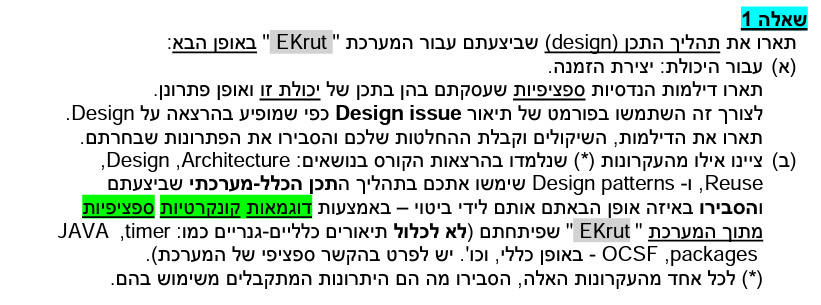
**תשובות לשאלות קבוצה G15**

תאריך הגשה: 18.01.2023





**תשובה לשאלה 1**

1. תהליך העיצוב עבור יצירת הזמנה:

* מטרה:

המטרה היא לאפשר ללקוחות רשומים ומנויים לבצע הזמנה במערכת ekrut. עבור מנויים נאפשר גם קבלת מבצעים שונים. המערכת תאפשר לבצע הזמנה מרוחקת בתצורת OL או הזמנה ישירות במכונה.

* עדיפויות:

העדיפויות בסדר יורד

ממשק ידידותי למשתמש, ברור ונוח לתפעול הכולל את הפונקציונאליות הנדרשת. (usability)

שימוש חוזר בקוד קיים (reuse).

הכנה לפיתוח עתידי ושינויים. (maintainability, portability).

שיפור הקוד והיעילות. (efficiency)

* תיאור:

המשתמש יתחבר למערכת ויוכל ללחוץ על כפתור "create new order" אם הוא ב-OL יוצג לו חלון בו הוא יבחר האם לעשות משלוח או לאסוף במכונה ספציפית. לאחר אישור עוברים למסך הקטלוג בו יוצגו המוצרים, השמות שלהם, המחירים ועבור מנויים גם מחיר אחרי הנחה(אם יש). הלקוח יוכל להוסיף מוצר לעגלה ולשנות כמות על ידי כפתורי '+' ו- '-'. ולצפות בעגלה בכל רגע נתון. שיסיים הוא ילחץ על כפתור "place order" ויעבור לסקירת ההזמנה שם ימלא פרטי משלוח אם בחר במשלוח ויוכל לצפות בתכולת הזמנתו ובהנחות שקיבל אם היו. השלב האחרון הוא מעבר לתשלום שלאחר אישור התשלום יוצג לו אישור עם הפרטים במידה של משלוח או איסוף.

* בעיות תכנון עיקריות:
  1. כאשר לקוח בוחר במשלוח, איפה הוא יכניס את פרטי המשלוח?

אלטרנטיבה א' - בהתחלה מיד לאחר הבחירה באפשרות משלוח.

אלטרנטיבה ב' - בסקירת ההזמנה לפני התשלום.

החלטה - בחרנו לשים בסקירת ההזמנה מכיוון שכשאר שמים בהתחלה מילוי פרטים זה יכול לגרום ללקוח "להתעצל" ולוותר על ההזמנה לכן באפשרות ב' הוא קודם כל בוחר את המוצרים שהוא רוצה וכך מילוי הפרטים יהיה חלק מסגירת ההזמה אחרי שהוא יודע בוודאות שהוא מעניין להזמין.

* 1. כאשר לקוח לוחץ על כפתור חזרה אחורה ממסך הקטלוג מה יהיה עתידם של בחירת מוצריו?

אלטרנטיבה א'- לאפשר ללקוח לזכור את בחירותיו האחרונות ולאפשר זיכרון זמני במערכת של העגלה הנוכחית .

אלטרנטיבה ב'- בלחיצת כפתור חזרה להתריע ללקוח כי כל בחירותיו יימחקו והוא יצטרך להתחיל מחדש את הזמנתו.

החלטה- בחרנו באלטרנטיבה ב', בכדי לשמור על תקינות מלאי המוצרים אשר מוצג במסד הנתונים, ובכדי שלא ייווצר מצב בו לקוח מוסיף מלאי שלא קיים במכונה לעגלת הקניות שלו ואז הוא לא יוכל לבצע את הזמנתו כהלכה.

