

תאריך: _____

לכבוד
יחידת הפרויקטים
מה"ט

הצעה לפרויקט גמר

* יש להדפיס את כל הנתונים הנדרשים

א. פרטי הסטודנטים

שם הסטודנט	ת.ז. 9 ספרות	כתובת	טלפון נייד	תאריך סיום הלימודים
נריה דוד לוי	207786047	רימון 121 בית שאן	0522859094	

שם המכללה _____ מכללה טכנולוגית כנרת סמל המכללה: _____72228

מסלול ההכשרה: הנדסאים.

מגמת לימוד: תוכנה _____ מקום ביצוע הפרויקט: מכללה טכנולוגית כנרת

פרטי המנחה האישי

שם המנחה *	כתובת	טלפון נייד	תואר	מקום עבודה/תפקיד

* עבור מנחה אישי חדש יש לצרף קורות חיים, ניסיון מקצועי ותעודות השכלה לאישור מה"ט.

_____ חתימת הגורם המקצועי מטעם מה"ט

_____ חתימת המנחה האישי

_____ חתימת הסטודנט

[Type here]

1. שם הפרויקט

BuildLocker

2. רקע

2.1. תיאור ורקע כללי

- בבנייני מגורים קיימת לעיתים קרובות בעיה של חוסר זמינות לכלי עבודה וציוד תחזוקה, עקב עלות גבוהה ושימוש מזדמן בלבד. כיום פתרונות קיימים כמו השאלה בין שכנים או השכרה מחנויות אינם נוחים, דורשים תיאום מראש ואינם מבוקרים
- הפרויקט מציע מערכת לוקרים חכמה בבניין, המאפשרת לדיירים להשאל ולהחזיר ציוד באופן עצמאי באמצעות אפליקציה/אתר, תוך ניהול מלא של זמני שימוש ותשלומים.

2.2. מטרות המערכת

- לאפשר לדיירים גישה נוחה ומהירה לציוד כלי עבודה מתוך הבניין.
- לייעל את תהליך ההשכרה באמצעות מערכת אוטומטית הזמינה 24/7.
- לספק למנהל המערכת כלי ניהול: הוספה/עדכון ציוד, ניהול השכרות ותשלומים.
- לאפשר חיוב לפי זמן שימוש בפועל.
- לשמור על בקרה ואבטחה – פתיחת לוקר רק למשתמש מאומת ורישום פעולות.

3. סקירת מצב קיים בשוק, אילו בעיות קיימות

1. **השכרת כלי עבודה מחנויות** – דורשת נסיעה, זמינות מוגבלת, ותשלום ליום שלם גם עבור שימוש קצר.
2. **השאלות בין שכנים / קבוצות קהילתיות** – חסר אמון, אין בקרה על זמני החזרה, אין אחריות במקרה של נזק.
3. **לוקרים חכמים קיימים (למשל לחבילות)** – זמינים ויעילים אך אינם מותאמים להשאלת כלים (תחזוקה, בלאי, ניטור).
4. **ארגוני ציוד משותפים בבניינים** – ללא מנגנון ניהול, נטייה לאיבוד או בלגן, ללא רישום מי לקח מה.
5. **מרכזי "ספריות כלים" עירוניות** – דורשים הגעה פיזית, שעות פעילות מוגבלות ולא זמינים קרוב לבית.

בעיות עיקריות:

- חוסר נוחות וזמינות
- היעדר בקרה וניהול
- תיאום ידני בין אנשים
- תשלום לא מדויק לפי זמן
- חוסר אחריות במקרה של נזק או אי החזרה

4. מה הפרויקט אמור לחדש או לשפר

- אוטומציה מלאה של תהליך ההשאלה – שימוש בלוקרים חכמים הנפתחים דרך אפליקציה/אתר ללא צורך במפגש פיזי.
- חיוב מדויק לפי זמן שימוש – במקום חיוב יומי/תקופתי, המערכת מחשבת תשלום מהפתיחה ועד ההחזרה בפועל.
- בקרה מלאה על הציווד – רישום מי לקח, מתי, האם הוחזר, ומעקב היסטוריית שימוש.
- שיפור אמינות ובטיחות – פתיחת לוקר רק למשתמשים מאומתים, שמירת לוגים ויכולת ניטור.
- חיסכון כלכלי לדיירים – אין צורך שכל דייר יקנה כלים יקרים לשימוש חד-פעמי.
- נגישות 24/7 בבניין עצמו – אין צורך בנסיעה לחנות או בתיאום עם שכנים.

5. דרישות מערכת ופונקציונאליות

5.1. דרישות מערכת

סביבת הטמעה ושימוש. שרידות, ביצועים\התמודדות עם עומסים.

סביבת הטמעה ושימוש:

- שרת Backend Node.js ו-Database (MySQL/Mongo).
- ממשק Web/אפליקציה לדיירים ולמנהל.
- לוקרים חכמים המחוברים לרשת. (Wi-Fi / LAN)
- שרידות והתמודדות עם עומסים:
- תמיכה במספר משתמשים ברזמנית.
- מנגנון טיפול בתקלות תקשורת מול הלוקר. (Retry)
- שמירה עקבית של נתונים בבסיס הנתונים. (ACID / Transactions)
- גיבוי נתונים תקופתי.

5.2. דרישות פונקציונאליות

רשימת דרישות המשתמש מהמערכת, מהן הפעולות בהן נדרשת המערכת לתמוך

למשתמש רגיל (דייר):

1. הרשמה והתחברות.
2. צפייה ברשימת ציוד זמין.
3. פתיחת לוקר וקבלת כלי.
4. החזרת ציוד וסגירת השכרה.
5. צפייה בהיסטוריית שימוש ותשלומים.

למנהל מערכת / מנהל בניין:

1. ניהול משתמשים – הוספה, עדכון והרשאות.
2. ניהול ציוד – הוספה, מחיקה, עדכון ומעקב מלאי.
3. ניהול לוקרים – סטטוס פתוח/תקין/בטיפול.
4. ניהול השכרות ותשלומים.
5. צפייה בלוגים ופעולות משתמשים.

6. בעיות צפויות במהלך הפיתוח ופתרונות (תפעוליות, טכנולוגיות, עומס ועוד):

6.1. תיאור הבעיות- הללו כפועל יוצא של דרישות המשתמש מהתוכנה.

