

המחלקה להנדסת תוכנה

פרויקט גמר – תשע"ו

מערכת לניהול משאבי הארגון

mobile ERP system

מאת

מיתר שוקרון

זאב מלומיאן

מנחה אקדמי: מר שי תבור	אישור:	תאריך:
אחראי תעשייתי: מר עופר פישלוביץ'	אישור:	תאריך:
רכז הפרויקטים: מר אסף שפיינר	אישור:	תאריך:

מיקום	מערכת
https://github.com/meitarsh/m.s-aluminium-manager-app	מאגר קוד
https://calendar.google.com/calendar/embed?src=t1f2ojv5arrqonei6h09i5fld0%40group.calendar.google.com&ctz=Europe/Athens	יומן
https://github.com/meitarsh/m.s-aluminium-manager-app/projects/1	ניהול פרויקט (אם בשימוש)
https://github.com/meitarsh/m.s-aluminium-manager-app/releases	הפצה
https://github.com/meitarsh/m.s-aluminium-manager-app/wiki/Alpha_video	סרטון גרסאות אלפא

תוכן עניינים

3	מילון מונחים, סימנים וקיצורים
4	תקציר
6	1. מבוא
6	2. תיאור הבעיה
6	דרישות ואפיון הבעיה
6	הבעיה מבחינת הנדסת תוכנה
7	3. תיאור הפתרון
7	מהי המערכת
8	מצבי עבודה ותכנון הפתרון
8	תיאור הכלים המשמשים לפתרון
9	4. תכנית בדיקות
11	5. סקר שוק
15	6. ריכוז פרויקטים דומים
17	7. נספחים
17	א. רשימת ספרות \ ביבליוגרפיה
18	ב. תרשימים וטבלאות
29	ג. תכנון הפרויקט
30	ד. טבלת סיכונים
30	ה. רשימת טבלת דרישות

תקציר

במסגרת התוכנית להנדסת תוכנה בעזריאלי המכללה להנדסה, הוטלה עלינו המטלה לבצע פרויקט גמר – פרויקט במסגרת של 400 שעות אשר ממש את כל מה שלמדנו במסגרת הלימודים, על מנת שנוכל לקבל ניסיון שטח החלטנו לעבוד עם חברה קיימת, כך נלמד על דרישות הלקוח המשתנות לאורך חיי הפרויקט ונקבל ניסיון ודרכי התמודדות איתם בזמן אמת, נחוה פגישות מקצועיות עם הלקוח שלנו ונוכל לפתור בעיה שקיימת ללקוח במקום להמציא בעיה ולפתור אותה, ואת הבעיה אנחנו לא פותרים בשבילנו אלא בשבילם – כלומר נצטרך לחשוב על חווית המשתמש וקהל היעד שלנו, כתוצאה מכך נחוה את כל התהליך של יצירת אפליקציה בעולם האמיתי.

הלקוח שלנו, חברת מ.ש אלומיניום אינו שונה מאיתנו בהרבה, הלקוח מספק שירות של יצירה והפקה של מוצרי אלומיניום עבור לקוחות שונים שרואים בכך צורך, כיוון שמטרת הרווח של הלקוח שלנו היא הרבה מעלינו, הלקוח שלנו גם מונה מספר עובדים הנקראים מנהלי פרויקטים שתפקידם להסתובב באתרי הלקוח כדי לוודא את שביעות הרצון של הלקוח וגם להציע פתרונות נוספים – כלומר למכור את עצמו, כלומר גם ללקוח שלנו קיימים לקוחות, וללקוחות קיימים פרויקטים, תפקידם של מנהלי הפרויקטים הוא לראות באתר הלקוח את הבעיה, לדווח אותה לחברה ולדאוג למצב הפרויקט.

פעולה זו של מנהלי הפרויקטים דורשת המון סדר ואחריות, יש חובה להפריד בין הפרויקטים השונים, כדאי למלא את הדוחות בפורמט ידוע וסטנדרטי ואף יש צורך בשליפת מידע ואחסון מידע בצורה נוחה ביותר, למזלם של מנהלי העבודה, מערכות המחשוב המודרניות מספקות את השירותים האלו כמעט בחינם, הבעיה היא לא באמצעי המחשוב או היכולות שנותנות מערכות המחשוב אלא אמצעי התקשורת, למנהל עבודה לא תמיד יש את אמצעי התקשורת כדי לדווח אל השרת את הדו"ח שלו מכל אתר לקוח אפשרי, מטרתנו היא לספק את היכולת המחשוב תוך התמקדות בבעיה זו, אנו ננצל הזדמנות זו כדי לספק לקהל היעד שלנו את השירות שהם צריכים ("שגר ושכח" של דו"חות) תוך כדי התמקדות בממשק נוח ואבטחה.

במהלך הפרויקט יש לנו שני יחידות מפתח להתייחס אליהם, הממשק משתמש – כלומר החווית משתמש, והחלק הלוגי, כלומר הפעולות הלוגיות שהאפליקציה מבצעת, כדי למקסם את התהליך, בחרנו לבצע את הפרויקט בזוגות, כאשר אחד מהשותפים (מיתר) מתמקד בחוויות המשתמש ויהיה אחראי על החלק הגראפי והחלונות השונים של האפליקציה, והשותף השני (זאב) יהיה אחראי על פיתוח פעולות צד שרת ופעולות שמירת נתונים בנוסף לפיתוח בדיקות וטסטינג ולאפליקציה.

1 מילון מונחים, סימנים וקיצורים
הלקוח – החברה מ.ש. אלומיניום, חברה להפקת מוצרי אלומיניום, אשר מוצר הפרויקט מיועד לשימוש שלה.

השרת – מחשב אשר נמצא בבניין הלקוח אשר מחובר אל האינטרנט (בשאיפה להיות מחובר כל הזמן) ותפקידו הוא להחזיק בנתוני החברה ולספק שירותי שליפה וכתובת נתונים אונליין, לספק שירותי מייל ושירותים אחרים שיתכן והלקוח מעוניין בהם.

המסד נתונים – מערכת האחסון בשרת שבה נשמרים הנתונים הארגוניים של הלקוח כולל משתמשים מורשים, פרויקטים שונים שהלקוח מבצע ומצבם, עובדים ושכריהם וכדומה.

המוצר/אפליקציה – התוצר הסופי של הפרויקט, יהיה אפליקציית אנדרואיד שמרצה את דרישות הלקוח ודרישות קהל היעד (מנהלי העבודה של הלקוח) גם מבחינת פונקצנאליות וגם מבחינת ממשקיות וחווית משתמש.

המכשיר - מכשיר סלולארי העובד על מערכת הפעלה אנדרואיד, יהיה בעצם הרכיב חומרה שעליו תרוץ המערכת שלנו.

מצב offline - מצב בו המכשיר לא מחובר אל האינטרנט (מצב לא מקוון) ומסיבה זאת אין חיבור אל השרת, במצב זה כל הפקודות שנרצה לשלוח אל השרת כדי לשנות את התוכן של השרת ישמרו על המכשיר וישלחו שוב במועד אחר כאשר אנו יודעים כי המכשיר עובד במצב מקוון.

SQLite - מערכת לניהול מסדי נתונים שיש לה תמיכה טבעית במערכת הפעלה אנדרואיד, אנו נשתמש בה בתור מנהלת תוכן הקבצים המקומיים ששמורים על המכשיר כזכרון cache בין השרת למוצר.

Windows Server 2012 R2 - מערכת ההפעלה (או התוכנה) שמופעלת כל הזמן על השרת, היא מספקת לנו שירותים דרך האינטרנט שמספקים את השרת ואת הלקוח, ביניהם Microsoft SQL Server ויכולת הזדהות (Active Directory).

Microsoft SQL Server - מערכת לניהול מסדי הנתונים בצד השרת, אחראית על לשמור מידע על השרת ולשלוף מידע החוצה ממנו ולספק רמת אבטחה מינימלית, המערכת תספק אותנו כשירות דרך האינטרנט.

1. מבוא

המערכת שלנו באה לתת מענה לציבור מנהלי העבודה של הלקוח שלנו שמוצאים את תהליך הדיווח של מצבי הפרויקט שלהם ועדכונים אחרים מאתרי הפרויקט אל השרת כתהליך ארוך ומייגע, תלוי מצב תקשורת באתר הלקוח ודורש ידע טכנולוגי רב.

המערכת שלנו תיתן מענה ישיר לקהל היעד שלה ותקל על תהליך התקשורת בין המכשיר הסלולארי לבין השרת כך שתהליך הדיווח יהיה לא יותר מאשר שליפת סמארטפון והקלדת נתונים, המערכת תדאג להסתיר מפני המשתמש את התקלות שהוא יקבל עקב כחוסר בתקשורת ככל שניתן ותדאג לשלוח את הנתונים בכוחות עצמה או לפחות ללא התערבות רבה מצד המשתמש.

ממשק המערכת יהווה פיצ'ר מפתח במערכת שלנו כיוון שבמהלך תוכנית העבודה וגם לקראת הסיום שלה ממשק המשתמש ישתנה ויתוחזק במקביל לדרישות קהל המשתמשים ומודולריות התוכנה תהווה יכולת שינוי והוספת פיצ'רים פשוטה מצד הלקוח ככל שעולה הצורך במהלך חיי המוצר.

