**תאריך הגשה: 04\05\2021**

**מטלה 2 – שאלות פתוחות – צוות 6**

**חברי הצוות:**

|  |  |  |
| --- | --- | --- |
| **מייל** | **תעודת זהות** | **שם מלא** |
| shaninosh314@gmail.com | 314760711 | שני פרויק |
| ayalachn@gmail.com | 312142946 | איילה כהן |
| yadika14@gmail.com | 319118733 | ירדן אדיקה |
| bashar.ali.121@gmail.com | 209063015 | בשאר עלי |
| samrr93@gmail.com | 204381388 | סאמר ארקאב |
| Alonamel13@gmail.com | 319540985 | אלונה מלניצ'נקו |

**שאלה 1:**

תארו את תהליך הניתוח/תכן ראשוני שביצעתם למרכיב: ביצוע בחינות (ממוחשב). קוים מנחים לתשובה : פרטו מה הם השאלות/הפרטים שהתייחסתם אליהם בתהליך הניתוח והחשיבה, וכן התייחסו לקשרים ולמעברים

1. ממודל Use-case למודל תהליכי המיוצג בעזרת Activity Diagram.
2. ממודל התהליכים הדינמיים לקראת מימושו בתכנה.

**תשובה:**

במהלך תהליך הניתוח שביצענו למרכיב "ביצוע בחינות ממוחשב", התייחסנו לשאלות הבאות:

1. מיהם השחקנים במרכיב זה
2. אילו פעולות מבצעים השחקנים
3. מהם האירועים המרכזיים אשר מרכיבים מרכיב זה
4. מהו רצף האירועים האלו
5. באילו תנאים יתרחשו האירועים האלו

בכדי לענות על שאלות אלו, בנינו דיאגרמות UML.

תהליך הניתוח ממודל Use-Case למודל Activity

דיאגרמת ה Use-Case שיצרנו עזרה לנו לענות על שאלות 1, 2 ו-3. כלומר, בעזרת הדיאגרמה יכולנו לראות בבירור מיהם השחקנים והאירועים המרכזיים המרכיבים את המרכיב "ביצוע בחינות". ואלו שחקנים יכולים לבצע כל אחת מהפונקציונליות שנמצאות במערכת.

במימוש דיאגרמת Activity, נעזרנו ב- Flow of events שנמצא בדיאגרמת ה-Use-Case בכדי לתאר את רצף האירועים.

בחלק הראשון של הפרויקט בנינו דיאגרמת Use-Case לתיאור הראשוני של המערכת, כחלק מהדיאגרמה הבענו את האירוע "ביצוע בחינות" כבועה שבתוכה רשמנו precondition ו flow of event. דיאגרמה זו לא מספיקה למידול הסופי של המערכת ולכן בשלב השני של הפרויקט יצרנו עוד דיאגרמות לכלל התהליכים ובפרט לאירוע "ביצוע בחינה ממוחשבת". באמצעות כך ענינו על שאלות 4 ו-5 –

באמצעות המידול הסברנו על תהליך וסדר הפעולות של האירועים השונים ופירטנו את רצף האירועים והתנאים עבור הפעולות הבאות:

* הנבחן מזין את קוד הבחינה
* הנבחן מזין את מספר תעודת הזהות שלו
* המערכת מתחילה למדוד זמן
* הנבחן יכול לסמן תשובות
* הנבחן "מגיש" את הבחינה
* המערכת סוגרת אוטומטית את הבחינה בסיום הזמן המוקצה או לאחר ההגשה, המוקדם מבניהם
* המערכת נועלת אוטומטית את הבחינה כאשר כל הנבחנים שהתחילו את הבחינה הגישו
* המורה יכולה לנעול את הבחינה בכל זמן
* לאחר אישור המנהלת, המורה יכולה לשנות את הזמן המוקצה לבחינה במסירת נימוק
* המערכת מבצעת עיבוד מידע למבחן באופן אוטומטי ושמירתו במסד הנתונים לצורך ניתוח עתידי
  + המערכת בודקת ונותנת ציון לבחינה
  + המערכת מבצעת אימות אותנטיות של הבחינות השונות על מנת לזהות העתקות
  + המערכת מייצרת דוחות סטטיסטים מגוונים

בכדי לתאר את הפעולות של המרכיב "ביצוע בחינה ממחושבת" בדיאגרמת ה-Activity, נעזרנו בסקיצות שיצרנו.