1. תקשורת בין השרת ללוקרים החכמים
ייתכנו ניתוקים, חוסר יציבות רשת, או עיכוב בפתיחת לוקר.
2. ניהול תקלות של אי החזרת ציוד
משתמש עשוי לא להחזיר ציוד בזמן או להשאיר ציוד פגום.
3. עומסים על המערכת
שימוש של מספר רב של משתמשים בריזמנית, במיוחד בשעות עומס בבניין.
4. אבטחת מידע
גישה לא מורשית ללוקרים, גניבת זהות, או ניסיון לגשת ל-API ללא אימות.
5. סנכרון נתונים בין הצד לקוח לצד שרת
שינוי סטטוס לוקר/ציוד בזמן אמת מחייב עדכון מהיר ואמין.
6. ניהול תשלומים לפי זמן
יש צורך במדידה מדויקת של זמן פתיחה ועד סגירת לוקר.

6.2. פתרונות אפשריים. (נא ציין פתרונות אפשריים וחלופות ארכיטקטוניות)

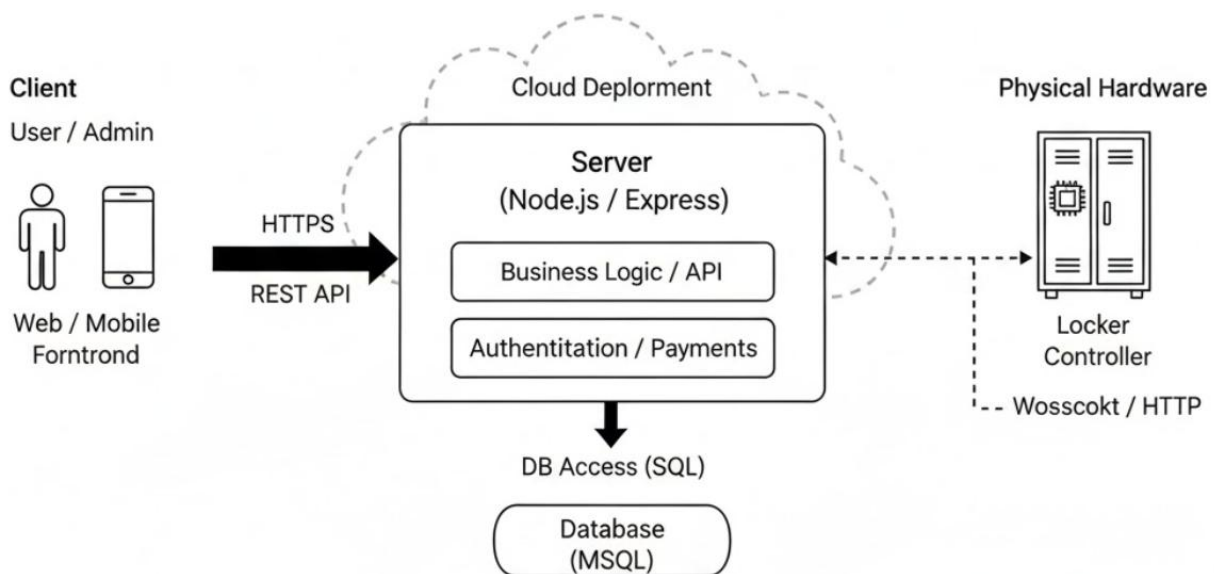
- שימוש ב־ WebSockets או MQTT לתקשורת רציפה ויציבה בין השרת ללוקרים → מאפשר עדכון בזמן אמת וגם fallback ל־HTTP.
- מנגנון Retry + Queue במקרה של כשל תקשורת, הפקודה נשלחת מחדש עד לאישור תקין.
- שימוש ב־ Load Balancing ו־Caching במקרה של עומסים – ניתן להרחיב את השרת בהמשך. (Scaling)
- אימות משתמשים JWT + HTTPS הגנה על תקשורת המשתמשים והלוקרים.
- תיעוד לוגים וניטור (Logging & Monitoring) מאפשר לזהות בעיות בזמן אמת.
- חלופות: שימוש בכלים כמו PM2, Grafana, Kibana מנגנון מדידת זמן בשרת
- השרת שומר זמן פתיחה וזמן סגירה → לא תלוי בצד לקוח. חלופה: שירות צד שלישי למדידת תעריפים (לא חובה).
- ארכיטקטורה מודולארית (MVC / Service-Repository) מקלה בניהול פרויקט, תיקון תקלות והרחבת המערכת בעתיד.

7. פתרון טכנולוגי נבחר:

7.1. טופולוגית הפתרון- כלומר: פריסת המערכת, היכן יתבצע יישום המערכת (deployment), מרכיבי הפריסה. הנ"ל ברמת מערכת (לדוג' פרויקט פיתוח אתר אינטרנט: המערכת מורכבת משרת, ממשק משתמש בצד הלקוח, DB's, טווח תקשורת-אינטרנט, המערכת תיושם ברשת האינטרנט, יש להציג את דיאגרמת המערכת וכו')

המערכת בנויה בתצורת Client-Server:

- **Frontend** – אתר אינטרנט/אפליקציה דרכה המשתמשים מבצעים הזמנות ופוזתחים לוקרים.
- **Backend Server (Node.js)** – אחראי על ניהול המשתמשים, הלוקרים, התשלומים ופעולות לוגיות.
- **Database** – מאגר נתונים (MySQL / MongoDB) לשמירת משתמשים, ציוד, השכרות וזמנים.
- **Locker Controllers** – בקרים פיזיים המחוברים לאינטרנט ומקבלים פקודות פתיחה/סגירה. המערכת נפרסת על שרת אינטרנט (מקומי/ענן) והלוקרים מתקשרים מול השרת דרך HTTP/WebSocket.



7.2. טכנולוגיות בשימוש. (איזה ומדוע בכמה מילים)

- **Node.js + Express** - פיתוח מהיר, גמיש, נפוץ ל-API.
- **React** - ממשק משתמש מודרני, מהיר וקל לתחזוקה.
- **MySQL** - שמירת נתונים בצורה עקבית ומסודרת.
- **WebSocket / MQTT** - תקשורת בזמן אמת עם לוקרים.
- **JWT Authentication** - אימות מאובטח של משתמשים.

7.3. שפות הפיתוח: (איזה שפות ומדוע בכמה מילים?)

- **JavaScript/Node.js** - מאפשרת פיתוח מהיר בצד שרת וקל לשילוב עם צד לקוח.
- **JavaScript/React** - בניית UI דינמי ונוח.
- **SQL** - להגדרת מבנה הנתונים ותקינותם.

