שיטות הנדסיות לפיתוח מע' תכנה, תרגיל 3:

קבוצה: 13

חברי הקבוצה: נתיב לנגל 315680058 nativlengel11@gmail.com

נדב מאיר 305781452 nadav491@gmail.com

תומר אבירם 206066052 tomerav83@gmail.com

שאלה 1:

1. דילמות הנדסיות – Design Issues

בכל הנוגע לתהליך ביצוע הבחינות הן באופן ידני והן באופן ממוחשב, אוטומטי, נתקלנו בכמה קשיים בשלב התכן, כדוגמות ספציפיות נבחר להתמקד בשני נושאים עיקריים – עיצוב מסד הנתונים, ועיצוב המחלקות הרלוונטיות.

ממבט מסד הנתונים –

התלבטנו איך לשמור את המידע הרלוונטי, אילו טבלאות ליצור, כשהשיקולים היו נוחות הגישה לנתונים, יעילות השאילתות אותן נאלץ לבצע על כל סכמה רלוונטית.

ממבט המחלקות –

פירוש והמרה- casting בין פורמטים שונים של שמירת הנתונים, וכמובן הלוגיקה וההיגיון שמאחורי הבחירה.

תחילה, דנו בנושא, ומצאנו 2 פתרונות אפשריים, שעליהן הסכמנו, בחרנו פתרון אחד וככל שהתקדמנו בקידוד התכנה, ראינו כי אנו נאלץ לבצע שינויים והתאמות, לאחר דיונים נוספים החלטנו כי נשלב את הפתרון השני שמצאנו יחד עם הפתרון שכבר התחלנו לממש, מפני שכל פתרון מבין השניים תיאר בצורה טובה יותר את הצרכים מבין שני הנושאים – המחלקות ומסד הנתונים.

יצרנו 2 מחלקות הקשורות, והמתארות את תהליך ביצוע הבחינות – האחת studentTest שמאפיינת בצורה הטובה ביותר את צד מסד הנתונים, שמירת המבחנים והנתונים שעליהן נבצע שאילתות לקבלת מידע סטטיסטי.

והשנייה – executedTest המאפיינת בצורה הטובה ביותר את צד המחלקות – וקוד התכנה, מעין controller המטפל בכל הקשור לנעילות מבחנים, שינויי זמן, ממשקים גרפים למילוי המבחן.

1. נבחר לדבר על נושא עיצוב התכנה, ותהליך פיתוחה מהרצאה 8, בהרצאה זו מופיע המושג design issues באופן קצת יותר מורחב, וכדוגמה מתוך מערכת AES שלנו, נבחר לתאר למשל את האופן בו בצענו את תכן המערכת –

בפגישה השבועית, דנו באופן כללי בעיצוב המערכת, ממבט על, כלומר בדרך הכי גנרית ואבסטרקטית –

מה הצורך ? – אילו מחלקות נצטרך ליצור, שמותיהן, מה הן יכילו ואילו בעיות הן פותרות, ארגון לפי המודלים אליהן נחשפנו (מודל 3 השכבות), ומודל התקשרות שרת- לקוח.

ככל שהתקדמנו בפגישות, נכנסנו יותר לעומק מבחינת עיצוב, והתחלנו לקחת בחשבון

מהן הדרישות הפונקציונליות והלא פונקציונליות ? – תארנו את הפונקציות אותן יכלו המחלקות, ועל איזה דרישות הן עונות מתוך מערכת AES, והתחלנו למפות ולמצוא פתרונות נוספים ואלטרנטיביים לדרישות המערכת שלא נוכל לבטא בצורה פונקציונלית.

דוגמות ספציפיות – טבלת המבחן במסד הנתונים, 2 המחלקות לטיפול ושליטה בביצוע הבחינות, טבלאות עזר לשמירת נתונים בצורה ממוקדת ועניינית יותר כדוגמת טבלת studentTest בכדי לחסוך בזמני ריצה של שאילתות וסינון תוצאות.

שאלה 2:

שני העקרונות שנבחר לתאר הם אלו שהתמקדנו בהם בצורה הבולטת ביותר והם עקרונות הבדיקה שבין נוחות ושימוש, כמאפיינים של usability & utility.

