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:

מערכת	מיקום
מאגר קוד	https://github.com/meitarsh/m.s-aluminium-manager-app
יומן	/https://calendar.google.com/calendar embed?src=t1f2ojv5arrqone i6h09i5fld0%40group.calendar.google.com&ctz=Europe/Athens 5arrqonei6h09i5fld0%40group.calendar.google.com&ctz=Europe/Athens
ניהול פרויקט (אם בשימוש)	https://github.com/meitarsh/m.s-aluminium-manager-app/projects/1
הפצה	https://github.com/meitarsh/m.s-aluminium-manager-app/releases
סרטון גרסאת אלפא	https://github.com/meitarsh/m.s-aluminium-manager-app/wiki/Alpha_video

תוכן עניינים

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, נחווה 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 נחווה 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 וכתיבת 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. תודות

ברצוננו להודות למר עופר פישלוביץ – על היוזמה והסבלנות להשאר איתנו איתן עם כל הקשיים בתקופת הפרויקט, למר שי תבור שלכל בקשה ומענה נתן ייעוץ מקצועי וחשב על הפתרון בכל יצירתיות אפשרית, לדר' שמרית צור דוד על העזרה בהבנת נכונות האבטחה במערכת אנדרואיד הקיימת ואיך יכולנו להסתמך עליה ומתי, לדר' אסף שפיינר על הסבלנות והקושי הרב שהוא עובר (לא רק איתנו) בכל פרויקטי המכללה

ובפרט אנו מודים לחברת מ.ש אלומיניום שנתנה לנו את ההזדמנות להוכיח את עצמנו ולהראות לה שגם סטודנטים יכולים לפתור בעיות באמצעות תוכנה

3. 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 התקשרות והזדהות מול שרת הלקוח, במצב הנוכחי שרת הלקוח מאובטח ודורש הזדהות לפני שתהליך ה session מתחיל,

לכן אנו צריכים ללמוד על ה 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 ל client VPN מינימליסטי שיהווה חלק מהאפליקציה (כיוון שהלקוח עובד כבר עם client קיים, זוהי אינה דרישת חובה).

4. 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 גראפית 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 שתיהיה בטוחה למניפולציות

אופציה שלישית היא להכריח את הלקוח לבצע גיבויים תקופים, 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"העתיד" באפליקציות אנדרואיד כדי למקסם תמיכה עתידית ככל שניתן

טכנולוגיות פיתוח:

נתבסס על ספריית 1. jtds כדי להתממשק מול המסד הנתונים בשרת

נתבסס 2.scytale כדי לספק גישה נוחה לספריות האחסון מפתחות של אנדרואיד על ספריית

נתבסס על 3.dokka בשביל יצירה של מסמכי תיעוד הקוד שלנו

5. תיאור המערכת שמומשה

המערכת שלנו מתחלקת ל 6 רכיבים:

1. רכיב offline service – רכיב שתפקידו לבצע סנכרון בעליה של האפליקציה מול השרת, וכל זמן מוגדר מראש לסנכרן שוב, במהלך הסנכרון המערכת תבצע בקשות להוריד את השינויים מהמסד נתונים, וכאשר השינויים נקלטו במסד הנתונים, המערכת טוענת לזכרון ווקטור המייצג את כל האוביקטים שנמצאים כרגע על הקובץ המסד נתונים עצמו – בגלל שה מסד נתונים יהיה בזכרון, זמני הטעינה שלו יהיו הרבה יותר מהירים ובכך נחסך הקושי של הקריאה החוזרת ונשנת של הקובץ, בשביל לעשות את זה אנחנו צריכים "לחשוף" את מסד נתונים החוצה, שאילתות עדכון בד"כ נשלחות דרך ה offline service, הוא דואג לשמור את השאילתת שינוי אצלו במסד נתונים מיוחד, וכל זמן מוגדר הוא ינסה להפעיל את השאילתה, השאילתת שינוי תמחק מהמסד נתונים המקומי רק כאשר השרת **הצליח** לבצע את השאילתה שביקשנו (כלומר תיאורטית, אם שאילתה לא מורכבת כמו שצריך היא "תתקע" לנצח), אבל הרווח שלנו הוא שאנחנו מצליחים לוודא ששינויים ישלחו גם אם היה חיבור לאינטרנט ולשירות VPN וגם אם לא.

2. רכיב הצפנה – מערכת שיודעת לקבל מידע ולהחזיר מידע מוצפן, וגם ההפך, המערכת הצפנה שלנו בעצם עובדת בצורה דו שלבית, תחילה, אנחנו מבקשים ממערכת ההפעלה מפתח סימטרי כלשהו, המפתח הסימטרי שקיבלנו מהמערכת הפעלה הוא ייחודי לנו (בגלל החתימה של האפליקציה) ואף אפליקציה אחרת או תוכנה אחרת או גוף אחר לא יכול לבקש אותו מהמערכת ההפעלה – זו האבטחה שקיבלנו ברמת מערכת הפעלה, אבל לא ניתן להשתמש במפתח הסימטרי הזה, כיוון שהוא מומש ע"י ווקטור נוסף כלומר במידה ואני יצפין את אותה המילה פעמיים, אני יקבל שני תשובות שונות, זה לא כלכך טוב לנו כי במידה ואחסנו מידע מוצפן בתוך המסד נתונים, כדי לבצע את השאילתת חיפוש נצטרך בעצם לשאוב את כל המידע ולפענח אותו, במקום לעשות שאילתת select .. where רגילה, וזה למה אנחנו משתמשים במפתח סימטרי אחר, שאותו אנחנו רוצים לאחסן כקובץ בתיקית האפליקציה של המערכת הפעלה, אבל אותו בעצם נצפין ע"י המפתח הסימטרי שקיבלנו מהמערכת הפעלה, את השאיבה של המפתח הזה נעשה חד פעמית בכל בקשה ראשונה להצפנה\פענוח, נוודא את זה בכך שרכיב ההצפנה שלנו יהיה singleton – כלומר יהיה רק מופע אחד שלו בכל האפליקציה, בתהליך ההצפנה עצמו אנחנו מבצעים הצפנה סימטרית, ממירים לבסיס 64 ואז שומרים את הנתון, בתהליך הפענוח אנחנו מבצעים decode מבסיס 64, מפענחים ואז מקבלים את הנתון – השמירה של נתונים בבסיס 64 נותנת byte dellusion של המידע, כלומר אנחנו יכולים לשמור את המידע המוצפן כ string כי לא יהיו ביטים שלא יוכלו לעבור ל charים מתאימים ומובנים, ההצפנה כמובן מתרחשת ברמת הבייטים.

3. רכיב גלובלי – בעצם זה רכיב singleton שמכיל את כל המשתנים שמייצגים את המצב הנוכחי של האפליקציה וכל הרכיבים שטוענים לזכרון שאנחנו רוצים להעביר לכמה מחלקות לפי הצורך, הדרך האנדרואידית זה להעביר מעין "אריזה" של הרכיבים האלו בין כל מחלקה אחד לשני, אבל בדרך זו קשה

יותר לשמור על שינויים ועדכון כל המחלקות, והקריאה הרבה יותר כבדה, לכן גילינו שעדיף לחסוך את הצורך בלהעביר את המידע לכל מחלקה בנפרד ולהחזיק את המידע והמצב הנוכחי במחלקה יחידה, חלק מהרכיבים שנטענים זה המופעים של המסדי נתונים, הווקטור של כל המסד נתונים, הפרויקט שבו אנחנו עובדים כרגע, וכמה פונקציות שאנו רוצים שיהיה גישה מכל מקום כמו פונקציה שוקראת למערכת ההצפנה.

4. themer – עוד רכיב singleton שאחראי על כל העניינים הוויזואלים במערכת שלנו, הרכיב בעצם יודע להחליט על "רקע" מסוג אחד וגם יודע לשלוף ולייצר לנו רכיבים גראפיים, האלטרנטיבה השניה שלנו זה להטמיע את הרכיב עמוק בתוך view כירושה או פונקציות נוספות, אבל רצינו שליטה מאוד מהירה על הרכיב והפונקציות שלו – לכן הפרדנו אותו, החסרון הוא שאנחנו צריכים להפעיל אותו בנפרד בכל activity, כלומר אנחנו צריכים לדאוג לקרוא לו שישנה את הרקעים, וזה חייב להתבצע בקטעים מאוד קריטיים ומסוימים, אבל הרווח הוא שאנו מסוגלים לבטל\לשנות את ההשפעה שלו בצורה ישירה.

