**קבוצה מספר 1**

**לי חוגי** - 311178552 - [leehugi93@gmail.com](mailto:leehugi93@gmail.com)

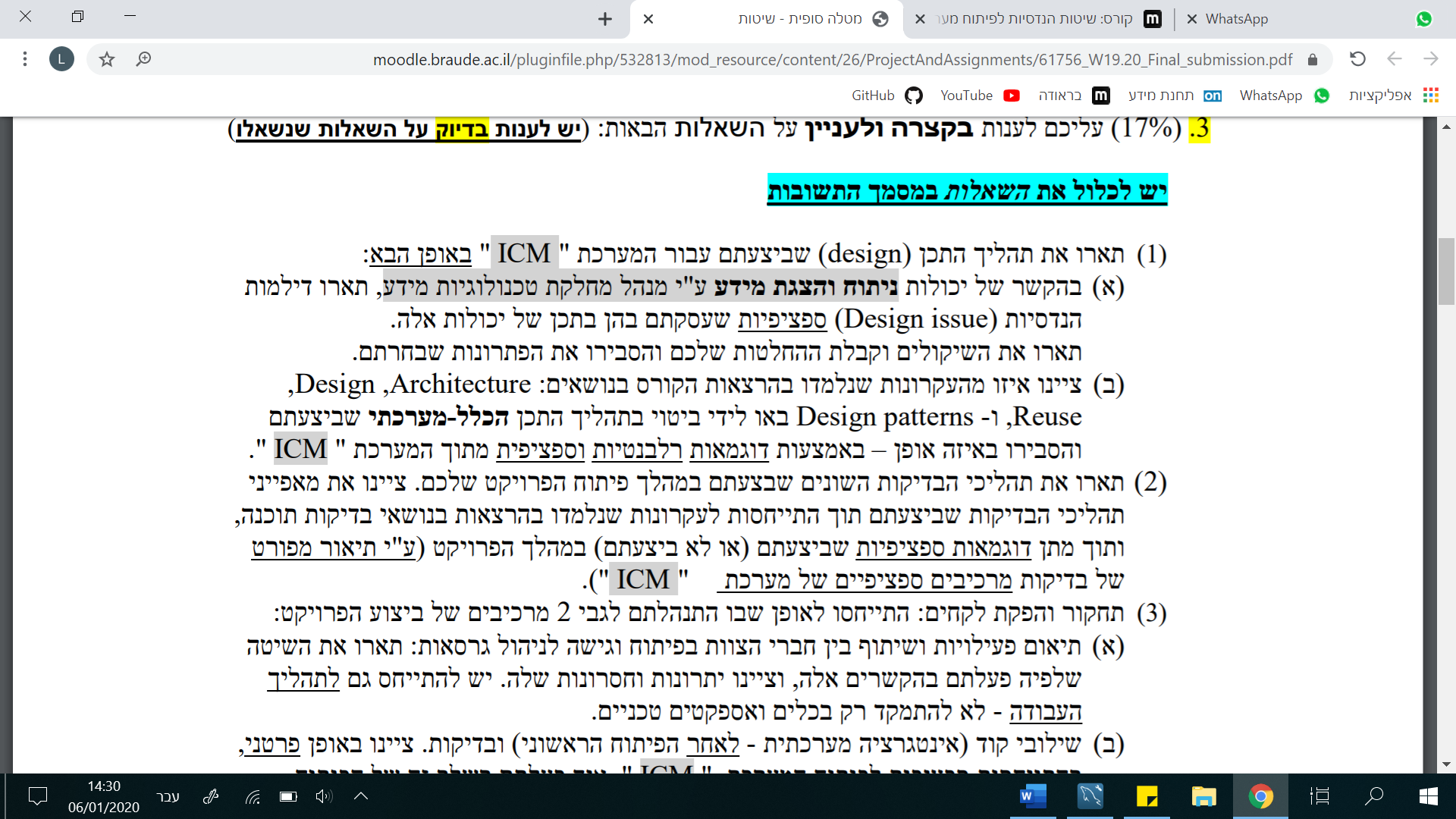
**איתי דוד** - 308021591 - [itaydavid22@gmail.com](mailto:itaydavid22@gmail.com)

