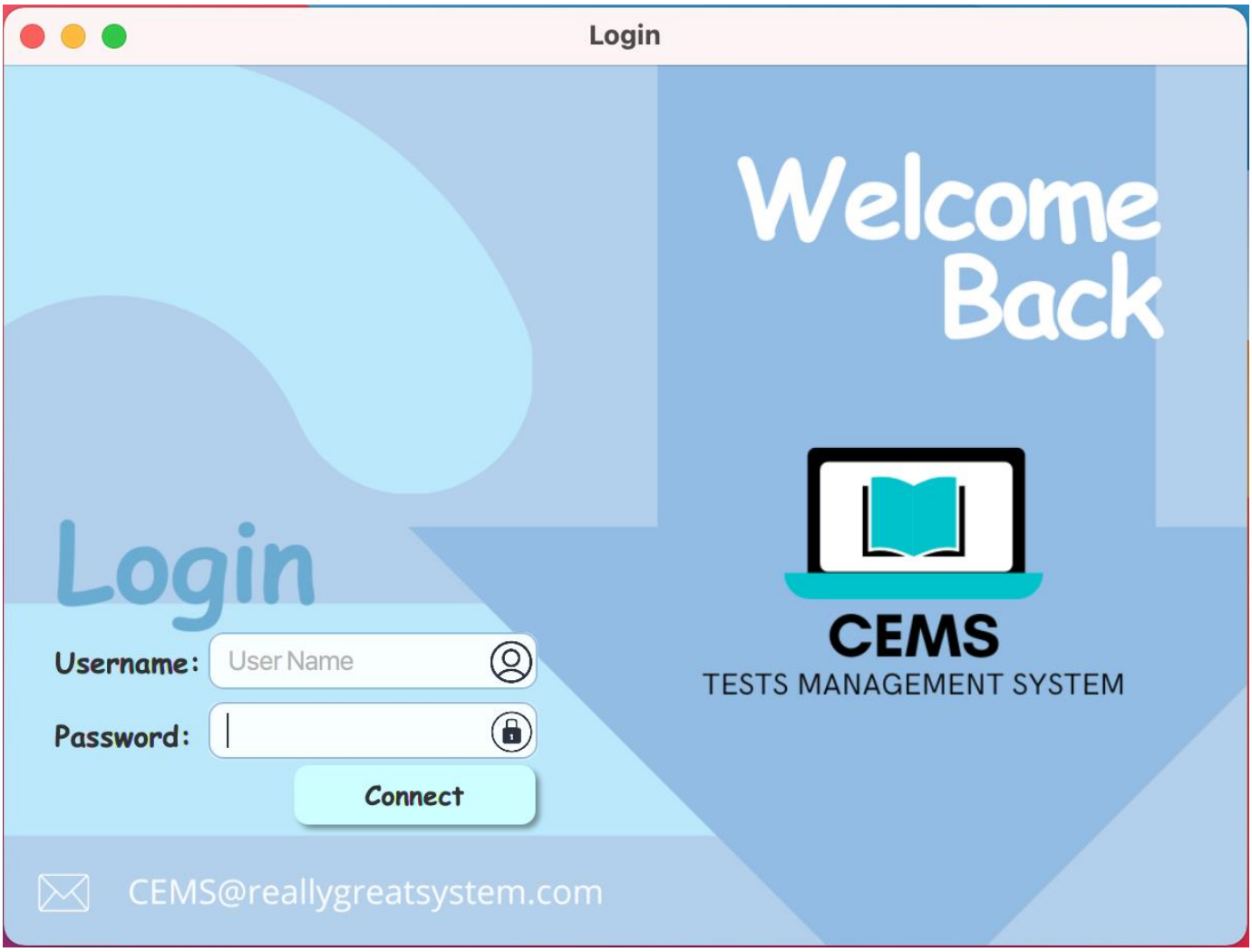
תשובות לשאלות פתוחות
מטלה 3 – קבוצה 10
תאריך הגשה: 20.06.2023

email	תעודת זהות	שם
mor.shmuel@e.braude.ac.il	315040980	מור שמואל
dor.david.shabat@e.braude.ac.il	316575620	דור שבת
lior.zucker@e.braude.ac.il	316375187	ליאור צוקר
yuval.rozner@e.braude.ac.il	207756552	יובל רוזנר



שאלה 1:

תארו את תהליך התכן (design) שביצעתם עבור המערכת " CEMS " באופן הבא :

(א) - תיאור תהליך התכן עבור היכולת :יצירת מבחנים .
- תארו דילמות הנדסיות ספציפיות שעסקתם בהן בתכן של יכולת זו ואופן פתרונן.
לצורך זה השתמשו בפורמט של תיאור Design issue כפי שמופיע בהרצאה על Design.
תארו את הדילמות, השיקולים וקבלת ההחלטות שלכם והסבירו את הפתרונות שבחרתם .