7.4. תיאור הארכיטקטורה הנבחרת- הסבר בכמה מילים מדוע

הארכיטקטורה בנויה לפי **Routers + Services + Repository Pattern**:

- **Routers** - מקבל בקשות HTTP מהלקוח.
- **Service** - מכיל את הלוגיקה העסקית (ניהול הזמנות/ציוד).
- **Repository (DB Access)** - אחראי על תקשורת עם בסיס הנתונים.

7.5. חלוקה לתכניות ומודולים

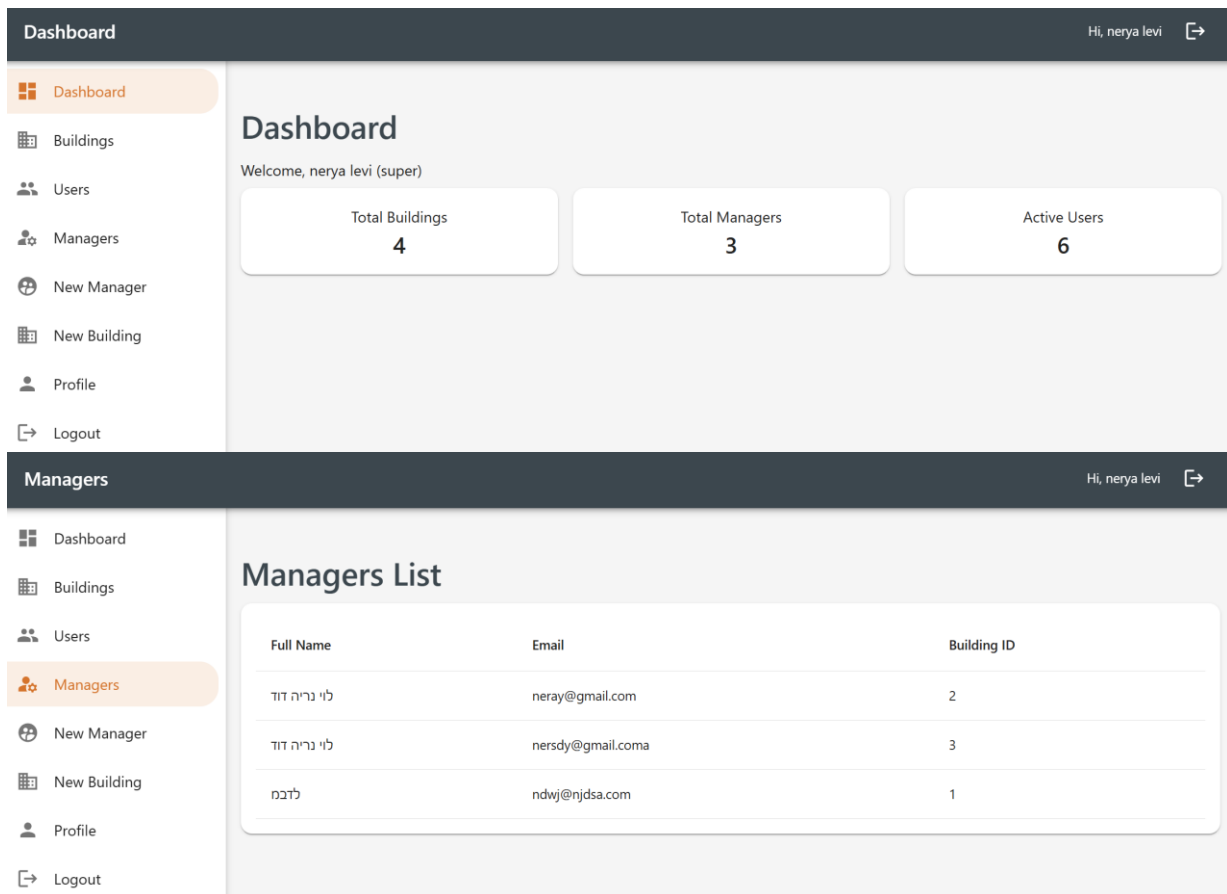
- משתמשים (Users Module)
- ניהול ציוד (Items Module)
- ניהול לוקרים (Lockers Module)
- ניהול השכרות (Rentals Module)
- אזור ניהול (Admin Module)
- Authentication & Authorization Module

7.6. סביבת השרת (מקומי, וירטואלי, ענן, שירות אירוח)

- ניתן להריץ מקומית לפיתוח.
- בפריסה – אפשרות לשרת ענן (כמו: Render, Railway, EC2).
- תמיכה ב־HTTPS ו־SSL לתקשורת מאובטחת.

7.7. ממשק המשתמש/לקוח – GUI

The screenshot shows a web application interface for creating a new user. The header is dark with 'New User' on the left and 'Hi, לובם' on the right. The sidebar on the left has a list of navigation items: Dashboard, New User (highlighted), Tools Shop, My Rentals, Profile, and Logout. The main content area is titled 'Create User' and contains a form with the following fields: ID (TZ), Full Name, Email, Phone, and Password. A 'Create User' button is located at the bottom right of the form.



7.8. ממשקים למערכות אחרות / API :