2. תיאור הבעיה

דרישות ואפיון הבעיה

מצב הנוכחי של הלקוח שלנו כיום, הוא שקיים אצלו שרת במבנה הארגון ע"ג מערכת ההפעלה Microsoft Windows Server 2012 אצל הלקוח יש כעשרה עובדים בתפקיד מנהלי פרויקטים אשר נדרשים לדווח את מצב הפרויקט באופן שוטף, אבל במסגרת הטכנולוגיה שקיימת אצלם כיום הם מזינים דו"ח אחת לחודש אשר מפרט את התקדמות הפרויקט והסטטוס שלו.

לעיתים קרובות מנהלי הפרויקטים נמצאים באזורים ללא קליטה ובכך אין להם גישה לאינטרנט, ולכן אין להם גישה ישירה לשרת ויכולת דיווח מידית.

הלקוח רוצה פתרון טכנולוגי אשר יאפשר למנהלי עבודה לדווח בזמן אמת ממיקום הפרויקט מהמכשיר הסלולארי שלו את מצב הפרויקט לשרת מבלי להיות תלוי בתקשורת שלו (האם קיים קליטה או לא) ובכך ללא צורך לזכור או לרשום בצד את הדוחות אלא לדווח במיקום ו"לשכוח מהעניין" – כלומר להקל על תהליך הדיווח מצד מנהל העבודה.

הלקוח שלנו מוכן לקבל עדכונים אחת לשעה (או כאשר למנהל עבודה יש תקשורת מקוונת לאינטרנט) ואצל כל מנהלי העבודה קיימים מכשירי אנדרואיד והלקוח מוכן לפיתוח למערכת אנדרואיד.

הבעיה מבחינת הנדסת תוכנה

הבעיה הראשונה שאנו צריכים ללמוד להתמודד איתה זה דרכי התקשורת והזדהות מול שרת הלקוח, במצב הנוכחי שרת הלקוח מאובטח ודורש הזדהות לפני שתהליך ה session מתחיל, לכן אנו צריכים ללמוד על API ועל תהליך ההזדהות ברמת התוכנה כדי שנוכל להטמיע אותו ולדאוג לאבטחה מקסימלית בצד המשתמש, לאחר מכן אנו צריכים לדעת איך לתקשר עם רכיבי ה SQL של הלקוח, זה כנראה יתבצע באמצעות jdbc

הבעיה השניה שנצטרך להתמודד איתה זה ההטמעה של התקשורת במצב offline - נצטרך לבנות מסד נתונים נוסף על גבי המכשיר שיחזיק נתונים מקומיים וגם ישמור עדכונים לשרת במכשיר עד שהמכשיר יחזור למצב מקוון, וכמובן שצריך להטמיע מעין service שידע לשלוח מחדש ולזהות חיבור מחדש לאינטרנט כדי לשלוח את הנתונים שוב.

הבעיה השלישית שנצטרך להתמודד איתה זה זיהוי בעיות אבטחה באפליקציה שלנו, אנו מניחים כי מכשיר המשתמש לא יגיע לידיים הלא נכונות ובגלל שללא גישת root אין גישה לזיכרון האפליקציה שלנו במכשיר אנו צריכים לבדוק ולאבחן את שקיפות החבילות שיוצאות מהמכשיר ולנסות ככל שניתן למנוע זליגת מידע לא רצוי, בנוסף לכך אנחנו צריכים לבדוק את שקיפות הנתונים שמאוחסנים ע"ג המכשיר ולדאוג לעשות אותם כמה שפחות ברורים ללא שימוש באפליקציה.

שרת הלקוח מחובר לאינטרנט באמצעות שרת VPN, אם יתאפשר – ישנה עדיפות לVPN client מינימליסטי שיהווה חלק מהאפליקציה (כיוון שהלקוח עובד כבר עם client קיים, זוהי אינה דרישת חובה).

3. תיאור הפתרון

מהי המערכת

המערכת תיתן למשתמש את הפונקציות הבאות

1. הזדהות מול השרת
2. שאיבת כל הנתונים וסנכרון עם המכשיר
3. כל שאר ממשקי האפליקציה יעבדו עם הנתונים המקומיים (אפשרות זו ניתן לבטל ולעבוד רק ע"י קריאות שרת – כבקשת הלקוח)
4. עדכון נתונים שהמשתמש בוחר לעדכן בשדות ייעודיים (ישנם שדות שלא ניתן לעדכן)
5. תצוגה גרפית של נתונים על מצב הפרויקט בהתאם לדרישות הלקוח
6. להצפין את כל הנתונים שמאוחסנים ע"ג המכשיר
7. התחברות אל שרת הלקוח באמצעות VPN (אפשרות)

מצבי עבודה ותכנון הפתרון

המערכת תכלול שני מצבי עבודה עיקריים:

- 2 מצב עבודה אונליין – בה יש תקשורת וסנכרון עם שרת הלקוח, במצב זה נשמרים ומתעדכנים הנתונים ע"ג המכשיר וכל פעולה שהמשתמש עושה ע"ג שרת הלקוח נשלחת על שרת הלקוח
- 3 מצב עבודה offline - במצב זה אין תקשורת וסנכרון עם שרת הלקוח, מוצגים נתונים שעודכנו לאחרונה ע"ג המכשיר וכל פעולה ע"ג שרת הלקוח נשמרת במכשיר ותשלח רק אחרי שנחזור למצב עבודה online

הפתרון יהיה מערכת שמשתמשת ברכיבי Microsoft כדי להזדהות מול השרת, שימוש ב driver של jdbc כדי לבצע פעולות שאיבת נתונים מהשרת או עדכון השרת, ושימוש ב sqlite כדי לשמור טבלאות ונתונים ע"ג המכשיר ולקרוא אותם לפי הצורך

בגלל שהתוכנה עובדת מול תוכנת שרת קיימת, אנו נצטרך למצוא סביבת עבודה מקבילה או להשתמש בתוכנת הלקוח בתור סביבת עבודה במהלך הפיתוח, בגלל שקיים סיכון למידע ששמור ע"ג השרת אם אנו עובדים על גבי השרת, ננסה להשתמש במכונה ווירטואלית כדי לעשות סביבה משלנו קרובה ככל שניתן לסביבה של הלקוח, או לבקש מהלקוח "נישה" בחלק השרת שתהיה בטוחה למניפולציות

אופציה שלישית היא להכריח את הלקוח לבצע גיבויים תקופיים, אופציה שננסה להתנער ממנה עקב אי הנוחות שנגרר כתוצאה מאותה האופציה

תיאור הכלים המשמשים לפתרון

בפרויקט נשתמש במסגרת הפיתוח של Android Studio כדי לפתח את האפליקציה שלנו, את סביבת הפיתוח אנו נריץ בשני מערכות הפעלה שונות (Windows ו Manjaro Linux), שפת התכנות העיקרית איתה נעבוד תהיה Kotlin שאליה יצורפו כל הכלים שנכללים בספריות הפיתוח של Android

הפרויקט ינוהל לחלוטין ע"י מערכת GitHub שתנהל לנו את

1. מאגר הקוד
2. יומן אירועים
3. לוח מטלות
4. מצב הפרויקט ולוח ניהול לפרויקט

על מנת שנוכל לעבוד על גבי השרת בסופו של דבר, נצטרך לעבוד מול שרת במהלך הפיתוח אבל אנו לא יכולים לסכן את הלקוח בתהליכי הפיתוח, לכן עומדים לשירותנו כמה אפשרויות

- 4 שימוש במכונה ווירטואלית
- 5 שירות docker למכונה ווירטואלית ברשת
- 6 שירות Amazon AWS לפתיחת שרת בתשלום על גבי השרת
- 7 שירות Google Cloud Server זהה לשירות של Amazon
- 8 לבקש מהלקוח "נישה" לצורכי פיתוח

בנוסף, נעשה לימוד קצר לפני תחילת העבודה על הפרויקט כדי לעבוד בשפת קוטלין – מטרתנו היא לעבוד בשפת תכנות שמקוטלגת כ"העתידי" באפליקציות אנדרואיד כדי למקסם תמיכה עתידית ככל שניתן

טכנולוגיות פיתוח:

1. נתבסס על ספריית [jtds](#) כדי להתממשק מול המסד הנתונים בשרת
2. נתבסס על ספריית [scytale](#) כדי לספק גישה נוחה לספריות האחסון מפתחות של אנדרואיד
3. נתבסס על [dokka](#) בשביל יצירה של מסמכי תיעוד הקוד שלנו

4. תכנית בדיקות

ללקוח שלנו קיימת דרישה ליציבות מקסימלית של האפליקציה (כלומר כמות נמוכה ככל שניתן של קריסות) כדי להשיג מטרה זו אנחנו מבצעים בדיקות יחידה על כל המרכיבים הלוגיים המורכבים שלנו, ביניהם.