- מטרה:
מסמך זה מתאר את יכולת המערכת ליצור מבחנים, מתן מענה לדרישות הקשורות ליצירת תוכן וארגון של מבחן.
על מנת לממש זאת בנינו מספר טבלאות עם מידע רלוונטי למבחן שיוצגו למשתמש לפי בחירתו :

• טבלת Test המתארת את כל המבחנים שקיימים במערכת.
• טבלת Test To Execute המתארת את המבחנים שהוצאו לפועל, מכילה בפנים מידע אודות כמות הסטודנטים שניגשו למבחן, כמות הסטודנטים שסיימו את המבחן וכמות הסטודנטים שלא הספיקו לסיים מבחן בזמן, בנוסף, התפלגות כלל הציונים של אותו המבחן.
• טבלת Student Test המתארת את כל המבחנים שסטודנט מסוים ביצע, מכילה בין היתר את תשובותיו ואת הציון שקיבל.
• טבלת Question המתארת את השאלות, מכילה את כל מבנה השאלה ומי כתב אותה.
• טבלת Question Course המתארת לאיזה קורסים השאלה שייכת, זאת מכיוון ששאלה יכולה להתאים לכמה קורסים ולכל קורס יש הרבה שאלות.
• טבלת User המתארת את כל המשתמשים שלנו, בסיווג לפי permission(מרצה, סטודנט או ראש מחלקה).

- סדרי עדיפויות כלליים:
תהליך התכנון מעניק עדיפות להיבטים הבאים -
1. ממשק ידידותי למרצים לבניית בנקי שאלות ומבחנים.
2. ניהול יעיל של שאלות ומבחנים.
3. זיהוי מדויק ושיוך של שאלות עם נושאים וקורסים.
4. גמישות לכלול מידע נוסף במבחנים, כגון הוראות והערות.

- קו התכנון:
התכנון כולל שני מרכיבים מרכזיים: הכנת שאלות ובניית מבחן.
מרצים יכולים ליצור בנק שאלות על ידי בניית שאלה, הנחיות לתשובה וארבע אפשרויות תשובה. תשובה אחת מסומנת כנכונה. כל שאלה מזוהה באופן ייחודי באמצעות מספר בן 5 ספרות ומשויכת לנושאים ולקורסים ספציפיים.
לאחר מכן, מרצים יכולים להשתמש בשאלות אלו כדי לבנות מבחנים, המזוהים באופן ייחודי באמצעות מספר בן 6 ספרות. למבחנים יש משכי זמן מוגדרים מראש, ערכי נקודות לכל שאלה וקטעי טקסט חופשי אופציונליים להנחיות/הערות של נבחן ומרצה.

- בעיות תכנון עיקריות ופתרונן:
-ארגון שאלות ומבחנים:
הדילמה שעלתה: כיצד לארגן ביעילות שאלות ומבחנים?
פתרונות אלטרנטיביים שהועלו:
1. אחסון שאלות ומבחנים בטבלה בודדת.
2. ניצול מלא של מסד הנתונים ליצירת קשרים בין שאלות, נושאים וקורסים(יצירת טבלאות מקשרות).
הפתרון הנבחר: החלטנו ליישם מבנה מסד נתונים יעיל לארגון שאלות ומבחנים. פתרון זה אפשר לנו לאחסן, ולנהל את המידע הדרוש לנו באופן היעיל ביותר. על ידי שימוש במסד נתונים זה, קל יותר ליצור קשרים בין שאלות, נושאים וקורסים, מה שמאפשר גמישות עבור שיפורים עתידיים.

-זיהוי ושיוך שאלות:

הדילמה שעלתה: כיצד לזהות במדויק שאלות המשויכות לנושאים וקורסים?
פתרונות אלטרנטיביים שהועלו:
1. הכנסת נתונים ידנית, כלומר, נאפשר למרצים להזין באופן ידני מידע על נושא וקורס עבור כל שאלה.
2. אינטגרציה אוטומטית, כלומר, שילוב עם מסד הנתונים שבנינו כדי להביא את פרטי הנושא והקורס שאליו אנו מעוניינים לשייך את השאלה.
הפתרון הנבחר: בחרנו בפתרון האינטגרציה האוטומטית. באמצעות שילוב עם מסד נתונים המטפל במספור נושאים וקורסים, המערכת יכולה להביא ולשייך את המידע הרלוונטי לכל שאלה באופן אוטומטי. גישה זו מפחיתה מאמץ ידני, מבטיחה דיוק ומפשטת את התהליך עבור המרצים.