1. ראשית, נעזרנו הרבה בהרצאות ולמדנו המון מן המצגת בנושא Architecture שמציגה את הפירוט הרלוונטי מאילו חלקים מורכבת מערכת ואיך צריך לבנות את הבסיס, שנית העקרונות שבאו לידי ביטוי ושימשו אותנו בתהליך התכן המערכתי שביצעו ב EKRUT הם :
   1. **REUSE**

יצרנו תבנית גנרית למסכי popup מסוגים שונים - הצלחה, שגיאה, אינפורמציה, סימולציה והחלטה. ובאמצעות יצירת פונקציה שמקבלת מחרוזת וסוג, הצגנו ללקוח Popup עם ההודעה המבוקשת, ובכך עשינו שימוש חוזר על ידי קריאה לפונקציה הנ"ל במקום ליצור מחדש מסך בכל פעם.

בנוסף, שמנו לב כי להתחברות באמצעות שם משתמש וסיסמה, והתחברות באמצעות התחברות מהירה (EKT) חולקים אותה לוגיקה. לכן במקום לשכפל את הקוד, ביצענו שימוש חוזר ב-login process תוך קביעת פרמטרי התחלה (set user).

כמו כן יצרנו מחלקה סטטית שהיא NavigationStoreController המנהלת את המעברים השונים במסכים. מכיוון שיש המון כאלה (כל הזמן כמעט), במקום לשכפל קוד של מעבר מסך, חזרה ממנו ורענון, החלטנו ליצור קוד אחד שנבצע ממנו REUSE.

יתרון מובהק של שימוש בעקרון זה הוא שימוש בקוד חוזר, שמקצר זמן ומונע באגים (ברוב המקרים).

* 1. **Architecture**

קודם כל השתמשנו במודל ה-ECB.

ה-entities שלנו שמרו מידע קבוע, למשך חיי התהליך כמו UserEntity, ItemInMachineEntity. את הישויות ברובם החזקנו כקלאסים המחזיקים מידע בלבד ואינם מבצעים פעולות נוספות (כמו שליחה וקבלה מהשרת). זו משום שרצינו לנתק את הלוגיקה ולהפוך את הישויות לאובייקטי מידע בלבד.

ה-boundries היוו את הממשק הויזואלי עם המשתמש, קבצי FXML בלבד כמעט ולא מבצעים לוגיקה.

ה-controllers היוו את מירב המערכת. את מחלקות אלה חילקנו למספר חבילות ראשיות:

* controllerGui- מכילה את כל הקונטרולרים של המסכים במערכת.
* Controller- אחראי לתקשורת בין ה-controllerGui השונים בעלי קשר משותף.

ה-controllers ניהלו את התקשורת בין הלקוח לשרת ושלטו בכל המתרחש ­כל העת.

בנוסף היו לנו עוד packages:

* Boundaries- כל המסכים שבמערכת.
* Client - אחראי ללוגיקה של שליחת וקבלת הודעות מהשרת.

יצרנו controller סטטי בשם NavigationStoreController השולט על זרימת המערכת. כלומר, הוא שולט על החלפות מסך, שמירת מסכים קודמים, רענון מסכים (והמידע שלהם), שליטה על כותרות ועוד. זאת עקב תכנון ארכיטקטוני גבוה.

יתרון של עקרון זה הוא הקפדה על ארכיטקטורה ברורה, חלוקה ברורה ועבודה מסודרת. עקרון זה עזר לנו להתמצא במערכת, לתכנן אותה ולעבוד בשיתוף פעולה עם מינימום שאלות.

בנוסף, הוא עקרון זה עזר לנו לחשוב בצורה הנדסית ולענות על דילמות שלפני כן לא ענינו עליהן (האם ישויות מטפלות גם בשליחת וקבלת מידע? אולי יש לעשות קונטרולר לזה?)

* 1. **Design patterns**

השתמשנו בתבנית העיצוב singleton מסוג Observable–המאפשרת מופע יחיד של מחלקה.