כדי להגדיר את רמת האמינות של התוכנה שלנו חילקנו אותה ל-8 בדיקות יחידה שונות של כלל המרכיבים השונים, כאשר נמצא מרכיב חדש שדורש בדיקת אמינות, נחליט להוסיף אותו לכאן, אל אחרי תכנון הפרויקט אנו בטוחים

שכלל המרכיבים האטומיים עד המנגנונים שלנו עוברים בדיקה, ושאר המרכיבים יסתמכו עליהם ולכן לא יצטרכו בדיקה עצמאית, חלק מהבדיקות שלנו אקטיביות וחלקם פאסיביות (כלומר חלקם מסתמכים על השפה וחלקם הם קוד בדיקה וחלקם הם חקירת קבצים).

- 1 את הקוד שלנו אנחנו כותבים בשפת קוטלין, קוטלין הינה שפה חדשה אשר יכולה להוציא כפלט קבצי bytecode, קבצים אלה יכולים להתקבל על ידי jvm (או במקרה של אנדרואיד, על ידי ה-ART) ומורצים על גבי המערכת בשפת מכונה שמתורגמת על ידי האימולטור של Java (JVM\ART).
- 2 קוטלין הינה שפה תכנות סטטית, כלומר השפה עושה בדיקות קלט בזמן הכתיבה ובזמן הריצה לתוכן המשתנה שלנו – אם המשתנה שלנו הוא null התוכנה לא עוברת קומפילציה, ואם בזמן ריצה משתנה שהוגדר כ-nullable הוגדר כ-null, יש קריסה של המערכת, בדיקות אלה שמבצעת שפת התכנות שלנו חוסכת לנו בדיקת יחידה של תקינות הקלט באופן רציף, הרי גם אם אנחנו נכתוב בדיקת יחידה תואמת לבדיקת קלט, הקומפילר והסביבת עבודה יתריעו לנו על המיותרות שלה ואף הקומפילר עצמו עלול להתעלם מהשורות האלה ויבצע עליהם אופטימיזציה שתוריד לנו את הבדיקה המדוברת, מעבר לכך מדובר על בדיקה חוזרת שמבזבזת משאבים ולכן היא מיותרת.
- 3 קוטלין הינה שפת תכנות חדשה שעדיין אינה בוגרת דיו, לכן ישנם כלים רבים שמספקת לנו סביבת העבודה (שבמקרה אותה חברה כתבה גם את השפה קוטלין) כדי לבדוק תקינות של אופטימיזציה ותרגום של הקוד ל-Android SDK וגם ל-Bytecode (כל בדיקה בנפרד), הבדיקות שלהם כמובן יותר מורכבות ויותר חזקות משלנו (הבדיקה היא ברמת ה-views וה-adapters) ולכן גם יותר חזקה, אנו מסתמכים על הבדיקות הני"ל מצד JetBrains כדי לספק עוד רמת אמינות של הקוד פלט שיוצא לנו, ובכל זאת, כיוון שהשפה עדיין לא בוגרת, יתכנו באגים שלא יהיו ברמתנו לתיקון, באגים אלה יטופלו ע"י קוטלין עד תאריך היעד של הפרויקט (שבה הכריזה קוטלין על להיות שפה בוגרת דיו) ולכן ניתן להסתמך על פלט השפה לפרויקט גמר שלנו.
- 4 בגלל אבסטרקטיות של הטבלאות המקומיות, הוחלט לעשות בדיקת יחידה לטבלאות רק בבדיקת המשתמשים, כל הטבלאות "מתנהגות" אותו הדבר כתוצאה מירושה מאובייקט אבסטרקטי שמנהל לנו את המסד נתונים, לכן בדיקה של טבלה אחד שקולה לבדיקה של כל הטבלאות, הוחלט לבדוק את טבלת המשתמשים, בבדיקת יחידה שלנו בדקנו מקרה של
 - חיפוש משתמשים לפני הוספת משתמשים (הגודל צריך להיות 0)
 - הוספת משתמשים וקריאת תקינותם
 - חיפוש משתמשים (הגודל צריך להיות 1)
 - חיפוש משתמשים דרך פרמטרים לקויים (אמור להיות 0)
 - הוספת משתמש זהה (אמור להתקבל כשגיאה)
 - הסרת משתמש לא קיים (אמור להכשל)
 - הסרת משתמש קיים (אמור להצליח)
 - חיפוש משתמש (אמור להכשל)
 בדיקת היחידה כבר קיימת אצלנו בקישור [הזה](#), ומורצת בכל ניסיון קומפילציה של האפליקציה שלנו, במידה והיא תכשל הקומפילציה תכשל (כתוצאה משימוש בפונקציות (assert)
- 5 נרצה להוסיף בדיקת יחידה של מנגנון הסנכרון, אבל מנגנון הסנכרון תלוי במסד נתונים חיצוני – כלומר הוא מסנכרן בין מסד הנתונים החיצוני לבין זיכרון המכשיר, מיקרוסופט לא מספקת כלי mock או בדיקת יחידה כלשהי ל-Microsoft SQL ולכן אין ביכולתנו לבצע בדיקות יחידה למנגנון הסינכרון או למנגנון שאיבת הנתונים בצורה חיצונית (אין mock object שיועד לאכול שאילות של MSSQL ולהביא פלט כלשהו, אפילו אם הפלט הוא mockable), לכן הוחלט להסתמך על ריצות אמת כדי לבדוק את מנגנון הסנכרון ושאיבת הנתונים, אנו עובדים כרגע מול שרת טסטינג שלנו (שיושב על המחשב כמכונה וירטואלית) שמכיל instance של Microsoft SQL Server ומוזנים בו נתוני דמה, הבדיקות בנתוני דמה מאוד

לא חזקות, אבל אילו, ללקוח שלנו קיים שרת טסטינג עם נתונים יותר מציאותיים בכמותם ובגדלם, לכן עצם האינטגרציה שלנו והרצה של האפליקציה שלנו ע"ג שרת הלקוח (גם אם שרת טסטינג של הלקוח) תבצע לנו בדיקות יחידה כל עוד האפליקציה רצה, מסיבה זו החלטנו להזניח את שלב ההמשך של הבדיקות יחידה שעוסקות במנגנון הסנכרון ומנגנון השאיבת נתונים

6 את מנגנון ההצפנה אנו בודקים באמצעות קריאות הנתונים, אנו ננסה לעיתים לשאוב את מסד הנתונים שעל זיכרון המכשיר ולשמור אותו על המחשב ולנסות לקרוא אותו באמצעות כלים מקצועיים, במידה ואחד הכלים הצליח לפענח חלק מהכתוב, זה יהיה כישלון בבדיקת יחידה, וכל עוד אף כלי עוד לא הצליח לקרוא אותו, זה יהיה הצלחת בדיקת היחידה, אנו נפרסם את המסד הנתונים (עם נתוני הדמה כמובן) בגיט שלנו, כדי לחזק את הבטחון של ההצפנה, בכללי, הצפנת מידע ואי קריאות שלה היא דבר שלא ניתן לבדיקה ישירה, הדרך היחידה לבדוק את זה היא להשתמש בכלים הזמינים למשתמש או באמצעות bruteforce לכל הביטים, כיוון שאין לנו את יכולת המחשוב לbruteforce ברמה הגבוהה דיו כדי לבצע בדיקות על מסד הנתונים, אנו נוניח בדיקות כאלו, על מנת לשפר את תהליך ההצפנה, נחליט להשתמש במזהה ייחודי ולאחסן אותו באמצעות Android Keystore System, מערכת חזקה לאחסון של מפתחות הצפנה, אנו שוב מסתמכים על כלים שניתנו לנו ע"י מערכת ההפעלה, אבל כלים אלה משתפרים ע"י מפתחי Android OS בכללי, ולכן ניתן להסתמך על חוזק זה כל עוד האפליקציה שלנו משוחררת ללקוח שלנו עם קוד פרטני

7 כרגע, נתונים שנשמרים ב Application Settings שלנו הם אינם מוצפנים, לכן אנו רוצים לבצע בדיקות יחידה על הנתונים הרלוונטים בApplication Settings, לדעתנו כרגע רק פרטי ההתחברות לשרת VPN הם הנתונים היחידים שדורשים הצפנה על המכשיר, ולכן, נצפין רק אותם, הבדיקת יחידה של הגדרות המכשיר מסתמכת על בדיקת יחידה של ההצפנה מבחינת אבטחה ועל ידי ניסוי הממשק מבחינת לוגיקה

8 כדי להוכיח את אמינות והצלחת האפליקציה שלנו – נבדוק אותה בעומס אמיתי בשרת הלקוח כשיגיע זמן השחרור, זה בעצם בדיקת העומס שלנו, כיוון שללקוח כמות נתונים מאוד גדולה יהיה זה גם בדיקה על יעילות האפליקציה שלנו מבחינת אחסנת parse על נתונים וגם על תוכן המידע

5. סקר שוק

פתרון טכנולוגי ראשון שעולה לנו לראש זה להשתמש בחשבון ענן כלשהו ולדווח מסמכים אליו באמצעות אפליקציות יעודיות לחשבון ענן הזה, לצורך עניין זה – נניח כי נבחר חשבון Microsoft Office 365.

