מספר קבוצה:	8			
חברי הקבוצה:	**שם מלא:** יובל הים	**ת.ז:** 314989484	**מייל:** Yuval.Hayam@e.braude.ac.il	
	שם מלא: נדב דרעי	**ת.ז:** 203327754	**מייל:** Nadav.dert@e.braude.ac.il	
	שם מלא: הדר אילוז	**ת.ז:** 312588601	**מייל:** Hadar.Iluz@e.braude.ac.il	
	שם מלא: מטר אסף	**ת.ז:** 209401439	**מייל:** Matar.Asaf@e.braude.ac.il	
	שם מלא: ידין אמסלם	**ת.ז:** 316325570	**מייל:** Yadin.Amsalem@e.braude.ac.il	
	שם מלא: ליאור קריש	**ת.ז:** 318267242	**מייל:** Lior.Karish@e.braude.ac.il	
תאריך הגשה סופי:	14.6.2021			

שאלה 1

תארו את תהליך התכן (design) שביצעתם עבור המערכת " CEMS " באופן הבא:
א. עבור יכולות <u>בניית בחינות</u> – כולל הכנת שאלות והכנת בחינות.
תארו דילמות הנדסיות ספציפיות שעסקתם בהן בתכן של יכולות אלה.
השתמשו בפורמט של תיאור issue Design כפי שמופיע בהרצאה על Design .
תארו את הדילמות, השיקולים וקבלת ההחלטות שלכם והסבירו את הפתרונות שבחרתם.
ב. ציינו אילו מהעקרונות שנלמדו בהרצאות הקורס בנושאים: Reuse , Design, Architecture ,
ו - patterns Design באו לידי ביטוי בתהליך ה<u>תכן הכלל -מערכתי</u> שביצעתם והסבירו באיזה אופן –
באמצעות דוגמאות קונקרטיות ספציפיות)לא כלליות(מתוך המערכת "CEMS "שפיתחתם.
לכל אחד מהעקרונות, הסבירו מה הם היתרונות המתקבלים משימוש בהם.

<u>תשובה:</u>

א. מבחינת ארכיטקטורה אנו משתמשים בארכיטקטורה של 3 רבדים- boundary שאחראי על ה-GUI . כל מרכבי ה-GUI נמצאים ב- קובץ fxml המתאים לו והוא מממש כל כפתור או טקסט. במידה וה-boundary צריך מידע מהמסד הוא פונה ל-controller המתאים לו והוא מייצר בקשה לשרת, השרת עצמו מייצר שאילתה מתאימה עם הערכים המתאימים שקיבל וסוג הבקשה ומבצע פנייה לDB. בנוסף הcontroller מקבל תשובות מהשרת, בעזרת מחלקה אליה הוא טוען את סוג התשובה והאובייקט, כלומר: הוא מאגד כל תשובה ושולח אותה חזרה לboundary. הentities הם אובייקטים המשתמשים בעיקר לאחסון מידע, לדוגמה בעזרת האובייקטים הללו אנו מעבירים מידע בין הclient ל- server על פי הבקשה שנוצרה.

<u>דילמות שנתקלנו בהם:</u>

בתהליך יצירת מבחן, המימוש הראשוני היה :
1. לבחור שאלה קיימת מתוך רשימת שאלות.
2. בחירת ניקוד השאלה ע"י הקלדתו.

תהינו איך המורה יכולה לדעת איזה שאלות כבר בחרה, כדי שלא תבחר פעמיים את אותה שאלה. הרגשנו כי המימוש שיצרנו מעט מסורבל. הפתרון היה לשנות את המסך ובהתאם לכך את הפונקציונליות מאחוריו: יצרנו טבלה שמופיעה במסך יצירת מבחן, שבה ניתן לראות את השאלות שבחרנו למבחן- תחילה הטבלה ריקה. כאשר לוחצים על לחצן browse question ניתן לבחור מתוך רשימת שאלות שאלה ספציפית. אותה שאלה ספציפית שנבחרה מתווספת כל פעם לטבלה המוצגת למורה. כך המורה רואה לאורך כל התהליך את ההוספה של השאלות אחת אחרי השנייה למבחן חדש.

