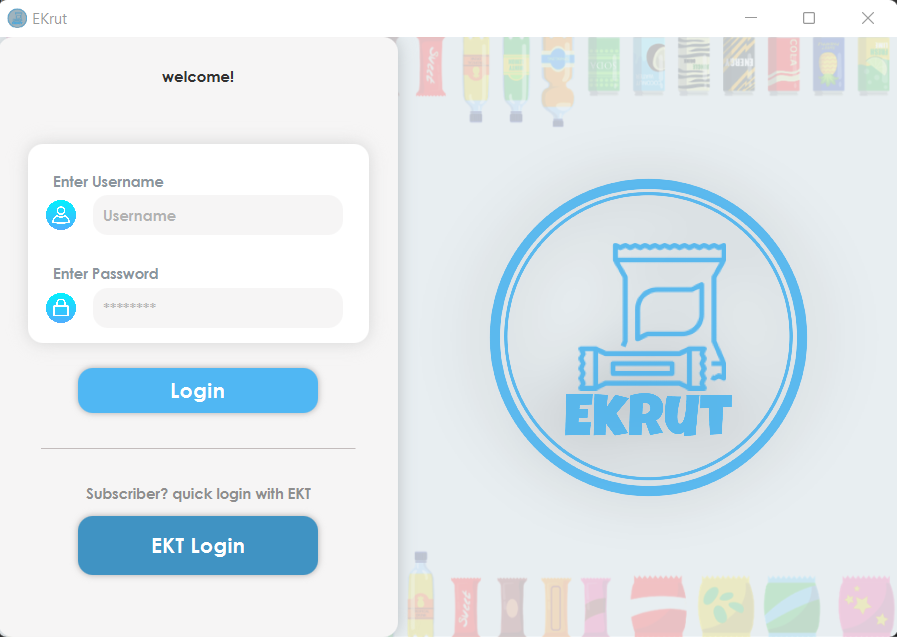
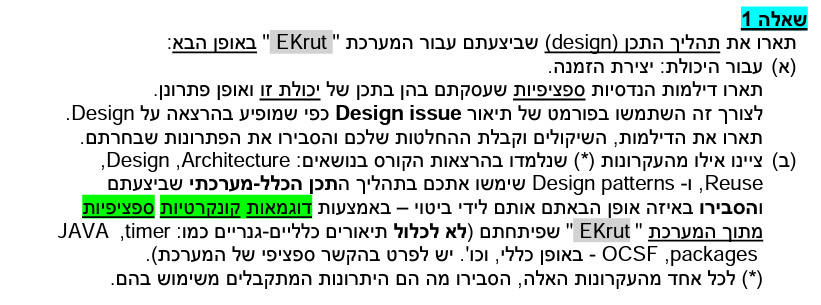
**תשובות לשאלות קבוצה G15**



שאלה 1



תשובה לשאלה 1

1. דילמות הנדסיות ספציפיות לתהליך ההזמנה שבהן עסקננו הן :
   1. דילמה שהייתה בנוגע לתהליך ההזמנה היא – האם לשים את המוצר בקטלוג בתור קובץ חיצוני או האם לבנות אותו ידנית בתוך הקוד דבר הלוקח זמן רב לבנות אך מאפשר הרצת קוד מהירה יותר
   2. דילמה נוספת היא מאחר והתנהגות הקטלוג ותהליך ההזמנה הוא שונה , נוצרה לנו הדילמה האם לאפשר ללקוח לשמור על פריטי העגלה שלו באפשור הכפתור חזרה אחורה או האם לאפס את כל בחירותיו
   3. דילמה נוספת היא האם ליצור מסך עגלה נפרד או לעשות אותו כחלונית קופצת
   4. דילמה נוספת היא איך לאפשר בדיוק יצירת הזמנה אחת החל מלחיצת כפתור תהליך ההזמנה ובחירת שיטת משלוח במידה ואנחנו בתצורת OL עד לREVIEW ORDER – SINGLETONE
2. 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 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.
   2. Architecture- השתמשנו ב-

חילקנו את ה-packages לפי השכבות הבאות:

* Boundaries-כל המסכים שבמערכת.
* controllerGui- מכילה את כל הקונטרולרים של המסכים במערכת.
* Controller- אחראי לתקשורת בין ה-controllerGui השונים בעלי קשר משותף.
* Client- אחראי ללוגיקה של שליחת וקבלת הודעות מהשרת.
  1. Design patterns – השתמשנו בתבנית העיצוב singleton מסוג Observable–המאפשרת מופע יחיד של מחלקה.

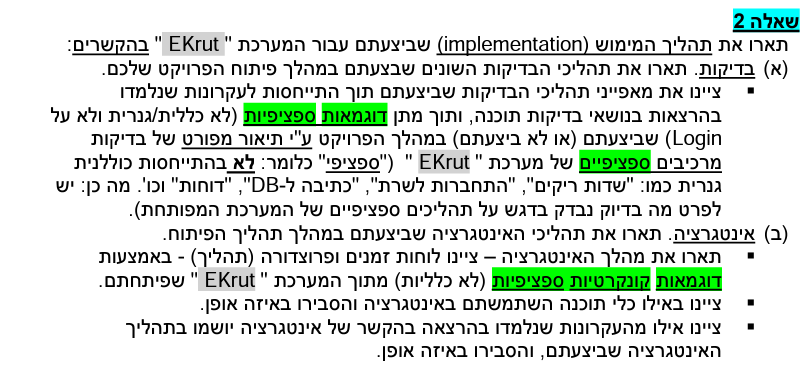
לדוגמא: אפשרנו שהחיבור ל- DBשמתבצע במחלקה MySqlClass הינו חיבור יחיד בלבד ולא ייווצרו מופעים נוספים של ה-connection אלא נקבל את אותו מופע על ידי שימוש במתודה getConnection().

REUSE זה גם OCSF אפשר לציין

ירושה, מנשק

??????

שאלה 2

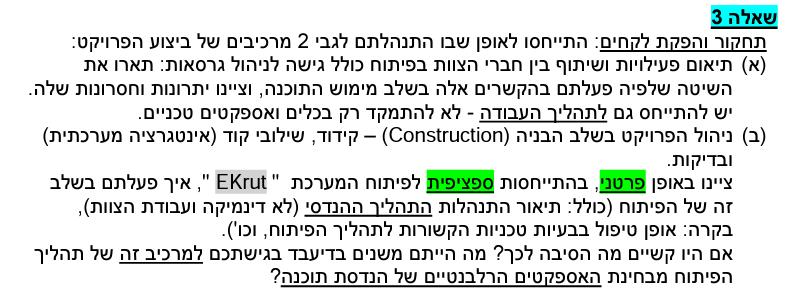


תשובה לשאלה 2

ב) מה הוא רוצה? מה זה כלי תוכנה עכשיוווווווווו מהההההההה די עם הקורס הזה

כלי תוכנה- git

שאלה 3



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: