מטלה 3 - שאלות פתוחות

תאריך הגשה: 20.06.2020

קבוצה 1

ולדיסלב ברקנס vladigr1@gmail.com
אלרואי כהנא elroye77@gmail.com
אלדד קרונפלד eldadkro@gmail.com
ליעד וקסמן liadvv@gmail.com
לי אור וונש liorwunsch@gmail.com

שאלה 1 סעיף א'

(א) בהקשר של יכולות **מעקב פעילות** ע"י מחלקת השיווק, תארו דילמות הנדסיות ספציפיות שעסקתם בהן בתכן של יכולות אלה. השתמשו בפורמט של תיאור Design issue כפי שמופיע בהרצאה על Design.
תארו את הדילמות, השיקולים וקבלת ההחלטות שלכם והסבירו את הפתרונות שבחרתם.

במהלך הפרויקט, עבור הדרישה למעקב פעילות ע"י מחלקת השיווק, נתקלנו בדילמות ההנדסיות הבאות:

1. **כיצד נמלא את טבלת 'המבצעים' בממשק המשתמש בעמודות הנדרשות של מילוי דו"ח תגובות עבור מבצע שיווק?**

שאלה זו נובעת מצורך בהצגת נתונים משתי טבלאות שונות במסד הנתונים, שנקבעו לפי ה- design שתוכנן בשלב התכן.

אופציה 1 - נפנה לשרת על מנת למשוך נתונים מטבלה אחת ולמלא את טבלת 'המבצעים' בעמודות השייכות לה ולאחר מכן נפנה לשרת פעם שנייה כדי למשוך את הנתונים בטבלה השנייה ולמלא את העמודות שנותרו בטבלת 'המבצעים'.

אופציה 2 - פונים לשרת פעם אחת, הוא יוצר מופע חדש של מחלקה חדשה וייעודית המכילה את כל העמודות הנדרשות בטבלת 'המבצעים' ובכך מושך את המידע הרצוי משתי הטבלאות הדרושות ומציב את הנתונים בתוך השדות המתאימים במחלקה.
לאחר סיום הצבת הנתונים, השרת מחזיר את מופע המחלקה שיצר וכך ברגע שה-client מקבל את אותו מופע בצד שלו, טבלת המבצעים תתמלא כולה בקריאה אחת.

דילמות :
באופציה 1 יש צורך בפניה לשרת פעמיים ולקבל את המידע בחלקים שונים.
לעומת זאת, באופציה 2 נדרש לבנות מחלקה חדשה שלא הוגדרה בשלב התכן כלל.

החלטה :
החלטנו לבצע מימוש של אופציה 2 על מנת לחסוך בפנייה נוספת לשרת ולקבל את כל הנתונים הרצויים בצורה נוחה יותר, על חשבון שינוי בתכנון המערכת הראשוני.

2. **כיצד נוכל לדעת מתי מבצע עדיין תקף או שכבר פג תוקפו?**

שאלה זו נובעת מהצורך למנוע יצירת דו"ח תגובות חדש על מבצע שעדיין לא הסתיים.

אופציה 1- ניצור שדה חדש בטבלת המבצעים במסד הנתונים, הנקרא finished שמייצג את המצב של כל מבצע (האם זמן הסיום של המבצע הגיע או לא)

אופציה 2 - נבדוק לפי זמן סיום המבצע, האם המבצע הסתיים ואם אכן הסתיים, ניתן ליצור דו"ח עבור אותו מבצע.

דילמות :
באופציה 1 יש צורך לעדכן את השדה שהוספנו לפי שעה, לכך נצטרך ליצור תהליך שרץ ברקע ודורש משאבים.
מצד שני, אין צורך לבצע השוואות מסובכות של תאריכים (שעבורם נתקלנו בקשיים מסוימים) כמו באופציה 2, אלא להשתמש בשאילתה פשוטה עבור השדה המתאים.
לעומת זאת, באופציה 2 אין צורך לתחזק את השדה finished.

החלטה :
החלטנו לממש את אופציה 2 כי ניהול השדה הנוסף שהוצע דורש מאמץ רב יותר ומימוש אופציה זו פחות יקר מבחינת זמן חישוב כי פניות לשרת הן יקרות ולוקחות זמן.