-הכללת מידע נוסף במבחנים:
הדילמה שעלתה: כיצד לספק הנחיות והערות בתוך מבחנים?
פתרונות אלטרנטיביים שהועלו:
1. חלקים ייעודיים, כלומר, יצירת חלקים נפרדים בתוך המבחן להנחיות והערות.
2. הערות מוטבעות, כלומר, נאפשר למרצים להוסיף הנחיות והערות ישירות לצד שאלות.
הפתרון הנבחר: לאחר הערכת החלופות, החלטנו לכלול קטעי טקסט חופשי ייעודיים במבחנים. פתרון זה מאפשר הפרדה ברורה בין הנחיות הנבחן (חלק מהמבחן) לבין הנחיות/הערות המרצה (אינן גלויות לנבחנים). הוא מספק גישה מובנית ומבטיח שמידע חשוב מועבר ביעילות לשני הצדדים.

- פרטים נוספים הקשורים לתכנון:
ממשק המשתמש עבור תכונת יצירת המבחן תוכנן תוך התמקדות בפשטות, אינטואיטיביות ויעילות. ההיבטים המרכזיים הבאים נלקחו בחשבון בתכנון:
ניווט וארגון ברור ופשוט:
ממשק המשתמש משתמש במבנה ניווט ברור ואינטואיטיבי, המאפשר למרצים לגשת בקלות לפונקציונליות של יצירת השאלה והמבחן. תפריט מאורגן היטב מבטיח שמשתמשים יכולים לעבור במהירות בין חלקים שונים של המערכת.
תצוגה מקדימה בזמן אמת:
ממשק יצירת המבחן כולל תכונת תצוגה מקדימה בזמן אמת, המאפשרת למרצים לראות כיצד המבחן ייראה לנבחנים בזמן שהם בונים אותו. זה מאפשר למרצים לסקור את ההוראות והפריסה הכוללת של המבחן בזמן אמת, מה שמבטיח את הבהירות שלו.

(ב) ציינו אילו מהעקרונות (*) שנלמדו בהרצאות הקורס בנושאים: Design ,Architecture, Reuse, ו Design patterns שימשו אתכם בתהליך התכן הכלל-מערכתי שביצעתם והסבירו באיזה אופן הבאתם אותם לידי ביטוי – באמצעות דוגמאות קונקרטיות ספציפיות מתוך המערכת " CEMS " שפיתחתם (לא לכלול תיאורים כלליים-גנריים כמו: JAVA packages, OCSF, timer - באופן כללי, וכו'. יש לפרט בהקשר קונקרטי מפורט של המערכת).
(*)לכל אחד מהעקרונות האלה, הסבירו מה הם היתרונות המתקבלים משימוש בהם.

- Reuse- שילבנו במערכת שלנו קטעי קוד השייכים ל-OCSF Framework, המכילה את הפונקציונליות בין השרת ללקוח, דבר הממומש את עקרון שימוש בקוד מוכן שלא אנחנו כתבנו אותו. בנוסף, יש לנו מחלקות כלליות שבהן אנו משתמשים בכל מקום בקוד. דוגמאות:
1. Abstract Controller Class: משמשת כמחלקה בסיסית עבור בקרים אחרים בחבילת ה-Client GUI. על ידי הפשטת פונקציונליות נפוצה ואספקת שיטות להפעלת הבקר, טיפול באירועים והצגת הודעות, הוא מקדם שימוש חוזר בקוד בין בקרים שונים. בקרים אחרים במערכת יכולים להרחיב את מחלקת AbstractController ולרשת את הפונקציונליות שלה, ולבטל את הצורך לשכפל קוד עבור פעולות נפוצות.
2. Msg Class: מייצגת אובייקט הודעה המשמש לתקשורת בין הלקוח לשרת. על ידי הקפצת רכיבי שאילתת מסד נתונים ומתן שיטות שירות לטיפול בחלקים שונים של ההודעה, הוא מאפשר שימוש חוזר בקוד בשכבת התקשורת של המערכת. מחלקת Msg יכולה לשמש רכיבים שונים שצריכים לשלוח או לקבל הודעות, מה שמבטיח קוד תקשורת עקבי וניתן לשימוש חוזר.
3. NotificationAlertsController class: מספקת שיטות להצגת סוגים שונים של התראות ב-GUI. על ידי ריכוז הלוגיקה הנדרשת להצגת התראות אזהרה, התראות אישור, התראות מידע והתראות שגיאה, הוא מקדם שימוש חוזר בקוד בכל האפליקציה. חלקים אחרים של המערכת