מאחורי הקלעים בשלב המימוש היינו צריכים שהטבלה תתעדכן בצורה דינמית. הבנו שצריך להשתמש ב property ולכן יצרנו מחלקה בשם Question in exam row אשר מנהלת את המידע בטבלה באופן דינמי בהתאם לשינוי שרצינו ליצור.

ב. בחרנו להשתמש ב design pattern להלן:

את המחלקה Logged in user מימשנו בעזרת singleton , יצרנו משתנה סטטי יחיד למחלקה שנוצר פעם אחת ע"י הבנאי. כאשר נוצר המשתנה כל קריאה לבנאי תחזיר מצביע לאותו משתנה. המטרה היא שעבור כל הרצה של client יהיה רק משתמש יחיד שמחובר למערכת CEMS.

במסך login ובעת יצירת משתמש ביצענו שליפה של כל הנתונים אודות אותו משתמש שהתחבר למערכת, כלומר עבור סטודנט שלפנו את כל רשימת הקורסים שלו והפרטים המזהים שמייחדים אותו. באותו אופן גם עבור המורה, שלפנו את רשימת המקצועות שהיא מלמדת ובכך איפשרנו לגשת לאוסף הפרטים הללו ללא גישות חוזרות לשרת, אלא קבלה של מידע מובן בעזרת גישה ל instance של המשתמש המחובר למערכת.

בנוסף השתמשנו גם בעקרון ה-Reuse באופן הבא: לאחר שנוכחנו לראות במהלך העבודה המשותפת על הקוד, בהרבה מקרים היו מתודות שחזרו על עצמן במחלקות רבות ולכן יצרנו מחלקה גדולה ומשותפת לכל המתודות הללו שנקראת "GuiCommon" שתפקידה להכיל את המתודות הרבות שמשותפות בין המחלקות השונות.
לדוגמא ניתן לראות במחלקה הנ"ל מתודות לבדיקת ID תקין של מבחן, בדיקת ID תקין של שאלה, הקפצת הודעה למשתמש בסגנון popUp ועוד.
היתרונות המתקבלים מעבודה בצורה זו הם: חיסכון במאמץ כתיבה - המתודה נכתבת פעם אחת בצורה טובה ואין צורך לכתוב אותה כל פעם מחדש.
יתרון נוסף הוא הימנעות מבדיקה עם שגיאות, כך שלאחר שישבנו וכתבנו לצורך הדוגמא את המתודה שבודקת ID תקין של מבחן פעם אחת כראוי ובצורה מקיפה, אין צורך לחזור ולבדוק את תקינותה כל פעם מחדש ומספיק שפשוט נקרא לה כל פעם שנצטרך.
כמו כן, לאורך זמן נחסך גם זמן רב כאשר השתמשנו במתודות משותפות מורכבות יותר כמו חלון הודעה למשתמש "popUp" שדרשנו עבודה מעט יותר מורכבת.

בנוסף, ביצענו שימוש במתודות set לטובת מעבר בין מסכים, באופן זה העברנו אובייקטים שונים בין רצף מסכים במערכת אשר מקטין את כמות הפניות לשרת.

זיהינו תבניות של נתונים שנוכל לאסוף עבור מסכים שונים ממקום אחד, לדוגמא כאשר נכנסים למסך exam Bank המורה יכולה לבצע פעולות שונות עבור מבחנים, כמו עריכת מבחן, יצירת מבחן אקטיבי וכו'. נעזרנו במתודה פרטית שמביאה את כל הנתונים של המבחן הזה כך שלפני כניסה למסך שהמידע הזה רלוונטי עבורו יוכל להשתמש בה. באופן זה איפשרנו מעבר חלק בין מסכים עם מינימום קריאות לשרת.

