📁 תיק פרויקט – GroceryList

1. תיאור כללי

GroceryList היא מערכת אינטרנטית לניהול רשימות קניות. המשתמשים יכולים להירשם, להתחבר, ולהוסיף מצרכים לרשימה אישית. כל משתמש רואה רק את הפריטים שלו, ויכול לבצע CRUD מלא עליהם. המטרה היא להחליף פתקים אבודים באפליקציה נוחה ומאובטחת.

2. דרישות

2.1 פונקציונליות

- הרשמה עם שם, מייל וסיסמה
- התחברות
- ניהול רשימת קניות (הוספה, עריכה, מחיקה)
- שליפת פריטים אישיים

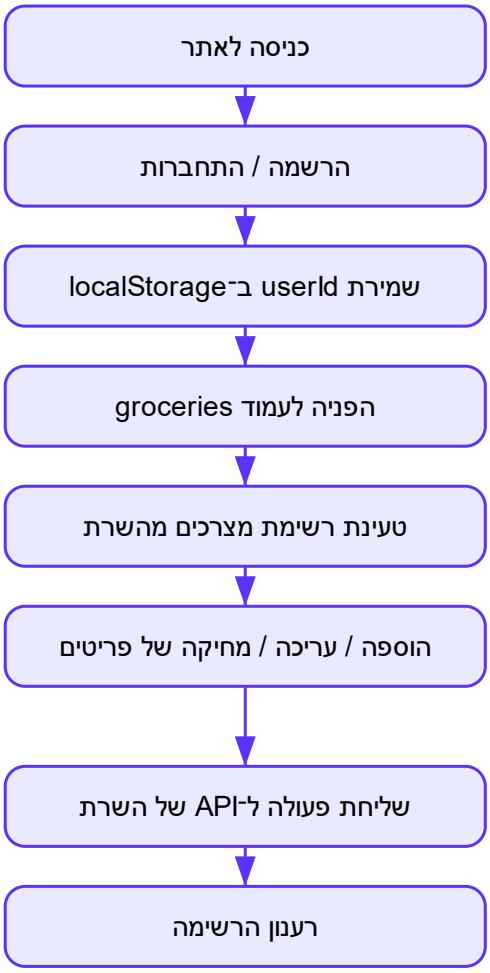
2.2 טכנולוגיות

- Frontend: HTML, CSS, JavaScript
- Backend: Node.js + Express
- Database: MySQL

3. ניווט בין מסכים

- / → עמוד הבית
- signup/ → הרשמה
- login/ → התחברות
- groceries/ → ניהול רשימה
- 404/ → דף שגיאה

🔄 תרשים זרימה - תהליך שימוש באפליקציה



4. קוד והסבר – צד לקוח

4.1 login.js

```
document.getElementById('loginForm').addEventListener('submit', async (e
  event.preventDefault();
  const email = document.getElementById("email").value;
  const password = document.getElementById("password").value;

  const response = await fetch('/api/users/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ email, password })
  });

  const data = await response.json();
```

```
  if (response.ok) {
    localStorage.setItem('userId', data.userId);
    window.location.href = '/groceries';
  } else {
    alert(data.error || 'Login failed');
  }
});
```

קוד זה מנהל את תהליך ההתחברות – מאזין לטופס, שולף ערכים, ושולח אותם לשרת. אם ההתחברות הצליחה, מזהה המשתמש נשמר ואנו מועברים לעמוד groceries.

signup.js 4.2

```
document.getElementById('signupForm').addEventListener('submit', async (
  event.preventDefault();

  const name = document.getElementById("name").value;
  const email = document.getElementById("email").value;
  const password = document.getElementById("password").value;
  const confirmPassword = document.getElementById("confirmPassword").val

  if (password !== confirmPassword) {
    alert("הסיסמאות אינן תואמות");
    return;
  }

  const response = await fetch('/api/users/register', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ name, email, password })
  });

  const data = await response.json();
  localStorage.setItem('userId', data.userId);
```

```
  window.location.href = '/groceries';
});
```

קוד זה מבצע ולידציה לסיסמאות ולאחר מכן שולח את נתוני ההרשמה לשרת. אם ההרשמה הצליחה, נשמר userId והמשתמש מועבר לעמוד הרשימה.

4.3 groceries.js

```
document.addEventListener("DOMContentLoaded", () => {
  const userId = localStorage.getItem("userId");
  fetch(`/api/list/${userId}`)
    .then(res => res.json())
    .then(data => renderGroceries(data));
});

function addGrocery() {
  const text = document.getElementById("groceryName").value;
  const userId = localStorage.getItem("userId");

  fetch("/api/list", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ text, userId })
  }).then(() => location.reload());
}
```

לאחר טעינת הדף, נשלחת בקשה לשליפת המצרכים של המשתמש. ניתן להוסיף פריט חדש דרך addGrocery – שנשמר במסד הנתונים.