**עידו קדוש** - 301727277 - [Idokadosh1@gmail.com](mailto:Idokadosh1@gmail.com)

**ליאור קאופמן** - 315837997 - [123liorkauffman@gmail.com](%20123liorkauffman@gmail.com)

**רביב קומם** - 316217751 - [ravivkomem@gmail.com](mailto:ravivkomem@gmail.com)

**תאריך הגשה: 19.01.2020**

**שאלה 1:**

**תשובה:**

סעיף א:

את תהליך התכן של ניתוח והצגת מידע ע"י מנהל מט"מ פירקנו לחלקים – הצגת בקשות לשינוי, צפייה ועריכת הרשאות לעובדי המחלקה, הצגת דוחות וצפייה בבקשות מושעות.

לאחר שפירקנו את התהליך ל-4 חלקים שונים החלטנו כי כל חלק יהיה עמוד נפרד בתוך העמוד של מנהל המט"מ.

אחת הדילמות שנתקלנו בהם היא איך לייבא את המשתמשים המתאימים מטבלת ה – MySQL לעמוד של מנהל המט"מ, האם לייבא את המשתמשים לפי הרשאות או לפי מחלקה.

תחילה חשבנו שלייבא את המשתמשים לפי שם המחלקה יהיה הכי נכון כיוון שמנהל המט"מ אחראי על כל מי שנמצא במחלקת המידע.

לבסוף החלטנו שהפנייה לטבלה ב-MySQL תיהיה לפי הרשאות ולא לפי מחלקה כיוון שיכול להיות במחלקה משתמשים שאינם מ"מ ואותם אנחנו לא רוצים לייבא כיוון שמנהל המט"מ לא יכול לערוך להם הרשאות.

דילמה נוספת שעלתה בעת העבודה על העמוד הנ"ל היא איך להציג את הדו"חות המבוקשים למנהל המט"מ, לפי הבקשה של הלקוח ישנם 3 דו"חות שהמנהל יכול לראות – דו"ח פעילות, דו"ח ביצועים ודו"ח איחורים בביצוע.

התלבטנו האם להציג את הדו"חות כטקסט רגיל המתאר לדוגמה כמה בקשות פעילות, דחויות, סגורות ומוקפאות קיימות במערכת, אך לבסוף החלטנו שנציג את הדו"חות בגרפים כיוון שכך ניתן להציג טוב יותר את הפרטים המבוקשים בכל דו"ח.

סעיף ב:

העקרונות שבאו לידי ביטוי במימוש התוכנה:

Design Patterns - יצרנו מספר interfaces אשר מחלקות אחרות ממשות, ע"י כך יצרנו במחלקות השונות אחידות בקוד, הקוד קריא ומסודר יותר וכך כל אחד מאיתנו יכול להבין טוב יותר הקוד ששאר חברי הצוות כתבו.

לדוגמה המחלקה BasicController אשר משתמשת במתודה getResultFromClient אשר כל ה-controllers במערכת ממשים את המחלקה, ולמתודה הנ"ל מגיעות כל התשובות מהשאילתות שפנו ל-data base.

דומה נוספת היא ה- DataInitializable, כל boundary אשר ממש מחלקה זו חייב לממש גם מתודה הנקראת initData, מתודה זו מקבל אובייקט המועבר לדף הנוכחי מהדף הקודם.

Reuse – יצרנו מספר מחלקות אשר בעזרתן אנו מייעלים את הקוד ומונעים שימוש חוזר בקוד, בעזרת שימוש במחלקות אלה הקוד שלנו קריא ומסודר יותר ואף יעיל יותר.

לדוגמה המחלקה PagingController במחלקה זו יש מספר מתודות אשר מדפדפות בין העמודים בעזרת path שמקבלת המתודה. חלק מהמתודות האילו מקבלות אובייקטים שאותם הם מעבירים לעמוד הבא שנפתח.

דוגמה נוספת היא המחלקה Utilaizer במלקה זו יש מספר מתודות לחישובים מתמטים שונים, במתודות אלה השתמשנו מספר פעמים בקוד במחלקות אשר ממשות את הדוחו"ת שונים.

דוגמה נוספת היא המחלקה TimeManager מחלקה זו אחראית על זמנים בקוד שלנו, במחלקה ישנם מתודות שונות אשר מייבאות את היום הנוכחי או משוות בין תאריך.