שאלה 2

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"**).**

תשובה:
תהליכי הבדיקות שביצענו בפרויקט היו אבן דרך חשובה בתהליך. בדקנו את נכונות הקוד וממשק המשתמש. הבדיקות שביצענו היו תוך כדי כתיבת הקוד וגם בסיום הכתיבה. כחלק מדרישות המטלה ביצענו בדיקות ב Jubula של בניית בחינות ויצירת שאלות, בדיקות Unit test על התחברות למערכת וסטטיסטיקות.

בין היתר, הבדיקות נעשו תוך כדי כתיבת הקוד בהתאם לעקרונות שנלמדו:
קופסא שחורה- בדיקות אילו הן בדיקות המבוססות על דרישות ופונקציונליות, שיטת בדיקה זו נעשית כאשר המבנה הפנימי לעיתים אינו ידוע לבודק, בדיקות אילו בדרך כלל בודקות את הפונקציונליות של המערכת בחנו את תפקוד המערכת לאחר תוך כדי כתיבת הקוד וגם בסיומו. בדקנו את נכונות הקלט/פלט - קלטים ופלטים חוקיים. בדיקות אלה בוצעו בעיקר כאשר המשתמש מזין נתונים, וכאשר יש מעבר בין מסכים כיוון שפלט של מסך אחד הוא קלט של מסך אחר.
דוגמא1- כאשר מכניסים examID שהוא יותר מ6 ספרות- מודיעים על כך בpopUp רלוונטי- כדי שהמשתמש יבין מה הטעות (מספר ספרות שגוי).
בנוסף כאשר המורה מבצעת עריכת מבחן לexamID מסוים אין באפשרותה לערוך את שדה זה ולכן תוך כדי תפעול כמשתמש אמיתי במערכת זיהינו מקרי קצה חשובים.

דוגמא2- בשלבים הראשונים של הפיתוח, לאחר שמימשנו תקשורת של שרת לקוח ושימוש במסד נתונים רצינו לבדוק תחילה שהתקשורת עובדת בצורה תקינה, כלומר בדקנו שלפי קלט ספציפי מצד הלקוח אכן מתעדכן מסד הנתונים בצד השרת ע"י שאילתה שנכתבה כטיוטה במסד כדי לדעת ולוושא שהיא אכן תקינה ושולפת את המידע הנכון. ולאחר מכאן ציפינו לקבל פלטים ידועים מראש חזרה בצד הלקוח

דוגמא3- ביצירת מבחן חדש , בשלב 2 כאשר מוסיפים שאלות ועבור כל שאלה מזינים ניקוד- מתבצעת בדיקה שעד אשר לא מגיעים לניקוד מלא של מבחן שווה ל100 לא ניתן לעבור למסך הבא. בנוסף, מציגים הודעה מתאימה. באופן זהה לעריכת מבחן כאשר משנים ניקוד לשאלות.
נציין כי בדיקה שחשובה ששמנו אלייה לב בשלב האינטגרציה בין מימוש המסכים הללו היה כאשר ביצענו מעבר בין המסכים ורצינו לחזור אחורה וציפינו שהמסך יזכור את הנתונים ויציג אותם למורה.
זיהינו שנדרש מנגנון שיסייע לכך ומימשנו פונקציית set המעבירה נתונים ומקבלת כאשר מבצעים מעברים שונים.