לדוגמא: אפשרנו שהחיבור ל- DBשמתבצע במחלקה MySqlClass הינו חיבור יחיד בלבד ולא ייווצרו מופעים נוספים של ה-connection אלא נקבל את אותו מופע על ידי שימוש במתודה getConnection().

תבנית עיצוב זו הובילה גם בהקשר של order controller מכיוון שאפשרנו מופע בודד של המחלקה הזו, שתפקידה לנהל את ההזמנה 'מאחורי הקלעים'. כלומר מבצעים, עגלה, מוצרים וכו'.

השתמשנו בעיקרון הפולימורפיזם ע"י הורשה (לדוגמה EKTController ירש מ-LoginController).

* 1. **Design**

בחלק זה יצרנו תבנית אחידה שתשמש את כל המסכים (גם בעתיד) המהווה את התבנית העיצובית הראשית של המערכת. היא כוללת border pane.

בחלקו של border pane ביצענו חלוקה בצורה הבאה:

בחלק המרכזי – החלק העיקרי של המסך

בחלק התחתון – מוזרק בר תחתון שכרגע כולל כפתור 'חזור'

בחלק העליון – מוזרק בר עליון הכולל pane מעוצב, עם כותרת העמוד, ומיקום המשתמש (ניווט) ביחס למסך הנוכחי.

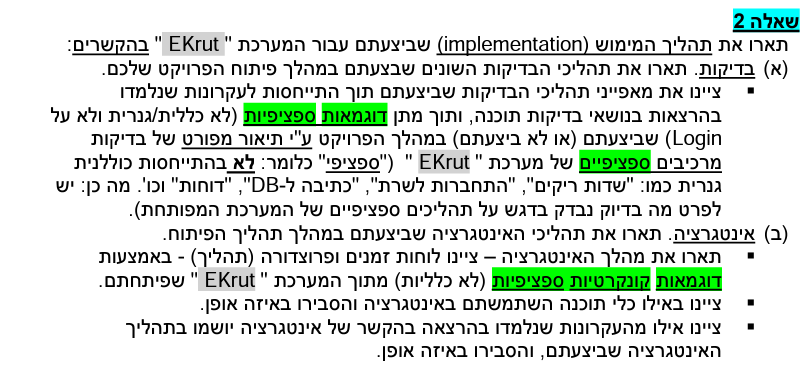
בנוסף ישנם מסכים רבים החולקים פונקציונליות דומה עם 'שקרים' למשתמש. לדוגמה:

HomePage – הוא אחיד לכל המשתמשים במערכת ומשתנה בהתאם לתפקיד שלהם ולכמות הכפתורים הזמינים עבורם.

Login – הוא משותף גם עבור התחברות בעזרת EKT

מסך סיכום הזמנה – משותף גם עבור משלוח (בו משתמש צריך להזין פרטים).

שיטת העיצוב שבחרנו לעבוד בה היא ממסכים ראשיים ולהתפתח למסכים המשניים (המהווים את המשך המערכת).



**תשובה לשאלה 2**

1. 2 גישות -

* Black box קלט ופלט- ללא התמקדות במה קורה בפועל
* White-box – עובר גם על הflow

נתאר את לדוגמא תהליך בנית דוח הזמנות תוך מימוש TDD ו-:

1. בניית המסך - בנינו מסך עם נתונים מוכנים מראש והגדרנו מה התוכן של הדוח אותו רצינו לממש, החלטנו כי אנחנו רוצים להראות את כמות ההזמנות לפי מכונה, וכמות הרווחים לפי מכונה בהתפלגות חודשית. בדקנו שהפונקציונליות הבסיסית של טעינת הגרפים עובדת. ובדקנו מה תהיה התצורה הפשוטה ביותר לצורך הכנסת המידע לטעינת הגרפים והצגתם.
2. הכנת שאילתה השולפת דוח מוכן מן מסד הנתונים – הגדרנו איך טבלת דוח ההזמנות תראה במסד, והערכים ההכרחיים לשמירת הרשומה, ובדקנו את נכונות שאילת השליפה באמצעות הכנת דוחות מוכנים.
3. עיבוד הנתונים, לאחר שליפת דוח מוכן- בנינו פונקציה המעבדת את הנתונים למבנה המתאים אותו הגדרנו בשלב 1, וביצענו אינטגרציה בין שלבים 1 ו 2. ובדקנו כיצד הפונקציונאליות שלהם עובדת יחד, ואת תקינותה. לאחר הכנת רשומות הדוח, בדקנו גם את לוגיקת הוולידציה, במידה והדוח לא קיים, או במידה ושדה מסוים לא נבחר.
4. הכנת שאילתת יצירת הדוח ועיבוד המידע בצד השרת, תחילה יצרנו שאילתה אשר מחזירה את הנתונים הדרושים לנו לצורך עיבוד מידע הדוח ושמירת רשומתו במסד. נתנו למתודה המעבדת את המידע ערכים המדמים הזמנות ובדקנו שהשמירה של הרשומה מתבצעת באופן הרצוי. בנוסף, ביצענו בדיקות למקרי קצה שונים, דוח לא קיים, אי הכנסת חלק משדות הדוח לשמירת הרשומה, או דוח ריק (אין מכירות). ולאחר שבניית הרשומה עבדה, באמצעות הזרקת התלות של הTIMER לתוך המתודה שבונה את הדוח, בדקנו את הפונקציונליות של הדוח.
5. בדיקת הTIMER חיברנו אליה את הפונקציונאליות של חילול הדוחות אחת לחודש- TIMER.

בדקנו את נכונותו בנפרד.

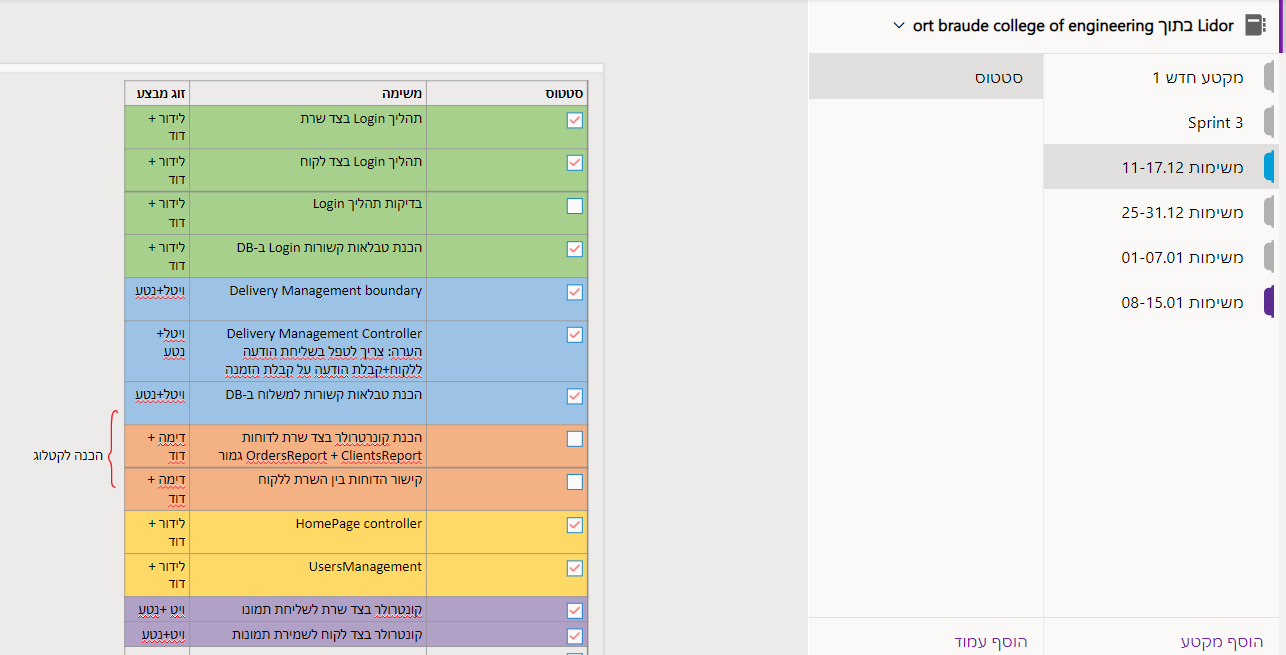
ולבסוף הרצנו דמו של התהליך על מנת לראות שהאינטגרציה בין תת התהליכים שיצרנו לתהליך בניה והצגת דוח הזמנות עובדת.