בשלבי הטיפול בכל בקשה השתמשנו רבות בזמנים ותאריכים, לכן המחלקה הנ"ל מנעה מאיתנו שימוש חוזר בקוד ובעזרתה ייעלנו את הקוד שלנו.

כמו כן השתמשנו בקומפוננטות חיצוניות בתוך הפרויקט עם התאמות מינימליות לצרכים שלנו, לדוגמא מימוש שליחת האימייל נעשה באמצעות קוד אשר חקרנו ומצאנו בפרויקט Open Source בGitHub. או לדוגמא אובייקט הToast אשר מאפשר לנו הקפצת הודעות בדומה למימושים שונים בעולם הAndroid.

כמובן כל תקשורת השרת-לקוח נעשית באמצעות מחלקות הocsf עם התאמות מינוריות לצרכי הפרויקט שלנו.

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

**תשובה :**

במהלך הפרויקט עבדנו בצורה מקיפה ועקבית לפי התוכן הנלמד בקורס מבוא לבדיקות תוכנה, תוך שימוש בכל הכלים השונים והתמקדות בתהליכים של פיתוח תוכנה נכון, המלווה בבדיקות תוכנה מתאימות. דאגנו ליישם את כל עקרונות הפיתוח שנלמדו ולאחר כל שלב פיתוח דאגנו לבדוק את שלבי הפיתוח בהתאם לבדיקות השונות.

במהלך שלבי הפיתוח יישמנו מספר סוגי בדיקות תוכנה שהתאימו לשלבים השונים –

**Black Box Testing**

בדיקות Black box בדיקות אילו הן בדיקות התנהגות המבוססות על דרישות ופונקציונליות, שיטת בדיקה זו נעשית כאשר המבנה הפנימי לעיתים אינו ידוע לבודק, בדיקות אילו בדרך כלל בודקות את הפונקציונליות של המערכת, עוד בשלבים הראשונים של הפיתוח ביצענו מספר בדיקות מסוג זה כאשר מימשנו עבודה עם OCSF, לאחר שמימשנו תקשורת של שרת לקוח ושימוש במסד נתונים רצינו לבדוק תחילה שהתקשורת עובדת בצורה תקינה, כלומר בדקנו שלפי קלט ספציפי מצד הלקוח אכן מתעדכן מסד הנתונים בצד השרת ולאחר מכאן ציפינו לקבל פלטים ידועים מראש חזרה בצד הלקוח, תהליך זה היה הכרחי עבורנו בכדי לדעת שתעבורת התקשורת עובדת נכונה בכדי לבנות עליה את המשך תהליך הפיתוח.

עם התפתחות הפרויקט ופיתוח התוכנה חילקנו בין כל חברי הצוות מטלות ודפים למימוש, כל חבר צוות עבד על מספר דפים למשל (Committee page, Tester Page , Execution page), כאשר כל חבר צוות סיים לממש את הדפים שלו דאגנו כי כל חבר צוות יבצע White Box Testing.

**White Box Testing**

בדיקה זו מבוססת על הידע וההיגיון הפנימי של קוד היישום, ידוע גם כ " קופסת זכוכית ", בבדיקות מהסוג הזה יש להכיר את הקוד הפנימי ובדיקות אילו מתבצעות לרוב על ידי המפתח. כאשר כל חבר צוות סיים את מימוש הדפים שהוקצו לו , כל חבר צוות דאג לבצע בדיקות מהסוג הזה על הדפים שלו בכדי לבדוק את הנתיבים השונים, לדוגמא בדף Execution page כאשר המבצע צריך להצהיר על תאריך סיום צפוי לביצוע הבקשה יש מספר בדיקות שצריך לבצע כמו הסיטואציה בה הוא מזין תאריך שכבר עבר, או מזין תאריך ומחכה כעת לתשובה מן המפקח האם התאריך אושר או לא, כמובן כל זה על ידי מימוש שאילתות נכונות הבודקות את שלב הבקשה הנוכחי שמתעדכן בכל סיום שלב.