בעזרת "צומת פעולה" בדיאגרמת ה-Activity, מציגים כל שלב של "ביצוע בחינות ממוחשב".

בעזרת "צומת החלטה" מציגים את התנאים. ובעזרת חצים, מביעים את הקשרים בין הפעולות והתנאים.

בצורה כזאת, ניתן לראות בבירור את הסדר הכרונולוגי של האירועים.

לסיכום, כאשר עוברים מדיאגרמת Use-case לדיאגרמת Activity, ניתן לראות בצורה ברורה ועמוקה יותר את התיאור של מהלך האירועים, מה שעוזר בהבנת אירועים אלו, מימושם והבחנה מיידית בתהליכים שמתבצעים בו זמנית. בנוסף, דיאגרמת ה-Activity מציגה את המחלקות השונות במערכת אשר מעורבות באירוע באמצעותSwim Lane.

מודל התהליכים הדינמיים לקראת מימושו בתכנה

את המעבר ממודל התהליכים לקראת מימושו בתוכנית נבטא בכך שהתהליך מתחיל כשהתלמידים מקבלים את הבחינה דרך המערכת ומזינים את קוד הבחינה (שנמסר להם בעל פה מהמורה). התלמידים ממלאים תעודת זהות, ומרגע זה, המערכת מתחילה למדוד זמנים והתלמידים יכולים להתחיל לפתור את הבחינה ולסמן את תשובותיהם. בסיום הזמן המוקצה או כאשר כל הנבחנים שהתחילו את הבחינה הגישו, המערכת סוגרת אוטומטית את הבחינה.

במהלך הבחינה, המורה יכולה לנעול בכל זמן את הבחינה ולשנות את זמן הבחינה שהוגדר מראש בתנאי שמסרה נימוקים למנהלת והמנהלת אישרה.

המימוש של האירוע בתוכנה יבוא לידי ביטוי בכך שהשחקן שמתחיל את האירוע הוא התלמיד – שניגש למסך המתאים ומקיש קוד ביצוע בחינה המורכב מ-4 שדות (ספרות ואותיות). דרך המסך הזה, קוד הבחינה נקלט במערכת לתוך מחלקת *Student* באמצעות מתודה *enterCode* .המתודה *authenticateCode* בודקת האם הקוד קיים ותקין (מה לרשום אם הקוד לא תקין)

והמערכת פותחת מסך בו הנבחן מקיש את תעודת הזהות שלו. תעודת הזהות נקלטת במערכת לתוך מחלקת *OnlineExamFrom* באמצעות מתודה *enterID*. המתודה *authenticateStudentID* בודקת האם תעודת הזהות קיימת ותקינה (מה לרשום אם התז לא תקין) ואז, המערכת מתחילה למדוד זמן באמצעות המתודה *timer* שמריצה חוט אשר יתעורר כאשר יקבל תעודת זהות תקינה . הנבחן מקבל את הבחינה למחשב שלו מתוך מחלקת *StudentController באמצעות מתודה getExam* ומתחיל לפתור את הבחינה ולסמן את תשובותיו באמצעות המתודה *chooseAns*.

במהלך הבחינה :

באמצעות צומתי החלטה, נראה שהמורה יכולה לנעול את הבחינה בכל זמן באמצעות מתודה *LockExamButton*

ולשנות את משך זמן הבחינה שהוגדר מראש באמצעות המתודה *ChangeExamTimeButton*.

שתי מתודות אלה, מגיעות מתוך המחלקה *EditExamForm*.