בדיקות אינטגרציה-
בדיקה זו מתמקדת בבדיקת מודולים משולבים, כלומר לאמת פונקציונליות משולבת לאחר אינטגרציה מודולי קוד ויישומים נפרדים, בדיקות אילו רלוונטיות במיוחד ללקוחות ושרתים ומערכות מבוזרות.
לכן, לקראת סוף שלב הפיתוח ובכדי לבדוק כי המערכת עובדת בשלמותה באופן תקין ביצענו המון בדיקות מהסוג הזה כדי להבטיח כי חיבור הדפים, התחברות של כמה לקוחות שונים או זהים כמו קבוצת סטודנטים תעבוד נכון. כלומר קלט מתוך מימוש של דף מסוים היווה פלט לתוך מימוש של דף אחר.
הדוגמא הבולטת ביורת הייתה בתרחיש המצבים הבא:
כאשר סטודנטים מבצעים מבחן פעיל ממוחשב או ידני המורה יכולה להגיש בקשה למנהלת לתוספת זמן.
ברגע זה מוצגת למנהלת התראה והיא יודעת שהיא נדרשת להכנס באופן יזום למסך אישור הבקשות.
ברגע שהיא מאשרת את הבקשה הסטודנטים מקבלים התראה על כך מהמורה שיש תוספת זמן לבחינה.
ובנוסף המורה מקבלת התראה גם כן כי בקשה אושרה או נדחתה.

דוגמא2- לאחר כתיבת קוד של מסך enter to exam רצינו לבדוק אותו. כאשר התחברנו למערכת באמצעות ה server- התחברות בוצעה בהצלחה והמשכנו למסך הרלוונטי. שם המערכת קרסה בעת לחיצה על לחצן start exam ולא היה ניתן להמשיך. היינו צריכים לבדוק מה הבעיה וכיוון שיש צורך שה client וה server יעבדו יחד ולא תתבצע קריסה. גם כאן הפתרון היה לדבג ולהבין מה נק' הנפילה והוספנו גישה לשרת הדוגמת אם לסטודנט כבר קיים אותו מבחן בטבלת exam_of_student או לא כאינקציה לכך שהוא כבר התחיל את הבחינה ולא יכול לבצע אותה בשנית. זאת על מנת להבטיח **שלא** ינסה לפתור אותה שוב וגם לא במגבלות הזמן שמותר לו להכנס לבחינה(חצי שעה ממועד התחלת הבחינה).

<u>קופסא לבנה-</u> בדיקה זו מבוססת על הידע וההיגיון הפנימי של קוד היישום, ידוע גם כ " קופסת זכוכית ", בבדיקות מהסוג הזה יש להכיר את הקוד הפנימי ובדיקות אילו מתבצעות לרוב על ידי המפתח. כאשר כל חבר צוות סיים את מימוש הדפים שלקח על עצמו הוא ביצע בדיקות מסוג זה, כדי לבדוק את מסלולי החישוב השונים בקוד והנתיבים השונים שיש בקוד:

דוגמא1- היות והשתמשנו בעיקרון ה**reuse** בקוד, אותו נפרט בהמשך, עשינו שימוש במסכים בעלי אותה פונקצונאליות על מנת לא לבצע חזרה כפולה של קוד או קבצי FXML . לכן הגדרנו סטטוס מצב למסכים מסוג זה האם הם נמצאים ברשות המורה לעריכה או בסטטוס צפייה בלבד ברשות המורה. הflow הזה דרש עדינות בכתיבת הקוד כדי לשמר פונקציונאליות ולהסיר ולהציג כפתורים שונים בעת זיהוי המשתמש המחובר למערכת במצב נתון כולשהו.

דוגמא 2- עבור מורה, קיים מסך של אישור ציון של סטודנט (Score Approval) שבתוכו ניתן לאשר סופית או לעדכן ציון של מבחן ספציפי עבור סטודנט שנבחר.
לאחר שהמורה בוחר את הפרטים הוא נדרש לרשום הערה + ציון חדש ולאחר מכן ללחוץ על כפתור העידכון.
ברגע שהמורה לוחץ על כפתור העדכון מתבצעות מספר בדיקות שיכולות להתפצל למספר נתיבים שונים :
ראשית נבדק שהציון החדש שהוזן והערה לשינוי הציון אינם ריקים, לאחר מכן נבדק שהציון נמצא בין 0 ל100 וכמו כן נבדק שהציון מורכב מספרות בלבד.