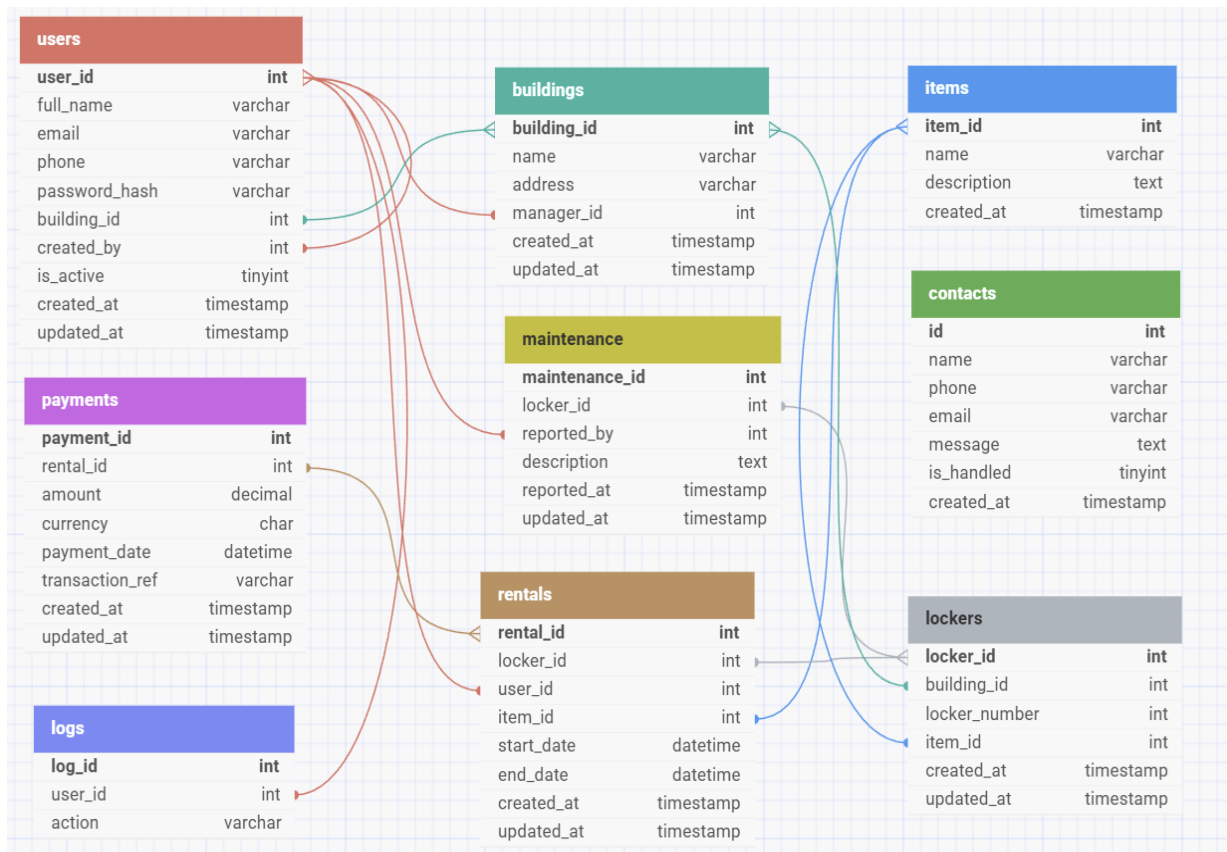
• אין

7.9. שימוש בחבילות תוכנה :

- bcrypt - הצפנת סיסמאות.
- JWT - לאימות.
- mysql2 - חיבור לבסיס הנתונים.
- fetch - תקשורת מול השרת.
- React Query - ניהול תקשורת נתונים בצד לקוח.
- CORS - הגנה ושיפור אבטחה.

8. שימוש במבני נתונים וארגון קבצים

8.1. נא פרט את מבני הנתונים.



8.2. נא פרט את שיטת האיחסון (מאגר, קבצים ובאיזה טכנולוגיה)

המערכת עושה שימוש בשיטת אחסון המבוססת על **מאגר נתונים רלציוני (Relational Database)** בטכנולוגיית **MySQL**, כפי שמוגדר בסכימת ה־SQL של הפרויקט.

א. מאגר נתונים MySQL – Database Storage

כל המידע במערכת נשמר במבנה טבלאות רלציוניות, הכוללות קשרים מסוג 1-N בין ישויות. הטכנולוגיה שנבחרה:

MySQL - בזכות יציבות, תמיכה ב־ Foreign Keys, ביצועים טובים ויכולת Scalability. במאגר נשמרים הנתונים הבאים:

- **users** - נתוני משתמשים והרשאות
- **buildings** - פרטי בניינים ומנהלים
- **lockers** - לוקרים פיזיים בבניין
- **items** - פריטי ציוד הנמצאים בלוקרים
- **rentals** - השכרות פעילות והיסטוריות
- **payments** - תשלומים עבור השכרות
- **maintenance** - תקלות ותחזוקה
- **contacts** - פניות משתמשים
- **logs** - רישום פעולות (לוגים)

אחסון קבצים (File Storage)

נכון לגרסה הנוכחית של הפרויקט

אין אחסון קבצים במערכת

(כמו תמונות, מסמכים או קונפיגורציות). במידת הצורך בעתיד:

- ניתן להוסיף תיקיית uploads בשרת
- או להשתמש באחסון בענן (לדוגמה AWS S3):

נימוק לבחירת הטכנולוגיה

- **MySQL** מתאים במיוחד לפרויקטים של מערכת השכרה ולוקרים בגלל:
 - ✓ תמיכה בקשרים בין ישויות (FK)
 - ✓ שמירה על עקביות נתונים (ACID)
 - ✓ נוח למימוש ולתחזוקה
 - ✓ זמינות גבוהה
 - ✓ אינדקסים לשיפור ביצועים
- אחסון קבצים הוחלט שלא להשתמש בו בשלב זה, כדי לפשט את המערכת.

8.3. נא ציין מנגנוני התאוששות מנפילה/קריסה/תמיכה בטרנזקציות.

- שימוש ב־ Transactions (ACID) פעולות קריטיות כמו פתיחת השכרה, סגירה ותשלום מבוצעות בתוך טרנזקציה כדי למנוע פגיעה בנתונים במקרה של כשל.
- Foreign Keys עם CASCADE / SET NULL שומרים על עקביות הנתונים ומונעים רשומות יתומות במקרה של מחיקות או עדכונים.
- אינדקסים לשיפור ביצועים
- אינדקסים על שדות חשובים מפחיתים עומסים ומונעים קריסות בזמני עומס.
- גיבוי נתונים תקופתי
- מאפשר שחזור המערכת במקרה של מחיקה או תקלה חמורה.
- Health Checks ובדיקת זמינות
- המערכת בודקת שהתשתיות (שרת ו־DB) פעילות ויציבות.
- מודולריות במערכת
- תקלה במודול אחד אינה מפילה את כל השרת.