מבחינת מבט המשתמש-

1. האם המערכת עונה על צרכי המשתמש ?

בדקנו האם מימשנו באופן מלא, ויעלי ככל האפשר, את דרישות המערכת כולן, הן הבסיסיות והן האפשריות (מבחינת נוחות ו/או יעילות).

קוד הפונקציות – בדקנו האם הפונקציות נותנות מענה לדרישות המערכת, למשל – פונקציות לקבלת דוחות סטטיסטים, פונקציית נעילת מבחן, פונקציית שליחת מבחן, פונקציות שונות לשמירת מידע במסד הנתונים.

כאן בחרנו להתייחס לעקרונות ה – USABILITY, EFFICIENCY.

1. האם המערכת ידידותית למשתמש ?

בדקנו היכן ניתן לבצע שיפורים מבחינת נוחות השימוש והמראה של התכנה, בכדי להקל וליצור סביבת עבודה ידידותית יותר למשתמש המערכת, למשל - חלונות גרפיים נוספים להוספת שאלות למבחן, אפשרויות עיצוב חלונות GUI ביצירת מבחן, שיתנו באופן עקיף מידע למשתמש אילו מבחנים הוא יכול ליצור ואילו לא ( מבחינת קוד המבחן), בכדי למנוע חוסר הבנה והתמצאות עד כמה שניתן.

כאן בחרנו להתייחס לעקרונות ה – FRIENDLY USE, SELF-LEARNING, SELF-UNDERSTATNDING, MINIMIZE DEVELOPER-CLIENT DEPENDENCY .

מבחינת מבט המערכת –

1. האם המערכת עונה על צרכיה ?

האם כל דרישות המערכת מומשו, האם כולן פועלות באופן תקין ואמין, האם יש מקום לשיפורים, ואם כן, האם נוכל לבצע זאת, האם בנינו את המערכת בצורה שנוכל להשתמש בחלקים דומים לשימוש אמנם שונה, אך חוזר.

כאן התייחסנו לכל העקרונות שלמדנו בהרצאה לגבי בדיקת איכות העיצוב של המערכת שלנו – USABILITY, EFFICIENCY, RELABILITY, MAINTAINABILITY, REUSABILITY.

1. האם המערכת עונה על צרכיה באופן יעיל?

למרות שבחנו את נושא היעילות, בחרנו לתת משקל נוסף לנושא זה ובחנו בשנית האם ניתן לבצע שינויים בכדי ליעל את המערכת ? – זמני ריצה, אחסון מיותר של נתונים וכו'..

כדוגמה ספציפית – פונקציות להפקת דוחות סטטיסטים – שינינו חלק מהפונקציות בכדי שסינון תוצאות רלוונטיות ואיחודם לכדי תוצאה מבוקשת יהיה כבר בשלב הבאת המידע ממסד הנתונים, על ידי כתיבת שאילתות מתאימות, למשל – חישובי ממוצע לפי סטודנט, לפי קורס, לפי מורה, לפי מבחן, ובפונקציות שאת תוצאתן לא מתאפשר לקבל בשלב זה בחרנו לשנות בכדי שרק הנתונים הרלוונטיים לחישוב יגיעו לפונקציות חישוב ב java, שעליה אחראית מחלקת הדוחות, יחד עם שילוב אלגוריתמים פשוטים ואולי אפילו יעילים יותר לביצוע החישוב, למשל – חציון.

האם ניתן לעבוד במקומות מסוימים בקוד בצורה של שלד ? , כלומר – האם יש מבנה שלדי לפונקציה מסוימת, שרק יעוד ספציפי משתנה בה ? – כלומר עבודה בצורה גנרית לשם שימוש חוזר.

כאן, כאמור – התייחסנו שוב לעקרון ה EFFICIENCY, ובנוסף גם לעקרון ה reusability.