3. **כיצד נשמור מידע ליצירת דו"ח אפיון תקופתי של לקוחות?**

שאלה זו נובעת מהצורך בשמירת נתוני לקוח ונתוני חברת דלק באותו דו"ח.

אופציה 1 - ניצור טבלה חדשה במסד הנתונים שבה יישמרו עבור כל לקוח - שקלול של רמת פעילות הקנייה שביצע בתקופה מסוימת לכל חברת דלק (מספר הרשומות לכל לקוח כמספר חברות הדלק שתידלק בהן).

אופציה 2 - ניצור טבלת ביניים בין לקוח לחברת דלק שיהיו בה נתונים מזהים משתי הטבלאות ורמת פעילות הקנייה המתאימה לכל תאריך מסוים.

דילמות :
באופציה 1, נצטרך לבצע השוואות של מספר רב של שדות בטבלה מכיוון שיהיו בה נתונים של כל הלקוחות בכל התקופות האפשרויות שבהן הייתה להם פעילות קנייה.
בנוסף, כמות הבקשות מהשרת תהיה גבוהה עבור משיכת כמות מידע קטן אך עדכון יחיד לכל רכישה בחברת דלק מסוימת יהיה פשוט ומהיר יותר.
מצד שני, לא נצטרך להתעסק עם שאילתות שפועלות על מספר טבלאות במקביל.
לעומת זאת, באופציה 2 הנתונים יהיו מחולקים לפי פרמטרים ספציפיים יותר ולכן יהיה ניתן ליצור את הטופס ביותר קלות.
בנוסף, ייקח פחות זמן לפנות פעם אחת למסד הנתונים במקום כמספר חברות הדלק במערכת.

החלטה :
החלטנו להשתמש באופציה 2, כי סידור הנתונים בצורה בהירה עזר לנו להבין בקלות איך לממש את התהליך.

שאלה 1 סעיף ב'

(ב) ציינו איזו מהעקרונות שנלמדו בהרצאות הקורס בנושאים: Architecture, Design, Reuse, ו- Design patterns באו לידי ביטוי בתהליך **התכן הכלל-מערכתי** שביצעתם והסבירו באיזה אופן – באמצעות דוגמאות רלבנטיות וספציפית מתוך המערכת "MyFuel".

- בפרויקט השתמשנו במספר דפוסי ארכיטקטורה:

1. **דפוס ארכיטקטוני שלושת השכבות: boundary, control, entity.**

המטרה הייתה ליצור הפרדה בין חלקים שונים בקוד ושייכות בין מחלקות שונות, למשל -
המחלקה שמציגה מידע לגבי הזמנת דלק ביתי היא **CustomerWindow** בשכבת ה-boundary והמחלקה **CustomerController** מנהלת אותה וקובעת את הלוגיקה שמאחורי המסך כמו יצירת בקשה ושליחה לשרת, מחלקה זו יושבת בשכבת ה-control.
על המידע השייך להזמנת דלק ביתי המחלקה **HomeFuelOrder** אחראית והיא מכילה שדות אשר מאפיינים את ההזמנה, לכן מחלקה זו שייכת לשכבת ה-entity.

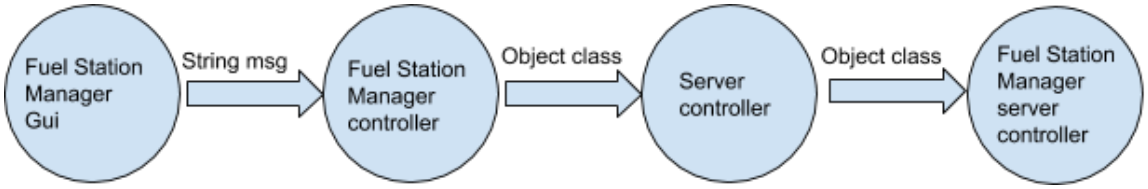
2. **דפוס ארכיטקטוני שרת לקוח: חלוקת הפרויקט לרכיב שרת ורכיב לקוח.**