סיום בחינה :

כאשר הנבחן מסיים לפתור את הבחינה, הוא מגיש את הבחינה באמצעות מתודה *submitExam*.

המערכת סוגרת אוטומטית את הבחינה באמצעות מתודה *closeExam* כאשר:

* כל הנבחנים שהתחילו את הבחינה הגישו
* עבר הזמן שהוקצה מראש לבחינה

כל בחינה שבוצעה, מתועדת במחלקת *DoneExam* אשר שומרת את המידע הבא :

* *Date* – תאריך ביצוע הבחינה
* *executionTime* – זמן ביצוע הבחינה
* *actualDuration* – משך הזמן המוקצה בפועל לביצוע הבחינה
* *numStudentsStarted* – מספר התלמידים שהתחילו את הבחינה
* *numStudentsFinishedSuccessful* – מספר התלמידים שסיימו בעצמם את הבחינה
* *numStudentFinishedUnsuccessful* – תלמידים שלא הספיקו לסיים את הבחינה

**שאלה 2:**

בהרצאה הוגדרה ***Reusability*** כתכונה של תוצר של תהליך הפיתוח אשר משקפת את היכולת לבצע reuse בהקשר לתוצר זה. בהתאם להגדרה זו, תארו בדיוק (ובהתייחסות ספציפית) ובפירוט איך באות לידי ביטוי 3 הדרישות ליישום מוצלח של Reusability בהקשר של אותם מרכיבים שלא אתם כתבתם או תכננתם ובחרתם לשלב במערכת שלכם באמצעות Reuse, תוך התייחסות בדוגמאות ספציפיות (לא 'עקרוניות' או 'כלליות') לדרישות הפונקציונליות של המערכת שתכננתם (התייחסות ספציפית בהקשר זה = התייחסות למרכיבים ספציפיים קונקרטיים (לא גנריים) מתוך התיאור המילולי הראשוני של פעולת המערכת שאתם מפתחים מהתחלת הסמסטר. לא כולל תהליך Login או זיהוי משתמש).

אם יש מי מ-3 הדרישות הנ"ל אשר לא באה לידי ביטוי ב-reuse שביצעתם – הסבירו את הסיבה לכך.

**תשובה:**

משמעות המילה Reuse היא שימוש חוזר. הכוונה לשימוש חוזר שמאפשר לנו לשלב קטעי קוד ועוד מרכיבים שלא אנחנו תכננו. ה-Reuse הוא כלי שימושי ביותר ומקל עלינו משמעותית כמתכנתים.

כאשר מימשנו את האבטיפוס, השתמשנו בתכונת . Reusability זאת אומרת, העברנו את פרויקט ה-OCSF שהיא תבנית מוכרת של client-server שהיה כתוב עבורנו בתרגולים לאבטיפוס עצמו וזה עזר לנו בהכנת האבטיפוס.

ראשית, מרכיב זה מקיים תכונת Availability . מכיוון שהפרויקט זמין ונגיש עבורנו יחד עם המחלקות והמתודות שלו ובנוסף הייתה לנו אפשרות לקחת אותו ולהשתמש בו.

מרכיב זה מקיים תכונת Understandability . הקוד בכל המחלקות היה ברור. בנוסף, המבנה הקבוע עם המחלקות האבסטרקטיות והמתודות הקבועות שחייבות להיות חלק מהפרויקט מראות שיש פורמט קבוע כלשהו לפרויקט וזה מחזק את תכונת ה- Understandability שלו. משום שבכל פעם שנרצה להשתמש בפרויקט הזה, נדע מה המבנה שלו ומה "צורת העבודה" שלו מראש.

