# קבוצה 4:

**יזהר אננייב** [Izhar\_ananiev@hotmail.com](mailto:Izhar_ananiev@hotmail.com)

**ליאור גל** [liorgal28@gmail.com](mailto:liorgal28@gmail.com)

**כפירי ביטון** [kfirbitonn@gmail.com](mailto:kfirbitonn@gmail.com)

**איתי עלמני** [itaialmani@gmail.com](mailto:itaialmani@gmail.com)

**רון בג-צבי** [benzvi.ron@gmail.com](mailto:benzvi.ron@gmail.com)

**הוגש בתאריך:** 28/01/2018

**שאלה 1:**

1. במימוש הקטלוג, נפגשנו עם מספר דילמות הנדסיות:

* העברת התמונות מה-server אל ה-client ולהפך: כאשר הקטלוג נטען, עלינו להציג את התמונות של כל המוצרים בו. על מנת להציג את התמונות הללו, עמדה בפנינו דילמה: כיצד להעביר את התמונות האלו דרך ה-socket. על מנת להעביר אובייקט בעזרת socket, עליו להיות (לממש את הממשק) serializable. את התמונות תכננו להציג בעזרת ImageView אשר מקבל path, או InputStream שנוצר על ידי path או מערך בתים. הבעיה היא שה-path הוא ב-server (או כאשר מדובר על עדכון הקטלוג, ב-client), אך התמונה אמורה לעבור אל הצד השני ולכן ה-path אינו חוקי.

לכן מצאנו את הפתרון הבא - עלינו להפוך את התמונה למערך של בתים בעזרת BufferedInputStream, ונשלח את המערך הזה אל הצד השני. בצד השני, המערך יפורש בחזרה בתור תמונה בעזרת ByteArrayInputStream.

* בכל טעינת קטלוג מחדש, עלינו לשלוף מחדש את הנתונים ממסד הנתונים למקרה שהנתונים במסד השתנו בגלל האפשרות שמספר clients יהיו מחוברים בו זמנית – נתקלנו בדילמה הזו ולא הצלחנו להחליט בהתחלה מה חשוב יותר: האם יותר חשוב שטעינת הקטלוג תיהיה מהירה יותר אך המידע בה לא יהיה מדויק, או שאולי ייקח יותר זמן לטעון את הקטלוג בגלל שליפה מחדש של המוצרים אך המידע שבו יהיה הנכון והמעודכן ביותר. לבסוף, הגענו למסקנה שיש אפשרות נכונה יותר, שהיא ללא כל ספק מידע מעודכן ונכון יותר, למרות זמן טעינה גבוה יותר. לכן, בעת טעינת הקטלוג, אכן נשלחת בקשה מה-client אל ה-server, והמוצרים שבו בכל פעם נטענים מחדש.

1. הנושאים שנלמדו בהרצאות 6-11 אשר באו לידי ביטוי בתהליך התכן המערכתי שביצענו:

* **הרצאה 6 – Reuse:**

לאורך כל תהליך התכנון המערכתי, רצינו ליצור תשתית מערכתית בסיסית (האב טיפוס הראשוני) שבה כל חברי הצוות ישתמשו. באמצעות שימוש חוזר בקוד הצלחנו למקסם את עבודתנו הן מבחינת תאום רמת הקוד והן מבחינת יעילות וחתירה לגנריות. כמובן, כמו שמתואר בהסברי ההרצאות, שימוש חוזר בקוד בא לידי ביטוי גם בשימוש בעיצובים קיימים, ספריות ומחלקות קיימות וירושה מהן (לדוגמה המחלקה OCSF והעבודה מולה).

בנוסף, היינו צריכים להתגבר על מספר מכשולים הקשורים לעבודה עם שימוש בקוד חוזר, מכשולים כדוגמת התאמת הקוד החוזר לצרכים שלנו.