5. remote SQL helper – מחלקה מסוג singleton שמימשנו שיודעת לבצע חיבור למסד הנתונים בתצורת microsoft SQL ע"י שימוש ב jdbc driver, הבחירה שלנו לגבי jdbc למרות שהוא לא הכלי הרשמי של microsoft נבעה בעיקר מהיעילות שלו, הדרייבר הנ"ל יודע להחזיק חיבור מתעדכן עם השרת, כלומר אם אני מבצע את אותה השאילתה פעמיים, ובין הפעם הראשונה לפעם השניה לא התבצע עדכון נתונים ע"ג השרת, הדרייבר ידע להחזיר לי את אותה התשובה מבלי לבצע את הבקשה מול השרת, בנוסף הוא שומר שאילתות שנשאלו בפורמט סטרינגי בצורה פנימית כך שהוא חוסך את הזכרון שנדרש להקצות עבור כך, הדרייבר עובד בחיבור ssl אפשרי, כלומר במידה והשרת תומך בכך, ולא מכריח אותנו להביא תעודה "רשמית" ע"י ארגונים אחרים, מה שחוסך ללקוח שלנו תקציבית את הצורך בבקשה לחתימה ע"י ארגון אחר המיועד לכך, המחלקה שלנו מוסיפה לשירות ה jdbc גם את היכולת להרכיב שאילתות שונות (שאילתת שליפה\עדכון\הוספה\מחיקה) שמימשנו בפונקציות נפרדות, בנוסף לכך שינינו את שאילתת העדכון שתדע גם להוסיף עמודה חדשה לפי הצורך, השאילתות שליפה\עדכון שלנו יודעות לעדכן את זמן העדכון, ולשאול לפי זמן העדכון, כלומר רשומה מסוימת התעדכנה בשעה 12, ואנחנו בעל מידע שהתעדכן לאחרונה ב 11, אנחנו נקבל רק את כל השדות החדשים שאין לנו ואת כל השינויים שאין לנו ,כמו לדוגמא את הרשומה המסוימת, וכך גם בכתיבה – בעצם מהלך זה חוסך את הצורך לשליפה מלאה של המסד הנתונים כל פעם (שזה עולה לנו כ 70MB כיום ע"ג שרת הטסטינג ו 700MB בשרת הסופי ושניהם נחסכים לרמת העד 100KB)

6. רכיב ה SQL הלוקאלי – בעצם מייצג את המסד הנתונים שנשמר על הפלאפון באמצעות sqlite, מימשנו מחלקה אבסטרקטית שכל מסד נתונים לוקאלי שלנו ירוש ממנה, המחלקה האבסטרקטית יודעת להגדיר לנו טבלה בצורה מאוד מהירה (עד כדי מספר דקות מועט של תכנות כדי להוסיף עוד טבלה), בנוסף לכך כל רשומה שנכנסת אנחנו מצפינים לפני שמכניסים מידע ומפענחים כשאנחנו שולפים מידע, בעצם כל המחלקות שיורשות ממחלקה זו לא מודעות להצפנה ולרמת המימוש, מבחינתם הנתונים נשמרים כ key-value ונטענים כ key-value (בעצם ה key זה שם השדה וה value זה ערך השדה, אנחנו מחזיקים ווקטור של hashmap כזה).

המערכת שלנו בעצם תחילה תבקש סיסמה כדי להתחבר, את הסיסמה אנחנו נאחסן ונשמור ב remote SQL helper, כך שהמחלקה לבד תדע לבצע חיבור מחדש במידה והחיבור כשל, לאחר מכן אנחנו קוראים ל offline service כדי לבצע סינכרון וקריאה של כל המסד נתונים לווקטורים תואמים, כאשר אנחנו מסיימים, אנחנו נותנים למשתמש יכולת לבחור פרויקט מהווקטורים שנטענו, כאשר נבחר פרויקט – אנחנו מאחסנים את המספר הייחודי של הפרויקט בצורה גלובלית, ומבצעים סינונים ע"ג הווקטורים בכל דף בנפרד ומציגים את המידע מהווקטורים האחרים – כל שינוי שנרצה לעשות, אנחנו תחילה נשנה בווקטורים, לאחר מכן נאחסן במסד נתונים המיועד לעדכונים, ואז נשמור במסד נתונים אצלנו, וכל כמה זמן אנחנו ננסה להקרין את השינויים מהמסד הנתונים שנועד לשינויים אל תוך השרת, כל שינוי שהוצלח יוסר ברמה שלנו, הדרך שלנו להקל על תהליך החיבור זה לזכור את הסיסמה שהמשתמש הקליד, תהליך זה יכול להוות פרצת אבטחה – אבל בגלל כמות העובדים המועטת אצל הלקוח שלנו ניתן לסמוך עליו

שבמידה ופלאפון יאבד הלקוח ידאג לשנות ולעדכן את הסיסמה של המשתמש ברמת השרת – ובכך לא יכול דרך האפליקציה לראות את הנתונים שנשמרו על המכשיר, וגם לא יכול לפענח אותם או את הסיסמה כי כולם מוצפנים.

מימוש נוסף אצלנו זה עצם העובדה שרוב הנתונים ושמות השדות של ההתנהגות של האפליקציה שלנו נמצאים בקבצי XML, כלומר בזמן הריצה אנחנו שולפים את הנתונים האלו, אם נרצה לשנות את התנהגות האפליקציה ברמת החיבור, אחסנת נתונים או שמות שדות במסד נתונים, שמות מסדי נתונים וכו' - זה יהיה תהליך פשוט ממש כמו בתכנות בסביבת אינטרנט, וחוסך את הצורך בידע בתכנות בכלל.

6. 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 ה jvm (או במקרה של אנדרואיד, על ידי ה ART) ומורצים על גבי המערכת בשפת מכונה שמתורגמת על ידי האימולטור של Java (ה 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 not 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, הקומפיילר והסביבת עבודה יתריעו לנו על המיותרות שלה ואף הקומפיילר 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 adapterים)
ולכן גם יותר חזקה, אנו מסתמכים על הבדיקות הנ"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 מאוביקט 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 שיודע לאכול שאילתות 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 insance 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 ל bruteforce ברמה הגבוהה דיו כדי לבצע בדיקות על מסד הנתונים, אנו נזניח 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 ב Application Settings , לדעתנו כרגע רק פרטי ההתחברות לשרת ה 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

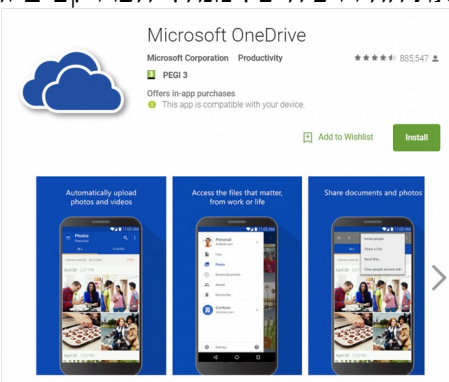
8 כדי להוכיח את אמינות והצלחת האפליקציה שלנו – נבדוק אותה בעומס אמיתי בשרת הלקוח כשיגיע זמן השחרור, זה בעצם בדיקת העומס שלנו, כיוון שללקוח כמות נתונים מאוד גדולה יהיה זה גם בדיקה על יעילות האפליקציה שלנו מבחינת אחסנת ו parse על נתונים וגם על תוכן המידע

7. 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

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, אייפון, ווינדוס[כולל הגרסאת 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

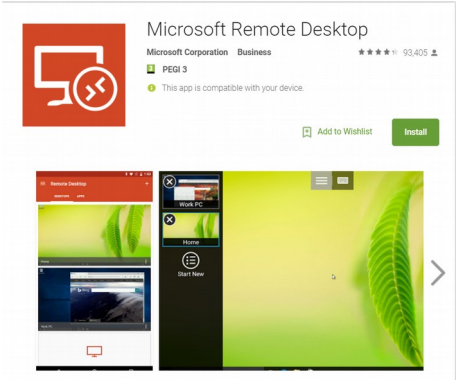
תכונות כלליות:

- 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 התיקייה 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



האפליקציה נותנת שירות Remote Desktop - כלומר השתלטות מרחוק, השירות ניתן למכשירי אנדרואיד, iOS, ווינדוס)כולל הגרסאת מובייל) ולינקס ו mac ותמיכה בו תלויה במיקרוסופט, כיוון שהיא פיתחה את הפרוטוקול המשומש, 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"ירוץ 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

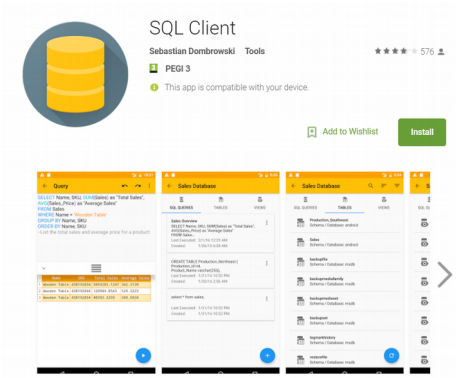
חסרונות:

- 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 האפליקציה הבאה SQL Client שמבצעת



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

- עבודה עם שירות 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 צרכיו מבחינת נוחות ופונקציונליות אלא גם נספק את צרכיו מבחינת אבטחת תוכנה ונתמוך בדרישה של הלקוח לעבודה במצב offline לחלוטין

8. מסקנות מהמימוש והפרויקטים

תהליך הכתיבה של החיבור המרוחק

בתחילת דרכנו לא ידענו מה התהליך של חיבור האפליקציה למסד נתונים דורש, או איך הוא מתבצע, היינו צריכים לראות את זה בעיניים שלנו – לכן היינו צריכים "שרת קטן" שייצג את החיבור הדרוש, בחרנו להשתמש ב virtual machine שיהווה שרת לוקאלי ובחרנו את ה ip להיות ה ip הלוקאלי, ובכך התחלנו לממש אבסטרקטיות של החיבור – בעצם ברגע זה, התקשורת שלנו עם המסד נתונים שנמצא בשרת הלוקאלי היה זהה לתקשורת מול מסד נתונים שהוגדר ע"י הלקוח עד כדי רמת השם מסד נתונים, הכתובת IP שלו והפורט שהוא בחר – לכן בחרנו להכניס את הקבצים האלה לקבצי XML שאנחנו שולפים בזמן ריצה באפליקציה – ובכך הקלנו על עצמנו, במקום לזכור באיזה קטע קוד אנחנו אחסנו את הכתובת IP לחיבור ועדכונה כל פעם, אנחנו עבדנו מול קבצי ה XML כל פעם, לאחר מכן רצינו לוח הגדרות להאפליקציה שבזמן ריצה נוכל לשנות את הרכיבים האלו – וגילינו שהמערכת עובדת צמוד עם הקבצי XML, לכן מעבר לנוחות שלנו קיבלנו נוחות גבוהה יותר במימוש לוח ההגדרות - כל ההחלטות האלו זירזו את כתיבת המנוע שלנו, ובחודש הראשון היה לנו מנוע שיודע לשלוף נתונים – מה שנשאר היה זה היכולת לאחסן נתונים.

תהליך הכתיבה של המסד הנתונים הלוקאלי

תחילה כתבנו קובץ dummy באמצעות sqlite וראינו שיש מחלקה ייסודית באנדרואיד שנותנת לנו מימוש חלקי של מה שרצינו (אומנם מוגבל), אותו רצינו לשפר קצת – רצינו דרך נוחה לגשת לנתונים כאילו הם לא היו מקובץ, אלא בצורה של key value, והגענו למסקנה שווקטור של hashmap יתן לנו את מה שתכננו – יש לנו ווקטור שכל אוביקט בו מייצג שורה במסד הנתונים וכל העמודות מתחלקות ב hashmap שהשם עמודה נכנס ב key והמידע על העמודה נכנס ב value, רצינו לראות איך המידע נשמר – וגילינו שכל "אפליקציה" באנדרואיד מקבל user ברמת הלינקס, ופותחת תיקיה באזור מסוים שרק ל user הזה יש גישה אליו, ובכך אפליקציות שונות לא יכולות לגעת בנתונים של האפליקציה שלי אלא אם כן הם מוגדרות בתור system app כלומר יש להם גישת root access במערכת הלינקס, בשביל להתגבר על כך ובכל זאת לראות את המידע, פרצנו את המכשיר וקיבלנו root access (שזוהי פעולה די פשוטה) וראינו את הנתונים, והבנו כמה זה קל לראות ולחשוף אותם, לכן כתבנו ומימשנו את המחלקת ההצפנה שלנו, ובדקנו שוב, ואחרי שאחסנו את המידע המוצפן – המידע לא היה חשוף יותר! הטמענו את המערכת הצפנה לבסיס שכתבנו לכתיבת הקבצי sql בתוך אנדרואיד ומנקודה זו הלאה ידענו שאנחנו מאחסנים נתונים בצורה מאובטחת, מיילסטון גבוהה עבורנו, לאחר שסיימנו עם זה, חיברנו את המערכת אחסון שלנו עם המערכת שליפה שכתבנו מקודם – והרי יש לנו פתאום מערכת שיודעת לשלוף את כל הנתונים מהשרת (מספר הנתונים היה מועט כיוון שהשרת שלנו היה עדיין virtual machine לוקאלי והתעצלנו לכתוב המון נתונים) ולאחסן אותם לוקאלית ובכך נחסך השליפה הנוספת כל פעם.

כתיבת המערכת offline

לאחר שכתבנו את המערכת סינכרון קבצים, רצינו שהיא לא תיהיה תלויה יותר בחיבור לאינטרנט, בעצם מנקודה שאנחנו מאחסנים את המידע ע"ג המכשיר פלאפון, לא היינו צריכים את האינטרנט כדי לשלוף את המידע אלא רק כדי זיהוי של המשתמש שנוכל לחשוף לו את המידע, אבל העדכון היה יותר טריקי, בשביל לעדכן בזמן אמת היינו צריכים אינטרנט, אחרת השאילתה הייתה כושלת ולא היינו מודעים לכך – לכן הרכבנו מסד נתונים שמחזיק את השינויים שרצינו לבצע ע"ג השרת, ודאגנו לשלוח את השינויים האלו לעיתים תכופות, כאשר כל פעם ששינוי מתקבל ע"ג השרת הוא מחזיר לנו תשובה חיובית ואנחנו נמחק את השינוי – הבעיה שנחשפנו זה רמת העדכון של המסד נתונים, כן אנחנו מבצעים "הורדה נוספת" של הנתונים מול השרת לפני שאנחנו עושים את השאילתה של השינוי – ובגלל שהמסד נתונים שלנו בצד השרת היה עודנו קטן, התהליך היה מאוד מאוד מהיר, ובכך בנינו מערכת שגם מורידה את המסד נתונים וגם מעדכנת את הנתונים שנמצא ע"ג השרת אצלנו ע"ג המכשיר על זמן מוגדר, זה היה המוצר אלפא שלנו, הודענו ללקוח שלנו על המימוש שלנו ושאלנו אם צריכים לבצע מנגנון גרסאות לכל רשומה כדי למנוע דריסה, הלקוח חשב שעדיף נתקדם הלאה במקום זאת כיוון שאין אצלו תרחיש אפשרי שהדריסה תתקיים כיוון שהאחריות של כל רשומה תיהיה מחולקת למשתמשים נפרדים

אינטגרציה ראשונית

בעצם לאחר המוצר אלפא היינו בטוחים שהאפליקציה שלנו ראויה לאינטגרציה בשרת הלקוח ולא להיות תמיד ע"ג virtual machine, תכילה פתחנו שרת אצל amazon ששקול לשרת שלנו וניסינו להתחבר אליו במקום לשרת המקומי – רצינו לעשות זאת כדי לקבל תחושה יותר "אמיתית" כיוון שהמשתמשים שלנו יתחברו אל שרת מרוחק ולא שרת virtual machine מקומי על המכשיר שלהם, ולכן גם מהירות החיבור תיהיה אחרת ורצינו לראות עד כמה מהיר החיבור הנ"ל – ומעט התאכזבנו, הדרייבר שמיקרוסופט סיפקה לנו כדי להתחבר למסד הנתונים המרוחק היה מאוד מגושם ולא ידע לנהל חיבור מרוחק בצורה מאוד יעילה, התחלנו לחפש תחליפים ומצאנו דרייבר אחר של jtds, בגלל שכל החיבור המרוחק היה תחת מחלקה אחת האינטגרציה של הדרייבר החדש היה מאוד מהירה והשיפור היה מדהים – החיבור לקח מספר בודד של שניות אם בכלל לקח שניות, וכל השאילתות והעדכונים נשלחו כמעט מיידית אפילו שהחיבור היה מרוחק לשרת בארה"ב, אז לשרת שנמצא בישראל בכלל...., מה שהביא לנו בטחון רב שהאפליקציה שלנו מאוד בשלה לכך, ביקשנו מהלקוח את הפרטי חיבור של השרת והופתענו לדעת שהשרת מאוחסן בתוך VPN, כלומר כדי להתחבר אליו אני צריך לראות את דרכי ההתחברות שלי ל VPN.