9. תרשימי מערכת מרכזיים

9.1 Use Case **אופן השימוש**

מנהל מערכת יוצר מנהל בניין	שמירת פרטי מנהל בניין במסד הנתונים שיוך מנהל לבניין
מנהל מערכת יוצר בניין	שמירת פרטי בניין במסד הנתונים עדכון רשימת בניינים פעילים
מנהל בניין מתחבר למערכת	אימות משתמש טעינת הרשאות מנהל בניין
מנהל בניין יוצר משתמשים בבניין	שמירת פרטי משתמש במסד הנתונים שיוך משתמש לבניין
משתמש רגיל מתחבר למערכת	אימות משתמש טעינת פרטי משתמש
המשתמש צופה ברשימת ציוד זמין	DB-שליפת ציוד זמין מה בדיקה שמוצרים אינם מושכרים
המשתמש בוחר ציוד להשכרה	בדיקת זמינות הציוד (זמני lock) נעילת הציוד
המשתמש מאשר השכרה	'צירת הזמנה במערכת שיוך ההזמנה ללוקר פנוי רישום שעת התחלת השכרה חישוב מחיר ראשוני
המערכת שולחת פקודת פתיחה ללוקר	הלוקר נפתח עדכון סטטוס: ציוד מושכר
המשתמש לוקח את הציוד	אישור פתיחת לוקר עדכון סטטוס ההשכרה
"המשתמש בוחר "החזרת ציוד"	איתור ההזמנה הפעילה הקצאת לוקר החזרה שליחת פקודת פתיחה ללוקר
המשתמש ניגש ללוקר	פתיחת לוקר החזרה התחלת מדידת זמן החזרה
המשתמש מחזיר ציוד וסוגר לוקר	אישור סגירת לוקר רישום שעת סיום חישוב זמן השכרה ותשלום סופי עדכון סטטוס: ציוד זמין

התרשימים יתארו שני תרחישים מרכזיים:

דייר: (User)

- התחברות למערכת
- צפייה בציווד זמין בלוקרים בבניין שלו
- בחירת לוקר / כלי וביצוע השכרה
- פתיחת לוקר וקבלת הציווד
- החזרת הציווד וסיום השכרה
- צפייה בהיסטוריית השכרות ותשלומים אישית

מנהל בניין: (buildingAdmin)

- התחברות לאזור ניהול
- צפייה בניהול משתמשים השייכים לבניין (אישור / חסימה / עדכון פרטים בסיסיים)
- ניהול לוקרים בבניין: סטטוס (available/occupied/maintenance), שיוך לוקרים לבניין
- צפייה בהשכרות הפעילות והיסטוריית השכרות בבניין
- צפייה בקריאות תחזוקה (maintenance) וטיפול/עדכון סטטוס שלהן

מנהל מערכת ראשי: (Super)

- התחברות לממשק ניהול על (Global Admin)
- ניהול משתמשים כלל מערכת (יצירה, עדכון, הגדרת תפקידים והרשאות)
- ניהול בניינים (הוספה/עדכון/שיוך מנהל בניין)
- ניהול ציווד: (Items) הוספה, עדכון, הסרה, שיוך ציווד ללוקרים
- ניהול לוקרים ברמת מערכת (הגדרת לוקרים חדשים למערכת)
- צפייה בכלל ההשכרות, התשלומים וקריאות התחזוקה בכל הבניינים
- צפייה בלוגים ובפעולות מערכת לצורך בקרה ואבטחה

9.2. Sequence diagram - רצף קריאות פונקציות מרכזיות בלוגיקה העסקית המרכזית של

הפרויקט

תרחיש: משתמש מבצע השכרה של כלי בלוקר

Actors:

- User (Frontend)
- Backend API
- Database (MySQL)
- Locker Controller (לוקר חכם)

Sequence Flow – High Level

- User → Frontend:
מבצע Login ושולח בקשת התחברות.
- Frontend → Backend:
בקשת POST /login עם פרטי המשתמש.
- Backend → Database:
בדיקת משתמש. (SELECT * FROM users WHERE email = ...)
- Frontend → JWT לחזרה.
אם תקין
- User → Frontend:
בוחר כלי / לוקר זמין.
- Frontend → Backend:
בקשת POST /rentals להתחלת השכרה:
 - user_id
 - locker_id
 - item_id
- Backend → Database:
Transaction Start
 - יצירת רשומה בטבלת rentals
 - עדכון סטטוס לוקר ל-occupied
 - יצירת רשומת תשלום ראשונית (pending)
- Backend → Locker Controller:
שליחת פקודת פתיחה ללוקר. (open_locker)
- Locker Controller → Backend:
מחזיר סטטוס. "Opened"
- Backend → Database:
עדכון rental כפעיל. (status='active')
- Transaction Commit
- Backend → Frontend:
החזרת תשובה. "Rental Started + Locker Opened"

Sequence - החזרת הציווד

User → Frontend: ○

לוחץ “סיים השכרה / החזרה”.

Frontend → Backend: ○

בקשת PUT /rentals/{id}/complete.

Backend → Database: ○

עדכון end_date

חישוב סכום סופי (לפי זמן)

עדכון תשלום ל-completed

עדכון לוקר ל-available

Backend → Frontend: ○

תשובה: “Rental Completed” :

9.3 Data flow

משתמש → (User / Admin / Super) ממשק משתמש (Frontend)

- המשתמש מבצע פעולות כמו התחברות, צפייה בציווד, פתיחת לוקר, ניהול לוקרים וכו'.
- ה־ Frontend שולח בקשות API לשרת.

2. Frontend → Backend (שרת) Node.js / Express

Backend: כל פעולה של המשתמש נשלחת ל־

- בקשות Login / Register
- בקשות להתחלת השכרה
- בקשות סיום השכרה
- ניהול משתמשים / לוקרים / ציוד
- פעולות מנהל ושליחת נתונים לניהול השרת אחראי על לוגיקה עסקית, חישובים ואימות הרשאות. (JWT)

3. Backend → Database (MySQL)

ה־ Backend מבצע קריאות קריאה/כתיבה לכל אחת מהטבלאות :

- users
- buildings
- lockers
- items
- rentals
- payments
- maintenance
- contacts
- logs

המידע נשמר במבנה רלציוני עם Foreign Keys וטרנזקציות.

4. Backend → Locker Controller (לוקרים חכמים)

לפעולות כמו :

- פתיחת לוקר
 - עדכון סטטוס לוקר
 - קבלת תגובה על פתיחה/תקלה
- השרת שולח פקודות ללוקרים דרך WebSocket / HTTP

5. Locker Controller → Backend

הלוקר מחזיר לשרת :

- “Opened” / “Failed”
- סטטוס דלת
- תקלות תחזוקה

6. Backend → Frontend → משתמש

השרת מחזיר תוצאות ועידכונים :

- הצלחת השכרה
- פתיחת לוקר
- סכום לתשלום
- היסטוריית השכרות
- נתוני ניהול

סיכום זרימת המידע

User ⇌ Frontend ⇌ Backend ⇌ Database

10. תיאור המרכיב האלגוריתמי – חישובי

10.1. איזה בעיה בא לפתור, איך יפתור?