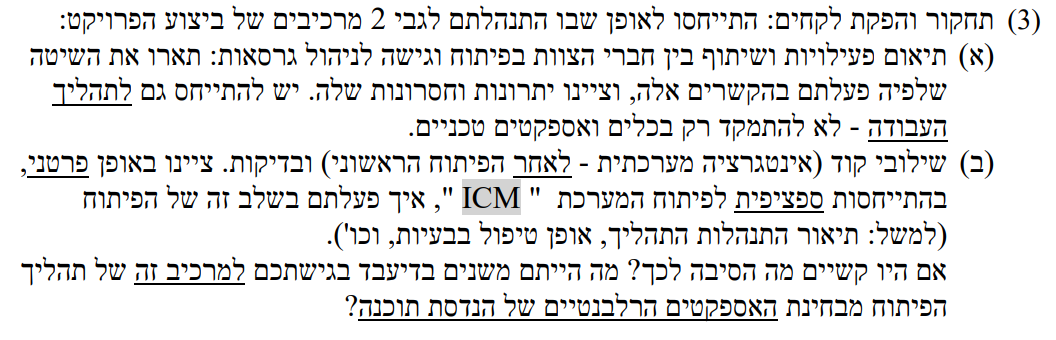
בהמשך שלב הפיתוח ולאחר שכל חברי הצוות ביצעו את המטלות השונות, תוך סיום מימוש הדפים השונים, בכדי לשלב את כל מרכיבי התוכנה ולבדוק האם המערכת עובדת בצורה תקינה כמערכת שלמה ביצענו Integration Testing.

**Integration Testing**

בדיקה זו מתמקדת בבדיקת מודולים משולבים, כלומר לאמת פונקציונליות משולבת לאחר אינטגרציה מודולי קוד ויישומים נפרדים, בדיקות אילו רלוונטיות במיוחד ללקוחות ושרתים ומערכות מבוזרות. לקראת סוף שלב הפיתוח ובכדי לבדוק כי המערכת עובדת נכונה בשלמותה ביצענו המון בדיקות מהסוג הזה על מנת לוודא כי חיבור הדפים ושילוב תנאים מדפים שונים עובדים בצורה נכונה, כלומר קלט מתוך מימוש של דף מסוים היווה פלט לתוך מימוש של דף אחר. לדוגמה כאשר כל תפקיד בביצוע הבקשה מבקש הארכת זמן , יש לעדכן את מסד הנתונים בצורה נכונה ולעבור לשלב של המפקח לאישור הזמן, בנוסף כאשר מסתיימים שלבים שונים בתהליך טיפול בבקשה יש לבדוק שהמעבר בין התפקידים השונים מתבצע נכונה תוך עדכון מסד הנתונים בשלב הנוכחי שבו הבקשה נמצאת ( דוגמה – כאשר מבצע הטיפול בבקשה מסיים את עבודתו כעת יש לעדכן את מסד הנתונים בכדי שבודק הבקשה יוכל לראות את הבקשה הספציפית בכדי להתחיל בשלב הבדיקה.)

**Usability testing**

בדיקות אילו הן בדיקות הבודקות ידידותיות למשתמש, נבדק בדרך כלל זרימת היישום, נשאלת השאלה – האם משתמש חדש יכול להבין את היישום, וכיצד לתת מענה ועזרה נכונה בכל פעם שהמשתמש "תקוע" בכל נקודה בשימוש התוכנה. בדיקות מהסוג הזה ביצענו במהלך כל שלב הפיתוח, בסיום הדפים השונים וכמובן בסוף שלב הפיתוח כאשר המערכת שלמה, דאגנו לתת משוב וביקורת בונה עבור הדפים השונים של חברי הצוות, כלומר כל חבר צוות בדק דפים שונים שחבר צוות אחר מימש על מנת לבדוק שהGUI תקין ונוח לשימוש, השתדלנו לחשוב כמו אנשים חיצוניים שנתקלים לראשונה בתוכנה ולראות האם התוכנה ברורה, האם כל עמוד של בעל תפקיד בטיפול בבקשה ברור והאם מובן מה לעשות בדפים בכל נקודה, דאגנו לממש הסברים קצרים על כל דף כלומר על מה יש ללחוץ בכל חלק ובכל שלב בכדי לטפל בבקשה בצורה תקינה ותואמת לדרישות הלקוח.