בעית ה VPN

אנדרואיד מספקת ממשק התחברות כלשהו אבל הוא לא מתאים לשרת VPN של הלקוח, לאחר המון מאמצים וחיפוש פתרונות הגענו למסקנה שזמן הפיתוח לשירות VPN שווה ערך לפרויקט שלהם וחייב בכלל להתבצע ב C++, לכן הודענו ללקוח והוא הסכים לקבל על עצמו שכל משתמש ירץ שני אפליקציות כדי לעבוד – אפליקציית חיבור VPN שסופקה ע"ג מביא השרות של הלקוח, והאפליקציה שלנו, האלטנרנטיבה שלנו היה להעתיק את הספריה שסופקה באפליקציה של המביא שירות של ה VPN, אבל זה היה סיכון שפרסום האפליקציה בחנות של גוגל יכול להיות צעד לא חוקי כי זה לא קוד שלנו והספריה לא חופשית לפי הרשיון שלה, לאחר שהתחברנו ל VPN עם ההאפליקציה של מביא השרות, הגענו לבעיה אחרת

זמני סינכרון נוראיים

זמן הסינכרון היה כ 40 דקות הורדה מהשרת של הלקוח (המסד נתונים שלהם היה גדול יותר ממה שציפינו), וכאשר הקבצים כאלה גדולים גם הטעינה שלהם לזכרון לוקחת המון המון זמן, התחלנו "לקצץ את השומן", ועדיין הסינכרון לקח חצי שעה, חשבנו רבות על פתרון טוב והגענו לפתרון מאוד נחמד – על כל שורה במסד נתונים ניתן עמודה של "מתי עודכן לאחרונה", ונשמור אצלנו באפליקציה "זמן עדכון אחרון", ובעצם ניתן לשירות SQL לבצע סינון עבורנו אם אנחנו צריכים את המידע שישלח דרך האינטרנט או לא – נבצע שאילתה שמסתיימת ב where local_update_time < remote_update_time כלומר נשלוף את הנתונים רק אם הזמן עדכון שלנו הוא קטן מהזמן עדכון במסד הנתונים – וכאשר נסיים את העדכונים רק אז נעדכן את הזמן עדכון המקומי להיות התאריך של **עכשיו**, הבעיה היא ההגדרה של זמן העדכון, אם נצמד לתאריכים של השרת, במידה והשרת יצא מסינכרון הזמן זה יפגע רבות באפליקציה שלנו, לכן אנחנו דואגים לעדכן גם את השדה הזה אצלנו לפי התאריך המקומי (שתמיד יהיה ישראלי) וגם לשלוף אותו כך, ניתן יותר לסמוך על שירות הזמן של הפלאפון מאשר של השרת כיוון שהמפעילה הסלולארית דואגת לעדכן את התאריך ע"ג הפלאפון ולרוב המפעילה מהירה ו"צודקת", לעומת רכיבי זמן שיכולים להכשל ע"ג מחשב השרת שיכולים "לחרבש הכל" אצלנו, אחרי השינוי הנ"ל – זמן העדכון הראשוני לא השתנה מדי, היה כ 20 דקות, אבל כל זמני הסינכרון אחריו היו מאוד מהירים (עד כמספר שניות), הבעיה היתה זמן הסינכרון הראשוני וזמני הטעינה, אותם פתרנו באמצעות מקבול תהליכים, הצלחנו כך להוריד את זמני הסינכרון הראשוניים לעד כ 8 דקות, ואת זמני הטעינה להיות כ 2 דקות – שיפור מאוד משמעותי, בדקנו אופציה של לא להשתמש בטעינה מהזכרון כדי שהשתמש יכול אוטומטית לעבוד עם האפליקציה, גילינו שהטעינה מהקבצים יותר כבדה, ועדיף לתת למשתמש לחכות – בכל זאת קבצים כבדים לוקחים זמן לטעון, והיו כמה טבלאות של הלקוח שהיו מעבר ל 700 אלף שדות – שזה מעבר חד לכיוון ה big data, אבל הטעינה לזכרון חסכה לנו את זה

צרות של הסוף?

בעצם לאחר הצרות של הסינכרון האיטי, האפליקציה היתה מאוד בשלה ומוכנה לשימוש – הבעיה עכשיו זה תאימות, זה להיות פחות אבסטרקטים ויותר מכוונים לצרכים של הלקוח (טבלאות של הלקוח וכו'), לכן מימשנו תחילה אבסטרקיות לכך – מימשנו אבסטרקיות של אוביקט שיוצא ממסד נתונים (מה אנחנו מצפים ממנו? שיהיה לו toString? שיהיה לו מעבר מהיר ל hashmap כדי שנוכל לאחסן במסד נתונים אצלנו? וכו'), מימשנו אבסטרקיות של מה זה אומר חיבור מרוחק לטבלה כלשהית (מה היא צריכה לדעת לעשות? לעדכן נתונים? לדעת מה הנתונים?) והאבסטרקיות של החיבור המקומי היתה מוכנה לנו – לכן התלבשנו עליה... לאחר מכן ביצענו ירושה לכל הרכיבים האבסטרקטיים והיינו הפעם יותר ממוקדים לאוביקטים שמייצגים את צרכי הלקוח (לצורך העניין טבלת פרויקטים שמכילה מספר פרויקט, שם פרויקט, טבלאות מידע על הפרויקטים שמכילות נתונים על הפרויקט ומצב עליו וכו') וכל טבלה של הלקוח קיבלה מופע מחלקתי, מופע של מחלקת עזר לעדכון במסד הנתונים בצורה מרוחקת ומופע של טבלה לוקאלית, ומימשנו בצורה קצת מלוכלת עבור כל מימוש לוקאלי את היכולת להוריד מהשרת, ולהוסיף לוקאלית כל שינוי שהשתנה, כמובן שההורדה מהשרת עכשיו יותר קלה מקודם, בעצם לאחר העבודה של המיקוד האפליקציה עבדה פלאים יותר מהר וסיפקה את השירות שהלקוח ביקש ישירות, מעבר לכך בגלל האבסטרקיות שם, כל שינוי נוסף שהלקוח "הפתיע" אותנו כמו טבלה נוספת או שינוי של

שדמות השדות התקבל בברכה – הרי הכל בתוך קובץ XML שהגדרנו מראש, ומכאן האפליקציה היתה בשלה ל release – ונתנו ללקוח לטעום – הלקוח לא התאכזב!

ועכשיו?

עכשיו נשאר לנו ללטש את ה UI – איזה נתונים להציג באיזה activity ומתי? הדרך שלנו ללטש את זה זה לשלוח ללקוח ולקבל ממנו פידבקים ("תגדילו את הכפתור", תוסיפו את המידע של X משדה Y") בחלק מהנתונים הלקוח דרש מיון – המימוש היה פשוט, הרי הכל בווקטורים, והכל מומש במופעים של מחלקות – והווקטור ימיין לפי השיטת compareable של המופע של המחלקה – לכן מימשנו את ה interface של compareable בכל המופעים שדרשו מיון כלשהו, אנחנו עדיין של שרת הטסטינג של הלקוח, ורוצים ללטש הכל לפני שנעבור לשרת ה"מרכזי" של הלקוח, אבל אנחנו בטוחים שהאפליקציה מלוטשת דיו אחרי כל התהליך הנ"ל שהחיבור לשרת המרכזי לא יהווה שינוי רב מהשרת טסטינג והכל יעבוד עד רמת השמות השדות ושמות הטבלאות.