אין

10.2. איסוף מידע וניתוחים סטטיסטיים (אנליטיקות)

המערכת אוספת נתונים ממספר טבלאות (rentals, payments, lockers, users) ומאפשרת ביצוע ניתוחים סטטיסטיים בסיסיים, לדוגמה:

- שימושיות המערכת:
 - מספר השכרות בתקופה (ליום/חודש/בניין).
 - ממוצע זמן השכרה לכל כלי / לוקר.
 - ניתוח הכנסות:
 - סך כל ההכנסות לתקופה מסוימת.
 - הכנסה ממוצעת להשכרה.
 - השוואת הכנסות בין בניינים שונים.
 - ניתוח ניצול ציוד:
 - אילו לוקרים/כלים בשימוש גבוה. (Top used)
 - אילו כלים כמעט ולא מושכרים – זיהוי ציוד לא כדאי.
- נתונים אלה יכולים להיות מוצגים בדוחות או גרפים באיזור הניהול, ולסייע בקבלת החלטות לגבי הרחבת ציוד, שינוי תעריפים או הוספת לוקרים בבניינים פעילים במיוחד.

11. תיאור/התייחסות לנושאי אבטחת מידע

נא לציין אזורים הדורשים אבטחה, כגון: שרת, בקרת גישה לאתר, חשבונות משתמשים, מאגרי מידע וכיצד ניתן מענה.

נא ציין מס' מקרים ותגובות להם ניתן מענה אבטחתי. **המערכת כוללת מספר אזורים הדורשים אבטחה ברמה גבוהה, מאחר שהיא מטפלת במשתמשים, לוקרים חכמים, תשלומים ונתוני בניינים. אזורים הדורשים אבטחה**

שרת (Backend API)

- שימוש ב־HTTPS לכל התקשורת.
- אימות משתמשים באמצעות JWT Tokens.
- סינון בקשות (Validation) למניעת הזרקות (SQL Injection / XSS).
- בקרת גישה והרשאות (Role-Based Access Control)
- הפרדה בין תפקידי משתמש. User / buildingAdmin / SuperAdmin :
- כל תפקיד יכול לבצע רק פעולות המותרות לו.
- חסימת פעולות מנהל בפני משתמש רגיל.
- חשבונות משתמשים
- סיסמאות מוצפנות באמצעות bcrypt.
- מנגנון חסימת משתמשים לא פעילים (is_active).
- מאגר נתונים (Database)
- שימוש ב־Foreign Keys לשמירת עקביות נתונים.
- הרשאות DB מוגבלות – השרת בלבד יכול לקרוא/לכתוב.
- גיבוי תקופתי ושחזור במקרה של קריסה.
- לוקרים חכמים (Locker Controller)
- שליחת פקודות מאובטחות מהשרת בלבד.
- אימות בין השרת ללוקרים כדי למנוע פתיחה לא מורשית.

מקרה 1 – ניסיון גישה ללא התחברות

תגובה:

ה־Backend בודק JWT; אם חסר או לא תקין → החזרת שגיאת 401 (Unauthorized).

מקרה 2 – ניסיון של משתמש רגיל לבצע פעולת מנהל

לדוגמה: מחיקת לוקר או ניהול משתמשים.

תגובה:

בדיקת תפקיד (Role) בצד השרת → החזרת 403 (Forbidden).

מקרה 3 – ניסיון הזרקת קוד (SQL Injection)

תגובה:

כל שאילתות DB מבוצעות באמצעות Prepared Statements. בנוסף Validation בצד השרת מונע הכנסת קלט מסוכן.

מקרה 4 – סיסמה שנגנבה או נחשפה

תגובה:

המערכת שומרת סיסמאות רק לאחר הצפנה (bcrypt), ולכן לא ניתן לשחזר אותן. בנוסף קיימת אפשרות להשבית משתמש (is_active=0).

מקרה 5 – ניסיון פתיחת לוקר בצורה לא חוקית

תגובה:

רק השרת יכול לשלוח פקודת פתיחה; כל פקודה נבדקת מול רשומת השכרה פעילה. ללא השכרה פעילה – הפקודה נדחית.

מקרה 6 – נפילת שרת או קריסת DB

תגובה:

- שימוש בגיבויים אוטומטיים
- תמיכה בטרנזקציות למניעת נזק לנתונים
- אפשרות לשחזור מלא מתוך Backup

12. משאבים הנדרשים לפרויקט:

12.1. מספר שעות המוקדש לפרויקט, חלוקת עבודה בין חברי הצוות

300

12.2. ציוד נדרש

- מחשבים: מחשבים אישיים לכל חבר צוות לצורך פיתוח, קימפול ובדיקה.
- שרת פיתוח/בדיקות: סביבת ענן בסיסית (כגון AWS EC2, Render או Railway) להטמעת השרת וה-Database לבדיקות סופיות.
- מודל לוקר (אופציונלי אך מומלץ): לוח פיתוח פיזי דוגמת Arduino/Raspberry Pi/ESP32 עם מנעול אלקטרוני (Solenoid Lock) ומודול תקשורת, Wi-Fi/LAN, המדמה את פעולת ה-Locker Controller.
- רשת: חיבור אינטרנט יציב לצורך תקשורת בין השרת ללוקר

12.3. תוכנות נדרשות

VS Code, MySQL, Insomnia, IDE

12.4. ידע חדש שנדרש ללמוד לצורך ביצוע הפרויקט

- תקשורת: Real-Time: הבנה מעמיקה של פרוטוקולי WebSocket או MQTT לצורך תקשורת יציבה ורציפה בין השרת לבקר הלוקרים.
- אבטחת מידע: יישום נכון של JWT Authentication ושיטות הצפנה (bcrypt) לסיסמאות.
- טיפול בטרנזקציות: יישום עקבי של טרנזקציות ב MySQL -לפעולות קריטיות (התחלה/סיום השכרה) למניעת אובדן נתונים.
- ספריית React מתקדמת: שימוש ב React Hooks ובספריות ניהול מצב נתונים מתקדמות (כגון React Query).