דפוס זה עונה על הצורך של התוכנה בטיפול במספר רב של משתמשים ובביצוע בקשות משתמשים במהירות.
זה בא לידי ביטוי אצלנו בכך שכל משתמש במערכת הוא בעל רכיב לקוח והמערכת בנויה בעיקר מרכיב השרת שמתקשר גם עם מסד הנתונים.
בפרויקט שלנו יש מגוון רחב של משתמשי מערכת כמו לקוחות של חברות הדלק, ספק של חברת דלק , מנהל תחנת דלק וכו'.
לאחר שאומתו פרטי ההתחברות של המשתמש, השרת מנתב את המשתמש לחלון המתאים לו לפי הפעילות אותה מורשה לבצע.
בנוסף, דפוס זה מקנה אבטחת מידע ומידור בין משתמשים, למשל - מנהל תחנת דלק פז לא יכול לראות נתונים של תחנת דלק סונול או תחנת דלק פז אחרת.

- העקרונות המרכזיים שהנחו אותנו בשלב התכן מבחינת Design:

1. **Divide and Conquer:**

מחלקות ה-boundary אחראיות בעיקר על האינטראקציה של המשתמש עם המסך שהוא רואה והשינויים שקורים במסך.
מחלקות ה-controller שבצד ה-client אמונות על תרגום בקשות מממשק המשתמש ושליחתן לטיפול השרת.
מחלקות ה-controller שבצד ה-server מטפלות בבקשות אלו ומנתבות אותן הלאה ל-database.
לדוגמא, כאשר מנהל תחנת דלק לוחץ על הפקת דו"ח רבעוני, בקשה מתאימה נשלחת לcontroller של מנהל התחנה בצד ה-client, שיוצר מופע של המחלקה "מנהל תחנה" עם הנתונים המתאימים ולאחר מכן שולח אותו לשרת.



2. **Increase Cohesion:**

בשלב התכן הגדרנו מחלקות אב שיכילו תהליכים המשותפים למחלקות הנדרשות שאיתן עבדנו.
למשל, בכל חלונות המשתמש לאחר התחברות יש כפתור sign-out והתהליך הוא זהה לכל המסכים, לכן הגדרנו מחלקת אב בשם UserWindow שתממש זאת.

- בפרויקט השתמשנו בעקרון ה-Reuse:

1. OCSF, JavaFX, JDBC.
2. מכיוון שיש תהליכים שמזינים נתונים לטבלאות משותפות במסד הנתונים, מימשנו מתודות שמכניסות רשומות חדשות לטבלאות והשתמשנו בהן שימוש חוזר המותאם לכל תהליך.

- בפרויקט השתמשנו ב-Design Patterns:

1. **Singleton:**

נועד למקרים בהם מעוניינים להגביל את יצירת המופעים של מחלקה מסוימת למופע יחיד.
לדוגמא, במערכת שלנו אין צורך ביותר ממופע אחד של CustomerContoller ולכן מימשנו אותו כ-Singleton.

2. **Abstract Occurrence:**

ישנן מחלקות שמכילות מידע המאפיין אותן, אך גם מחלקה אחרת.
לכן, למחלקה כזאת יהיה מופע של המחלקה המקבילה כמתואר.
לדוגמא, תחנת דלק מאופיינת גם ע"י חברת דלק ולחברת דלק יש רכיבים המאפיינים אותה, לכן למחלקה "תחנת דלק" נוסיף מופע של המחלקה "חברת דלק".

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 Login) שביצעתם (או לא ביצעתם) במהלך הפרויקט (ע"י תיאור מפורט של בדיקות מרכיבים ספציפיים של מערכת "MyFuel").