יכולים להשתמש ב-NotificationAlertsController כדי להציג הודעות התראה עקביות וסטנדרטיות מבלי לשכפל את קוד הטיפול בהתראה.

היתרון:
מחלקות אלו מדגימות שימוש חוזר בקוד על ידי מתן פונקציונליות ניתנת לשימוש חוזר והטמעת פעולות נפוצות בפרויקט. גישה זו משפרת את יכולת התחזוקה, מפחיתה כפילות קוד ומקדמת עקביות בחלקים שונים של המערכת.

- Architecture- השתמשנו ב- Layer pattern ובעקרונות ה-ECB:
Boundary Layer- שכבה זו מקיפה את חבילת ה-GUI, המכילה קבצי ממשק הן בצד הלקוח והן בצד השרת. שכבת הגבול משמשת כממשק בין המשתמש למערכת. הוא מטפל בקלט משתמש, מציג מידע ומתקשר עם מסד הנתונים והשרת. שכבת הגבול משויכת ל-Controller המתאים לה, המשמש בתוך קבצי FXML כדי להגדיר את ההתנהגות של רכיבי ממשק המשתמש.
Control Layer- בצד השרת, שכבת הבקרה מטפלת בלוגיקת השרת, כגון פעולות ניתוב ואינטראקציה עם מסד הנתונים ב- SQL. הוא מבטיח שהפעולות המתאימות מבוצעות על סמך המידע הנדרש. בצד הלקוח, שכבת Control אחראית על פעולות הממשק וניתוח המידע המוחלף בין הלקוח לשרת. הוא מנהל את זרימת הנתונים ומתאם את התקשורת בין שכבת הBoundary לשרת.
Entity Layer- שכבת הישויות מייצגת את ישויות המערכת, שהן הנתונים והאובייקטים הקבועים. כדי להבטיח מודולריות ושימוש חוזר, הפרויקט מאמץ פרויקט נפרד בשם Shared, שבו מאוחסנים בין היתר, הישויות והבקרים שלהם. גישה זו מקדמת אנקפסולציה ועקביות בייצוג של ישויות מערכת.

היתרון:
ה-ECB מפריד תחומי אחריות לישויות, בקרה וגבולות. עקרונות אלה תורמים לארכיטקטורת מערכת מובנית היטב, מודולרית וניתנת לתחזוקה.

- Design- בתהליך התכנון של הפרויקט, אימצנו **גישה איטרטיבית** לבניית המערכת. בתחילה התמקדנו בהקמת תשתית הבסיס ופיתחנו בהדרגה רכיבי מערכת בשלבים. תהליך איטרטיבי זה אפשר לנו להעמיק בדרישות הפרויקט ולחדד את ההבנה שלנו בכל שלב.
דוגמאות לפיתוח הדרגתי של המערכת-
1. Use Case Diagram: בנינו דיאגרמת Use Case כדי להבין את האינטראקציות של המערכת עם הסביבה ולהגדיר את דרישות המערכת. תרשים זה שימש בסיס להבנת התנהגות המערכת ולזיהוי גורמי מפתח והאינטראקציות שלהם עם המערכת.
2. Class Diagram: התבסס על דיאגרמת Use Case, יצרנו תרשים Class. תרשים זה סיפק סקירה כללית של מבנה המערכת, כולל היחסים בין בקרים, ישויות ומסכים. זה עזר לנו להבין את ארכיטקטורת המערכת והקל על היישום של פונקציות שונות.
3. Activity and Sequence Diagrams: דיאגרמות אלו אפשרו לנו לדמיין את רצף הפעולות ואת הזרימה ההגיונית של הפעולות. הם הדריכו אותנו בשלב ההטמעה, ודאגו שהמערכת תפעל בצורה חלקה ויעילה.
4. Package Diagram: דיאגרמה זו תיארה את החבילות והתלות שלהן בתוך המערכת, ונתנה לנו תובנה לגבי הארכיטקטורה הכוללת וסידור הרכיבים.
5. Database Design: תכננו והטמענו את מסד הנתונים לפרויקט, כולל כל הטבלאות הנדרשות והקשרים ביניהן.

