
**שאלות פתוחות מטלה**



1. דיאגרמת Use-case מתארת קשר בין שחקנים הפועלים במערכת בתרחישים שונים, כך הבנו כי "הפקת דירוגים" הוא תרחיש שעובד מחלקת השיווק לוקח חלק בו, בעזרת תרחיש זה ניתן לבצע את תרחיש "הפקת מבצעי מכירות" שחלק מרכזי בו נעזר בדירוגים המופקים.

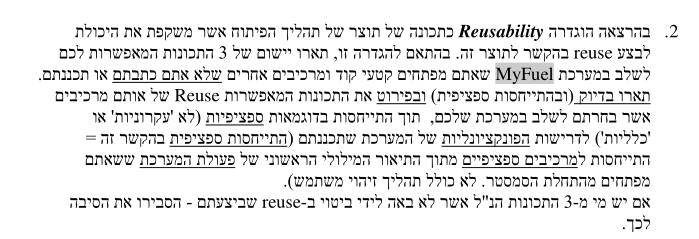
בעקבות זאת הסקנו כי בשלב האימפלמנטציה נצטרך לתת התייחסות לצורך של הפקת הדירוגים,שיהיו זמינים לאיש מכירות, וכתוצאה מכך הוספנו ב- Use-case, בחלק ה- Logic של המערכת תרחישי עזר.

לאחר מסקנות אלו ניתן לתת ביטוי לסדר הפעילויות ולתאר זאת בעזרת מודל activity.

מה-activity יעלו השאלות באיזה שלב הדירוגים מופקים ואיזה חלק במערכת מפיק אותם וכיצד נתונים אלו באים לידי ביטוי.

ובכך נסיק כי נצטרך למשוך מהשרת את הדירוגים לפני שניצור מבצע מכירות ואיך הנתונים יבוטאו במסך.

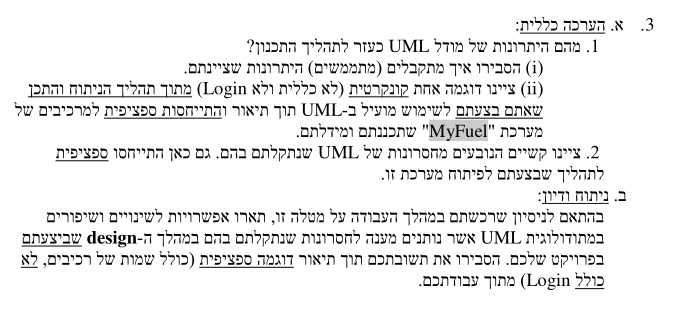
דיאגרמה זו תציג לנו בבירור בזמן מימוש המסך "הפקת מבצע מכירות" את הפרטים שהוא יכלול והמידע שנצטרך להציג על המסך



1. בתהליך הפיתוח באה לידי ביטוי עיקרון ה-"reusability"המקיים את התכונות גמישות,נגישות,וניתנות של המערכת להבנה.

לדוגמה בפרויקט שלנו:

* השתמשנו ב-OCSF שזה הוא framework של שרת-לקוח. נראה כיצד באה לידי ביטוי תכונת **ניתנות להבנה.** OCSF מקיים את התכונה "ניתנות להבנה" בזה שה-framework מחולק לשני חלקים של שרת ולקוח ברגע שמממשים את המחלקה האבסטרקטית מיד ניתן לדעת מה הקוד החסר ואיך להתאים אותו עבור המערכת. בקוד שלנו תכונה זו מתבטאת בכך שמימשנו בעזרת ה-framework פעולות של משיכת מידע מהשרת.
* השתמשנו ב-JBDC המתאר את תכונת ה**גמישות**, תכונה זה באה לידי ביטוי על ממשק JBDC בזה שניתן להשתמש בשימוש חוזר בקטעי בקוד לחלקים שונים במערכת, כמו לדוגמה שמירת לקוח חדש ב-DB ושמירת דו'ח רבעוני חדש ב-DB מתבצעים ע"י פקודת insert של database controller .
* השתמשנו באובייקטים של javaFX ספרייה זו היא חלק מספריות ה-utility של java ולכן התקיימה תכונת ה**נגישות**, מפני שעשינו שימוש חוזר בממשק באובייקטים שהספרייה מציעה אובייקטים אלו הם קטעי קוד מוכנים מראש הנגישים לכולם ואין צורך לכתיבת קוד מחדש למשל בממשק שלנו עשינו שימוש חוזר ב-label,gridpane,vbox וכדומה.



1. א. מידול בעזרת UML כלי מרכזי בתהליך התכנון בפרויקט.

היתרונות במודל ה-UML הם:

* זהו תהליך שאינו צורך זמן תכנות, כלומר כדי להבין מערכת המעוניינים לבנות לא צריך להתחיל ולבנות חלקים ממנה מספיק להבין את הקשרים ולהיעזר בדיאגרמות שונות .