בחבילת המוצרים של מיקרוסופט קיים מוצר בשם [Microsoft OneDrive](https://onedrive.live.com/)

המוצר הנ"ל בעצם מתפקד בתור מערכת אחסון קבצים באינטרנט, כלומר ניתן לאחסן קבצים בכל היררכיה אפשרית, כולל את ההיררכיה של מנהלי הפרויקטים והפרויקטים של הלקוח



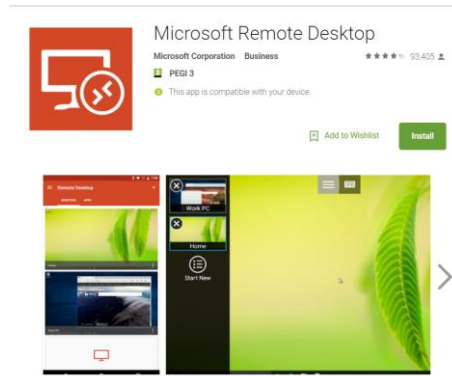
האפליקציה היא אפליקציית ענן, כלומר היא עובדת על כל מערכות ההפעלה הידועות (אנדרואיד, איפון, ווינדוס) כולל הגרסאות מובייל, לינקס, מאק.. ואף גם על חלק ממערכות ההפעלה הפחות הידועות (אקסבוקס, פליסטיישן), המערכת הושקה ב-2007 ע"י מיקרוסופט כדי לתת מענה לחברות ואנשי עסקים ואפילו למגזר הפרטי כדי להיכנס לעולם האחסון קבצים בענן ולעולם הענן בפרט תכונות כלליות:

- היררכיה של מערכת הקבצים על גבי השרת דומה מאוד לכל מערכת קבצים שאנו רגילים אליה ומאפשרת היררכיה ע"פ שם חברה, שם פרויקט, שם מנהל עבודה וכו'...
- במצב מקוון ניתן לעדכן קבצים מול השרת ונשמרים נתוני cache על המכשיר שמייצגים את מערכת הקבצים.
- במצב לא מקוון עדיין ניתן לראות את ההיררכיה של הקבצים איך שהיא הייתה בפעם אחרונה שהתחברנו אל השרת, מה שמציג לנו כמות מינימלית של נתונים
- ניתן לשמור קבצים במכשיר כך שניתן לגשת אליהם במצב offline
- ניתן לשלוח קבצים גם במצב offline, הקבצים ישלחו באמת רק אחרי שהמכשיר התחבר אל השרת
- את מערכת הקבצים ואת הקבצים כל בן אדם שמחובר לחשבון או שהמנהל חשבון "שיתף" איתו את התיקיה בענן – יכול לראות ולשנות אותה, או רק לראות אותה, לפי הגדרות מנהל החשבון

חסרונות:

- האפליקציה מייצרת תיקיה חדשה בענן וחשבון חדש בענן, כל המידע הזה לא מסונכרן עם השרת ולכן צריך להגדיר את השרת לעבוד עם החשבון וגם להעביר את כל המידע שלקטנו עד היום לחשבון זה.
- חריגה מגודל זיכרון אחסון מסוים תצרוך תשלום חודשי וקבוע

מוצר נוסף שיכול להוות פתרון אפשרי ללקוח הוא [Microsoft Remote Desktop](https://www.microsoft.com/en-us/remote-desktop)



האפליקציה נותנת שירות Remote Desktop - כלומר השתלטות מרחוק, השירות ניתן למכשירי אנדרואיד, iOS, ווינדוס (כולל הגרסאות מובייל) ולינקס maci ותמיכה בו תלויה במיקרוסופט, כיוון שהיא פיתחה את הפרוטוקול המשותף, הפרוטוקול הוטמע ב-2003 במערכות הפעלה של מיקרוסופט למחשבים הניידים והנייחים, כולל השרתים, ואילו האפליקציה יצאה במאי 2015 תכונות:

- שליטה מרוחקת ישירות על מחשב השרת
- עדכון ישיר של מסד הנתונים
- אלמנט חווית המשתמש מעודכן כל פעם על ידי מיקרוסופט כדי להשיג את חווית המשתמש המושלמת אפילו כאשר משתלטים על מכונה יותר מורכבת
- ישנה אפשרות לשרת לפתוח Virtual PC לכל Client של Remote Desktop, כלומר כל חיבור מרחוק יקבל "משתמש" משלו ו"ירוך על הרקע" של השרת כמכונה ווירטואלית עם גישה למסד הנתונים, מה שנותן מקביליות
- תיעוד של התחברויות
- ניתן לפצל הרשאות של תיקיות ב Virtual PC של מיקרוסופט כך שלמנהלי עבודה יהיה גישה רק לפרויקטים שהוצמדו להם
- מאובטח כיוון שהתקשורת מוצפנת

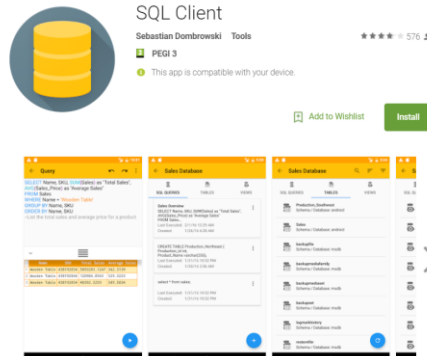
חסרונות:

- חובה חיבור אל האינטרנט כדי לבצע את כל הפעולות, כלומר אין מצב עבודה Offline
- מהירות האינטרנט תהווה צוואר בקבוק ל"מהירות המחשב" או תגובת האפליקציה

יתרונות על פני OneDrive

- השירות נותן לנו את היכולת לעדכן את המסד הנתונים על גבי השרת, לעומת OneDrive שפותח תיקיה חדשה ונפרדת בענן.

פתרון שלישי הוא להתחבר ישירות על מסד הנתונים באמצעות דרייבר, פעולה שמבצעת האפליקציה הבאה [SQL Client](#)



האפליקציה נותנת שירות להתחברות ל Microsoft SQL Server באמצעות דרייבר JDBC האפליקציה שוחררה בפברואר 2011, ונכתבה למערכת android בלבד, האפליקציה מאפשרת:

- עבודה עם שירותים Microsoft SQL server, Postgres SQL, MySQL
- הפעלה ושמירה של שאילתות SQL
- להפעיל מספר פקודות SQL בשאילתה בודדת
- לבצע פעולות סינון וחיפוש על תוצאות של שאילתות
- שיתוף פקודות SQL
- שמירת נתונים שנשלפו ממסד הנתונים במכשיר בפורמטים שונים
- תיקון סינטקס בזמן ריצה
- תמיכה באלמנט הנוחות של undo/redo

האפליקציה משתמשת במנגנון שאנו רוצים להטמיע באפליקציה שלנו, כלומר היא תומכת ברוב הדרישות שלנו למעט:

- שמירת שאילתות במצב עבודה Offline ושליחתם שהמכשיר עובר למצב online
- שמירת נתוני cache של המסד הנתונים בצורה אוטומטית
- מנגנון אבטחה כתוצאה מהזדהות של המנהל עבודה עם שם משתמש וסיסמה

וזו הסיבה שללקוח שווה להשקיע בפרויקט, אנו נספק לא רק את צרכי מבחינת נוחות ופונקציונליות אלא גם נספק את צרכי מבחינת אבטחת תוכנה ונתמך בדרישה של הלקוח לעבודה במצב offline לחלוטין

6. ריכוז פרויקטים דומים

קיימים מספר פרויקטים שיש להם רמת דמיון לפרויקט שלנו ביניהם

WhatsApp

אפליקציית WhatsApp מספקת שירות הודעות בין משתמשים שונים דרך שרת WhatsApp שמספק את השירות הזה דרך האינטרנט, הודעות WhatsApp שנשלחו לשרת והתקבלו בשרת (או אפילו אצל הנמען) בהצלחה מקבלות אינדקציה באמצעות סימני וי



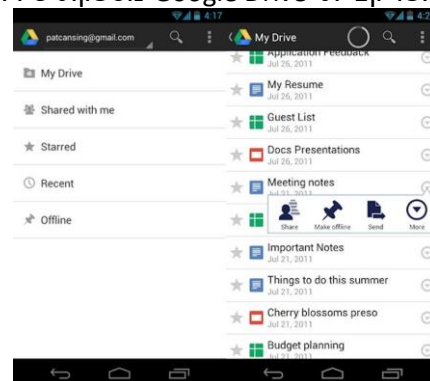
והודעות שלא הצליחו להתקבל בשרת מקבלות סימן של שעון, ונשמרות באפליקציה (בזכרון cache מקומי) עד כאשר ישנו חיבור חוזר אל האינטרנט ואז נשלחות