היתרון:
על ידי שימוש בעקרונות וטכניקות אלו, השגנו גישה שיטתית לפיתוח הפרויקט. השימוש בדיאגרמות ופיתוח איטרטיבי אפשרו לנו להבהיר דרישות, לתכנן מערכת מובנית היטב ולהבטיח יישום ותפעול חלקים.

- Design Patterns-
1. Singleton Pattern: כדי להבטיח קיום Server אחד בלבד במערכת הפועלת, להבטיח קיום instance בודד של Client עבור כל לקוח.
2. Observer pattern: כדי להבטיח שהרכיבים יישארו מעודכנים בנתונים העדכניים ביותר. לדוגמה: ה-TableView פעל כ-Observer במקור הנתונים, כגון אוסף או טבלת מסד נתונים,

ורענן את התצוגה שלו בכל פעם שהתרחשו שינויים. באופן דומה, ה-BarChart רשם את עצמו כ-Observer למקור הנתונים הסטטיסטיים והתאים את הייצוג הגרפי שלו בכל פעם שהנתונים השתנו. על ידי יישום דפוס ה-Observer, השגנו עדכונים בזמן אמת והדמיה מדויקת של הנתונים בשני הרכיבים, תוך שיפור חווית המשתמש.

<u>היתרון:</u>
שימוש בתבנית Singleton מבטיח מופע יחיד, מקדם יעילות במשאבים ובקרה מרכזית. תבנית ה-Observer מאפשרת עדכונים בזמן אמת, שימוש חוזר ברכיבים לנתונים מסונכרנים וגמישות משופרת.

שאלה 2:

תארו את תהליך המימוש (implementation) שביצעתם עבור המערכת " CEMS " בהקשרים:

(א) בדיקות. תארו את תהליכי הבדיקות השונים שבצעתם במהלך פיתוח הפרויקט שלכם.

- ציינו את מאפייני תהליכי הבדיקות שביצעתם תוך התייחסות לעקרונות שנלמדו בהרצאות בנושאי בדיקות תוכנה, ותוך מתן דוגמאות ספציפיות (לא כללית/גנרית ולא על Login) שביצעתם (או לא ביצעתם) במהלך הפרויקט ע"י תיאור מפורט של בדיקות מרכיבים ספציפיים של מערכת " CEMS " ("ספציפי" כלומר: לא בהתייחסות כוללנית גנרית כמו: "שדות ריקים", "התחברות לשרת", "כתיבה ל-DB", "דוחות" וכו'. מה כן: יש לפרט מה בדיוק נבדק בדגש על תהליכים ספציפיים של המערכת המפותחת).

במהלך הפיתוח של מערכת 'CEMS', הטמענו תהליכי בדיקה שונים להבטחת אמינות ותקינות התוכנה. תהליכי בדיקה אלו תאמו את העקרונות שנלמדו בהרצאות בנושא בדיקות תוכנה. להלן הפרטים של תהליכי הבדיקה שביצענו:

1. <u>בדיקת רכיבים בודדים</u>: כל חבר צוות שפיתח מסכים ופונקציות שונות ערך בדיקות מקיפות על הרכיבים המתאימים לו. זה כלל בדיקת תקינות הקוד, אימות פונקציונליות ממשק המשתמש והבטחה שהרכיבים ביצעו את המשימות שלהם כנדרש. העיקרון של code review and peer testing יושם כדי לאמת את האיכות של רכיבים בודדים.
2. <u>White Box Testing</u>: השתמשנו בעקרונות White Box Testing כדי להבטיח את נכונות עיבוד הנתונים, החישובים וההחלטות הלוגיות בתוך המערכת. בדיקות אלו התמקדו בדיוק של תהליך התחלת מבחן(הכנסת קוד, ת.ז ובדיקה שהסטודנט שייך למבחן זה), כניסה למערכת(שלא יהיה ניתן להיכנס מאותו המשתמש פעמיים, ורק במידה והמשתמש קיים), יצירת שאלות ויצירת מבחן(ביצוע שמולאו כל השדות המחייבים ליצירה תקינה). על ידי בחינת הקלט והפלט של המערכת, וידאנו שהמערכת ביצעה את הפונקציות המיועדות לה כהלכה.
3. <u>Black Box Testing</u>: השתמשנו בעקרונות Black Box Testing כדי לבדוק את התנהגות המערכת על סמך קלט ופלט מבלי להתחשב ביישום הקוד הפנימי. מקרי קצה ותרחישים חריגים נבדקו כדי להבטיח שהמערכת טיפלה בהם כראוי. לדוגמה, בדקנו את יכולת המערכת לטפל בתהליך הוספת מבחן לאחר יצירתו כראוי למסד הנתונים, הוספת התפלגות ציונים למבחן שבוצעו כבר על ידי סטודנטים. בדיקות אלו נועדו להעריך את חוסנה וחוסן של המערכת במצבים שונים בעולם האמיתי.