**פתרון שאלה 3:**

סעיף א:

כפי שלמדנו במהלך הסמסטר, על ידי צוות קורס, אחד ההיבטים החשובים והמשמעותיים ביותר בעבודה בתור מהנדסי תכנה לעתיד הינו ניהול גרסאות. כאשר על פרויקט ממונה צוות המכיל אפילו יותר מבן אדם אחד, יש צורך בכלי לניהול גרסאות וכמובן לניהול תצורות.

טרם פיתוח הקוד במהלך השלבים הראשוניים של הפרויקט, הרגשנו בצורך של ניהול גרסאות בצורה אופטימלית. כמו כן עם ניסיון עבר של חלקנו בתוכנות של ניהול גרסאות ידענו כי ישנו צורך בהתנסות מעשית "Hands On" עם הכלים טרם עבודה ממשית בכדי למנוע טרגדיות עתידיות.

החלטנו לחלק לעצמנו Action Item של ביצוע מחקר על מספר סוגים של כלים לניהול גרסאות, כפי שציינו במסמך הToDoList שהוגש כבר בהגשה הראשונה. במהלך המחקר גילינו כי ישנן שתי גישות מרכזיות לניהול גרסאות:

* **גישה ריכוזית** – כל המסמכים וקבצי הקוד נמצאים בתיקייה אחת Repository אשר כל אחד מחברי הצוות יכול לגשת אליה ולשנות אותה על ידי התחייבות של השינויים ((Commit.
* **גישה מבוזרת** – הקבצים נמצאים בענן, כאשר כל אחד מחברי הצוות יכול להעתיק את הקבצים ולעבוד עליהם לוקאלית ורק כאשר הוא מרוצה מהעבודה או הרגיש שהוא סיים תכולה גדולה הוא יכול לבצע דחיפה (Push) לענן.

התייעצנו והתדיינו בתוך הצוות והגענו להחלטה לפעול בגישה מבוזרת, כעת השלב היה למצוא כלי שיהיה ידידותי למשתמש שנוכל להתרגל אליו די במהירות ולהקים את התשתית של הפרויקט, מצאנו מספר כלים בשוק כמו GitHub, GitLab, BitBucket, BeansTalk, AWSCodeCommit.

לאחר סקר שוק קצר החלטנו להתקדם באמצעות GitHub כיוון שזהו כלי חינמי בעל יתרונות רבים כגון: אין צורך בהתקנה מוקדמת או יצירת סביבת עבודה, מאפשר התחברות נוחה מכל מקום פיזי שונה, וכמובן היתרון המרכזי היה שזהו אחד הכלים הכי פופולאריים כיום ולכן ישנם מסמכים ומדריכים רבים כיצד ניתן לפעול ולהשתמש בכלי זה.

כמו כן בכדי להקל על השימוש בכלי בסביבת העבודה של GitHub, החלטנו להשתמש באחד מהכלים והתוספים הקיימים בשוק כיום, לאחר מחקר קצר הגענו לשני כלים עיקריים שהם גם חינמיים וגם מאפשרים פיצ'רים נוחים לשימוש: TortiseGit, GitExtensions. לבסוף בחרנו להשתמש בTortiseGit כיוון שזהו כלי שמאפשר אינטגרציה מאוד נוחה ופשוטה עם מכשירים בעלי מערכת הפעלה של Windows אשר יש לכלל חברי הצוות (זה בעצם shell extension). כמו כן הוא מאפשר icons overlay כך שניתן לראות בקלות איזה קבצים שוני באופן מקומי מחשבים, על אילו קבצים בוצעו commit ומהם הקבצים החדשים שנוספו.

כמו כן ידענו שעל אף הזהירות ומקדמי הבטיחות שניקח יהיו מקומות בהן לא יהיה מנוס מפעילות על קטעי קוד משותפים וכמובן כצורך יוצא מכך נצטרך לראות את ההפרשים והשינויים בין קבצים בגרסאות שונות (diff) ולמזג ולאחד את הקבצים לגרסה אחת עדכנית (merge), כלומר בנוסף לכל הכלים נצטרך לכלי השוואה של קבצים. לשמחתנו TortiseGit מכיל בתוכו גם כלי השוואת קבצים פנימי והוא גם מאפשר להשתמש בכלים אחרים אשר מותקנים במחשב, ולכן בחרנו להשתמש בכלי חינמי של Beyond Compare 4.