דוגמא 3- כאשר אנחנו מבצעים יצירה של מבחן חדש עבור כל שאלה חדשה שהמורה מכניס לשאלות של המבחן הוא נדרש לתת ניקוד ולאחר שהוא מוסיף כל שאלה מתבצעת בדיקה של הניקוד של השאלה האם הוא תקין בפני עצמו - לא שלילי ולא 0, ולאחר שראינו שהניקוד של השאלה נכון אנו בודקים שגם הניקוד הכללי של סך השאלות במבחן תקין - האם לא עובר את ה100 והאם בדיוק 100.

שאלה 3

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

א. תיאום פעילויות ושיתוף בין חברי הצוות בפיתוח וגישה 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.

ב. שילובי קוד (אינטגרציה מערכתית - לאחר הפיתוח הראשוני) ובדיקות.

ציינו באופן פרטני, בהתייחסות ספציפית לפיתוח המערכת " CEMS ," איך פעלתם בשלב זה של הפיתוח (למשל: תיאור התנהלות התהליך ההנדסי (לא עבודת הצוות) , אופן טיפול בבעיות, וכו').

אם היו קשיים מה הסיבה לכך? מה הייתם משנים בדיעבד בגישתכם למרכיב זה של תהליך הפיתוח מבחינת האספקטים הרלבנטיים של הנדסת תוכנה?

תשובה:

3.א

שיתוף הפעולה בין חברי הצוות וגישת ניהול גרסאות נעשה באופן הבא:

כאשר מתפרסמת מטלה, נפגשנו בזום כל חברי הצוות ועברנו על המטלה, ציינו את הנקודות שבהם נצטרך לתת דגש, חלוקת המשימות וקביעת יעדים.

בנוסף, חילקנו כל המשימות לתתי משימות ראשיות על פי סעיפי המטלה.

לאחר מכן, פרשנו את המטלות בצורה יותר ממוקדת עם פירוט על כל משימה.

החלטנו לנהל את המשימות שלנו לפי קטגוריות, עדיפויות, וסטטוס בעזרת "לוח משימות צוות"- Trello (ראה תמונה)

יתרונות:

- הגדרת כל משימה ופירוט בפנים במידה וצריך.
- שיוך חבר צוות למשימה שלו.
- הוספת סטטוס: complete/ finish not tested…וכו'.
- מונע הצטברות של הודעות ווצאפ/ בעל פה שילכו לאיבוד..
- כל חבר צוות יודע מי ביצע כל משימה וידע לפנות אליו במידת הצורך.
- כאשר מישהו מסיים משימה הוא משתבץ למשימה חדשה וכך מונעים כפילות.

(בנוסף, היות ואנחנו עובדים ב branch -ים יש לכך חשיבות גדולה).

חסרונות:

דורש מחברי הצוות להיות מאורגנים ולעדכן סטטוס אך מדובר בפעולה פשוטה, זריזה ומיידית כך שהיתרונות שבה עולים מעל כל חסרון שיכולנו להעלות.

סביבת הפיתוח שבה אנו משתמשים היא GIT, היה לנו מאוד חשוב שכל חברי הצוות ידעו כיצד להשתמש נכון בכלי ולכן החלטנו שכל חבר צוות יקדיש זמן ללימוד עצמי. בנוסף, היה חבר צוות שהכיר יותר לעומק את הכלי וכאשר הועלתה תובנה לגבי השימוש ערכנו פגישת זום של הסבר תפעול נכון, השיחה הוקלטה וכך כל אחד יכל לחזור להזכר בפעולות.

דוגמא להסבר שביצענו היה כיצד לפתוח branch חדש, וכיצד למזג את החידושים ב main אל הbranch עליו עובדים בשוטף.