1. תהליך הפיתוח בו בחרנו, בהישענות על תהליך אג'ילי, הוא אינטגרציה העובדת מלמטה למעלה, BOTTOM UP כלומר, העבודה ולוחות הזמנים, התפלגו בצורה בה פיצלנו עבודה לתהליכים שכמעט זרים אחד לשני, ובנינו אותם בתור, תת מערכות אשר עובדות באופן עצמאי, ביצענו בדיקות , ולאחר מכן בכל שלב הוספנו עוד רובד של אינטגרציה, כלומר תחילה בדקנו את נכונותה הלוגית של המתודה, באמצעות בדיקות פשוטות, לאחר מכן בנפרד שלפנו נתונים ממסד הנתונים, ובדקנו את תאימותם, ואת נכונות השאילתה. לבסוף חיברנו את שני החלקים ובדקנו האם הפונקציונאליות עובדת ומתאימה למה שהגדרנו בתחילת העבודה. לאחר ביצוע בדיקות מקיפות ומקרי קצה ממשיכים לחבר לשאר המערכת.

כלי התוכנה בהם השתמשנו, הוא GIT ו- JUNIT , לכל תהליך פתחנו ענף ובו פיתחנו את הפונקציונליות של התהליך, אחת לשבוע ביצענו MERGE משותף והרצנו בדיקות מקומיות על מנת לוודא כי האינטגרציה בין התהליכים עובדת עד כה, ועל מנת לסנכרן את התהליכים יחדיו, ביצענו REFACTORING לכל תהליך והשארנו בקוד, מקום עתידי לצורך הכנסת התלות, לדוגמא, תלות תהליך ההזמנה בהתחברות המשתמש.

אל הספרינט השלישי, נכנסו כאשר מימשנו פונקציונליות בסיסית של מעבר בין מסכים, נתאר לוחות הזמנים וביצוע התהליכים השונים:

לדוגמא מצורפת חלוקת משימות :



בכל שבוע פיצלנו את עבודתנו לענפים שונים בGIT ובסוף השבוע ביצענו MERGE משותף, ביצענו REFACTORIG לקוד, ושמרנו בענף ה-MAIN את הגרסה העדכנית ביותר שעובדת בפרויקט.

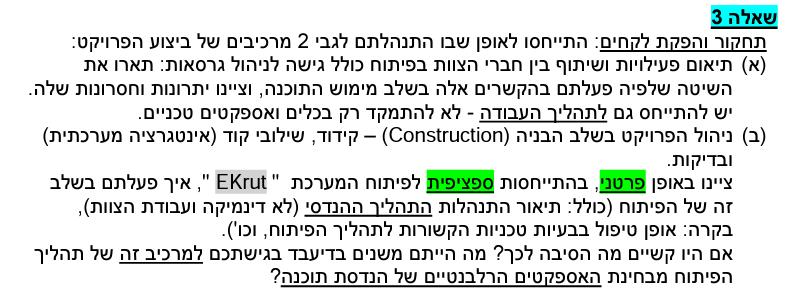
שבוע 1 – לאור ההחלטה לעבוד בשיטת ה BUTTOM UP, ניתחנו את התהליכים השונים אותם יש לבצע, וחילקנו אותם לתהליכים אשר תלותם זה בזה מינימלית ככל האפשר, ופיצלנו את העבודה, בשבוע זה מימשנו ניהול משלוחים, תהליך התחברות למערכת, פונקציונאליות דף הבית, ניהול משתמשים ושליחת תמונות ובניית דוחות.

שבוע 2- מימשנו את סימולציית ייבוא המשתמשים, ניהול המלאי, ניהול המבצעים משימות כלליות במערכת כמו בניית TIMER , ופופ אפ כללי.

שבוע 3 – מימוש תהליך ההזמנה, לאחר סגירת "הקצוות" לכל התהליכים אותם מימשנו, נותר תהליך ההזמנה בו האינטגרציה עם שאר התהליכים היא הגדולה ביותר, לאור העובדה שכלל המערכת נבנתה ותהליך הבנייה, היה תוך הסתכלות עתידית על פונקציונאליות תהליך ההזמנה, בניית התהליך הייתה בעיקר בניית מסך הקטלוג, סיום ההזמנה, ועדכון ההזמנה במסד הנתונים, תוך שימוש בפונקציונליות קיימת מתהליכים אחרים. הפעם כל הצוות השקיע את מאמציו בבניית התהליך ועבדנו יחדיו על ענף משותף.

