מטלה 2 / שאלות פתוחות - קבוצה 7

חברי הצוות

שם מלא: היא קבלאן / ת"ז: 313312373 / מייל: haiakablan96@gmail.com

שם מלא: שלומי לוי / ת"ז: 305129652 / מייל: xxsmgxx@gmail.com

שם מלא: אריק סקיגין / ת"ז: 312360449 / מייל: arik30000@gmail.com

שם מלא: סמי עודה / ת"ז: 315654962 / מייל: rsgunz.sami@gmail.com

שם מלא: אור אליהו / ת"ז: 307884890 / מייל: oreliyahu93@gmail.com

שאלות

1. תארו את תהליך התכנון שביצעתם לתהליך השאלה. פרטו מה הם הדילמות התכנוניות שהתייחסתם אליהם? מה היו החלופות ומה השיקולים לבחירת הפתרון שיושם? בתשובה התייחסו לעקרונות שנלמדו בהרצאות בקשר לנושאים Architecture, Reuse ,Design ו-Design patterns.
2. ציינו עקרון של גישת התכן :Object-Oriented Design שבו עשיתם שימוש במהלך התכן (design) של הפרויקט. תארו בדיוק ובפירוט את תרומתו של עקרון בתהליך התכן שביצעתם ובתוצאתו הסופית, תוך התייחסות ספציפית (לא 'עקרונית') לדרישות הפונקציונליות של המערכת שתכננתם. (התייחסות ספציפית בהקשר זה = התייחסות למרכיבים ספציפיים מתוך התיאור המילולי של המערכת שקיבלתם בהתחלת הסמסטר).
3. ציינו לפחות עקרון אחד של OOD שלא בא לידי ביטוי במודלים שלכם והסבירו למה.
4. הערה כללית:
5. מהם היתרונות של מודל UML כעזר לתהליך התכנון?
6. הסבירו איך מתקבלים (מתממשים) היתרונות שציינתם.
7. ציינו דוגמה אחת קונקרטית )לא כללית( מתוך התהליך שאתם בצעתם לשימוש מועיל ב-UML תוך תיאור והתייחסות ספציפית למרכיבים של מערכת "GCM" שתכננתם ומילדתם.
8. ציינו קשיים הנובעים מחסרונות של UML שנתקלתם בהם. גם כאן התייחסו ספציפית לתהליך שבצעתם לפיתוח מערכת זו.
9. ניתוח דיון:

בהתאם לניסיון שרכשתם במהלך העבודה על מטלה זו, תארו אפשרויות לשינויים ושיפורים במתודולוגית UML אשר נותנים מענה לחסרונות שנתקלתם בהם במהלך ה-design שביצעתם בפרויקט שלכם. הסבירו את תשובתכם תוך תיאור דוגמה ספציפית (כולל שמות של רכיבים) מתוך עבודתכם.

תשובות

1. ניהול גרסאות:

כאשר עובדת חברה רוצה לעדכן גרסה מבוצע התהליך הבא:

1. העובדת בוחרת עיר.
2. המערכת מציגה את רשימת המפות של העיר שנבחר.
3. העובדת בוחרת רשימת המפות שיכללו בגרסה ומגדירה להם גרסה.
4. המערכת יוצרת אוסף מפות עם הגרסה ורשימת המפות שנבחרה.
5. המערכת בודקת אם הגרסה של אוסף המפות למול הגרסאות הקיימות במערכת.
6. אם הגרסה שהתקבלה חדשה יותר נמשיך ל-ז', אחרת העובדת חברה תקבל שגיאה ומפסיק.
7. המערכת מעדכנת את רשימת הבקשות לאישור של המנהלת.

הדילמות התכנוניות:

1. היכן מתבצעת הבדיקה של הגרסה הנוכחית למול החדשה?, אנחנו רוצים למנוע עדכון גרסה ישנה יותר?
2. אם קיימת בקשה לאישור של אותה גרסה (זהה או יותר גבוהה) של אוסף מפות?
3. איזה רשימה תכיל אוסף המפות?

חלופות:

* בדיקה אצל הclient
* בדיקה אצל ה-server (מול מסד הנתונים)
* שולחים כל בקשה לאישור מנהל.
* סופרים גרסאות לאוספי מפות לאישור.
* יכלול reference לכל מפה
* העתק של כל מפה.

שיקולים:

אמינות, גודל מסד הנתונים, נכונות ויעילות.

לבסוף הוחלט כי בזמן בחירת העיר המערכת תבקש מה-server את המידע העדכני על העיר, יבוצע נכונות בצד ה-client וכאשר יישלח לאישור במערכת גם ה-server יבדוק את הגרסה העדכנית במערכת למול הגרסה של האוסף שנשלח, בנוסף במידה והגרסה עומדת בקריטריונים היא תתווסף לבקשות לאישור במידה ולא קיימת בקשה על אותה גרסה.