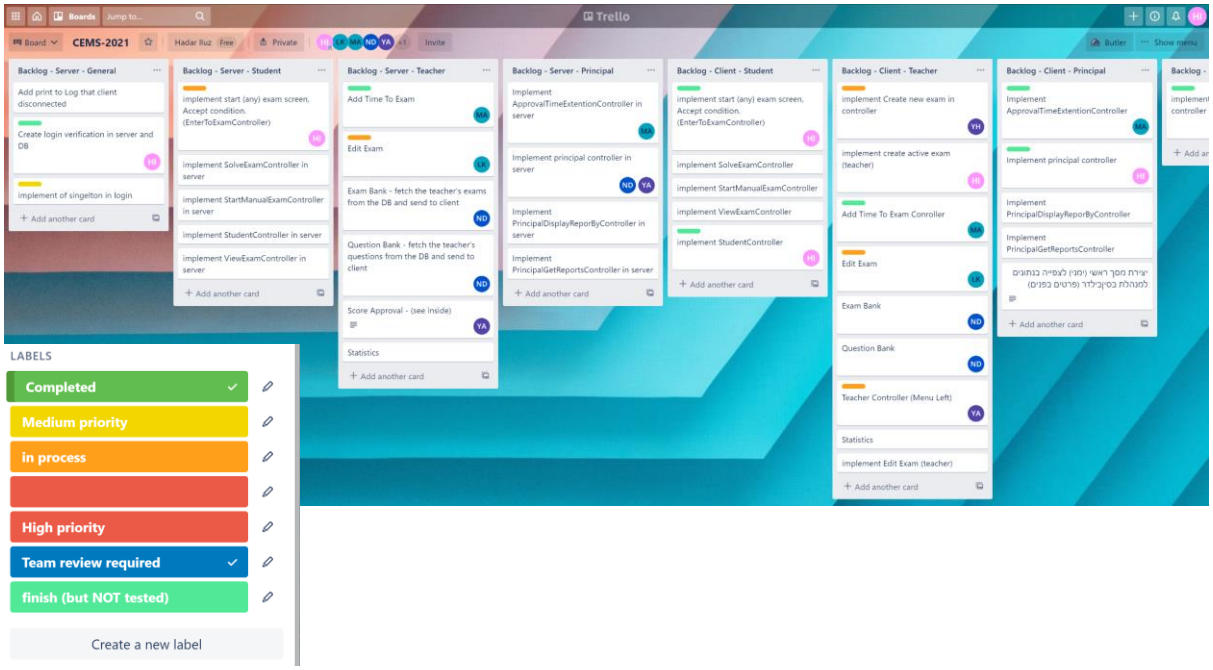
זיהינו כבר בשלב פיתוח האבטיפוס שיהיו מחלקות שנרצה לשמר ואף להוסיף מנגנונים נוספים וזאת על מנת לצמצם קטעי קוד חוזרים.

יצרנו מחלקות נוספות של RequestToServer ו- ResponseFromServer שירכזו את יצירת הבקשות מהלקוח לשרת החזרת התשובה בין השרת ללקוח.

בנוסף, השתמשנו בgoogle drive עבור כלי שבו ריכזנו עבודה שוטפת על מסמכים, כמו מענה על קובץ שאלות זה, הכנת מצגת, הכנת מטלת מסכי אפליקציה בממשק וכדומה.

העדפנו לרכז מסמכים שבהם אנחנו עובדים באופן רציף במקום בו כל חברי הצוות יכולים לערוך מסמך אחד בכל זמן נתון וזאת על מנת לא להגביל חברי צוות שונים.

לסיכום יש ברשותנו 3 ערוצים עיקריים לניהול הפרויקט באופן שוטף, כל חברי הצוות מאוד מאורגנים ושלטו בכל הכלים במהירות.