* **הרצאה 7- גישות שונות לעיצוב:**

הגישה שאנו נקטנו בעיצוב המערכת שלנו היא הגישה הנפוצה ביותר לתכנון מערכת והיא "Mix of top-down and bottom-up approaches"

גישה זו, הינה שילוב של שתי גישות עיצוב, כאשר תחילה עיצבנו את המערכת מהרמה הכללית ביותר ולאחר מכן ירדנו אל הפרטים הקטנים, בשילוב של פיתוח תשתית שתאפשר לנו לעשות שימוש נרחב בקוד חוזר.

* **הרצאה 7- עקרונות המובילים לעיצוב טוב:**

במסמך ההרצאה מוצגים 11 עקרונות מובילים לעיצוב טוב, נתייחס למספר עקרונות שנלקחו בחשבון בתהליך התכן המערכתי.

1. "הפרד ומשול" – חלוקה של הפרוייקט לתתי משימות שחולקו באופן מסודר ושווה בין חברי הצוות על מנת שכל חבר בקבוצה יהיה ממוקצע בחלקו ויוכל לחלוק את הידע עם שאר החברים.בנוסף, חלוקה לתתי משימות באופן שווה גורמת לייעול זמנים מיטבי. כמובן, כל חבר צוות למד לעומק קומפוננטה בפרוייקט, מה שאיפשר לו ללמד במהירות גדולה יותר את שאר חבריי הצוות עליה, מאשר שכולם יצטרכו ללמוד הכל יחד.
2. רמת גנריות מקסימלית – אחד הנושאים העיקריים שנתנו להם דגש בעבודת התכן. ביצענו עוד בשלב האב-טיפוס בניה של קוד גנרי ככל שניתן, כאשר הבנו כי ככל שהקוד יהיה יותר גנרי כך השינויים שיהיו בהמשך יהיו מינוריים והן המערכת תוכל להיות מלמדת עבורנו הרבה מעבר לפרוייקט זה.
3. שימוש חוזר בקוד – הוסבר בפירוט בסעיף הראשון.
4. שימוש חוזר בעיצובים ארכיטקטוניים קיימים – בתכנון המערכת, ידענו כי קיימים מספר עיצובים ארכיטקטוניים שניתן להשתמש בהם ולחסוך זמן בעבודה. אחד השימושים הבולטים באים לידי ביטוי בשימוש במודל ה-Client Server (OSCF).

* **הרצאה 9 – עיצוב ממשק המשתמש:**

לדעתנו, עיצוב ממשק המשתמש הינו חלק השווה ברמתו לתכנון ועיצוב הפונקציונליות של המערכת. השקענו המון זמן בעיצוב ממשק המשתמש תוך התחשבות ב"חווית משתמש" ויכולת למידה מהירה של המערכת ע"י המשתמש, כל זאת ממומש באמצעות דיאלוגים ברורים, מעבר פשוט בין מסכים, פונטים מיוחדים, תמונות וכו'.

**שאלה 2:**

במהלך העבודה על הפרוייקט (החל משלב פיתוח האב טיפוס) ביצענו בדיקות רבות שהמטרה העיקרית שלהן היא בדיקת נכונות הקוד וממשק המשתמש שנכתב. בנוסף, כאשר מדובר בפרויקט תוכנה גדול המשלב מספר חברי צוות, ביצענו מעקב ובדיקות עבור שינויים בחלקי קוד חופפים. דוגמה ספציפית הממחישה את השימוש בתאוריה הנלמדה בקורס הינה העבודה הנרחבת בביצוע בדיקות יחידה (עבודה עם JUnit) למספר רכיבים שונים (ביניהם, הרכיבים שנדרשו במטלה מספר 3).

