

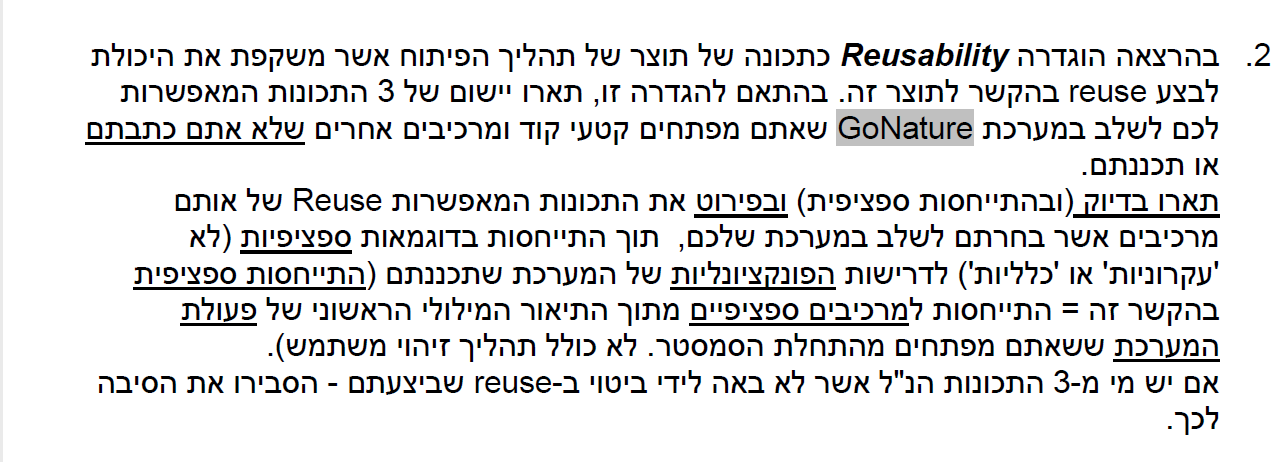
1. התהליך שביצענו עבור המרכיב : "ניהול ביקור לא מתוכנן (מזדמן)" הוא תהליך שמחולק לשני חלקים:  
מצד אחד אין אפשרות לממש אותו במערכת או להראות אותו באמצעות הדיאגרמות, ומצד שני אנו יכולים לבצע תהליך "אישור כניסה לפארק" ע"י עובד בפארק כאשר מטייל מגיע לפארק ומבקש להיכנס ללא הזמנה מראש .

השאלות והפרטים שהתייחסנו אליהם:

* מי יפעיל את המערכת בסיטואציה זו
* אילו פרטים ילקחו מהמטייל וירשמו במערכת.
* מה המערכת צריכה לבצע כאשר מגיע מטייל מזדמן (לדוגמא בדיקת זמינות בפארק).

במודל הuseCase התייחסנו למטייל מזדמן כ- preOrederTraveler כאשר הוא מגיע ומבצע הזמנה חדשה.   
הוא אינו שונה משאר השחקנים במודל, אך כאשר נבצע את ההזמנה למטייל מזדמן במודל ה- activity, נדרש לבצע זאת ע"י גורם חיצוני שהינו עובד בפארק שהוא מבצע עבורו את ההזמנה.  
ביצוע ההזמנה יהיה יותר קצר מתהליך הזמנת ביקור מראש וידרש מהמערכת לבצע בדיקת זמינות תפוסה בפארק. (המערכת תבצע פחות פעולות).  
במודל זה נמדל את המטייל המזדמן אחרת מכיוון שנתיב ההזמנה שונה משאר המטיילים שמבצעים הזמנה מראש. ביצוע המידול בactivity של סוג מטייל זה יבוצע ב-partition ששייך לעובד בפארק ודרכו יתחיל המסלול של ביצוע ההזמנה למטייל מזדמן.

לקראת מימוש התוכנה נבנה מסך עבור העובד בפארק שאחד מתפקידיו יהיו לבצע הזמנות עבור מטיילים מזדמנים ותהייה אפשרות לבצע בדיקה מהירה לזמינות בפארק ובכך ליידע את המטייל המזדמן אם יכול להיכנס לפארק או לא.



תיאור ישום 3 תכונות של REUSE – איך נשלב בgonature קוד שלא אנחננו כתבנו

->לתאר בדיקו ובפירוט את התכונות של המרכיבים שבחרנו לשלב (הקוד מהאינטנרט)

->דוגמאות ספציפיות אל מול הדרישות הפונקציואונליות -> התייחסותלמרכיכיבים ספציפיים מהסיפור (ללא זיהוי משתמש)

->אם יש תכונה של REUSE שלא באה לידי ביטוי להסביר למה

התכונות שהוגדרו בהרצאה עבור REUSABILITY הן : זמינות , גמישות והבנה .  
נתאר כיצד שלושת התכונות הללו מיושמות במערכת שלנו ומאפשרות שילוב של קוד מוכן :  
מערכת GONATURE אמורה לפעול בארכיטקטורת שרת - לקוח, על מנת לאפשר ממשק משתמש נוח ושמירת נתונים בצד השרת . מכיוון שמדובר בארכיטקטורה נפוצה וגנרית ניתן להשתמש ב framework קיים ולבצע reuse במקום לפתח קוד מחדש.  
לשם כך אנו זקוקים למחלקות שרת לקוח ותקשורת ביניהן. בחלק הזה במערכת אנו מבצעים REUSE של קוד OCSF שנחשפנו אליו בתרגול. הקוד המוכן עונה לנו על הצרכים של מימוש הארכיטקטורה בכך שהוא זמין , מתועד היטב , קל לשינוי והתאמה של פוקציות למערכת והצרכים שלנו . הקוד קל ופשוט להבנה , הוא מכיל את כל הפונקציונאליות המרכזית של ארכיטקטורת שרת -לקוח ולכן כדאי להשתמש בו .

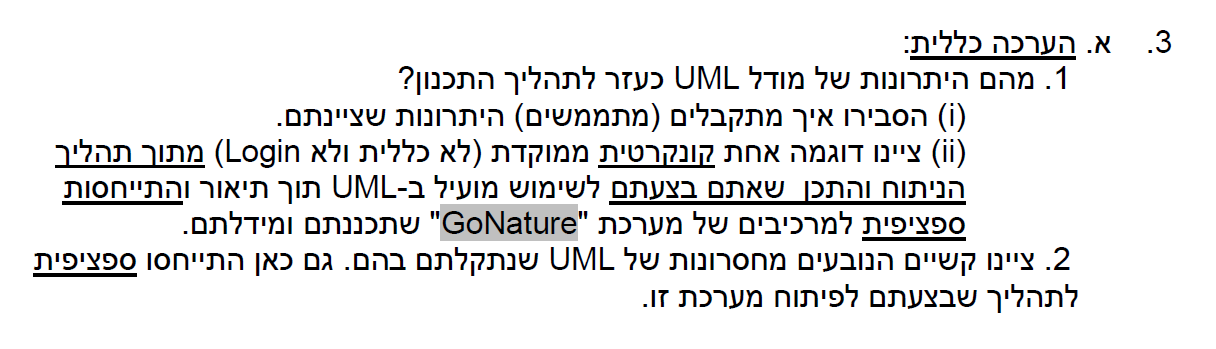
נתאר כיצד זה בא לידי ביטוי אל מול הדרישות הפונקציאונליות: ניקח כדוגמה ביצוע הזמנה במערכת .   
כאן באה לידי ביטוי ארכיטקטורה של שרת לקוח. הלקוח בצד שלו רואה את ממשק המשתמש – מכניס פרטים לצורך ביצוע הזמנה :האתר שבו יבקר , זמן ביקור ,מספר מבקרים וכתובת מייל . לאחר מכן בעת לחיצה הפרטים מועברים לבדיקה אל מול הDB. מסד הנתונים יושב בצד השרת ובצד הלקוח יכולים להיות לנו מספר לקוחות שמתחברים ממקומות שונים (ריבוי של לקוחות ) כלומר השרת והלקוח לא יושבים באותו מחשב. צריך שתהיה מחלקה כלשהי מקשרת בין השרת ללקוח שדרכה יועברו הנתונים אל השרת . בכך אנו גם שומרים על הפרדה בין השרת ללקוח.  
באמצעות פונקציית ה handlemessagefromclient שנמצאת ב ECHOSERVER אנו מבצעים קריאה של הנתונים ומעבירים למחלקה המטפלת בחיבור ושליפת נתונים מהDB. נבצע בדיקה אל מול תפוסת המבקרים בDB ולפיכך נאשר או לא את הביקור בפארק. באמצעות פונקציית handlemessagefromserver שנמצאת ב- ChaeClient נעביר את המידע מהשרת לבדיקה ונחזיר תשובה ללקוח האם ההזמנה אושרה או לא.

ה REUSE בא לידי ביטוי בכך שפוקציות handlemessagefromserver , handlemessagefromclient ניתנות לשינוי בהתאם להודעות שנרצה להעביר .

השינוי שנבצע יהיה בפונקציות אלו ופונקציות נוספות שנוסיף בהתאם למערכת שלנו, אך מעבר לכך כל התשתית הבסיסית של הארכיטקטורה תהייה מובנת על ה- OCSF שהיא מוכנה וזמינה לשימוש שלנו.

ניתן להגיד שהמחלקות ב OCSF הן אדפטיביות.

לסיכום , כל שלושת התכונות של REUSE באות לידי ביטוי ב בשימוש החוזר שביצענו בארכיקטורה של שרת – לקוח.



1.יתרונות ה UML –> איך מתממשים היתרונות שציינו

תיאור דוגמה אחת ספציפית מתוך תהליך הניתוח שבצענו לשימוש מועיל בUML ->התייחסות ספציפית למרכיבים שמידלנו בגו נייטור

2. קשיים נובעים מחסרונות UML -> התייחסות ספציפית לתהליך

1. נמנה את היתרונות של ה UML : דיאגרמות פשוטות להבנה, שפה אחידה – סטנדרט יחיד ליצירת אפיוני מערכת, מעבר קל מדיאגרמה למימוש , התאמה לדרך החשיבה של המשתמש.
2. היתרונות מתקבלים מכך שהUML מאפשר הסתכלות מכמה זוויות ראייה שונות על אותה מערכת שנרצה לפתח, מאפרת "להתסכל במבט מלמעלה " על התמונה הכוללת של המערכת , ממחישה את המבנה של המערכת והקשרים המשותפים. ניתן לראות ויזואלית וזה קל יותר להבנה מתיעוד במסמכים.
3. נתייחס לרכיב במערכת של הפקת דו"ח.

כאשר ביצענו ניתוח וקריאה ראשונית לרכיב הדו"חות לא היה לגמרי ברור מהם המעברים בין השלבים השונים בתהליך הפקת הדו''ח. בעזרת יתרונות הUML שאחד מהם הוא הבנה של התהליכים מהדיאגרמות תרמו לנו בכך -מדיאגרמת המחלקות הבנו אילו ישויות נצטרך על מנת לשמור את המידע , באיזה controller זה יתבצע , הקשרים שיהיו לנו במימוש בין המחלקות והמידע שנצטרך על מנת בעצם להפיק את הדו"ח.

תיאור המעבר בין התהליכים בדיאגרמת ה activity תורמים למימוש הזרימה והמעבר בין השלבים השונים , בין המחלקות השונות , כיצד המידע מועבר ומהם השלבים השונים והתנאים השונים שצריך לבדוק בכל שלב ושלב.