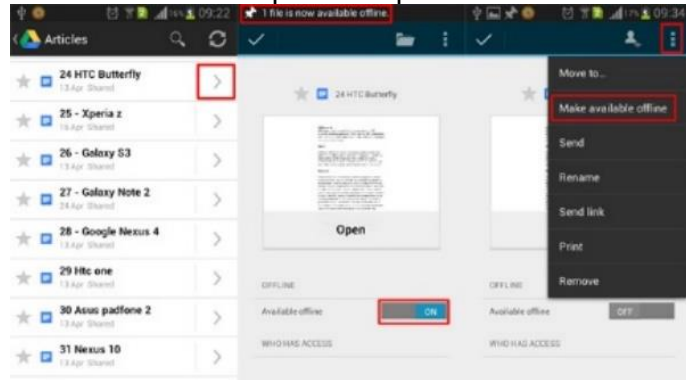
באותה רמה – אנו רוצים לממש מנגנון זהה אבל לשרתי Microsoft SQL Server 2012, שימומש בצורה דומה לצורה שבה הוא ממומש באפליקציית WhatsApp

Google Drive

אפליקציית Google Drive מספקת שירותי ענן לאחסון קבצים



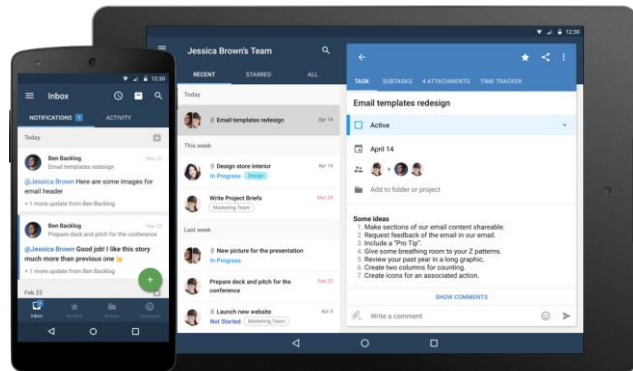
מעבר לסיפוק שרות זה, Google Drive נותנת למשתמש יכולת לשמור קבצים מסוימים בזיכרון המכשיר שהמכשיר יכול לקרוא את הקבצים האלו ללא כל גישה אל האינטרנט



באופן דומה אנו רוצים שהאפליקציה שלנו תציג רשימת פרויקטים ומידע נוסף שנשמר על זיכרון המכשיר על הפרויקט ללא כל גישה אל האינטרנט

Wrike

אפליקציית Wrike מספקת שירות לניהול פרויקטים במאגר כלשהו ברשת, אפליקציית Wrike נותנת שירות לשמירת ל"ז, מיון לפי נושאים, ניהול משימות ותתי משימות ותצוגה של חיי הפרויקט בצורה גראפית



באופן דומה אנחנו רוצים לספק שירותים דומים ברעיון אבל לא זהים במהותם, כלי עזר למנהלי הפרויקטים של הלקוח שלנו, שיספקו את השירותים שהם צריכים במדויק לסיוע לניהול הפרויקט במהלך הפרויקט נרצה להטמיע שירות זהה למסופק ב Wrike שמציג את חיי הפרויקט בצורה גראפית, אבל בצורה יותר מפורטת ומותאמת לפרויקטים של הלקוח שלנו.

7. נספחים

א. רשימת ספרות \ ביבליוגרפיה

1. Professional Microsoft SQL Server 2012 Integration Services by Brian Knight, Erik Veerman, Jessica M. Moss, Mike Davis, Chris Rock, 14 March 2012

https://books.google.co.il/books?id=7_g0SocQ3CcC&sitesec=buy&hl=iw&source=gb&atb

2.

Database Programming with JDBC and Java by George Reese, 2000

<https://books.google.co.il/books?id=oPbGi0I0ZHEC&sitesec=buy&hl=iw&source=gb&atb>

3.

The Android Developer's Collection (Collection) by James Steele, Nelson To, Shane Conder, 9 December 2011

<https://books.google.co.il/books?id=3Wi2gwGoZZ0C&sitesec=buy&hl=iw&source=gb&atb>

4.

Getting Started with JDBC By David Reilly, 5 June 2006

<http://www.javacoffeebreak.com/articles/jdbc/>

5.

אתר ממשק הפיתוח של אנדרואיד

<https://developer.android.com/index.html>

6.

אתר ממשק הפיתוח של Microsoft SQL Server

<https://docs.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server>

7.

אתר ממשק הפיתוח של מגנון ההזדהות של Microsoft Server

<https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-developers-guide>

8.

ממשק פיתוח משני ל-Microsoft SQL Server

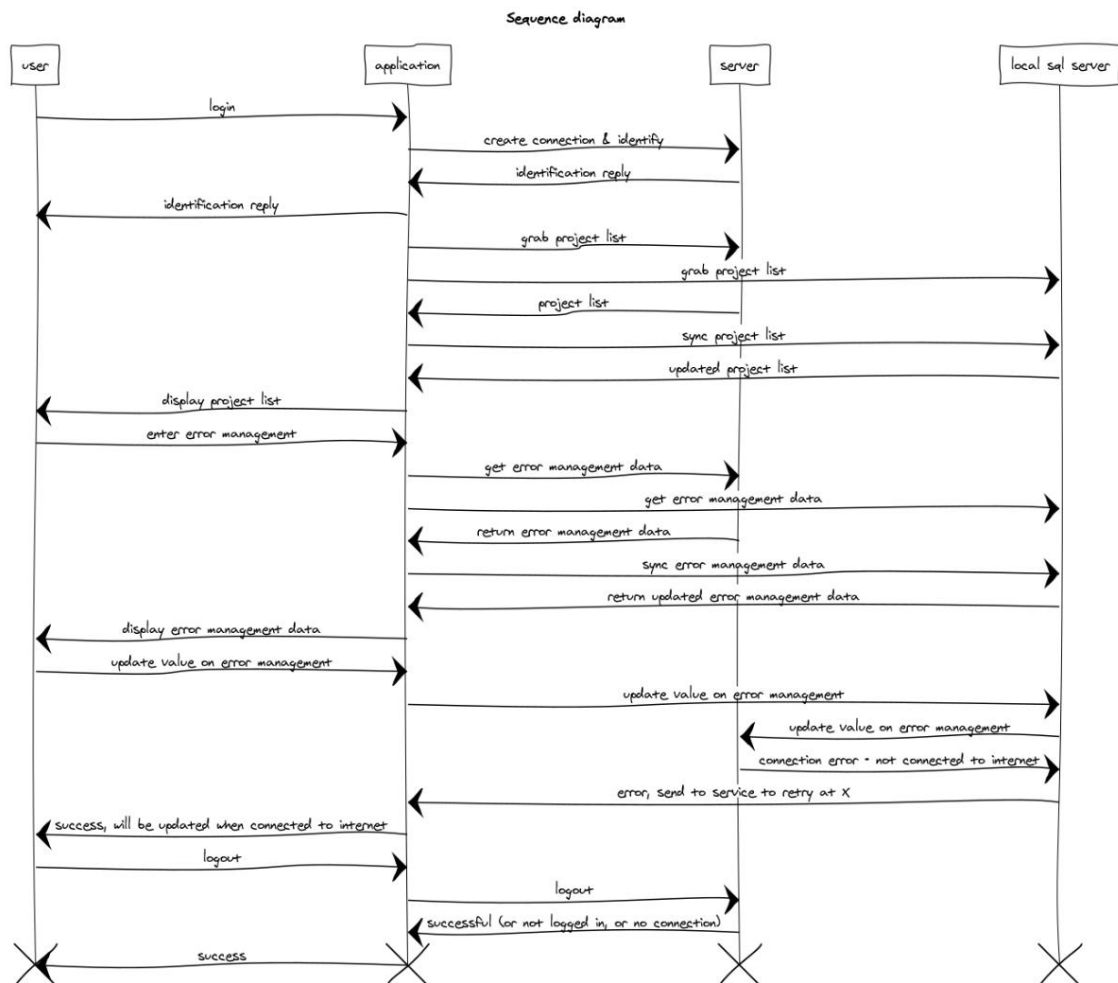
<http://jtds.sourceforge.net>

ב. תרשימים וטבלאות

כל הדיאגרמות הבאות הם להמחשה בלבד ואינם סופיות!

