סיכום גרסה 2

אליסה פיינגולד – 205576937
איתי דגן – 206226524
יערה רמני – 204434906
עינבר צור – 312493927
תומר גודלי – 204606495

מימוש המערכת

משימות שהושלמו במלואן

- מימוש כלל המחלקות כפי שהוצגו בגרסה מס׳ 1.
- כלל תרחישי השימוש מומשו מלבד לתרחיש שימוש 8.5.
- אבטחת סיסמאות במסד הנתונים.

משימות השייכות לאיטרציה מספר 3 שהושלמו באופן חלקי

- מערכת שליחת התראות למשתמשים אודות שינויים שהתבצעו ואירועים לאוהדים.
- תיעוד כלל האירועים במערכת בקובץ Log.

משימות שנדחו לגרסה עתידית

- מערכת שליחת מיילים למשתמשים בעקבות תרחישי השימוש הרלוונטיים.
- תרחיש שימוש מס׳ 8.5 – מנהל מערכת מפעיל את בניית מודל מערכת ההמלצה.

אתגרים שהתמודדנו עימם בבניית המערכת

1. אפשור למנהל ובעל קבוצה להיות בעלי תפקיד נוסף – שחקן/מאמן

אתגר זה נפתר באמצעות החזקת אובייקט נוסף של additional Role, במידה ולאחד מהמשתמשים שנחשבים Management User (Team owner/Team manager) קיים אובייקט זה אזי שהוא בעל תפקיד נוסף. בהינתן שיש לו 2 תפקידים אזי שהוא יכול לבצע את אותן פעולות כפי שהתפקיד הנוסף שלו יכול לבצע. כמו כן במידה והמשתמש כבר לא יהיה מנהל או בעל קבוצה וקיים לו תפקיד נוסף הוא יחזור למערכת על תקן התפקיד הנוסף בלבד.

בעצם עשינו כאן מעין שימוש של תבנית העיצוב Decorator באופן מותאם למערכת שלנו. כך אנחנו יוצרים סוג של מבנה שכבות עבור התפקידים כאשר התפקיד מחזיק מצביע לתפקיד הקודם במידה וקיים. חשוב לומר כי בDB יש user אחד עבור משתמש בעל מספר תקפידים שונים ועל כן המשתמש לא ירגיש בכך.

2. מתן הרשאות למנהל קבוצה ע״י בעל הקבוצה

אתגר נוסף היה הענקת הרשאות מתאימות למנהל קבוצה על ידי בעל הקבוצה לביצוע פעולות המשויכות לבעל קבוצה.

אתגר זה נפתר באמצעות החזקת שדה שנקרא permissions עבור מנהל קבוצה ובאמצעות ENUM מתבצע מיפוי של כל הפונקציות האפשריות והאם יש לו גישה עבור אותה פונקציה או לא.

3. ניהול מסד הנתונים של המערכת

ניהול בסיס הנתונים היווה עבורנו סוגיה בהתחלה שהחלטנו לפתור אותה באמצעות מחלקה שנקראת System Controller והיא בעצם מדמה עבורנו את בסיס הנתונים.
קיבלנו החלטה זו מתוך המחשבה שגם בעתיד נרצה מחלקה אחת שתהיה אחראית על התקשורת אל מול בסיס הנתונים.

4. <u>שיבוץ המשחקים ושיטות הניקוד</u>

אתגר זה נפתר באמצעות שימוש בתבנית תכן Strategy, זאת אומרת שאנו מבצעים injection של הקבוצה/ עונה אל המחלקות של שיבוץ משחקים ושיטות הניקוד ובכך המימוש הינו גנרי ותלוי זמן ריצה איזו מדיניות לקבוע עבור כל קבוצה/ עונה.

5. <u>שליחת התראות למשתמשים אודות שינויים שהתבצעו ואירועים לאוהדים</u>

משימה זו שייכת לאיטרציה מס' 3 אך כבר מימשנו את התשתית עבור הפונקציונליות הנדרשת.
ביצענו שימוש בתבנית Observer כך שבעצם Fan יכול להירשם על מנת לקבל עדכונים והתראות אודות שינויים שהתבצעו וברגע שהשינוי יתבצע יישלח עדכון עבור כל מי שנרשם לקבלת העדכונים.