בנוסף, התקיימה גם תכונת Flexibility משום שהפרויקט גמיש וניתן לשינוי (לא כולל מהמחלקות האבסטרקטיות), ניתן ליצור מחלקות שיורשות מהמחלקות האבסטרקטיות של הפרויקט בהתאם לצרכי המפתח ולאפשר מימוש מתודות בצורה אחרת כמו handleMessageFromServer ו- handleMessageFromClient

* במערכת שלנו, השתמשנו ב-Reuse במחלקה \_\_\_\_\_\_\_\_\_\_
* אם נרצה לפרט לפי התכונות שמאפשרות שימוש חוזר אז נחלק את המימוש והשימוש עפ"י כל תכונה.
* שימוש חוזר בשרת ולקוח
* אחת התכונות, אשר לא באה לידי ביטוי היא תכונת הגמישות.

לדוגמה, למרות שקיימות מספר מחלקות המממשות " טופס רישום", היות ומבנה המחלקות הנ"ל לא תואם באופן מדויק את הצרכים שלנו בפרויקט, ועקב כך שבעקבות שימוש בהן, היינו צריכים לבצע התאמות לצרכינו ושימוש במחלקות אלו לא בהכרח היה חוסך לנו זמן כתיבה, העדפנו שלא להשתמש בהן, אלא לממש את המחלקה בעצמנו.

* תכונה נוספת שלא באה לידי ביטוי בתכן שלנו היא שהמערכת שלנו צריכה להיות מקושרת למערכת סליקה שאחראית על קבלת התשלום מהלקוח.

**האם 2 הבולטס רלוונטים ?**

**שאלה 3:**

1. הערכה כללית:
2. מהם היתרונות של מודל UML כעזר לתהליך התכנון ?
3. הסבירו איך מתקבלים (מתממשים) היתרונות שציינתם.

**תשובה:**

למודל UML קיימים יתרונות רבים המשמשים כעזר לתהליך התכנון:

* באמצעות המודל ניתן ליצור דיאגרמות ברורות הממחישות את המתרחש במערכת שאנו בונים וכך גם להציג בפשטות לבעלי עניין שונים את בניית המערכת
* באמצעות המודל מתבצעת חלוקה גרפית של המחלקות במערכת
* באמצעות המודל ניתן לצפות את תהליך הפיתוח. זה נותן לנו אינדיקציה איך להתכונן בהתאם
* באמצעות המודל ניתן לראות פעולות שחוזרות על עצמן ואז באמצעות Reuse ניתן לחסוך בכתיבת קוד
* דיאגרמת Use-case נותנת לנו להבין את כל האינטראקציות שיש בין השחקנים השונים וגם תיאור של צעדים אפשריים לביצוע האינטראקציה
* דיאגרמת Class נותנת לנו אפשרות להראות את כל הישויות הקיימות במערכת ואת הקשרים הקיימים בין ישויות אלו. בנוסף, המפתח יכול "להעביר" מחלקות לקוד עצמו ולחסוך את החשיבה כיצד לממש מחלקות אלו בקוק
* דיאגרמת Activity נותנת לנו אפשרות להראות זרימת תהליכים אשר מתבצעים ע"י המערכת ובנוסף, מתארת בצורה ויזואלית פעולות שמתרחשות במקביל
* דיאגרמת Sequence נותנת לנו אפשרות לדעת כבר לפני כתיבת הקוד מאילו מחלקות תתבצע קריאה למתודות ששייכות למחלקות אחרות, אילו מחלקות מעורבות בביצוע של פעולה מסוימת ולהראות רצף של אירועים (מי קורא לאיזה מתודה ובאיזה סדר) דבר זה עשוי לפשט את המימוש בקוד של התהליכים
* דיאגרמת Package נותנת לנו לבצע חלוקה של המערכת הגדולה לתתי מערכות

**האם מספיק מפורט ???????**

1. ציינו דוגמה אחת קונקרטית ממוקדת (לא כללית ולא Login) מתוך תהליך הניתוח והתכן שאתם בצעתם לשימוש מועיל ב- UML תוך תיאור והתייחסות ספציפית למרכיבים של מערכת “CEMS” שתכננתם ומידלתם.

**תשובה:**

**לחשוב על דוגמא...**

1. ציינו קשיים הנוסעים בחסרונות של UML שנתקלתם בהם. גם כאן התייחסו ספציפית לתהליך שבצעתם לפיתוח מערכת זו.