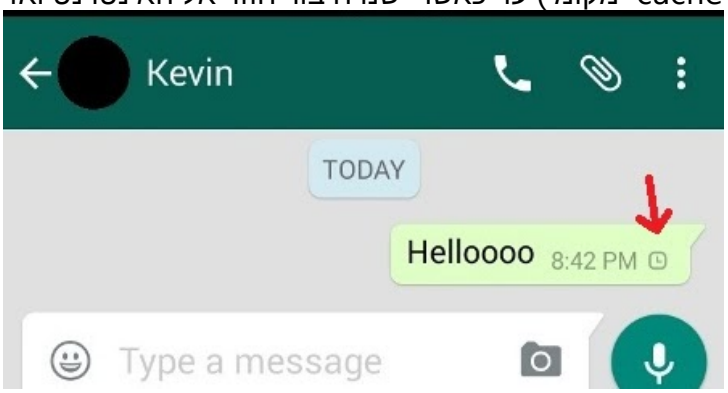
9. ריכוז פרויקטים דומים

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

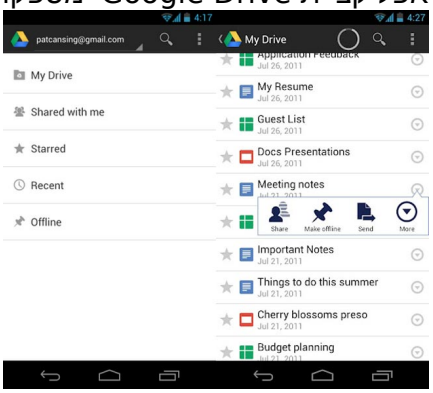
והודעות שלא הצליחו להתקבל בשרת מקבלות סימן של שעון, ונשמרות באפליקציה (בזכרון cache מקומי) עד כאשר ישנו חיבור חוזר אל האינטרנט ואז נשלחות



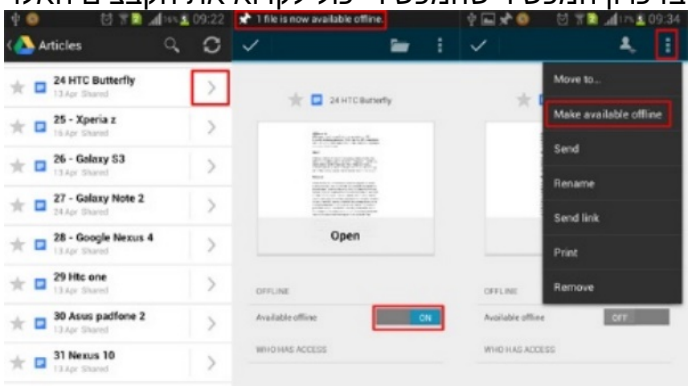
באותה רמה – אנו רוצים לממש מנגנון זהה אבל לשרתי Microsoft SQL Server 2012, שימומש בצורה דומה לצורה שבה הוא ממומש באפליקציית WhatsApp

Google Drive

אפליקציית Google Drive מספקת שירותי ענן לאחסון קבצים



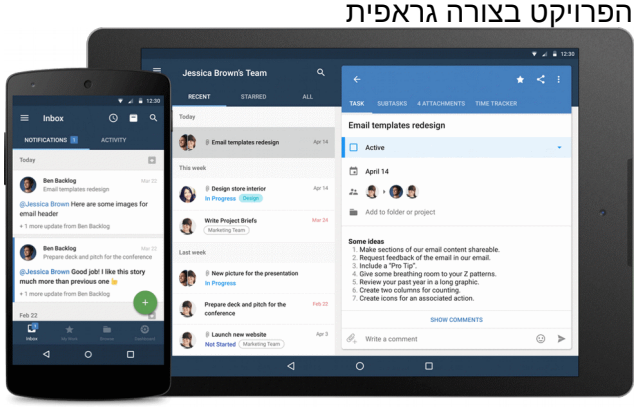
מעבר לסיפוק שרות זה, Google Drive נותנת למשתמש יכולת לשמור קבצים מסוימים בזיכרון המכשיר שהמכשיר יכול לקרוא את הקבצים האלו ללא כל גישה אל האינטרנט



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 לו"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.

10. 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=gbs_atb

.2

Database Programming with JDBC and Java by George Reese, 2000

https://books.google.co.il/books?id=oPbGi0l0ZHEC&sitesec=buy&hl=iw&source=gbs_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=gbs_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 ממש הפיתוח של מנגנון 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!