על ידי שילוב תהליכי בדיקה אלו, הצלחנו לזהות ולטפל בפגמים בתוכנה, להבטיח את דיוק החישובים ועיבוד הנתונים, ולאמת את התנהגות המערכת בתרחישים שונים. השילוב של אלו עזר לנו לשפר את האיכות והאמינות הכוללת של המערכת.

(ב) אינטגרציה. תארו את תהליכי האינטגרציה שביצעתם במהלך תהליך הפיתוח.

- תארו את מהלך האינטגרציה – ציינו לוחות זמנים ופרוצדורה (תהליך) – באמצעות דוגמאות קונקרטיות ספציפיות (לא כלליות) מתוך המערכת " CEMS " שפיתחתם.
- ציינו באילו כלי תוכנה השתמשתם באינטגרציה והסבירו באיזה אופן.
- ציינו אילו מהעקרונות שנלמדו בהרצאה בהקשר של אינטגרציה יושמו בתהליך האינטגרציה שביצעתם, והסבירו באיזה אופן.

במהלך תהליך ההטמעה של מערכת "CEMS", נקטנו בגישה מובנית שהתמקדה באינטגרציה ובתיאום בין מודולים שונים. אימצנו את שיטת הפיתוח האיטרטיבי, המאפשרת לנו לבנות את המערכת בהדרגה ולהבטיח אינטגרציה חלקה בכל שלב.
מלכתחילה, ערכנו בדיקות יסודיות של כל מודול בנפרד. הזנו ערכים באופן ידני דרך ממשק המשתמש או ישירות למסד הנתונים כדי לאמת את הפונקציונליות והנכונות של כל מודול.
לדוגמה: כשעבדנו המבחן הממוחשב של סטודנט, בדקנו את הפונקציונליות על סמך נתונים זמניים מאחר שפיתוח השאלות טרם הושלם. זה עזר לנו להבטיח שהתהליך עובד נכון עם נתונים

זמניים.
לאחר מכן המשכנו בתהליך הפיתוח וביצענו הוספה של תוכן לכל מודול, תוך תיאום ישיר בין המודולים השונים בהתאם לתלויות שיש להם אחד בשני. בכל פעם שהוספנו פונקציונליות חדשה למערכת, ביצענו עליו אינטגרציה. **דוגמה לשימוש באינטגרציה רחבה שבוצעה במערכת שלנו:** ניתן לראות בין מודול של ביצוע מבחן ממבט הסטודנט לבין מודול של הפקת הדוחות ממבט המרצה. לאחר שסיימנו להוסיף את כל הפונקציונליות, חברי הצוות האחראים והמפתחים של חלקים אלו נפגשו לצורך בדיקות אינטגרציה על המערכת. ביצענו מספר רב של מבחנים תוך כדי תיעוד של מה שאנחנו מצפים לקבל בדו"ח וכמובן, במסד הנתונים. תחילה, מצאנו לא מעט בעיות הקשורות במידע הנלקח לדוחות ובחוסר מידע שנשמר בתהליך המבחן, דבר זה החזיר אותנו בחזרה לשלב הפיתוח במקביל לבדיקות אינטגרציה חוזרות בחלק זה.

<u>כלי התוכנה בהם השתמשנו בתהליך האינטגרציה :</u>
1. Eclipse – התוכנה בה רשמנו את כל הקוד של המערכת. בתוכנה זו פיתחנו את הלוגיקה של המערכת, וכמובן, את כל מסכי ה-GUI.
2. MySQL – התוכנה בה מימשנו את מסד הנתונים שלנו. מימשנו טבלאות שיכילו את מסד הנתונים שלנו בהתאם לסיפור. גם בזמן הפיתוח נאלצנו לשנות את מסד הנתונים, לעדכנו מחדש ולשתפו בין חברי הפרויקט.
3. Git – כלי שסייע לנו כמפתחים בניהול הקוד. בזכות כלי זה התאפשר לנו לבצע תיאום בין עבודת הצוות, לעקוב אחר השינויים בקבצי המערכת ולא פחות חשוב, לשחזר קבצים קודמים וקוד שכתבנו, מכל רגע במהלך הפיתוח.
4. Zoom – האינטגרציה התבצעה ע"י כל חברי הפרויקט יחדיו, תוך כדי תיאום ציפיות ופעולות במערכת. חברי הפרויקט נפגשו יחדיו ב-Zoom וביצעו את הבדיקות בצורה שיתופית.