דיאגרמת מצבים – מצב offline פתע

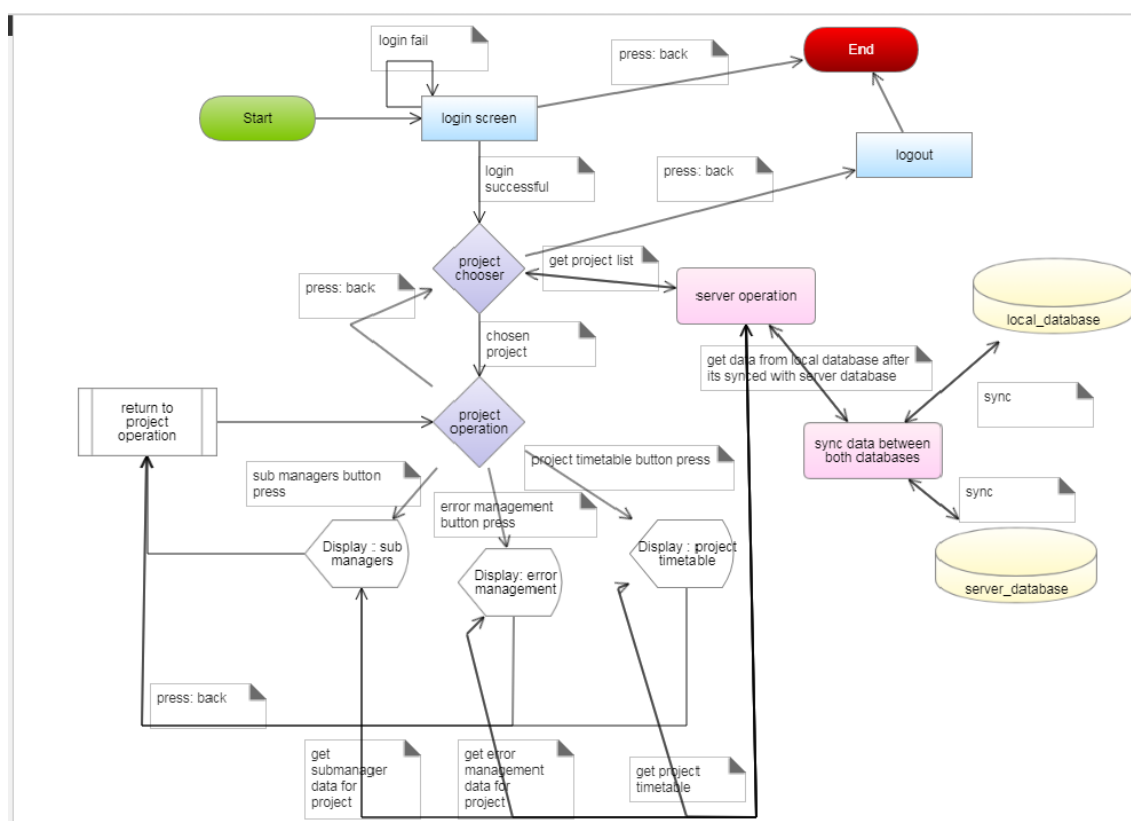
על מנת שנוכל להסביר את חשיבות המימוש של מצב offline אנו מתארים
בנספח זה את מקרה בודד מבין המצבים האפשריים כאשר שירות האינטרנט
או החיבור אל השרת נופלים



כמו שמתואר המנהל עבודה ניגש לאפליקציה, מתחבר ובוחר בפרויקט ובוחר לעדכן שדה כלשהו, ובשלב זה האפליקציה מאבדת חיבור אל השרת, האפליקציה דואגת לשמור את הפעולה בזיכרון המכשיר ומציגה למנהל עבודה הודעת הצלחה ותוספת שהשינוי יעודכן בשרת אחרי שחיבור אל האינטרנט יוקם מחדש, מצב זה מראה בצורה אידיאלית שקיפות והגינות כלפי מנהל העבודה ומקל על המנהל עבודה – כי במקרה אחר הוא היה צריך לשמור את הנתונים בצד בצורה

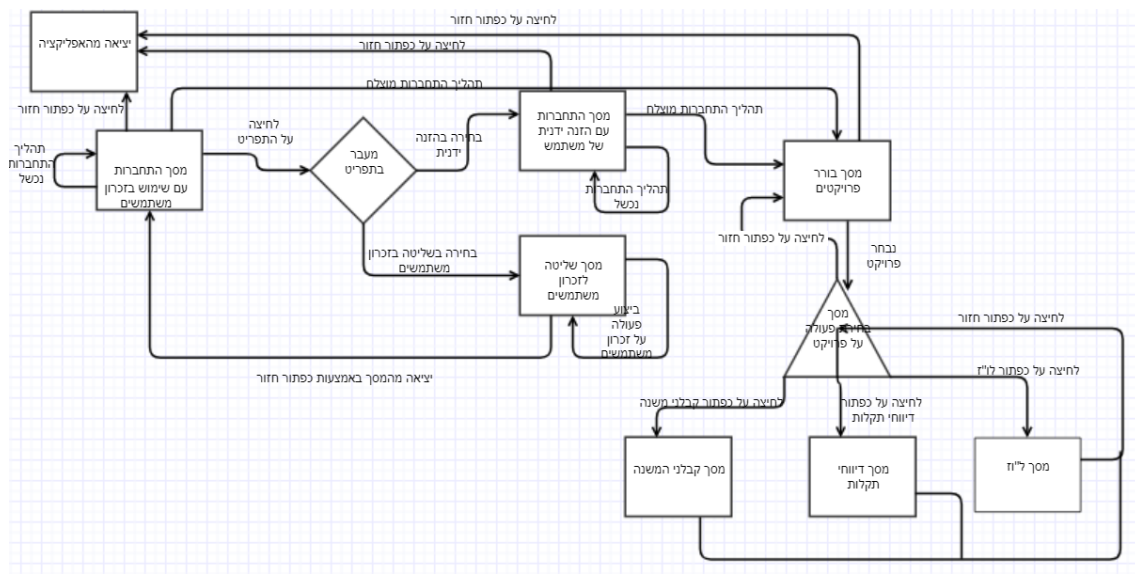
Flow-chart

בתרשים זה אנו מציגים את מערכת המצבים משלב ההתחברות עד תצוגת המידע במסכים ואפילו עד היציאה מהאפליקציה והסגירה שלה



כמו שניתן לראות לאחר ההתחברות ישנו בורר מצב פרויקטים ולאחר בורר מצב הפרויקטים יש בורר מסכים שמפרט איזה מידע נרצה לראות על הפרויקט כל המידע הזה מתקבל מגוף כלשהו שמסנכרן את המידע בין הזיכרון cache המקומי לבין השרת, ולאחר מכן המידע נשלח אל המסכים הרלוונטים

דיאגרמת מצבים למסכי הפרויקט



הדיאגרמה הבאה דומה מאוד לדיאגרמה הקודמת, אבל כאן יש התמקדות במסכים ולא במצבים, כיוון שמסכים חדשים לרוב מייצגים מצבים חדשים ניתן לראות דמיון רב בין דיאגרמת ה flow-chart לבין הדיאגרמה הנוכחית, אבל כאן רואים יותר התמקדות בין מעברי המסכים ואין התייחסות למסד הנתונים או למקור שלילת הנתונים למסכים.

מסך התחברות לשרת – עם זכרון של חיבורים אחרונים שהתבצעו



התפריט שנפתח עם אופציות שונות במסך ההתחברות



מסך ההתחברות – אופציית ההזנה הידנית של משתמש וסיסמה במידה ויש משתמש חדש להוסיף לזכרון המכשיר



במידה והמשתמש רוצה למחוק משתמשים מהזכרון מכשיר או לשנות את הסיסמה המוזנת אוטומטית בתהליך ההתחברות, יש מסך שליטה לזה