12.5. ספרות ומקורות מידע

W3
MUI

13. תכנית עבודה ושלבים למימוש הפרויקט

שלב 1 – אפיון וניתוח דרישות (Requirement Analysis)

- הגדרת מטרת הפרויקט
- זיהוי תרחישי שימוש (Use Cases)
- הגדרת תפקידי משתמשים SuperAdmin / buildingAdmin / User :
- אפיון מבנה מסד הנתונים (DB)
- אפיון תקשורת בין המערכת ללוקרים חכמים

שלב 2 – תכנון ארכיטקטורה (System & DB Design)

- תכנון ארכיטקטורת Client-Server
- יצירת תרשימי ERD, Use Case ו-Sequence
- תכנון API endpoints
- בניית סכימת MySQL עם קשרי PK/FK
- תכנון רכיבי Frontend וזרימת מסכים

שלב 3 – הקמת מסד נתונים (Database Implementation)

- יצירת הטבלאות, users, buildings, lockers, items, rentals, payments, maintenance, contacts, logs :

- הגדרת טרנזקציות ואינדקסים
- בדיקת שלמות קשרים (Constraints)

שלב 4 – פיתוח (Node.js / Express) Backend

- בניית שכבת API מלאה
- יישום לוגיקה עסקית:
 - השכרות
 - תשלומים
 - סטטוס לוקרים
 - ניהול משתמשים
- יישום אימות משתמשים (JWT)
- פיתוח חיבור למאגר הנתונים
- פיתוח תקשורת עם הלוקרים (WebSocket / HTTP)

שלב 5 – פיתוח (React) Frontend

- בניית מסכי משתמש:
 - התחברות
 - דף ציוד זמין
 - ניהול השכרות
 - היסטוריית תשלומים
- בניית מסכי מנהל בניין:
 - ניהול לוקרים
 - ניהול משתמשים
 - תקלות תחזוקה
- מסכי Super Admin: ניהול בניינים וציוד
- שימוש בממשק MUI לעיצוב אחיד

שלב 6 – בדיקות (Testing)

- בדיקות יחידה – (Unit Tests) בדיקת פונקציות לוגיקה מרכזיות
- בדיקות תהליכיות – (Full Flow) התחברות, השכרה, פתיחת לוקר, תשלום
- בדיקות תקשורת לוקר-שרת
- בדיקות עומס ותגובה לבעיות רשת
- תיקון באגים ושיפור ביצועים

שלב 7 – אינטגרציה בין המערכות

- שילוב מלא בין Frontend ↔ Backend ↔ Database
- בדיקת שליחת פקודות ללוקרים וקליטת תגובות
- בדיקה מקיפה של תרחישים אמיתיים

שלב 8 – תיעוד

- תיעוד תהליך הפיתוח
- כתיבת הספר הטכני
- הוספת תרשימים (ERD, Sequence, Use Case, Data Flow)
- הצגת הפרויקט ומסקנות

14. תכנון הבדיקות שיבוצעו

14.1. נא פרט בטבלה, בדיקות תהליכיות ברמת משתמש בהן נדרשת המערכת לעמוד (full Flow).

14.2. נא פרט בטבלה, מס מייצג של בדיקות יחידה למודולים המרכזיים בהן נדרשת המערכת לעמוד. (unit test)

מודול מרכזי	מספר בדיקות יחידה	תיאור הבדיקה	תוצאה צפויה
התחברות משתמש (Login)	3	בדיקה עם סיסמה נכונה, לא נכונה, משתמש לא קיים	הצלחה/שגיאה
הרשאות משתמש	2	בדיקה של גישה למודולים לפי סוג משתמש	רק משתמשים מורשים יכולים לגשת
ניהול לוקרים	4	בדיקה של יצירת לוקר, מחיקה, עדכון סטטוס, בדיקת לוקר תפוס	לוקר מתעדכן בהתאם
ניהול ציוד	3	בדיקה של הוספת ציוד, מחיקה, עדכון	ציוד נוסף/מעודכן/נמחק בהצלחה
השכרת לוקר/ציוד	3	בדיקה של השכרה, החזרה, מצב לוקר לאחר השכרה	סטטוס נכון אחרי פעולות
הצגת היסטוריה	2	בדיקה של הצגת היסטוריית השכרות	נתונים מוצגים נכון לפי תאריכים
התראות / הודעות	2	בדיקה של שליחת התראה במצב חריגה	הודעה מתקבלת על ידי המשתמש

15. בקרת גרסאות (version control)

<https://github.com/Nerya253/BuildLocker>

חתימת המנחה האישי

חתימת הסטודנט

ב. הערות ראש המגמה במכללה

ג. אישור ראש המגמה

שם: _____ חתימה: _____ תאריך: _____

ד. הערות הגורם המקצועי מטעם מה"ט

ה. אישור הגורם המקצועי מטעם מה"ט

שם: _____ חתימה: _____ תאריך: _____

נספח להצעת הפרויקט:

קווים מנחים בבחירת נושאים והיקפי עבודה בפרויקט הגמר.

1. דגשים ארכיטקטוניים ושיקולים במימוש:

- 1.1. מומלץ להתנסות בארכיטקטורות השלבות שימוש בתצורת שרת לקוח.
- 1.2. שימוש ב- design patterns במודולי התוכנה השונים- באיזורים מתאימים.
- 1.3. דגש על הפרדה בין לוגיקה עסקית השייכת לצד לקוח וצד שרת. (FrontEnd,Backend, ServerBL, ClientBL)
- 1.4. חלופות ארכיטקטוניות נדרשות לתמוך או לספק מענה לדרישות כגון:
 - 1.4.1. תמיכה והתמודדות בוויסות עומסים.
 - 1.4.2. תמיכה והתמודדות עם שיקולי אבטחה והגנה על מידע.
 - 1.4.3. תמיכה בשרידות והתאוששות מתקלה(טרנזקציות שמירה למאגר, והתאוששות)
- 1.5. תמיכה בשיקולי חווית משתמש (צד מנהל מערכת וצד משתמש קצה)
- 1.6. תמיכה היכן שניתן בניהול פרופילי משתמשים.