<u>עקרון שיישמנו בתהליך האינטגרציה–</u>
1. <u>Ensure everyone's changes integrate</u> : רצינו להבטיח שכל השינויים האישיים שבוצעו על ידי חברי צוות שונים משולבים בצורה חלקה במערכת הכוללת. זה הצריך תיאום וסנכרון קבוע של שינויי קוד באמצעות שימוש ב-Git. על ידי מיזוג ושילוב קבוע של הקוד, צמצמנו את הסיכויים להתנגשויות וחוסר עקביות.
2. <u>Reduce merge conflicts</u> : התנגשויות מיזוג יכולות להתרחש כאשר מפתחים מרובים מבצעים שינויים באותו קוד בו זמנית. כדי למזער את הקונפליקטים הללו, עקבנו אחר שיטות פיתוח טובות כמו חלוקת העבודה למודולים או רכיבים נפרדים, הקצאת תחומי אחריות ספציפיים. גישה זו עזרה לנו להימנע מקונפליקטים והפכה את תהליך האינטגרציה לחלק יותר.
3. <u>Catch and fix bugs</u> : אינטגרציה היא שלב קריטי שבו באגים או בעיות עלולים להתעורר עקב אינטראקציה של רכיבים שונים. נתנו עדיפות לבדיקות יסודיות במהלך תהליך האינטגרציה כדי לזהות ולתקן באגים או חוסר עקביות.
4. <u>Maintain a stable executable system</u> : היה חיוני עבורנו שתהיה לנו מערכת יציבה ופונקציונלית בכל עת. כדי להשיג זאת, עקבנו אחר מודל הסנכרון והייצוב. סנכרנו את הקוד על ידי שילובו באופן קבוע. זה עזר לנו לזהות בעיות בשלב מוקדם ולשמור על בסיס קוד יציב.

על ידי יישום העקרונות של אינטגרציה רציפה יציבה ואימוץ מודל הסנכרון והייצוב, הצלחנו לנהל ביעילות את תהליך האינטגרציה, למזער קונפליקטים, לטפל בבאגים באופן מידי ולשמור על מערכת הפעלה יציבה ואמינה עבור הפרויקט.

שאלה 3:

תחקור והפקת לקחים: התייחסו לאופן שבו התנהלתם לגבי 2 מרכיבים של ביצוע הפרויקט:

(א) תיאום פעילויות ושיתוף בין חברי הצוות בפיתוח כולל גישה לניהול גרסאות:
תארו את השיטה שלפיה פעלתם בהקשרים אלה בשלב מימוש התוכנה, וציינו יתרונות וחסרונות שלה.
יש להתייחס גם לתהליך העבודה - לא להתמקד רק בכלים ואספקטים טכניים.

ביצוע הפרויקט מתחלק לשני חלקים –
בשלב הראשון תכנון המערכת ובשלב השני מימוש המערכת על בסיס התכנון.

- בחלק של התכנון, הבנו את דרישות המערכת ובנינו את הדיאגרמות לפי המימוש הטוב ביותר שראינו לנכון לפי סיפור המשתמש ומענה על כל דרישות הלקוח כך שהתוכנה תהיה יעילה, נוחה למשתמש, וקלה למימוש. ביצענו את התכנון ב-Visual Paradigm כאשר כל שני חברי צוות היו אחראים על ביצוע דיאגרמה. מכיוון שאפשר ליצור מclass diagram, template ראשוני, בניית הדיאגרמות כלל תהליך עמוק של חשיבה וירידה לפרטים הקטנים. מהלך העבודה התרחש בזום/פרונטלי, ובסיום כל דיאגרמה הצגנו אותה לשאר חברי הצוות על מנת לקבל פידבקים, ובהתאם לכך ביצענו תיקונים, בנוסף בשביל לעמוד בתאריכים ויעדים, השתמשנו בכלי ToDoList.
- בחלק של המימוש, כאשר מימשנו את הפרויקט ב-Eclipse, על מנת לבצע איחוד של הקוד שכל חבר בקבוצה כתב, השתמשנו בGit, אך השימוש הראשוני ב-Git גרם לנו לדריסת קוד אחד של השני.