שבוע 4- הכנת קבצי הגשה מענה על שאלות ביצוע בדיקות שונות וייצוא ל קובץ סופי

בתהליך העבודה שלנו מימשנו את עיקרון ה- synchronize & stabilize בכך שחילקנו עבודה בין חברי הצוות וביצענו איחוד של כל החלקים בתדירות שבועית.



**תשובה לשאלה 3**

1. שיתוף חברי הצוות בפיתוח התבסס על מפגשים שבועיים קבועים כאשר בכל אחד מהם, עברנו על המשימות לשבוע שנגמר, וחלוקת משימות לשבוע העוקב.

התנהלנו אך ורק עם Git. הכלי אפשר לנו להתפצל לענפים שונים, כך שחברי צוות היו יכולים לעבוד על חלקים שונים, ובתום השבוע לעשות Code review ולמזג את הקוד לגרסה עובדת.

היתרון הרב שבניהול גרסאות שכזה, הוא שמירת ההיסטוריה ללא הפחד למחוק דברים, שמירת גיבוי בענף המרכזי, צפייה נוחה בשינויים עם הערות לכל Commit.

החסרון המועט הוא שלפעמים קוד התנגש, דבר שלקח זמן על-מנת לבצע מיזוג. התגברות על קושי זה נעשה ע"י שימוש בסביבת הפיתוח VsCode המאפשרת מיזוג קל.

1. התהליך ההנדסי בו בחרנו לעבוד היה להתחיל מהיחידות המבודדות והקטנות ולאחר מכן להגיע ל'ליבת המערכת' שהיא תהליך ההזמנה.

המעגל הראשון, לדוגמה: מודול התחברות, מסך בית כללי, מודול טיפול במשלוח, מודול הפקת דוחות, יצירת ממשקים.

לאחר מכן הרחבנו את המעגל ועברנו למודולים המשניים, לדוגמה: מודול רישום וייבוא משתמשים, ניהול מלאי, ניהול שיווק.

המתוארים מעלה הם חלקים עיקריים המדמים את הרעיון הכללי, שבבסיסו עומדת התפישה ההנדסית שיש לחלק את העבודה לחלקים מבודדים ולחבר מחדש "הפרד ומשול".

המטרה שלנו הייתה להגיע לתהליך ההזמנה, התהליך העיקרי, כאשר אנחנו סגורים בוודאות כיצד התהליכים מסביב לו צריך להראות ולמנוע שינויים גדולים תו"כ.

באספקט טיפול בבעיות – ניהלנו קובץ אקסל משותף (Google docs) שהוא קובץ באגים. מטרתו הייתה לשמור תיעוד של הבאגים: איפה קרו (מחלקה, מתודה אם ידוע), מי גילה, מי פתר ותיאור נוסף. הרעיון בלשמור מסמך שכזה, שגם אם חבר צוות לא יכול היה לטפל בבאג מסוים, חשוב לשמור תיעוד. לאחר טיפול בבאג מסוים, ביצענו שוב מעבר על כל הבאגים על מנת לראות שלא הרסנו את הפונקציונליות.

הרעיון ההנדסי שהנחה אותנו היה לבצע את הקוד ביעילות אך לשים לב לזמנים. לכן, גם אם לחבר צוות היה פתרון לא יעיל במיוחד – מימשנו ולאחר מכן ביצענו Refactor ושיפרנו, ע"י זמן שהקצבנו לכך מראש.

שינויים שהיינו מבצעים:

* היינו מבצעים REFACTORING יותר מעמיק כבר באמצע העבודה, ומתכננים לעומק את עיצוב המערכת מבחינה ארכיטקטורת, מבחינת הCONTOLLERS המטפלים.
* מפתחים מתודות יותר גנריות , תוך חשיבה על תכנון וגזירת משמעויות מעמיקות על צרכי המתודה, והסתכלות על היבט נוחות הבדיקות שיש לבצע בפרויקט.