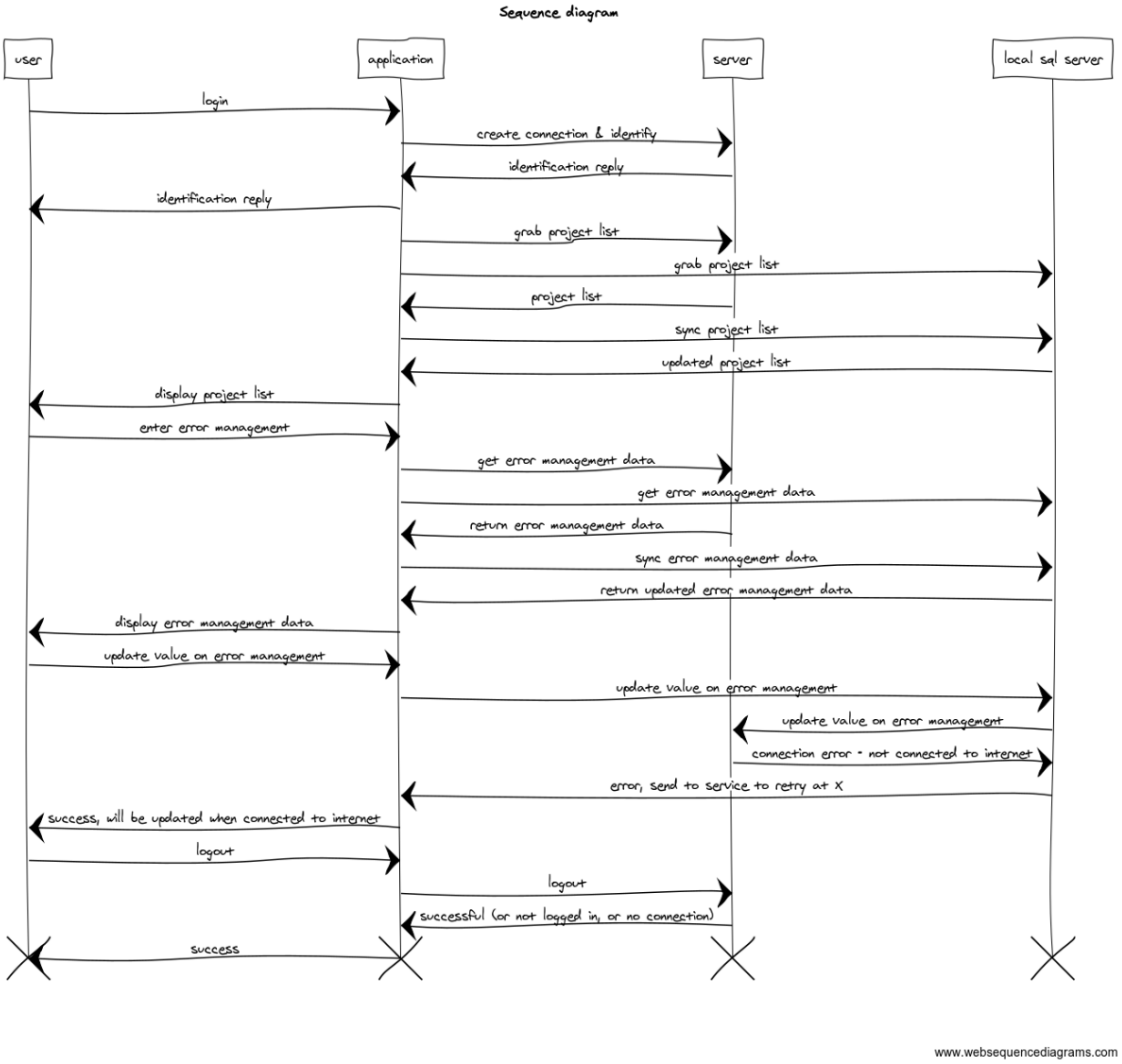
דיאגרמת מצבים – מצב offline פתע

על מנת שנוכל להסביר את חשיבות המימוש של מצב ה offline

אנו מתארים

בנספח זה את מקרה בודד מבין המצבים האפשריים כאשר שירות האינטרנט

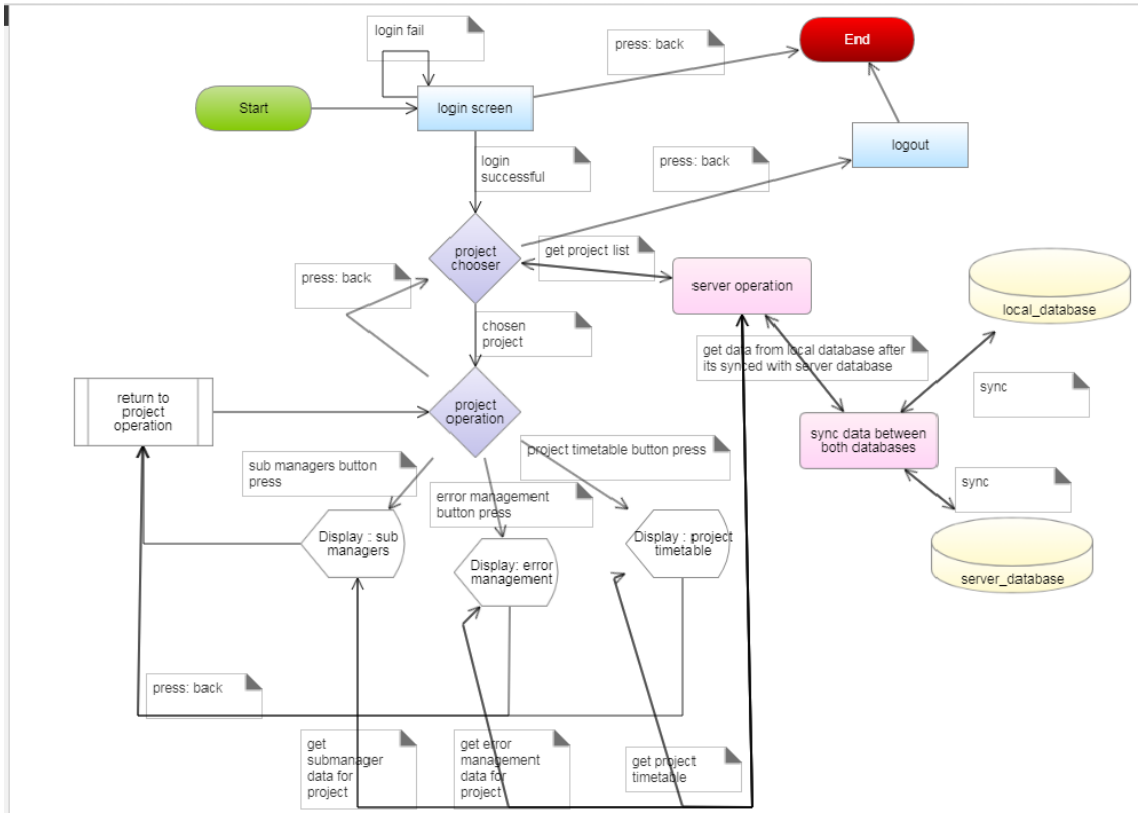
או החיבור אל השרת נופלים



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

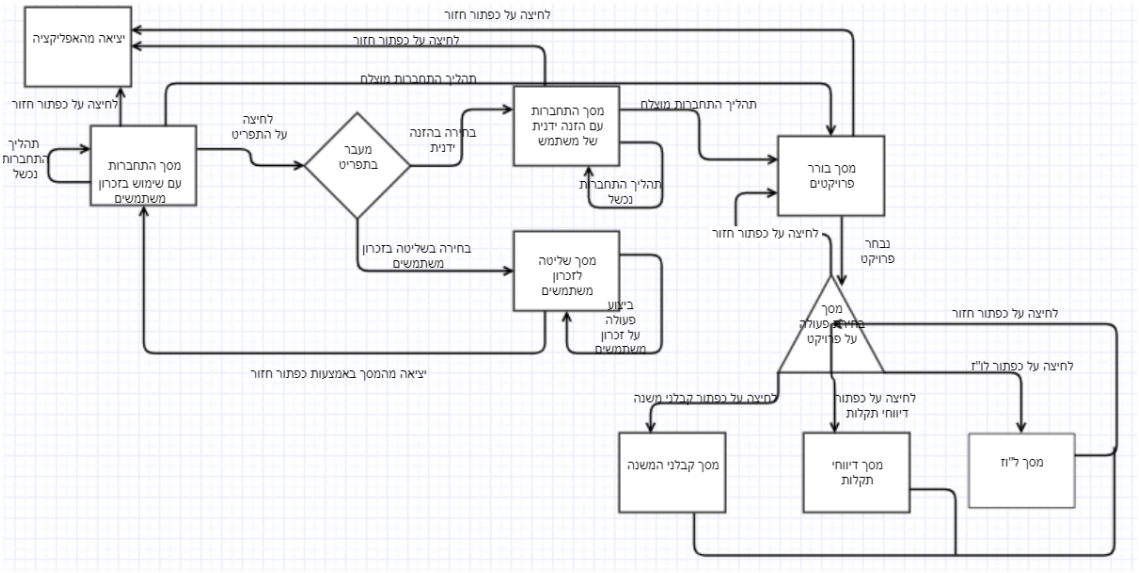
בתרשים זה אנו מציגים את מערכת המצבים משלב ההתחברות עד תצוגת המידע במסכים ואפילו עד היציאה מהאפליקציה והסגירה שלה



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 הרלונטים

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 שליפת 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

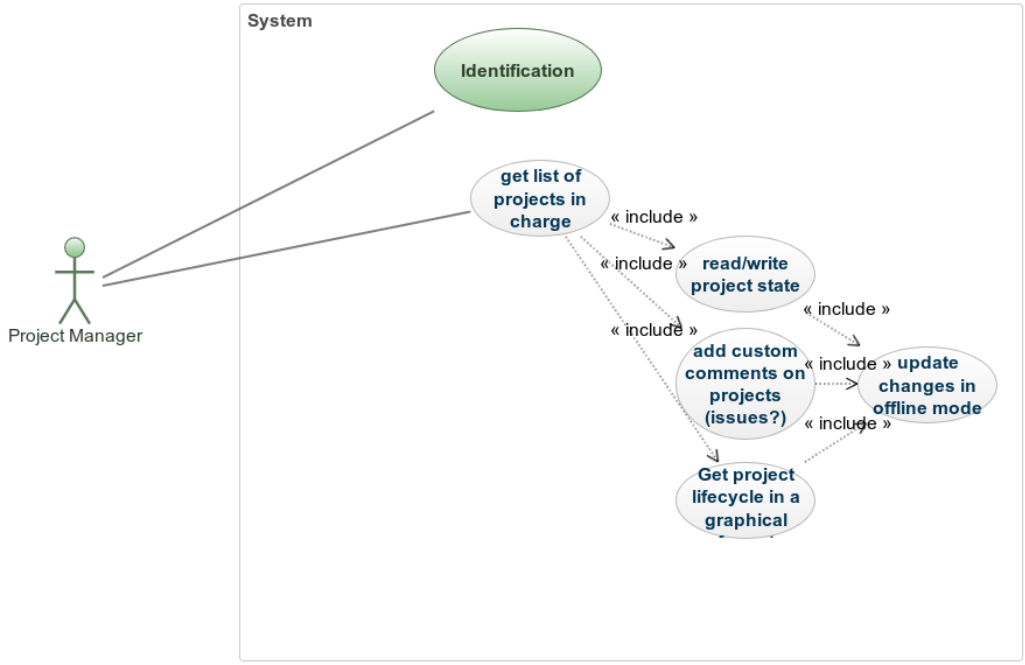
התרשים הבא מייצג את המנהל עבודה ואת השירותים שהוא יכול לקבל