לדוגמה בפרויקט שלנו עשינו במלואה Class ו-Usecase לכל תרחישים הזמן שלקח לנו לעשות אקח הרבה פחות מאשר לנסות לדמות בקוד חלקים ממערכת.

* כלים ב UML (קשרים/בועות) הם סטנדרטים לכן הוא מובן לכל חברי הצוות בפרט לגורמים חיצוניים.

לדוגמה בגלל שהוא סטנדרט אנחנו יכולים להתייעץ עם חברי הסגל ואין צורך רב לפרש כל תרחיש או חלק מהדיאגרמה בזמן שעת הקבלה.

* מגדיר את גבולות הגזרה של המערכת, כלומר הבנו מה המערכת שלנו צריכה לכלול ומה מבצעת בגורמים חיצוניים.

לדוגמה אנחנו יודעים שמערכת שלנו אינה צריכה לממש את התשלום, וזה מתקבל ישירות מ-Usecase היות ולא יצרנו תרחיש כזה.

* יוצר הבנה רחבה של המערכת. יוצר חיבור בין הדרישות השונות בסיפור למבנה המערכתי ומעלה דרישות נוסופות שאינם נכללות בסיפור בגלל הצורך העולה. בנוסף מראה את הקשר בין המרכיבים השונים במערכת. לדוגמה בסיפור קיימת דרישה להוספה לקוח למערכת, מתהליך המידול הבנו כי בעת הוספת הלקוח המערכת צריכה לדרוש הוספה של רכב על מנת לסיים את שלב הרישום.
* מפתחת כלומר קיים הרבה דוגמאות ברשת למידולים וגם frameworks רבים מתוארים בעזרת. לדוגמה OCSF הוא חלק קוד שנרצה להשתמש וקיים לו class diagram ידוע.

(אופציות)

* הוא מועיל בכך שהוא קריא מאוד,הדיאגרמות מובנות על ידי כל סוג של מתכנת ולפיהם ניתן להסביר את היחסים במערכת באופן ישיר.באופן מסורתי כדי להבין תוכנית או מערכת המתכנת יקרא את הקוד באופן ישיר,לדוגמה בפרויקט שלנו השתמשנו בדיאגרמות Class, ו-Usecase למידול המערכת דבר זה תרם לכל חברי הצוות להבין את החלקים והיחסים במערכת גם לאחר שינויים במודל
* UML הוא כלי תכנון עוזר לתכנן תוכנית עוד לפני שלב המימוש. כתוצאה ממידול המערכת מתקבל שלד ויזואלי של המערכת והקשרים בין האובייקטים השונים. זה יכול לעזור להפחית תקורה בשלב המימוש של המערכת. בנוסף, קל לשנות את תרשים מודל UML, בעוד שתכנות מחדש של קטע קוד יכול להיות מייגע וארוך זמן.

1. ב. במהלך תהליך המידול עלו גם החסרונות בשיטת UML :

* רב צורתית כלומר אפשר למדל דרישה בדיאגרמה כלשהי במספר צורות. לדוגמה עלו בפגישות הצוותיות דיונים כיצד למדל תרחיש כאשר כל אחד מחזיק בדעתו צורה שונה למידול לכן נתקלנו בבעית מהו המידול המתאים ביותר בסיפור.
* המידול הוא תהליך ארוך, אפילו שיחסית לקוד מידול בעזרת דיאגרמות קצר יחסית אבל עדיין מידול איכותי של מערכת דורש זמן לדוגמה N שבועות הראשונים היו ברובם לשלב זה.
* מידול בעזרת UML דורש ניסיון רב, ניקח את הצוות שלנו לדוגמה אפילו שלמדנו ועסקנו בחומר זה בקורס "מבוא להנדסת תוכנה" עדיין היינו צריכים להתייעץ עם גורמים מקצועיים.
* במידה והתהליך מידול מסובך נפגעת מידת הקריאות הבנה מתקבל מדיאגרמה. לדוגמה במידול הראשוני שלנו ב-Usecase היינו צריכים לעבור שוב ולפשט אותו מספר פעמים כדי שנוכל להשתמש בו בהמשך.

אנו השתמשנו במתודולוגית "מפל מים" כאשר שלב בפיתוח יתבצע רק אחרי סיום שלב קודם לו.

לדוגמה בשלב העיצוב מידולנו את ה Activity "הוספת לקוח למערכת" במהלך העבודה התגלו בפנינו צורך בשינויים בדיאגרמת ה- Use-Case. היינו יכולים למנוע שינויים אלו אם היינו משתמשים בצורה מתודולוגית אחרת למשל מתודולוגיה "איטרטיבית" , הרואה בפיתוח התוכנה תהליך מחזורי או ספירלי, שבו התוכנה נבנית בהדרגה. אפשר היה לעלות על הטעויות לפני שקיבלנו החלטה כלומר באיטרציה כשלהי הינו מתרכזים בתרחיש בודד והינו ממדלים עליו Use-case ו- Class.