סה"כ אנחנו מפיקים מההתנהלות הכללית שלנו בהיבט זה מספר תובנות חשובות:

יתרונות:

- צמצום התלויות של חברי צוות אחד בשני ויכולת לעבוד עצמאית והתקדמות בזמן האישי במטלות.
- צמצום היקף כל קובץ, ובכך יותר קל לקרוא אותו ולהבין בצורה לוגית מה הוא אמור לשמש
- שליחת מתודות ואובייקטים למחלקות שניתן להשתמש בהן ביותר ממקום אחד ובכך לחסוך זמני פיתוח
- כמות מיזוגים וקונפליקטים מינימליים בעקבות עבודה נכונה בgithub.

חסרונות:

- מסד הנתונים מתעדכן על בסיס קבוע ויש צורך בפעולה אקטיבית בכדי לטעון את הקבצים לWorkBench של כל אחד מחברי הצוות, אף על פי שמצאנו שיטה יעילה של יצירת קובץ אחד עבור כל הטבלאות כך שכל פעם שמישהו מבצע עידכון הוא מפרט המ השינוי ושולח לכולם גרסאת DB מעודכנת שדורשת 2 צעדי טעינה מהירים וקידום מספר הקבוצה בשם הקובץ לצורך מעקב.
- חוסר היכרות עם קטעי קוד שלא כתבת, או ביצעת עליהם Code Review, על מנת להפוך מכשול זה ליתרון החלטנו שניעזר בTerllo כדי לבצע עקיבה על Review בין חברי הצוות. כאשר משימה עוברת לcomplete ומסומנת עם תגית DONE וזאת לאחר שחבר הצוות שמימש זאת הוסיף הערות לדוקומנטצית JavaDoc אזי חבר צוות אחר ביצע מעבר במטרה לזהות/ להעלות הערות/ הארות ובכך יכלנו לוודא שהדוקומנטציה שביצע אכן טובה.

3.ב. בתחילת מטלה 3 ביצענו מפגש קבוצתי בזום וקבענו מספר כללים שינחו אותנו במהלך שלב הפיתוח:

1. כל המחלקות יסווגו בהתאם לפורמט אותו קבענו בפגישה.
2. כל חבר קבוצה עובד על מסך שונה אבל לוקח על עצמו את החלק של המסך בclient וגם בserver, כך שכולם יתנסו בפיתוח בשני הצדדים.
3. נתינת שמות משמעותיים למתודות ומשתנים כדי להקל על הבנת הקוד.
4. בהיבט ממשק המערכת, עבודה עם ממשק אחיד שהוכן מראש במפגש זה.
5. שימוש עקבי בgit וbranch ומעקב אחר עדכונים מחברי הצוות.
6. במקרה של חפיפה יש לבצע תיאום מראש ולהחליט יחד מה הכי נכון ויעיל.

שימוש בכללים האלה תרם לכך שיהיה תיאום מלא בין חברי הצוות באופן כתיבת הקוד והמחשבה הלוגית. כתוצאה משימוש בgit, העבודה בין חברי הצוות הייתה לרוב בלתי תלויה- כך שכל חבר צוות יכל לנהל לוח זמנים משלו. כתוצאה משימוש בbranch לכל חבר צוות שעובד על חלק שונה במטלה, לא נעשו דריסות קוד, ובמידה והייתה חוסר התאמה יכלנו לראות את שני הקבצים ולהשוות ולבדוק מה יותר נכון ויעיל.
הקשיים בהם נתקלנו היו כאשר הgit לא עבד בצורה חלקה, ולפעמים בעת ביצוע איחוד של קוד נאלצנו לבצע בצורה ידנית ומסורבלת. קושי נוסף הוא כאשר שני חברי צוות זקוקים לאותה מתודה, וצריך לבחור מי המתודה העדיפה, או כאשר נדרשנו לבצע שינוי במסדר הנתונים תוך כדי עבודה וחלק היו צריכים לשנות את הקוד שעליהם הם עובדים לטובת השינוי ההכרחי.

שינוי שהיינו מבצעים בדיעבד: לסיים את שלב כתיבת הקוד מוקדם יותר- לצורך השמת דגש על בדיקות התוכנה.