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

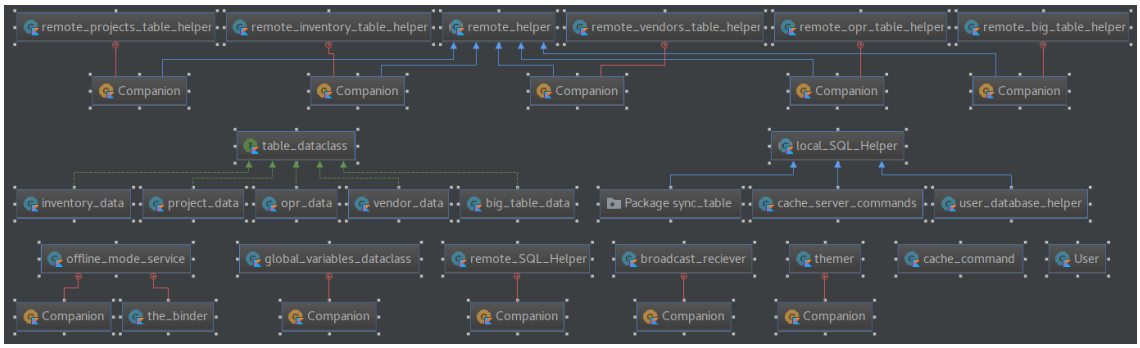


UML

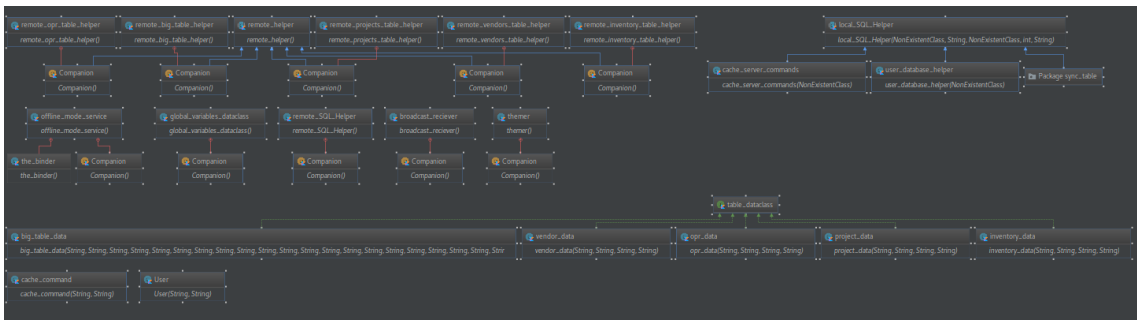
התרשים הבה מייצג את יחסי המחלקות שידועות לנו עכשיו בצד הלוגי, כמובן שקיימים מחלקות שמטרתם היא יותר גראפית (כמו ה Activity של Android או ה OpenSQLHelper של Java\Android או ה JDBC), אנחנו לא נתייחס אליהם אפילו שחלקם מתייחסים לחלק הלוגי של האפליקציה, אלא נחשוב כי קיים Interface בשם SQLite שמטפל

בחלק המקומי של המסד הנתונים ו Interface בשם Microsoft SQL driver שיטפל בחלק המקוון של המסד הנתונים.

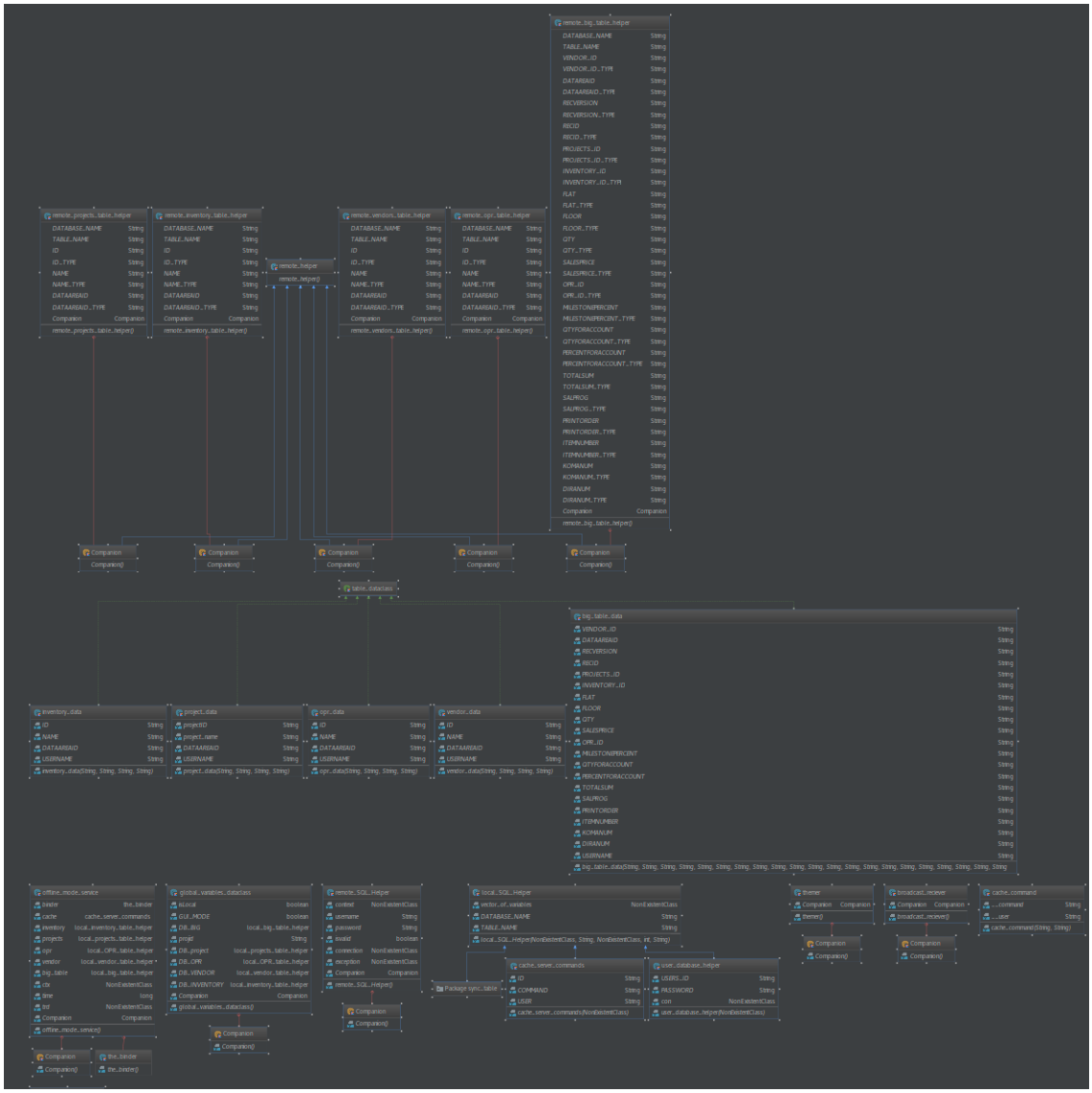
UML ראשון – מבט על ללא ירידה לפרטים במחלקות הראשיות בלבד