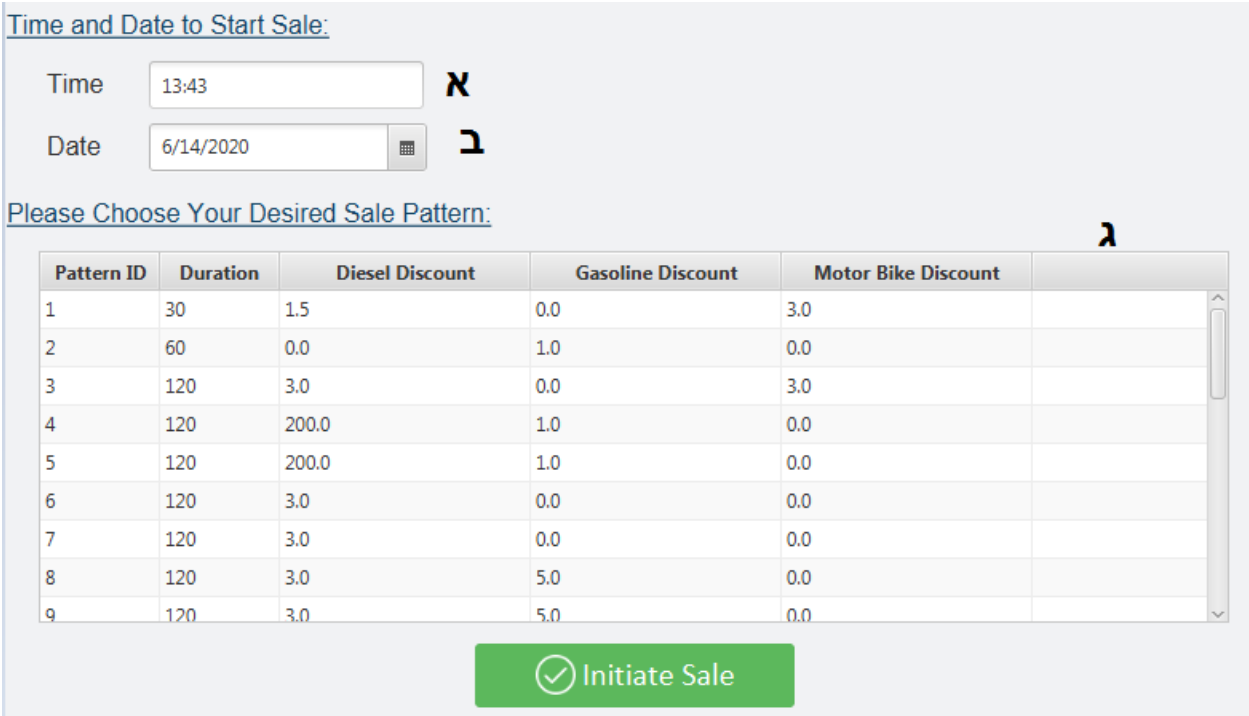
במהלך פיתוח הפרויקט, ביצענו בדיקות שונות שהמטרה העיקרית שלהן הייתה בדיקת נכונות הקוד שנכתב ונכונות ממשק המשתמש.

- לפי עיקרון **הקופסה השחורה**, נשלחות שאילתות למסד הנתונים, במידה והתהליך נכשל אנו תופסים את השגיאה ושולחים חזרה אל השרת הודעת שגיאה מתאימה.

דוגמא לבדיקת קופסה שחורה:
בתהליך "תדלוק מהיר", אנו רוצים לקבל ממסד הנתונים את נתוני הלקוח שמשפיעים על המחיר שישלם ולכן בזמן משיכת המידע ממסד הנתונים, מתבצעות בדיקות על נכונות המידע לפי הציפיות שלנו והחזרת הודעת שגיאה מתאימה למשתמש במקרה הצורך.

- לפי עיקרון **הקופסה הלבנה**, ביצענו בדיקות קלט שהמשתמש הכניס בממשק המשתמש.
  למשל, מספר עם תוים לא תקינים, מספר שלילי, תאריך שכבר עבר וכו'.

דוגמא לבדיקות קופסה לבנה שהתבצעו בתהליך "התחלת מבצע":



Pattern ID	Duration	Diesel Discount	Gasoline Discount	Motor Bike Discount
1	30	1.5	0.0	3.0
2	60	0.0	1.0	0.0
3	120	3.0	0.0	3.0
4	120	200.0	1.0	0.0
5	120	200.0	1.0	0.0
6	120	3.0	0.0	0.0
7	120	3.0	0.0	0.0
8	120	3.0	5.0	0.0
9	120	3.0	5.0	0.0

א. בדיקת הכנסת זמן:
בשדה זה המשתמש מתבקש להכניס זמן ביום שבה יתחיל מבצע, לכן על מנת למנוע מהמשתמש מלהכניס ערך לא רצוי אנו בודקים תחילה כי הערך שהכניס מכיל בתוכו ':'.
במידה ואכן מכיל ':', אנו בודקים שהשעה והדקות אכן תקינות כלומר בין 0 ל- 23 ובין 0 ל – 59 בהתאמה.
אם אכן הם בטווח הרצוי אנו גם רוצים למנוע מצבים שבהם הוכנסו תווים או אותיות, משום שמתבקש להכניס רק ערך מספרי.
בדיקה זו מבוססת לפי הערכים שנקבעו **בבדיקות הקבלה**.
בעזרת בדיקות אלו אנו שומרים על ערך זמן נכון על מנת שהמשתמש יוכל להמשיך בפעולות הרצויות.

ב+ג.
המשתמש מתבקש לבחור תבנית מבצע מהטבלה, לכן לאחר לחיצה על הכפתור "initiate sale" אנו בודקים האם נבחרה שורה כלשהי מהטבלה, אם לא נבחרה שורה תופיע הודעה במסך על שגיאה ומיקום השגיאה.
במידה והמשתמש בחר שורה מסוימת יש לאמת עם מסד הנתונים כי לא קיים מבצע בתאריך שהוכנס בשדה ב'.
אם אכן קיים מבצע עתידי בתאריך זה, המשתמש יקבל הודעה כי לא ניתן להתחיל מבצע באותו תאריך (תופיע הודעה במסך על שגיאה ומיקום השגיאה).

שאלה 3 סעיף א'

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, וציינו יתרונות וחסרונות שלה. יש להתייחס גם לתהליך העבודה - לא להתמקד רק בכלים ואספקטים טכניים.

כדי לנהל פרויקט מוצלח, הפרויקט כלל מספר שלבים שהוסכמו בקבוצה.
אחד השלבים היה לקבוע משימות ותיאום זמן לכל פעילות.
שלב זה כלל:

1. קביעת פגישה קבוצתית שהתקבלה מטלה חדשה.
הפגישה כללה:
   - לעבור ביחד על משימות המטלה ולקבוע תעדוף מבין סעיפי המטלה
   - משימות גדולות חילקנו לתתי משימות
   - להתחלק לצוותים
   - לקבוע תאריך משוער לסיום המשימה
   - לוודא שאכן ההגשה של המטלה אפשרית לפי החלוקה שנקבעה

2. בשלב העבודה, הצוותים מתאמים בין עצמם על איזו פלטפורמה תתבצע העבודה
3. אחד מאנשי הצוות יוצר מוסכמות עבור כל תת צוות שלו ואז קובע משימות
את המשימות הוא מציג בדומה למתואר בשלבים הנ"ל

החסרון המרכזי בשיטה זו היא שצריך שמישהו ייקח את המושכות בשלב 3 וזה לא מתאים לכל צוות.
היתרון הוא שהמטלות לא נופלות על בן אדם אחד ובכל צוות יש פרספקטיבות שונות על המטלה.

פלטפורמות שאיתם עבדנו היו :

1. סביבת google suite חינמית (docs, slides):
יצירת מסמך שיתופי שכל אחד יכול לעבוד לפיו ולראות את ההתקדמות של חברי הצוות האחרים, מה שמונע מעבודה כפולה ובזבוז זמן יקר.
היתרון בכלי זה הוא שכולם יכולים לעקוב אחרי תהליך הפתרון ולהתאים את עצמם אליו.

2. visual paradigm repository:
בשלב התכן, כל אחד מחברי הצוות עבד בסנכרון של הדיאגרמות השונות מכיוון שרוב הדיאגרמות היו מבוססות על דברים דומים.
חסרון שנוצר משימוש ב-repository הוא חוסר ניסיון בשימוש בו וקשיים בהעלאת דיאגרמות שהועלו אליו בין מחשבים.

3. git:
בעזרת כלי זה שיתפנו קבצי קוד, דיאגרמות ומסמכים שונים כך שכל חברי הצוות עבדו עם תוכן מעודכן.
לאחר בניית קובץ ראשוני לכל תת-משימה שיהיה משותף לכולם, כל אחד מתתי-הצוותים עבד ב-branch שמיועד לאותה משימה וזאת על מנת לא לגרום לדריסה של עבודה של צוותים אחרים.
אף שלא נעשה שימוש בשחזור גרסה, יתרון גדול היה השוואה של תוכן הפרויקט ובעיקר קבצי הקוד במטלה 3 עם גרסאות קודמות ומעקב אחרי ההתקדמות בהם.
חסרונות בולטים הם שהכלי דורש הכנה מראש וניסיון בשימוש בו, בנוסף merge עבור ה-branch-ים לא תמיד נעשה בצורה חלקה וגרם לבאגים ולכן נדרש שילוב קוד פרטני שלקח זמן.

4. google calendar:
לאחר הגשת todolist עברנו לפלטפורמה שמופצת אצל כולם.
אופן העבודה עם הכלי הייתה לקבוע מפגשים והגשות ביומן.
אירועים שהוספנו כללו:

- פגישות צוות
- הגשות
- תאריכי סיום משימה

היתרון בכלי זה שהוא נמצא בהישג יד, כל אחד יכול להתעדכן בטלפון שלו על הלו"ז הקיים ולעתים להציע שינוי שלו.

שאלה 3 סעיף ב'

(ב) שילובי קוד (אינטגרציה מערכתית - לאחר הפיתוח הראשוני) ובדיקות. ציינו באופן פרטני, בהתייחסות ספציפית לפיתוח המערכת " MyFuel ", איך פעלתם בשלב זה של הפיתוח (למשל: תיאור התנהלות התהליך, 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?

בתחילת המטלה השלישית כבר היה קוד מוכן אשר שימש את בסיס המערכת.
התקיים מפגש ובו :

- התבצעה חלוקת עבודה לכל חבר צוות אשר לא תגרום לקונפליקטים בין העבודות
- הסכמה על פורמט כתיבת הקוד
- הסברים על המחלקות הקיימות בבסיס (core)
- העלאת הקוד ל-branch האישי בפרויקט ב-GitHub
- תאריך יעד שבו יתקיים המפגש הבא ויוצגו תוצאות העבודה

שימוש במוסכמות הנ"ל הביא לתיאום מלא בין חברי הקבוצה באופן כתיבת הקוד ומימושו.
במהלך פגישת הביקורת, נבדקים העבודות של חברי הצוות הכוללים:

- פורמט הקוד
- פעולת הממשק
- הזנת נתונים תקינה במסד הנתונים
- האם הקוד עובד לפי הדרישות שנקבעו לו

כל פעם שחבר צוות סיים לבנות את החלק שלו, חבר צוות אחר התנדב לבצע בדיקות מתאימות ולהעיר על קונפליקטים שנמצאו.
לאחר שתוקנו כל השגיאות, היה ניתן לבצע אינטגרציה של חלק זה עם שאר הפרויקט.
האינטגרציה התבצעה בעזרת חבר צוות קבוע שלקח על עצמו את תפקיד שילובי הקוד ע"י שימוש בתוכנה TortoiseGit.

הבעיות שעלו:

- חלוקת עבודה לא הייתה תמיד נכונה
- שכפול קוד בלתי נמנע בתכולות שונות בקוד
- זליגה של חברי צוות לעבודה של אחרים
- קצב עבודה לא אחיד בין כולם
- חוסר ניסיון של חלק מחברי הצוות בדברים נחוצים
- הבנה לא נכונה של הדרישות

טיפול בבעיות:

- עזרה של חברי הצוות אחד לשני מעבר לתכולה שקיבלו
- צמצום הקוד המשוכפל כמה שאפשר בסוף תהליך הפיתוח
- התייעצות עם המרצים ותיקון אי הבנות
- הערכה מחדש של חלוקת העבודה וחלוקה חדשה לפי הצורך

השינויים שהיינו רוצים לעשות:

- מעקב אחרי עבודות של חברי הצוות ושינויים נחוצים ב-core
- תיאום בבניית הקוד בכל התכולות
- להפוך את התהליך של העברת מידע מהשרת ללקוח ולהיפך ליותר פשוט
- תיקון באגים ב-core לפני התחלת פיתוח התכולות