לאחר מסך ההתחברות – בורר הפרויקטים, בחירת הפרויקט בו המשתמש ירצה

לעסוק



האופציות השונות שניתנות לאחר בחירת הפרויקט



אופציית קבלני המשנה

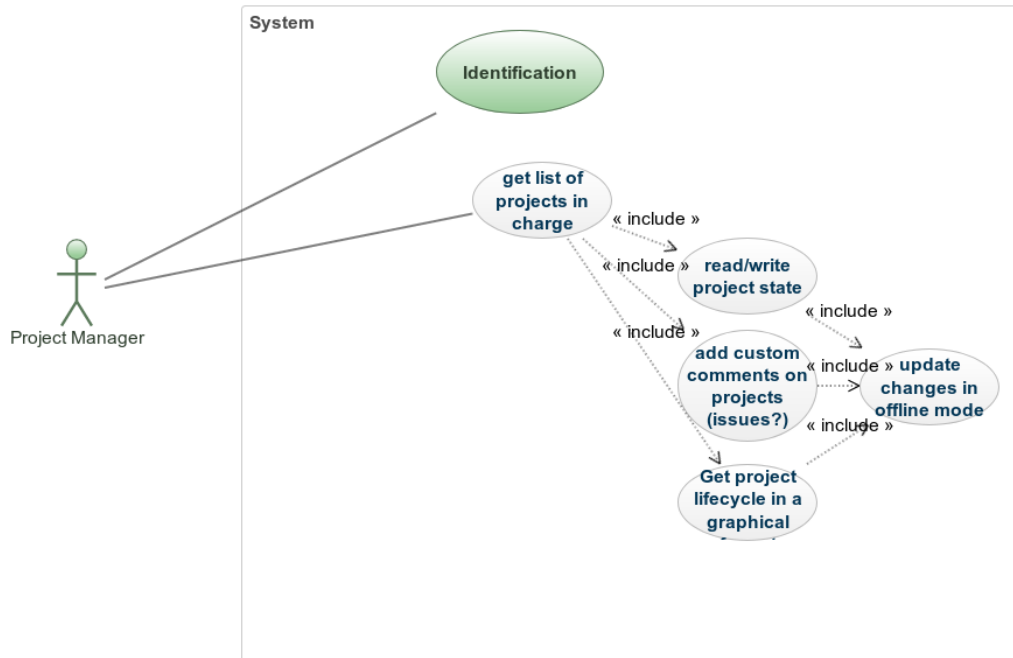


אופציית הדיווחי תקלות



Use-cases

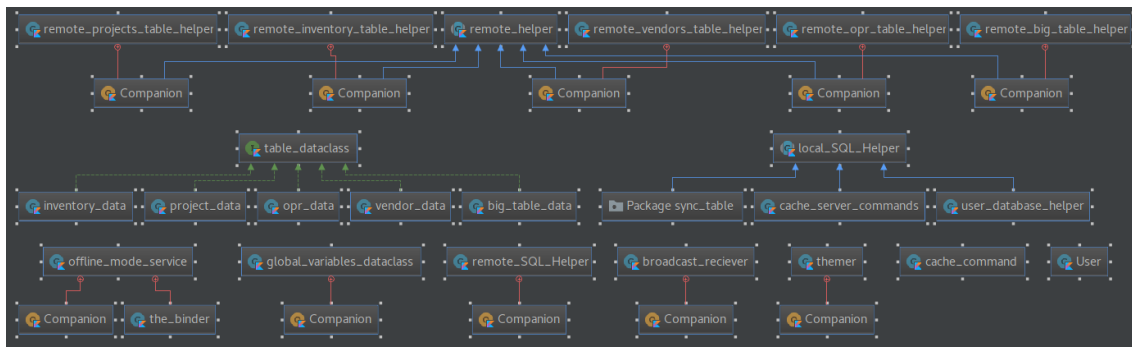
התרשים הבא מייצג את המנהל עבודה ואת השירותים שהוא יכול לקבל מהאפליקציה בשלב הנוכחי שלה, התרשים לא מייצג את המוצר הסופי כיוון שעוד לא ניגשנו אל מסד הנתונים ואנחנו לא יודעים איזה מידע נמצא במסד הנתונים אנחנו מראים רק חלק מהמידע שאנחנו מודעים שהוא יהיה נגיש אלינו והמנהל עבודה יהיה מעוניין לראות



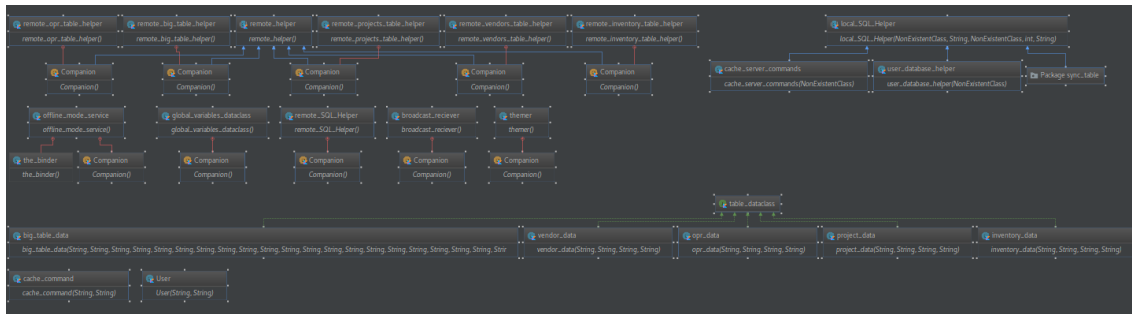
UML

התרשים הבה מייצג את יחסי המחלקות שידועות לנו עכשיו בצד הלוגי, כמובן שקיימים מחלקות שמטרתם היא יותר גראפית (כמו ה-Activity של Android או ה-OpenSQLHelper של Java\Android או ה-JDBC), אנחנו לא נתייחס אליהם אפילו שחלקם מתייחסים לחלק הלוגי של האפליקציה, אלא נחשוב כי קיים Interface בשם SQLite שמטפל בחלק המקומי של המסד הנתונים ו Interface בשם Microsoft SQL driver שיטפל בחלק המקוון של המסד הנתונים.

UML ראשון – מבט על ללא ירידה לפרטים במחלקות הראשיות בלבד



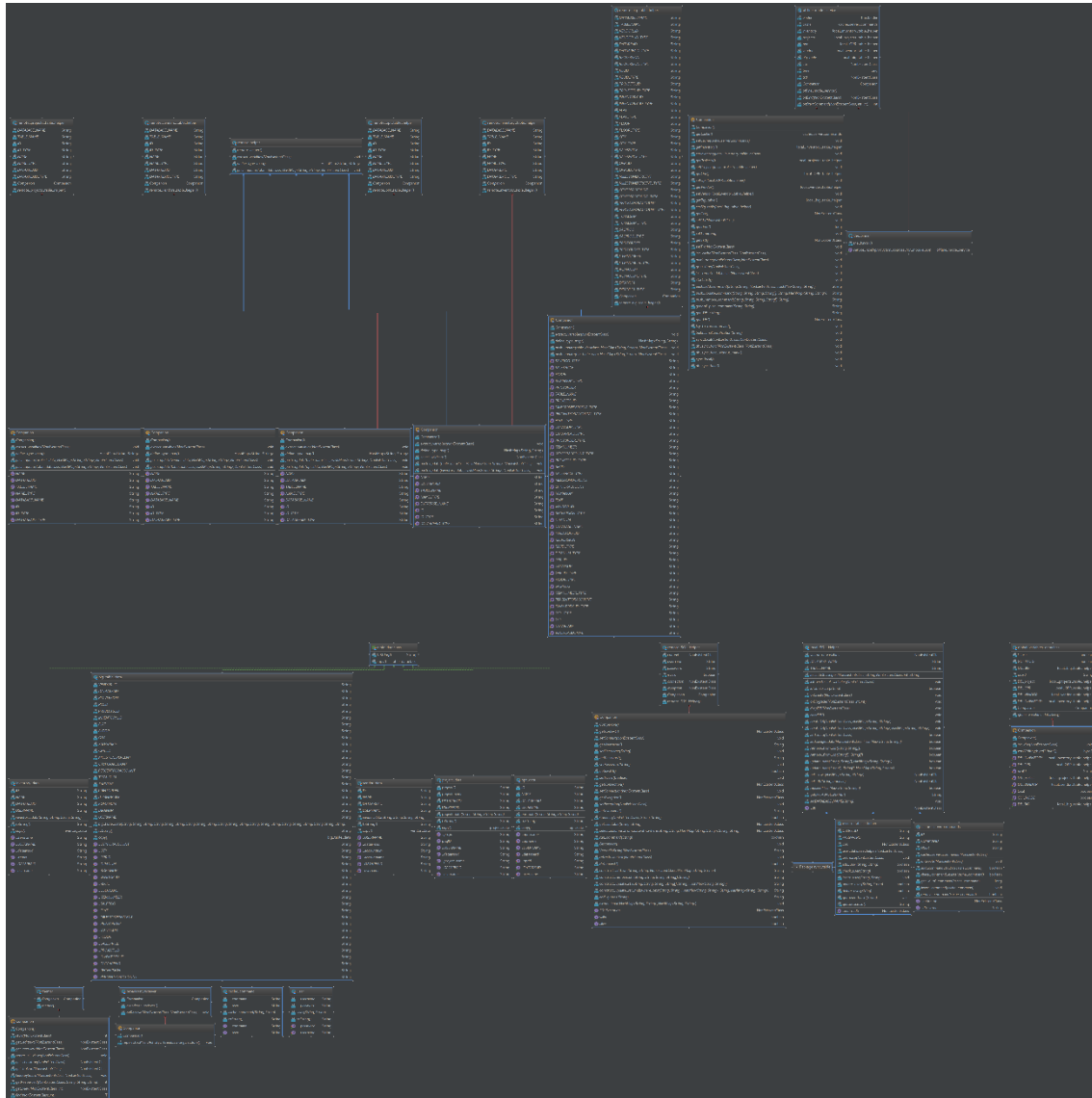
UML שני – UML למעלה עם סוגי הבנאים (לשם הבנת מורכבות בלבד)



UML שלישי – כמו הUML הקודם, בתוספת הclass Members של כל מחלקה



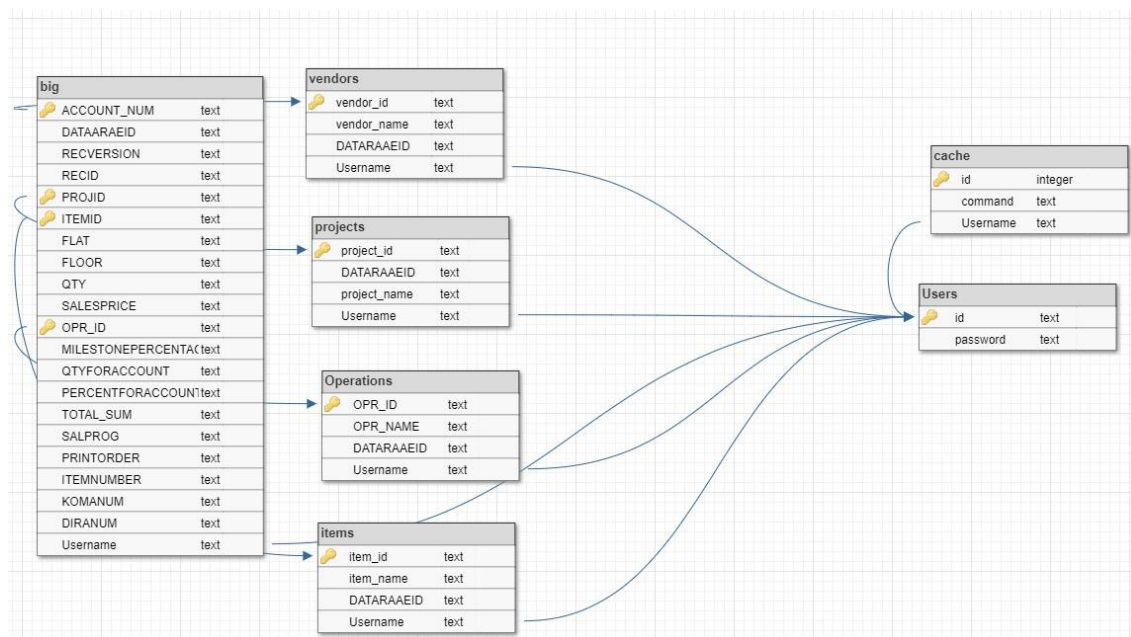
UML רביעי – כמו הUML הקודם בתוספת פונקציות (עדיין רק מחלקות ראשיות)