פתרון בצד ה-client יצמצם בקשות ל-server בטווח הרחוק והבדיקה בצד ה-server תשמור על נכונות הגרסאות במסד הנתונים.

כאשר ניצור אוסף מפות לאישור ניצור העתק של המפות במסד הנתונים, וזה כיוון שבמידה ונרצה לראות את היסטורית הגרסאות שאושרו כל גרסה תהייה נכונה באותה העת שנוצרה.

עריכת מפות:

כאשר עובדת רוצה לערוך מפה מבוצע התהליך הבא:

1. העובדת בוחרת מפה.
2. במידה וזאת עריכה ראשונה היא יכולה ליצור גם עיר.
3. העובדת מוסיפה ועורכת תכנים (מנהלת גם יכולות למחוק).

הדילמות התכנוניות:

1. איפה המנהל יוכל לממש את פונקציית המחיקה שלו?
2. איך המערכת תתנהג כאשר עובדים שונים מבצעים פעולות על אותה המפה (הוספה / עריכה / מחיקה)?

חלופות:

* מסך ניהול מיוחד.
* הוספת אפשרויות ע"פ הרשאה על אותו מסך עריכה.
* נעילת המפה כאשר אחד העובדים מבצע עליה עריכה כלשהי עד לסיום.
* נעילה חלקית.
* נכונות על אותה פעולה תתבצע בצד ה-server (מול מסד הנתונים).

שיקולים:

צמצום מסכים שלא לצורך, נכונות, יעילות ומורכבות המימוש.

לבסוף הוחלט כי ע"מ לצמצם מסכים למשתמש במידה וקיימת לנו הרשאה עבור כל עובד נוכל לצמצם מסכים ע"י הוספת פעולות (מחיקה) באמצעות שימוש בהרשאה של העובד.

כאשר תתבצע אחת הפעולות על תוכן המפה, לדוגמה הוספה של תוכן, העובד יזין את הנתונים, המפה תישלח ל-server ויבוצע בדיקה מול המסד נתונים אם המפה קיימת שמה או לא. באופן זה נגדיל במעט את הבקשות ל-server אבל נשמר את נכונות המידע.

1. עיקרון Object-Oriented Design בו עשינו שימוש הוא Reuse, עקרון זה מאפשר לנו לשלב קטעי קוד ומרכבים אשר לא אנו תכננו וממישנו אבל עונה על חלק מדרישות התכנון ויכול להשתלב בו בנוסף לתפקודי הליבה של המערכת.

באמצעות שימוש בעיקרון ה- Reuse השתמשנו ב-Framework קיים המממש תקשורת client-server (Object Client-Server Framework-OCSF).

כחלק מעיקרון ה- Reuse(עבור Framework) המשתמש ברובו אינו צריך לקרוא את הקטע קוד הממומש אלא לעיין היטב ב-API כדי להבין כיצד להשתמש נכון בספריית הקוד.

באמצעות ה-OSCF המבצע קשר בין ה-server ל-client נעביר מידע הכולל ערכים, פקודות, שגיאות וכו', בכל צד נממש חלקים היודעים לתרגם את המידע שהגיע ולהמשיך את הטיפול במידע.

לדוגמה, כלל המידע שנשמר במסד נתונים מסוג MySQL, המשתמש היחידי שמבצע שאילתות למול המסד נתונים הוא ה-server ע"י בקשות שנשלחות מה-client.

כשאנו מוספים מפה במערכת צד ה-client מעבד את המידע שהוא רוצה לשלוח באופן בו ה-server יודע לתרגם אותו, כאשר המידע על הוספת מפה חדשה מגיע ל-server (שימוש במתודה listen()) מבוצע תרגום המידע שהתקבל, הפיכתו לשאילתה (הוספה) למול המסד נתונים, במידה המפה כבר קיימת או שהיא נוספה בהצלחה ה-server מעבד את התגובה ושולח ל-client בהתאם.

שליחה וקבלת המידע בין ה-client-server מבוצע ע"י מתודות של OCSF.

1. דריסת מתודות (Override), הוא עקרון שלא בא לידי ביטוי ב- Object-Oriented Design.

כדי להתאים באופן אופטימלי את ה-OCSF לליבת המערכת שלנו ביצענו דריסות לחלק מהמתודות המובנות, דריסה זאת לא באה לידי ביטוי במודלים שלנו מכיוון שלא ניתן לתאר דריסה ב-UML.