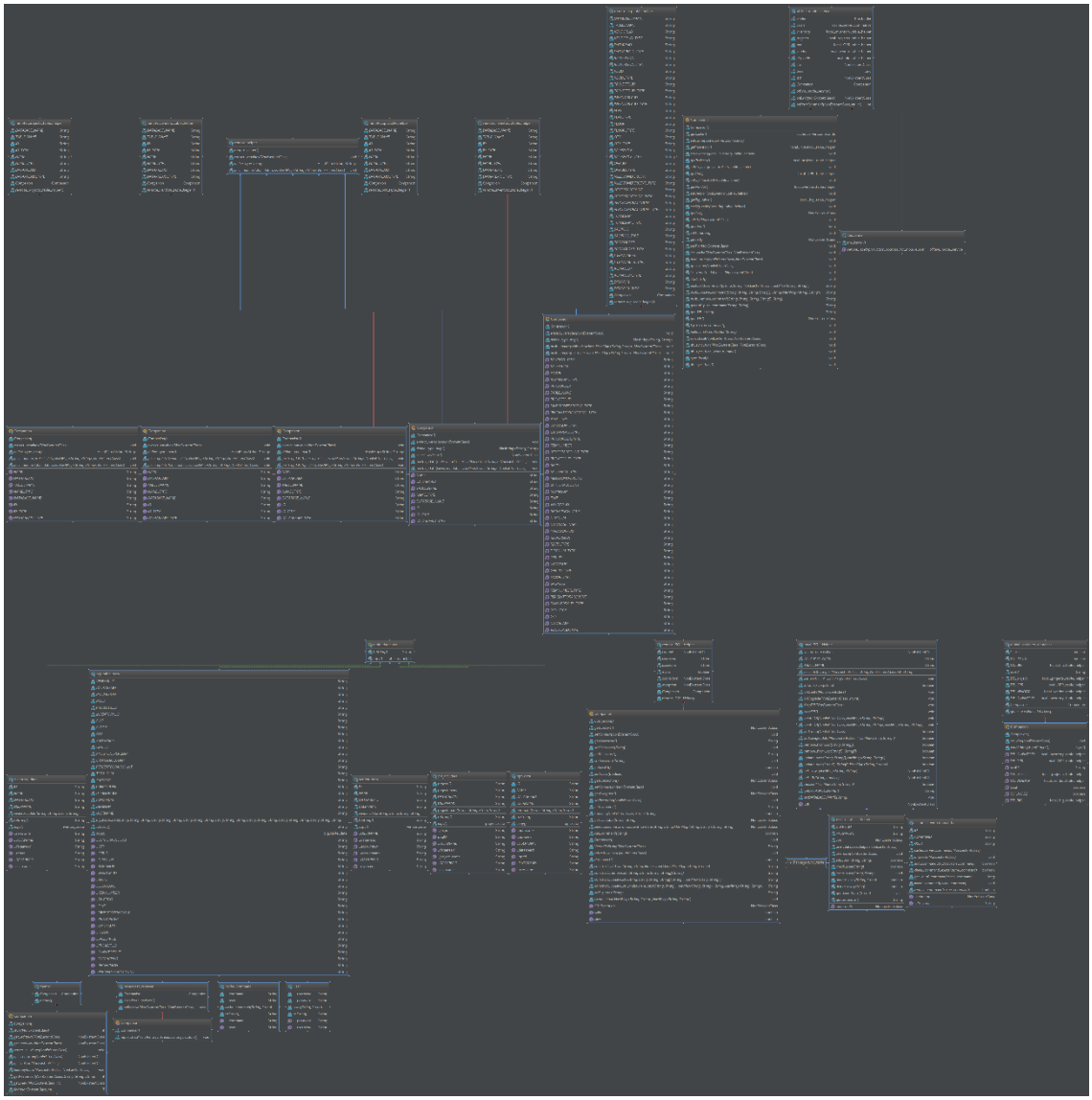
UML שני – ה UML למעלה עם סוגי הבנאים (לשם הבנת מורכבות בלבד)



UML שלישי – כמו ה UML הקודם, בתוספת ה class Members של כל מחלקה



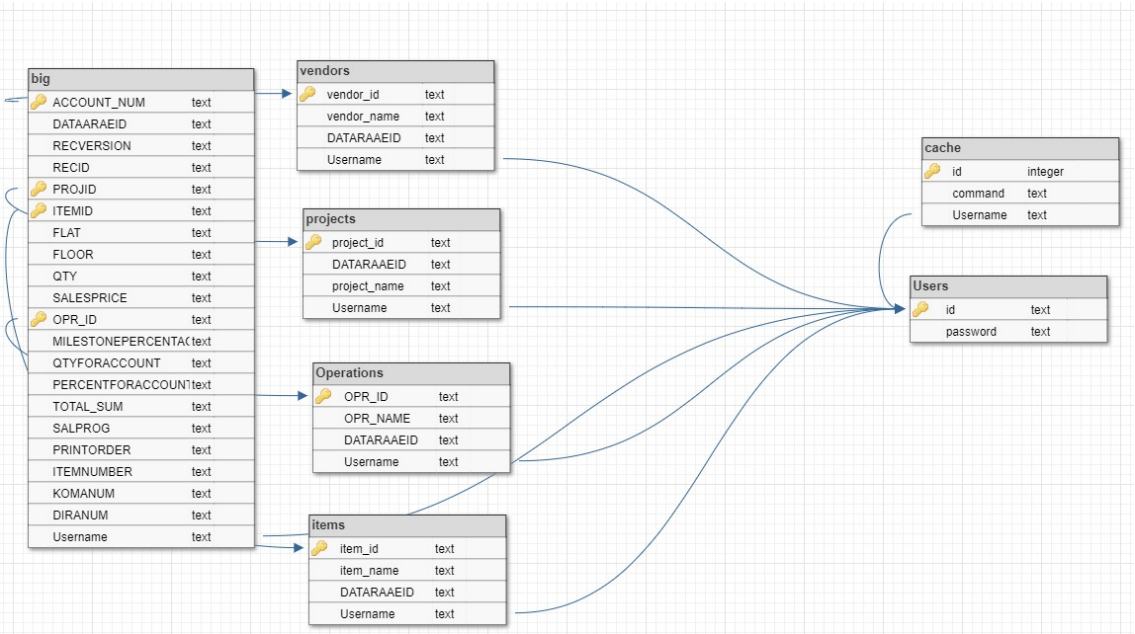
UML רביעי – כמו ה UML הקודם בתוספת פונקציות (עדיין רק מחלקות ראשיות)



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 ל offline 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

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

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 הספריה 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 וננסה ככל שניתן ביכולתנו לפתור את הבעיה טרום פיתוח, אחרת פשוט נודיע למשתמש בצורה גראפית על הסיכון
5	ישנם שירותים במכונה הווירטואלית שיצרנו שהתבססנו 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 client VPN מינימליסטי באפליקציה שלנו	1	ללקוח יש פתרון VPN נפרד, ואפילו עדיף שיתוחזק 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
2	פונקציונאלי	המוצר יכול לתקשר עם שרת הלקוח כדי לשאוב נתונים ולכתוב נתונים על הפרויקט הנתון – תוכנה
3	פונקציונאלי	המוצר ידע לעבוד במצב offline, כאשר לא קיים חיבור אל האינטרנט פעולות על השרת ישמרו בזכרון המכשיר וישלחו בהזדמנות – תוכנה
4	תמיכה לעתיד, הבנת הקוד	המערכת תהיה מודולרית על מנת שתוספים ופיצ'רים חדשים שהלקוח ירצה להוסיף בהמשך חיי הפרויקט לא יהוו מכשול לפרויקט – תוכנה
5	תמיכה נוכחית, ממשקיות	המוצר חייב לתמוך במערכת הפעלה אנדרואיד בגרסאות המקובלות לשימוש כיום (6 ומעלה) – חומרה\תוכנה
6	אבטחה	במהלך חיי הפרויקט רמת האבטחת מידע של המוצר חייבת לרצות את הלקוח כגוף ארגוני – אבטחה\תוכנה
7	ממשק	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
8	יציבות	כתיבת טסטים מרובה תאפשר יציבות מאוד חזקה ותמנע את הצורך לתחזק את האפליקציה - תוכנה

טבלת סיפורי משתמשים (User Case Stories)

1	בתור מנהל עבודה, אני יכול לשלוח נתונים לעדכן את השרת בנתונים חדשים מבלי להתייחס למגבלות המכשיר לחיבור לאינטרנט
2	בתור חבר בצוות ניהול החברה, אני יכול לראות את רשימת כל הפרויקטים בלחיצת כפתור פשוטה מבלי להכנס לדפדפן ולשירותים טכנולוגיים כמו rdm ו vpn כדי לראות את מצבי הפרויקטים על מסך הפלאפון שלי
3	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
4	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
5	בתור מנהל עבודה, יש לי את היכולת לראות את הנתונים האחרונים שראיתי על הפרויקט בצד השרת ללא צורך בלשמור אותם בצד בצורה עצמאית, וללא תלות בחיבור שלי אל הרשת
6	בתור מנהל עבודה יהיה לי בטחון שאם אאבד את המכשיר פלאפון שלי וגוף שלישי מחזיק בו – הוא לא יוכל לפגוע בלקוחות שלי בקלות