2. שפת מימוש הפרויקט-

- ישנו משקל גבוה במימוש הפרויקט ביותר משפת מימוש אחת לפרויקט, תוך מתן דגש ליתרונות היחסיים של כל שפה, עבור מודול תוכנה במכלולי הפרויקט. למשל במקרה של תצורת שרת לקוח (אתר אינטרנט):
- 2.1.1. לצד הניהול העיסקי של השרת, בחירה בשפות עיליות JAVA, C# או nodeJS.
 - 2.1.2. לתכולה חישובית/ אלגוריתמית- מימוש בשפת native נניח C, C++.
 - 2.1.3. לצד לקוח AngularJS, Asp.net וכו'

3. מאגר נתונים Database:

- 3.1. ישנה חשיבות גבוהה להתנסות בעבודה עם מאגרי נתונים למשל, מאגר רלציוני ומאגר FS Based למשל:
 - 3.1.1. עבור מאגר רלציוני נבחר ב- 'Sql server, Sqlite, etc'
 - 3.1.2. עבור מאגר לא רלציוני נבחר ב- mongoDB או NoSql.
- 3.2. ישנה חשיבות רבה להגדרת שכבת גישה למאגר הנתונים כזו שתנהל מרכיבים טרנזקטיביים וסנכרון. נין להשתמש גם במסגרת frameworks קיימים כדוגמת dotNet.

4. מרכיב אלגוריתמי/ חישובי-

- ישנה חשיבות רבה להתנסותו של התלמיד והתמודדותו עם יכולות חישוביות במסגרת מכלולי הפרויקט. ניתן לשלב היבטים אלגוריתמיים או לחילופין ניתוחים וחיתוכים סטטיסטיים בסיסיים מעל מאגר נתונים, למשל:
- 4.1. במקרה של אתרים כניסת משתמשים, גיאומטריה חישובית וכו'.

5. בדיקות תוכנה:

5.1. יש לגזור מדרישות המוצר אוסף בדיקות שיפרדו לפחות לשתי קטגוריות מרכזיות ויכסו את מרבית הקוד:

5.1.1. בדיקות יחידה (Unit-Test) - אלה הן אותן בדיקות אותן יממש המפתח ברמת פנים מכלולי התוכנה ועד לרמת הפונקציות הציבוריות באותם מכלולי תוכנה.

5.1.2. בדיקות תהליכיות (Full Flow) - הללו יעסקו בעיקר בבדיקות בקשר שבין מכלולי תוכנה מרכזיים ויבחנו את הפונקציונאליות האינטגרטיבית של המוצר, מקצה לקצה.

5.1.3. גישות מבורכות לתהליך ניתן לאמץ מתוך גישות שונות

5.1.4. למשל:

5.1.4.1. code a little test a little

5.1.4.2. , test driven development,

5.1.4.3. Regression Tests

פרקים מרכזיים נוספים לספר הפרויקט:

16. דרישות מערכת ופונקציונאליות

16.1. הנחות יסוד

16.1.1. הנחות יסוד הקשורות לסביבה הטכנולוגית ולתפקוד.

16.2. דרישות מערכת

סביבת הטמעה ושימוש. שרידות, ביצועים והתמודדות עם עומסים.

16.3. דרישות פונקציונאליות.

רשימת דרישות המשתמש מהמערכת, מהן הפעולות בהן נדרשת המערכת לתמוך.

5.2. חלופות ארכיטקטוניות-

דיון בבחירת ארכיטקטורות ברמת המערכת וברמת מכלולי התוכנה.

5.2.1. ברמת המערכת - חשוב להציג בספר הפרויקט בחינה של מספר חלופות

ארכיטקטוניות (לפחות 3) בהן יבחן התלמיד את האפשרויות השונות תחת מספר אילוצים רלוונטיים נשוא הפרויקט, בין יתר השיקולים ניתן לשקול: התמודדות עם עומסים וויסותם, שרידות, יכולת מימוש, זמינות טכנולוגית ועוד. יש לשקול שילוב במקומות המתאימים של design patterns מקובלים, הן ברמת המערכת והן ברמת מכלולי התוכנה.

5.2.2. ברמת מכלולי התוכנה - ניתן להציג דיון חלופות מצומצם יותר, אך לשקול בחיוב

שילוב של design patterns מקובלים במקומות המתאימים (להימנע משימוש מיותר).

6. טופולוגיית הפתרון הנבחר - הצגה סכימתית של פרישת המערכת.

7. ארכיטקטורה נבחרת: הצגה בגישת UML את פריסת מרכיבי הפתרון בחלוקה למכלולי

תוכנה ראשיים ומשניים כמו-גם הדיאגרמות הרלוונטיות.

7.1. שימוש במבני נתונים וארגון קבצים

7.1.1. נא פרט את מבני הנתונים.

7.1.2. נא פרט את שיטת האיחסון (מאגר, קבצים ובאיזה טכנולוגיה)

7.1.3. נא ציין מנגנוני התאוששות מנפילה לקריסה והתמיכה בטראנזקציות.

7.2. תרשים זרימת המידע במערכת

7.2.1. Use Cases

7.2.2. Sequence diagram

Data flow 7.2.3

7.3. חלופות שפת מימוש-

במסגרת ספר הפרויקט חשוב להציג בחינה של מספר חלופות עבור שפה/ות מימוש הפרויקט. הנ"ל צריך לכלול דרישות אותן יגדיר התלמיד בבחירת השפה המתאימה. בין יתר השיקולים ניתן לכלול: זמני ריצה, היבטי אבטחה והגנה, הגנה על זכויות יוצרים (בינארי או interpreter, קלות במימוש, התאמה לממשקי משתמש או צד שרת) וכו'.

8. חלופות אבטחתיות והגנה –

במסגרת ספר הפרויקט יש לבחון לפחות 3 חלופות אבטחתיות להגנה ושמירה על נתונים, יש לכלול התמודדות עם מקרים ותגובות בתהליך הבחינה (לפחות 10 מקרים).

8.1.1. יש לשים לב- לפרוטוקולי תקשורת, http, https, ssl###,

8.1.2. מכלולי תוכנה צד שלישי (אנטי וירוס, מצפינים, firewalls וכו').

8.1.3. ברמת הקוד- ווידוא סכימות הודעות בין מכלולי התוכנה ובתקשורת בניהם.

8.2. פירוט בדיקות תוכנה ואופן ביצוען (STP-DOC) - לכלול את רשימת בדיקות התוכנה, בדיקות יחידה, בדיקות תהליכיות- full Flow במסגרת מסמך תכנון בדיקות ובדיקות. הנ"ל יוצג בטבלה : תיאור הבדיקה, תוצאה רצויה, תוצאה מתקבלת.

ניתן לאמץ את נוהל מפתח באופן מושכל!

http://www.methodacloud.com/content/pages/kit_maxsum/H_Guide-map.asp