כעת ברגע שהגענו להסכמות לגבי הכלים שבהם נשתמש התחלנו לתכנן את שיטת העבודה, ואת דרכי הפיתוח בדגש על ניהול הגרסאות והימנעות מדריסה ופגיעה בקוד של חברי הקבוצה בצורה לא רצונית.

כפי שציינו הפרויקט שלנו מנוהל בספרייה בGitHub וכל אחד מחברי הצוות יוריד עותק לוקאלי של הספרייה אליו למחשב ויעבוד על מטלותיו באופן מקומי. בעת סיום המטלות וקבלת ביטחון בעבודה (confidence) שהיא מצליחה להתקמפל ולרוץ ללא שגיאות יבוצע commit אל הספרייה. מעת לעת יש צורך לבצע בקשת משיכה (pull request) מהספרייה בGitHub בכדי לוודא שאנו עובדים על הגרסה העדכנית.

החלטנו להתנהל בשיטה של ביצוע תכן מקדים לכל קטעי קוד שנרצה לכתוב כאשר החלטנו שנקפיד על מספר כללים, זיהוי והוצאת קוד שייתכן ויהיה בו צורך במספר מוקדים שונים אל מחלקות כלליות שכל אחד יוכל לגשת אליהן. טרם מימוש בפועל של המחלקות הכלליות נגיע להסכמה לגבי פעולות שהן אמורות לספק והכנת API כך שגם אם הן אינן ממומשות נוכל להשתמש בהן. וכמובן חלוקה מודולרית של הפרויקט לתתי רכיבים וקומפוננטים כך שנקטין את הסיכון לביצוע שינויים באותם קטעי קוד.

כלומר תהליך הפיתוח שלנו כלל את השלבים הבאים:

**יתרונות השיטה:**

* צמצום התלויות של חברי צוות אחד בשני ויכולת לעבוד עצמאית והתקדמות בזמן האישי במטלות
* צמצום היקף כל קובץ, ובכך יותר קל לקרוא אותו ולהבין בצורה לוגית מה הוא אמור לשמש
* שליחת מתודות ואובייקטים למחלקות שניתן להשתמש בהן ביותר ממקום אחד ובכך לחסוך זמני פיתוח
* כמות מיזוגים וקונפליקטים מינימליים, רק כאשר יש צורך בכך

**חסרונות השיטה:**

* כל אחד עובד לוקאלית אצלו ובמחשב וישנן מתודות (בעיקר של הקבצים) אשר תלויות במיקומים של קבצים, אשר ישתנו בין מחשב למחשב.
* מסד הנתונים מתעדכן על בסיס קבוע ויש צורך בפעולה אקטיבית בכדי לטעון את הקבצים לWorkBench של כל אחד מחברי הצוות.
* חוסר היכרות עם קטעי קוד שלא כתבת, או ביצעת עליהם Code Review
* כאשר היה צורך במיזוג, כנראה שנדרש לנו יותר זמן כיוון שהיה מספר יותר מצומצם שלהם.

**פתרונות שלנו לחסרונות:**

* כיוון שיש צורך במיקומים זהים של הקבצים בפרויקט בהתחלה ניסינו להתחייב על סטנדרט מסוים, כל אחד מחברי הצוות יקרא לתיקיית הפרויקט באותו השם וישים אותה בתיקייה קבועה מראש בכונן C. כמובן שהיה לנו קושי לעקוף את הסטנדרט הזה ונוצרו לנו מספר בעיות של חוסר התאמה.

לכן הבנו שעלינו למצוא פתרון יותר יעיל שידרוש פחות מאמצים והתעסקות, לאחר מחקר הגענו למסקנה שהכי נוח יהיה לייצר סקריפט אשר ימפה את כל תיקיית הפרויקט לכונן לוגי R ובכך גם אם כל חבר צוות ייתן לה שם שונה או ישמור אותה במקום אחר, הרי שבכונן R הכל יראה אותו הדבר.

* החלטנו להתלבש על הפתרון הקודם ולהכניס לסקריפט גם את ייבוא הטבלאות החדשות אל הWorkbench