1. לתרשימי UML יש כמה יתרונות כעזר לתהליך התכנון והם:
   * בניית ישויות וקשרים לפי עקרונות OOP דבר שמבטיח תכנון נכון.
   * ניתן לראות פעולות שחוזרות על עצמם ובכך לחסוך במימוש הקוד (Reuse).
   * מדד לסדר הגודל של הפרויקט – ראייה רחבה של הנדרש מהמערכת וקבלת אינדיקציה על גודלה וזה מאפשר תכנון פיתוח לטווח ארוך ע"י שלבים.
   * שפה משותפת - באמצעות הדיאגרמות נוכל להציג את המערכת לאנשי מקצוע שונים, באופן בו ההסבר יהיה קל וידידותי.
   * בעזרת תרשימי activity ו-sequence, ניתן להראות רצף וסדר של אירועים במערכת, מרגע בו האירוע מתחיל ועד לסיומו.
2. בתרשים שיצרנו, רכישה היא מחלקת אבא (שמוגדרת disjoin) אשר יורשים ממנה שתי מחלקות: מנוי ורכישה חד פעמית, ולכל אחת משתי מחלקות אלה יש לה את המאפיינים הרלוונטיים.

ז"א לפי עיקרון מונחה עצמים עבור שתי המחלקות היורשות קיימות תכונות משותפות שבאופן כללי נוכל לקרוא להם רכישה, עיקרון זה חוסך מונע כפילויות של קוד שמתבטא לנו בזמן פיתוח, ובנוסף נותן לנו ביטוי מופשט ונכון יותר של עצמים וקשרים.

* + - 1. UML היא שפה, וכדי לדעת את השפה מבלי לבצע שגיאות צריך ללמוד אותה באופן בו לא נבצע טעויות דבר שמעכב ולוקח זמן.

בנוסף, מידול מורכב של דיאגרמות יוצרים קשיים בבניית תרשימים הנסמכים עליה, ז"א כל תרשים צריך להיות תואם לוגית לתרשים המתבסס עליו הן מבחינת משתמשים והן מבחינת עקביות של אירועים.

בהתחלה יצרנו את ה-use case (עם הflow of events), אחר כך בהסתמך על התרחישים והשחקנים בו, בנינו את ה-activity ואת ה-class ובהסתמך על שלושתם בנינו את ה-sequence.

לדוגמה: במערכת GCM, קיימת אפשרות לעריכת מפה, ב-use case הגדרנו שעובד חברה יכול לערוך מפה (הוספה ועריכת תוכן), מכיוון שב-flow of events שכתבנו היה ארוך ומפורט מאוד היינו צריכים להיות עקביים אחר התהליך ולפיכך לבנות את ה-activity וה-sequence דבר שלקח לנו המון זמן בניתוח ובנייה של תרשימים אלה.

בנוסף, בחלק מתרשימי ה-UML אין אינדיקציה לסדר האירועים ולהגבלות מסוימות.

לדוגמה: בהגדרת ה-use case שלנו הגדרנו שחקן אורח הכולל כמה פעולות שביניהן התחברות, מכיוון שרוב הפעולות תקפות לשחקנים נוספים כמו לקוח ועובד חברה, שחקנים אלה יורשים ממנו, אבל מכיוון שאחרי שאורח מבצע התחברות למערכת, הוא הופך לשחקן אחר שיורש ממנו אבל לא יוכל לבצע התחברות.

בנוסף לכך, כאשר משחררים גרסה מדובר בתהליך הכולל אינטראקציה בין כמה שחקנים (צריך להגדיר, לשלוח בקשה, לקבל אישור משחקן מנהלת ואז לעדכן במערכת)

ב-use case ניתן להגדיר את הפעולות אבל לא יכולים לדעת את סדר האירועים שצריך לבצע.

1. אם הייתה לנו את האופציה לשנות את מתודולוגית ה-UML, אז היינו משדרגים את

ה- use caseבכך שנוכל למספר סדר פעולות הכרחיות ונוכל לסמן פעולות שלא רלוונטיות לשחקנים יורשים כפי שכתבנו בדוגמאות ב-3.א.2

עבור תרשום ה-use case נוסיף את מספור לסדר פעולות לדוגמה:

(1) הגדרת הגרסה ע״י עובדת -> (2) תישלח בקשה -> (3) אישור ע״י המנהלת ->

(4) עדכון ע״י העובדת.

ועבור ביטול הפעולה ליורשים נסמן על בועת הפעולה &.

עבור הגבלת פעולות ב-class diagram למחלקות נסמן באותו סימון & ליד הפונקציה שאנחנו לא רוצים להוריש.

אצלנו ב-class diagram תחת ה-controllers package קיימת פונקציה login במחלקת אורח GuestCC, נרצה כי פונקציית ה-login לא תהייה זמינה ליורשים ClientCC ו-CompanyWorkerCC.

שינויים קטנים אלה ישפיעו מאוד על הבנת הקורא ויתרום לנכונות הכללית של ה-UML.