טבלאות במסד נתונים

מסד נתונים לזכרון משתמשים שחוברו לאחרונה – בו נרשמים שמות משתמשים וסיסמאות של משתמשים אשר הצליחו להזדהות מול השרת, מקל על תהליך ההתחברות החוזר ונשנה מאותו המכשיר

מסד נתונים לזכרון cache של נתונים מצד השרת כדי לספק את התשתית לoffline mode (כלומר ישמרו נתונים כמו שמות פרויקטים ונתונים אחרונים שעודכנו, ופעולות על השרת שלא הצליחו להגיע אל השרת כתוצאה ממצב לא מקוון של המכשיר ישמרו במסד הנתונים הזה וימחקו רק לאחר הצלחת שליחת הנתונים אל השרת)



ג. תכנון הפרויקט

פגישת הכרות עם הלקוח	24.8
פגישה נוספת על הלקוח כדי להבין את דרישות הלקוח	7.9
פגישות קבועות בימי רביעי עם צוות הפיתוח לצורך קידום הפרויקט, פיתוח וחלוקת עבודות (מעין scrum)	22.10 – הגשת הפרויקט
פגישות עם הרכז	קישור ליומן
הגשת מסמך ההצעה, יצירת מכונה ווירטואלית מקבילה לשרת	19.11.2017
פגישה עם הלקוח בה נציג PoC של המוצר	סוף סמסטר א'
הגשת מוצר במצב ע"פ דרישת הרכז בסוף סמסטר א', זה כנראה יהיה מוצר אלפא	2.1.2018
הגשת מוצר בטא למנהלי העבודה של הלקוח שלנו לצורך קבלת feedback לאפליקציה (כנראה בעיקר חווית משתמש)	[תאריך שיקבע ע"י הרכז]
נגיש מוצר בטא נוסף עם תיקונים ע"פ בקורות מהשחרור בטא האחרון	שבועיים לאחר הגשת מוצר הבטא
הגשת הפרויקט	[תאריך שיקבע על ידי הרכז]
הגשת האפליקציה ללקוח והעלאה לחנות של גוגל	[תאריך שיקבע לאחר סיום תהליך הפיתוח]

ד. טבלת סיכונים

#	הסיכון	חומרה	מענה אפשרי
1	עיקוב רב עקב תהליכי פיתוח שאנו חשבנו כי יהיו פשוטים אך קשים יותר בפעולה מול שרת הלקוח	4	קריאת הספרייה של התוכנה שמורצת ע"ג שרת הלקוח ושליחת מיילים במידה ולא נמצא פתרון בחיפוש ברשת
2	סיכון למידע השמור ע"ג השרת עקב באגים שיתרחשו בתהליכי הפיתוח	9	ניסיון למצוא סביבת פיתוח מקבילה לסביבת העבודה שבה המערכת שלנו תעבוד, או לבקש מהלקוח מעין "נישה" בטוחה בשרת
3	המוצר הסופי לא יספק את חווית המשתמש הרצויה של קהל היעד שלנו	2	המוצר ישוחרר בכמה גרסאות כמה פעמים במהלך חיי הפרויקט כדי לקבל feedback מצד קהל היעד שנוכל לשפר את חווית המשתמש בתהליכי הפיתוח ואף לקראת סיומה
4	גילינו לאחר פיתוח הבטא ולאחר הגשת האפליקציה כי חלק מהחבילות שנשלחות מכילות מידע רגיש בצורה גלויה	7	כיוון שאבטחת מידע היא התמקדות מאוד גבוהה בסביבה הארגונית, נודיע מיד ללקוח על הבעיה ונסה ככל שניתן ביכולתנו לפתור את הבעיה טרום פיתוח, אחרת פשוט נודיע למשתמש בצורה גראפית על הסיכון
5	ישנם שירותים במכונה הווירטואלית שיצרנו שהתבססו עליהם אבל השרת לא תומך בהם	7	נצטרך לבצע בדיקות יזומות לאחר כל Milestone על גבי השרת
6	שימוש במסד הנתונים כזיכרון cache יהיה מכשול אבטחת מידע בגלל מעבר של משתמשים באפליקציה	3	נוסיף למסד הנתונים המקומי שדה מזהה למנהל העבודה ונעשה טעינה לכל משתמש בנפרד
7	לא נצליח להטמיע VPN client מינימליסטי באפליקציה שלנו	1	ללקוח יש פתרון VPN נפרד, ואפילו עדיף שיתחזק ע"י חברה שמתמחה בתחום האבטחה, הרצון שלנו לספק client VPN הוא אך ורק כדי להגביר את רמת השלמות של הפרויקט.

ה. רשימת טבלת דרישות

פורמט טבלת הדרישות יהיה לפי המקובל בארגון. להלן דוגמא:

טבלת דרישות (User Requirement Document)

מס' דרישה	סוג	תיאור
1	פונקציונאלי	המוצר יכול להזדהות מול שרת הלקוח

בצורה ייחודית כל משתמש לכל מנהל עבודה – תוכנה		
המוצר יכול לתקשר עם שרת הלקוח כדי לשאוב נתונים ולכתוב נתונים על הפרויקט הנתון – תוכנה	פונקציונאלי	2
המוצר ידע לעבוד במצב offline, כאשר לא קיים חיבור אל האינטרנט פעולות על השרת ישמרו בזכרון המכשיר וישלחו בהזדמנות – תוכנה	פונקציונאלי	3
המערכת תהיה מודולרית על מנת שתוספים ופיצ'רים חדשים שהלקוח ירצה להוסיף בהמשך חיי הפרויקט לא יהוו מכשול לפרויקט – תוכנה	תמיכה לעתיד, הבנת הקוד	4
המוצר חייב לתמוך במערכת הפעלה אנדרואיד בגרסאות המקובלות לשימוש כיום (6 ומעלה) – חומרה/תוכנה	תמיכה נוכחית, ממשקיות	5
במהלך חיי הפרויקט רמת האבטחת מידע של המוצר חייבת לרצות את הלקוח כגוף ארגוני – אבטחה/תוכנה	אבטחה	6
ממשק המשתמש יהיה מיועד למנהלי העבודה, קהל מפתחי העבודה של הלקוח שלנו יהיה בעל הדעה החזקה של איך אמור להראות ולהתנהג הממשק, וע"פ דרישה יהיה בעל מספר מינימלי של "קליקים" – ממשק משתמש/תוכנה	ממשק	7
כתיבת טסטים מרובה תאפשר יציבות מאוד חזקה ותמנע את הצורך לתחזק את האפליקציה – תוכנה	יציבות	8

טבלת סיפורי משתמשים (User Case Stories)

בתור מנהל עבודה, אני יכול לשלוח נתונים לעדכן את השרת בנתונים חדשים מבלי להתייחס למגבלות המכשיר לחיבור לאינטרנט	1
בתור חבר בצוות ניהול החברה, אני יכול לראות את רשימת כל הפרויקטים בלחיצת כפתור פשוטה מבלי להכנס לדפדפן ולשירותים טכנולוגיים כמו vpn ו rdm כדי לראות את מצבי הפרויקטים על מסך הפלאפון שלי	2
בתור מנהל עבודה, אני יכול לראות תרשים המדמה כלי מאוד חזק בעולם ניהול הפרויקטים מבלי לחשב את הנתונים בצורה עצמאית – מכשיר האנדרואיד שלי מחשב אותם עבורי	3
בתור אחראי אבטחה המידע של הארגון, יותר פשוט מבחינתי להתייחס למידע שזולג מהאפליקציה מאשר להתייחס למידע שזולג מכל דפדפן או כל שירות בנפרד, כל ה"סיפור" נשאב למקור בודד	4
בתור מנהל עבודה, יש לי את היכולת לראות את הנתונים האחרונים שראיתי על הפרויקט בצד השרת ללא צורך בלשמוע אותם בצד בצורה עצמאית, וללא תלות בחיבור שלי אל השרת	5
בתור מנהל עבודה יהיה לי בטחון שאם אאבד את המכשיר פלאפון שלי וגוף שלישי מחזיק בו – הוא לא יוכל לפגוע בלקוחות שלי בקלות	6