את בדיקות היחידה חילקנו בין חברי הצוות, כאשר כל חבר צוות עשה בדיקות עבור חלקו בפרוייקט. במהלך כתיבת הבדיקות לנגד עינינו עמדו מספר אפשרויות לביצוע בדיקות היחידה:

**"קופסה שחורה" –** בחינת התפקוד של הרכיב בתוך המערכת, בדיקה של נכונות הפלט והאם הפלט חוקי או לא. בדיקות אלו בוצעו בעיקר בתהליכים מסובכים הדורשים מעבר בין מסכים והידיעה כי פלט שנוצר במסך אחד יהווה קלט למסך אחר (תהליך ביצוע הזמנה).

**"קופסה לבנה" -** בחינת המבנה הפנימי של הרכיב, בדגש על בדיקת מסלולי החישוב, נכונות החישוב ונכונות ההחלטות הלוגיות. בדיקות אלו עזרו לנו מאוד בבדיקות פונקציונליות מעט מורכבת יותר של חישובים (ביטול הזמנה הדורש עדכון מלאי, החזריים כספיים וכו').

**שאלה 3:**

1. תחקור והפקת לקחים שביצענו עבור תיאום הפעילויות ושיתוף הפעולה בין חברי הצוות בפיתוח וניהול הגרסאות:

כאשר התחלנו את שלב מימוש הפרויקט, מהר מאוד הבנו כי העבודה בתיאום עם מספר חברי צוות היא נושא מורכב ורגיש המשפיע על שאר חברי הצוות באופן ישיר (לא אחת, בעת עדכון גירסה ע"י אחד מהחברים, אשר מתעדכן אוטומטית אצל כל חברי הצוות בעזרת ה-git, הפרויקט לא פעל לשאר חברי הצוות, לרוב בעדכון השרת ומסד הנתונים). ואכן כמו שחזינו, בעיה זאת חזרה מספר פעמים ואף הביאה לתסכול וייאוש. לכן, מהר מאוד פיתחנו לנו מעין הרגל שכל שינוי שחבר צוות אחד מבצע בקוד הרלוונטי לחבר צוות אחר, באחריותו לעדכן על כך בקבוצת הפרויקט המשותפת ב-WhatsApp. היתרון המרכזי בפתרון זה מובן מאליו – בפתרון זה, כל חברי הצוות מעודכנים על שינויי קוד, וכך גם נחסך זמן רב ותסכול.

יתרה מכך, נהגנו להיפגש אחת לשבוע, כאשר כל חבר צוות הציג את עבודתו לשאר הקבוצה. היתרון המרכזי בשיטה זו הוא קבלת משוב תמידי ותיאום ציפיות מול שאר חברי הצוות.

1. תחקור והפקת לקחים שביצענו עבור שילובי קוד ובדיקות:

כחלק מתהליך בנית פרויקט הממומש ע"י מספר מפתחים ישנו החלק הקריטי של איחוד קטעי הקוד ושילוב הקומפננטות השונות. עבור חלק קריטי זה, הקדשנו מפגש מיוחד ועמוס כאשר את האיחוד ביצענו באיטיות וזהירות תוך ביצוע משותף של התאמות לחלקי הקוד המשותפים. יתרה מזאת, עבור הבדיקות קיבלנו החלטה כי כל מפתח יבדוק את הקוד שהוא כתב, שכן, ברור כי הוא יודע הכי טוב את חלקו ובמידה ויגלה טעויות שיתגלו בשלב ה-QA הוא יוכל לפתור את הבעיות בקוד במהירות.

לדעתנו, התנהלות זו תרמה רבות להתקדמות הפרויקט בדרך לתוצר הסופי, באמצעות עבודה זו, כל חבר היה מעורה בחלקי הקוד שנכתבו ע"י שאר המפתחים. אנו חושבים כי כאשר מפתחים פרויקט גדול, זמן QA צריך להיות ארוך יותר ממה שניתן בפועל, שכן יש שגיאות שמתגלות לאורך זמן ושגיאות שקורות בעקבות ריבוי משתמשים.

לדעתנו, הגישה שבה עבדנו היא הנכונה ביותר לנו כקבוצה, ולכן, גם בדיעבד, היינו מתנהלים בצורה זהה.