**תשובה:**

הקשיים הנוסעים בחסרונות של UML שנתקלנו בהם היו למשל, לאחר שסיימנו לבנות את דיאגרמת ה-Class, אכן ראינו בבירור את המחלקות השונות והקשרים הקיימים ביניהן, אך לא ראינו איך קשרים אלו באים לידי ביטוי באופן מעשי במערכת שלנו.

דיאגרמת ה-Class לא מאפשרת לנו לתאר קשר בין מתודות במחלקה אחת לשניה. כלומר, במידה ולא היינו נותנים שמות משמעותיים ודומים יחסית אחד לשני, לא היינו רואים בבירור שלמשל, מתודה *GetReport* שנמצאתבמחלקה *ReportController* יכולה להשתמש ב-Attributes מהמחלקה *StatisticData*. בנוסף, לא רואים האם קיימות איזשהן לולאות או משפטי תנאי כאלה או אחרים. למשל, במחלקה *EditExamForm* יש מתודה *AddQuestionToExam*, במחלקה, לא רואים שמתודה זו הולכת להיות בתוך לולאה.

על מנת שכן נוכל לראות את הקשרים האלו, נעזרנו בדיאגרמת Sequence . אך שימוש בדיאגרמות נוספות יכול ליצור עומס על המפתח.

קושי נוסף שנתקלנו בו, הוא שאין אפשרות למדל את ה-GUI. לכן, אין לנו עדיין מספיק מידע לגבי איך המערכת תוצג למשתמשים. למשל \_\_\_\_\_\_\_\_\_\_

על מנת להתגבר על קושי זה, השתמשנו בכלי Scene builder בו הרכבנו את החלונות שלנו עבור המשתמשים למטלה של האבטיפוס. בדרך זו, יצרנו תמונה הרבה יותר ברורה של מה עלינו לממש.

1. ניתוח ודיון:

בהתאם לניסיון שרכשתם במהלך העבודה על מטלה זו, תארו אפשרויות לשינויים ושיפורים במתודולוגית UML אשר נותנים מענה לחסרונות שנתקלתם בהם במהלך ה-design שביצעתם בפרויקט שלכם. הסבירו את תשובתכם תוך תיאור דוגמה ספציפית (כולל שמות של רכיבים, לא כולל Login) מתוך עבודתכם.

**תשובה:**

בהתאם לניסיון שרכשנו במהלך העבודה על מטלה מספר 2, נתאר מספר אפשרויות לשינויים ושיפורים במתודולוגית UML :

אחד הרעיונות שלנו לשיפור מגיע בהשראת התכנה Eclipse . כפי שב-Eclipse יש אפשרות לעמוד עם העכבר על מילה מסוימת והתכנה מקפיצה רעיונות, כך גם עבור ה-Class. למשל, שתהיה אפשרות לעמוד עם העכבר על מתודה מסוימת בקלאס מסוים ויקפוץ חלון שמציג שלל פרטים הנוגעים למתודה זו. למשל, אילו מחלקות נוספות קשורות למתודה זו ועוד מיני חלון שמציין האם קיימות לולאות כלשהן או תנאים מקדימים. רעיון זה יכול בנוסף לייעל את המפתח ברגע שהמפתח יממש מחלקות.

בנוסף, כשנרצה לבנות דיאגרמת Sequence/Activity , נכתוב שם מחלקה רלוונטית לנו ויקפוץ חלון שמראה מחלקות, מתודות ו-Attributes קשורים מתוך ה-Class ונותן לנו כל מיני הצעות.

רעיון נוסף, הוא לנסות לבנות כל פעם קטעים קטנים מתוך הסיפור ועבור כל קטע קטן כזה, לבנות גרסה ראשונית של Class ומיני Activity ו- Sequence , היינו מציינים את הקשרים ובסוף מחברים את הכל ביחד באמצעות התכנה.

**יש עוד רעיונות ???**