<u>אכיפת אילוצי נכונות המערכת</u>

1. בעת עליית המערכת נוצר משתמש עבור מנהל המערכת והוא מקבל זימון לכתובת המייל שבעל המערכת הראשי הזין.
2. כלל בעלי העניין חייבים לעבור תהליך רישום/ קבלת הזמנה ממשתמש קיים במערכת.
3. את הקבוצה יוצרים רק בעלי קבוצות, כמו כן במידה ונרצה למחוק משתמש שהינו בעל קבוצה והוא בעל הקבוצה היחיד – המחיקה לא תתאפשר.
4. אין אפשרות להחזקת יותר מדף אישי אחד לקבוצה, שחקן ומאמן.
5. כל ליגה בעונה מסוימת יכולה להחזיק רק מדיניות אחת לחישוב ניקוד ושיבוץ משחקים.

<u>הנחות שנלקחו במהלך המימוש</u>

1. מנהל קבוצה, כדורגלן ומאמן יכולים להתווסף רק על ידי בעל הקבוצה.
2. בעל קבוצה יכול להתווסף רק על ידי מנהל המערכת באופן דומה להוספת שופט חדש למערכת.

בדיקות המערכת

ביצענו את תהליך הבדיקות עבור גרסה זו בשיטת Button-up כאשר שלבי העבודה היו:

1. ביצוע בדיקות Unit
2. ביצוע בדיקות Integration
3. ביצוע בדיקות Acceptation

פירוט השלבים

1. בדיקות ה-Unit

עבור כל Class בתרשים המחלקות ממשנו Test_Class אשר בודקת את השיטות והפונקציונאליות של המחלקה.

- במהלך בדיקות אלו הושם דגש על שיטות בעלות פונקציונאליות במחלקות אלו.
- לא נבדקו שיטות טריוויאליות מסוג: getters ו-setters.

2. בדיקות Integration

עבור כל שני Controllers במערכת, אשר יש ביניהם יחסי אינטראקציה, (לדוגמה: פונקציונאליות של controller מסוים אשר משפיעה על controller אחר) ביצענו בדיקות עבור האינטגרציה בין ה-controllers.

- נבדקה כל פונקציונאליות המקשרת בין ה-controllers.
- הבדיקה מאמתת כי הפעולה הביאה לתוצאה הרצויה בשני ה-controllers הרלוונטיים.

3. בדיקות Acceptation

בבדיקות אלה, עבור כל UC שהוגדר כ-UC במערכת ביצענו מבחני קבלה חיוביים ושליליים כדי לאמת כי הפונקציונאליות הנדרשת ב-UC מתקיימת בצורה הרצויה במערכת.

- עבור כל UC-controller ממשנו מחלקת UC_Test אשר בה ממומשים מבחני הקבלה החיוביים והשליליים.
- עבור כל UC ביצענו כיסוי קשתות (branch coverage) מלא, ואף בחלק מן המקרים גם ביצענו predicate coverage מלא.

בסיום כל שלב בתהליך הבדיקות, צוות הפיתוח בקבוצה ביצע תיקונים מתאימים בקוד במידה ונמצאו טסטים אשר כשלו.

שיקולים שנלקחו בעת בחירת אופן ביצוע הטסטים

- ביצוע הבדיקות בשיטה זו מאפשר גילוי מוקדם של שגיאות בפונקציונאליות של המחלקות.
- ביצוע הבדיקות בשיטת Button-up מאפשר עבודה מקבילית על הקוד כך שבזמן בדיקות היחידה, צוות הפיתוח המשיך בכתיבת הפונקציונליות של ה-Controllers במערכת.

אופן חלוקת העבודה

חלוקת העבודה בין צוות הבדיקות בתהליך זה התבססה על בדיקה רוחבית של מחלקות ופונקציונאליות במערכת. כל חבר צוות מונה להיות אחראי על מספר שווה של UC במערכת.

חבר הצוות אחראי לביצוע בדיקות ה- unit , integration, acceptances באופן רוחבי, עבור ה-UC עליהם הוא אחראי.

במהלך כל שלב בבדיקות, האחראי ביצע את הבדיקות עבור המחלקות הרלוונטיות עבורו.

מסקנות מתהליך ביצוע הבדיקות

הבחירה בשיטת ביצוע הבדיקות כ-button-up אפשרה לנו לגלות בעיות במימוש בקוד בשלבים מוקדמים, הדבר אפשר לנו להעלות את הבעיות ולתקנם ובכך להמשיך את תהליך הבדיקות.

בעקבות זאת נוכחנו כי מרבית הטעויות שנמצאו בקוד התגלו בשלבים מוקדמים, ובמהלך בדיקות הקבלה מספר השגיאות שהתגלו היה נמוך.

ביצוע תהליך הבדיקות באופן מקבילי לתהליך הפיתוח של הקוד אפשר לשני הצוותים בקבוצה לבצע את התיקונים הנדרשים באופן מידי.