לאחר מקרה זה הפקנו לקחים, גיבינו את הקוד בצורה לוקאלית לאחר כל יום עבודה ודאגנו שמקרה זה לא יחזור על עצמו. התחלקנו לבראנצ'ים כדי לא לפגוע בקוד של חבר צוות אחר ובמשך כל שבוע היינו מבצעים Merge לקוד שכתבנו, הגדרנו בצוות מישהו שיהיה אחראי לאחד ולסדר את כל השינויים שבוצעו, ולבסוף כל חברי הקבוצה האחרים היו מבצעים Pull לקוד על מנת להוריד את הגרסה האחרונה של הקוד לאחר האינטגרציה.

<u>יתרונות השימוש ב-Git:</u>
• אפשרות לשחזור גרסאות קודמות ולניהול גרסאות של הפרויקט.
• אפשרות למיזוג קוד של מספר חברי קבוצה.

<u>חסרונות השימוש ב-Git:</u>
• מצריך למידה עצמית של הכלי, והכרת האפשרויות הרלוונטיות לפרויקט שלנו על מנת למצות אותו.
• לאחר שביצענו Commit לקוד, חלק מהקוד לא עבד עקב דריסת קוד אחד של השני, בעקבות כך אחד מחברי הקבוצה נאלץ לוותר על כל הקוד שכתב ולבצע שכתוב ותיקונים של הקוד בשנית לאחר שחברי הקבוצה העלו גרסה חדשה.

(ב) ניהול הפרויקט בשלב הבניה (Construction) – קידוד, שילובי קוד (אינטגרציה מערכתית) ובדיקות.
ציינו באופן פרטני, בהתייחסות ספציפית לפיתוח המערכת " CEMS ", איך פעלתם בשלב זה של הפיתוח (כולל: תיאור התנהלות התהליך ההנדסי (לא דינמיקה ועבודת הצוות), בקרה: איך טיפלתם בבעיות טכניות הקשורות לתהליך הפיתוח, וכו').
אם היו קשיים מה הסיבה לכך? מה הייתם משנים בדיעבד בגישתכם למרכיב זה של תהליך הפיתוח מבחינת האספקטים הרלבנטיים של הנדסת תוכנה?

לאחר סיום חלק התכנון, ביצענו פגישת Milestone שבו נכחו כל חברי הקבוצה, ובה הוחלט על אופי העבודה של הפרויקט. החלטנו על דרך העבודה הנכונה למימוש קוד תוך כדי התבססות על שלבי התכנון, באופן הבא:

- מימוש הפרויקט: לפי המבנה שתוכנן – צד שרת, צד לקוח, בניית מסד הנתונים כאשר לכל מסך במערכת יש Controller שממנו נשלוט עליו.
- מהלך העבודה: בוצע בעיקר בזום, שם חילקנו בין כל חברי הצוות את העבודה לפי החוזקות של כל אחד ולפי עדיפויות אישיות של כל אחד ואחד מחברי הפרויקט. כל שבוע היינו נפגשים ובודקים באיזה מצב כל אחד על מנת להבין את ההתקדמות שלנו בפרויקט ביחס לתאריך ההגשה הסופי.
- אופן המימוש: בתחילת המימוש בנינו את בסיס המערכת – מסך הכניסה וכל החלקים הקשורים אליו. לאחר מכן, עברנו למימוש המסכים האחרים והחלטנו יחד על שני חברי צוות שיהיו אחראים על בניית כל מסכי המערכת ועיצובם על מנת לא ליצור חפיפה בשלב האיחוד של הקוד ב-Git, ועל מנת לשמור על קו אחיד של עיצוב. שאר חברי הצוות קיבלו את האחראיות לממש אבטיפוס של התקשורת והעברת הנתונים בין השרת ללקוח והפוך.

הקושי המרכזי שחווינו היה בתחילת העבודה עם ה-Git, בגלל שלמדנו את אופי השימוש בו תוך כדי למידה מקוונת, אבל כאשר הבנו את הכלי הנפלא הזה, העבודה המשיכה בצורה מאוד יעילה וקלה.
בדיעבד היינו לומדים כיצד לעבוד עם ה-Git לעומק, לפני שבכלל התחלנו לממש את המערכת, בכך היינו חוסכים המון טעויות ודריסות של הקוד, דבר שעיכב אותנו הרבה.