בדיאגרמת ה sequence הצגנו את הפוקציות הספציפיות במערכת ותיארנו את הזרימה והמעבר בין המחלקות והפונקציות כולל המידע שיוחזר , מרגע כניסת משתמש (מנהל הפארק) לחלון המתאים במערכת ועד להחזרת המידע דרך מעבר בשרת ,בDB וחזרה למסך בצד הלקוח. מתאר את כל השלבים בדרך ואיך זה בדיוק נראה בקוד בפירוט.

1. קשיים שנבעו מחסרונות הUML שנתקלנו בהם :

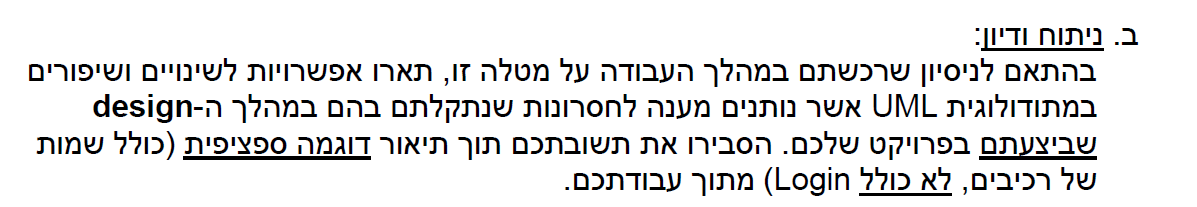
* ריבוי דיאגרמות :

\* נדרשנו לבצע package וזוהי דיאגרמה שהולכת "יד-ביד" עם דיאגרמת הclass כך ששילובן היה אמור להיות טבעי אך זה לא היה קל לשלב ולהבין את החלוקה לקטגוריות.

\* לא מכל דיאגרמה ברורה הזרימה הנכונה של התהליכים היות וכל דיאגרמה היא מאוד ספציפית למטרתה. כגון הclass מתאר רק את מבנה מערכת התוכנה אך לא את רצף האירועים במערכת כמו activity.

* התעסקות רבה בעיצוב : עיצוב הדיאגרמות והבאתן למצב אסתטי וקריא לקח הרבה מאוד זמן יקר שהיה יכול להיות מושקע בקוד יעיל ונקי יותר.

בדיאגרמת הCLASS היה עלינו לבצע PACKAGE ולשם כך לבחור קטגוריות מסויומות ולשייך אליהם את המחלקות שיצרנו בCLASS . האיחוד בין 2 המודלים היה מאתגר בבחירת הקטגוריות עצמן וההשמה של המחלקות שלנו בהם.   
ההתעסקות הרבה בעיצוב התהליך מקשה על אפיון רצף הפעילויות של הסיפור.לדוגמא בactivity היה עלינו לפרט בפרטי פרטים את הזרימה של האירועים וזה גרם לנו לחשוב איך בכל פעם אפשר להרחיב יותר את זרימת האירועים במקום לשים דגש על תהליכים מרכזיים. זה יכול להיות מבלבל מכיוון שלא ניתן למדל כל תרחיש אפשרי במערכת. בנוסף בדיאגרמה תהליך מסויים ממודל בדרך מסויימת וקשה להתאים שינויים מדרישות לקוח(כמו ההודעות בפורום).  
עלינו להתעסק בתכנון המערכת עצמה לפני חשיבה על תהליך המימוש עצמו.(להשאיר או להוריד?).



ב. ריבוי דיאגרמות : הוספת אפשרות של חיבור הקלאס והactivity, כך שאם מסמנים את הclass של הuser וההזמנה ,ניתן יהיה לראות את התהליך המתבצע במערכת בקשר בין שניהם והמעבר יהיה קבל ללא פתיחה של לשוניות מרובות ומסובכות בvisual paradigm.

התעסקות רבה בעיצוב : הוספת אופציה של עיצוב וסידור אוטומטי של התוכנה כך שיהיה קל יותר להזיז רכיבים עם הקווים המחוברים אליהם מבלי לשנות אחד אחד.