בעקבות אופן ביצוע הבדיקות:

- הגענו לאחוז כיסוי גבוה של שורות קוד.
- כיסוי מלא של כלל הפונקציונאליות במערכת:
  - בגרסה זו כיסינו בבדיקות את הפונקציונאליות של כל ה-UC אשר מומשו עבור איטראציה.

נספחים

1. מצ"ב תיאור הכיסוי אודות הבדיקות עבור ה-controllers השונים של המערכת:

100% classes, 96% lines covered in package 'domain.ServiceLayer.Controllers'

Element	Class, %	Method, %	Line, %
AssociationRepresentativeController	100% (1/1)	100% (10/10)	98% (79/80)
ComplaintSystemController	100% (1/1)	100% (4/4)	100% (24/24)
FanController	100% (1/1)	100% (6/6)	100% (41/41)
GuestController	100% (1/1)	100% (1/1)	93% (15/16)
PersonalPageSystem	100% (1/1)	100% (10/10)	100% (85/85)
RefereeController	100% (1/1)	100% (7/7)	100% (52/52)
SignedInController	100% (1/1)	100% (3/3)	100% (29/29)
SystemMangerController	100% (1/1)	100% (9/9)	93% (43/46)
TeamOwnerController	100% (1/1)	100% (17/17)	90% (137/152)
SystemController	100% (1/1)	100% (11/11)	100% (64/64)

ניהול הגרסה

את הגרסה ניהלנו ע״י שימוש בכלים הבאים:

- **Trello** – לאחר כל שיחה שניהלנו בנוגע לפרויקט, פרטנו את המשימות שיש לבצע והעלנו אותן למערכת. דאגנו לתקף ולערוך את המשימות בהתאם לאילוצים ולשינויים שהתגלו תוך כדי ביצוע המשימות. כמו כן, כאשר התגלו בעיות ע״י צוות הבדיקות הם העלו את הבעיה למערכת תחת הרשימה ״issues״ וכאשר הבעיה תוקנה דאגנו למחוק אותה מהרשימה או לסמנה בתווית המתאימה.
- **GitHub** – גם צוות הפיתוח וגם צוות הבדיקות יצרו branch ייעודי עבור כל חבילת קוד שנכתבה. לאחר שביצענו merge מוצלח ל-master דאגנו למחוק את ה-branch ע״מ לשמור על סדר בניהול הגרסה.
- **מטריצת מעקב** – השתמשנו במטריצת מעקב עבור שני שימושים: מיפוי ה-UC לאיפה בקוד הם מומשו ע״מ ליצור סדר ויכולת לבחון האם ה-package-ים שיצרנו הגיוניים ונכונים עבורנו, וע״מ למפות בין טסטים לבין UC-ים. בנוסף להעלאת באגים שנמצאו ל-Trello דאגנו לסמן באדום טסטים שלא עברו.
- **גוגל דרייב** – את כל הקבצים, התיעודיים והדברים שיצרנו עבור גרסה זאת העלינו לתיקייה ייעודית בגוגל דרייב. בנוסף, גם את הקבצים שיצרנו לגרסה הקודמת והיה עלינו לתקנם העלינו לתיקייה של גרסה זו, וכך יכלונו לעבוד במקביל תוך ידיעה שכל הדברים מעודכנים ואין גרסאות שונות אצל כל חבר צוות.

אתגרים שהתמודדנו עימם בבניית המערכת

1. עבודה צוותית על הקוד מרחוק

עבודה משותפת על קוד היא תמיד מאתגרת, בטח כאשר מדובר בקוד מורכב אשר מתחילים אותו מאפס וללא קונבנציות מסוימות, ובטח כאשר לא ניתן להיפגש עם חברי הצוות ע״מ להבהיר את הדברים.

תחילה ניסינו לחלק בנינו את הפונקציונליות ולנסות לקבוע קונבנציות, אך כשראינו הדבר גוזל מאיתנו זמן רב ועדיין איננו מצליחים לעבוד בצורה אחידה, החלטנו שתחילה יעבדו רק שלושה חברי צוות על הקוד (איתי, תומר ואליסה) ואילו השניים האחרים (יערה ועינבר) יתחילו בכתיבת טסטים.

בהמשך, כאשר עבודת הטסטים נהיתה יותר מאסיבית תומר החל לעבוד גם על הטסטים, בעוד שאליסה ואיתי נשארו בצוות הפיתוח והיו אחראים על תיקון הבאגים שנמצאו.

ע״י צורת עבודה זו, כך שעל כל משימה עבדו 2-3 אנשים, הצלחנו להימנע מקונפליקטים רבים בקוד, לשמור על תיאום ואפשרנו לכל חבר צוות להיות משמעותי בתחום שעליו הוא היה אחראי.