מבחינת מבט המתכנת / מפתח –

האם אנחנו מתכננים משהו אפשרי ? משהו יעיל כלל הניתן שאכן נוכל לבצע ובאופן מלא ?האם זה בר ביצוע – מבחינת אילוצי הזמן אליהם אנו כפופים? עקרון ה reusable ועקרון ה MAINTAINABILITY בכדי להקל על עבודתנו עד כמה שאפשר, (שימוש בפונקציות במובן גנרי), וניסיון לבנות את הפונקציות בצורה קלה להבנה על ידינו בשלב מאוחר יותר לשם תחזוקה ושינוי במקרה הצורך.

שאלה 3:

1. תיאום פעילות, וניהול גרסאות –

תיאום הפעילות בינינו היה מצוין, כל אחד ידע מה הוא אמור לעשות, על איזה חלקים הוא אחראי, ועד מתי עליו לבצע זאת.

ניהול הגרסאות היה קצת יותר בעייתי מבחינתנו, זו הפעם הראשונה בה נאלצנו להיות תלויים באחרים, במקרים קיצוניים בהם עובדים על אותו אזור בקוד, או שינויים בקוד של חבר צוות אחד הגוררים שינוי בקוד של חבר צוות אחר, הנושא היה מאוד מאתגר עבורנו.

השיטה בה בחרנו לעבוד – היא באמצעות סיכום על חלוקת המשימות ואיחוד וביצוע שינויים שצריך בכל פגישה שבועית, נעזרנו באמצעים טכניים כדוגמת toDoList ו GitHub, בכל פגישה שבועית עדכנו מה עלינו לעשות, סיכמנו בינינו מה נותר לעשות, ומה עשינו עד עכשיו, והאם הכל תקין, עובד ונכון.

במהלך העבודה העצמאית של כל חבר צוות, מחוץ לשעות המפגש השבועי, דאגנו להתעדכן ולעדכן אחד את השני ולשאול שאלות, לברר ולהבין באיזו נקודה אנו נמצאים כעת, הדבר שימש גם כמעיין בקרה – הן מבחינת העבודה הכוללת והן מבחינת העבודה השבועית, האם קצת העבודה הוא בסדר, והאם נספיק.

1. שילובי קוד – אינטגרציה –

את חשיבות האינטגרציה המתמדת, ובקרת גרסה הבנו בשלב קצת יותר מאוחר, לאחר שכבר התחלנו להיכנס למצב בו כל חבר צוות כבר כתב קטעים מסיבים של קוד.

באחת הפגישות השבועיות, נפגשנו במטרה לאחד את הקודים לכדי קוד אחד וממנו להמשיך, פגישה זו הייתה הרבה יותר ארוכה מפגישות קודמות, למרות שמבחינת תיאום הפעילות היינו במצב טוב, נחשפנו בפעם הראשונה שזה לא מספיק וכי צצות המון שגיאות ובעיות לאחר אינטגרציה של כל חלקי הקוד, שאותן נאלצנו לפתור, השיטה שלנו הייתה לעבור פונקציה-פונקציה בקוד, ולתקן שגיאות קיימות, עד למצב שהמערכת כולה ללא שגיאות, ולאחר מכן להריץ פונקציות ולבדוק בדיוק האם היא עושה את מה שצריך, האם היא עובדת, אם לא למה ואיך לתקן אותה, אילו שינוים זה גורר, האם זה משפיע על חלקים אחרים. לאחר שענינו על כל השאלות האלו ככל שהתקדמנו לעומק הקוד, הצלחנו להכיל שינויים ולתקן את השגיאות.

הסיבה לקשיים הייתה שלא נתנו משקל רב לנושא בקרת הגרסות כמו לתיאום הפעילות, וכשראינו שהתיאום הוא מצוין, ואנו מגיעים להבנות וכל אחד מודע למה שחברי הצוות האחרים עושים, הרגשנו קצת יותר מדי נוח, ולא חשבנו שעלולות לצוץ הרבה בעיות באיחוד הקוד.

בדיעבד, היינו צריכים להתייחס לנושא זה הרבה יותר, ולא להסתמך על כך שאנו מתואמים, ולהתחיל כבר מראשית בניית המערכת, לעבוד בצורה מקוונת מבחינת הקוד והקבצים הרלוונטיים, ולא להתחיל כך שכל חבר צוות עובד על החלק שלו ואחר כך לאחד.