5. קוד והסבר – צד שרת

5.1 users.js

```
// רישום
router.post('/register', (req, res) => {
```

```
  const { name, email, password } = req.body;
  db.query('INSERT INTO users (username, email, password) VALUES (?, ?,
    [name, email, password], (err, result) => {
      if (err) return res.status(500).json({ error: 'שגיאה ביצירת משתמש' })
      res.json({ message: 'משתמש נוצר בהצלחה', userId: result.insertId });
  });
});

// התחברות
router.post('/login', (req, res) => {
  const { email, password } = req.body;
  db.query('SELECT * FROM users WHERE email = ?', [email], (err, results
    if (err || results.length === 0) return res.status(401).json({ error

    const user = results[0];
    if (password == user.password) {
      res.json({ message: 'התחברת בהצלחה', userId: user.id });
    } else {
      res.status(401).json({ error: 'סיסמה שגויה' });
    }
  });
});
```

בקשת POST אחת מוסיפה משתמש חדש, והשנייה בודקת התחברות קיימת. לאחר התחברות תקינה מוחזר userId לזיהוי בצד הקליינט.

list.js 5.2

```
// שליפה
router.get('/:userId', (req, res) => {
  db.query('SELECT * FROM list_items WHERE user_id = ?', [req.params.use
    if (err) return res.status(500).json({ error: 'שגיאה בשליפה' });
    res.json(results);
  });
});
```

```
// הוספה
router.post('/', (req, res) => {
  const { text, userId } = req.body;
  db.query('INSERT INTO list_items (text, user_id) VALUES (?, ?)', [text
    if (err) return res.status(500).json({ error: 'שגיאה בהוספה' });
    res.json({ message: 'נוסף בהצלחה', id: result.insertId });
  });
});
```

הנתיבים האלו שולפים פריטים לפי משתמש ומוסיפים פריט חדש לרשימת המשתמש במסד הנתונים. בקריאה GET – מתבצעת שאילתה על פי userId ונשלפת רשימת המצרכים. בקריאה POST – מתווסף פריט חדש לשורת הטבלה עם הקישור למשתמש הרלוונטי.

5.3 list.js – עדכון פריט (PUT)

```
router.put('/:id', (req, res) => {
  const { text } = req.body;
  db.query('UPDATE list_items SET text = ? WHERE id = ?', [text, req.par
    if (err) return res.status(500).json({ error: 'שגיאה בעדכון' });
    res.json({ message: 'עודכן בהצלחה' });
  });
});
```

בקשת PUT מאפשרת למשתמש לעדכן את שם המצרך עבור פריט מסוים. הפריט מזוהה לפי מזהה (id) והטקסט החדש נשלח לגוף הבקשה. אם העדכון מצליח – השרת מאשר את השינוי ומחזיר הודעת הצלחה.

5.4 list.js – מחיקת פריט (DELETE)

```
router.delete('/:id', (req, res) => {
  db.query('DELETE FROM list_items WHERE id = ?', [req.params.id], (err)
    if (err) return res.status(500).json({ error: 'שגיאה במחיקה' });
    res.json({ message: 'נמחק בהצלחה' });
```

```
  });
});
```

קוד זה משמש למחיקת פריט מהרשימה של המשתמש לפי מזהה (id). ברגע שנלחץ כפתור "מחק", מתבצעת קריאת DELETE אל השרת. אם המחיקה בוצעה בהצלחה – המידע נמחק מהמסד והקליינט מקבל אישור.

6. מסד נתונים – SetProject.sql

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL
);

CREATE TABLE list_items (
  id INT AUTO_INCREMENT PRIMARY KEY,
  text VARCHAR(255) NOT NULL,
  user_id INT NOT NULL,
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```

המסד בנוי משתי טבלאות: users ו־list_items. טבלת users שומרת את פרטי המשתמשים עם אימייל ייחודי. טבלת list_items כוללת פריטים שכל אחד מהם מקושר למשתמש באמצעות מפתח זר. כך כל משתמש מקבל הפרדה ברורה ואישית לפריטי הרשימה שלו.

7. אבטחת מידע וולידציה

- ולידציה בצד הלקוח: בדיקה של שדות ריקים, אימייל תקין, סיסמה תואמת
- טיפול בשגיאות בשרת: החזרת תשובות משמעותיות במקרה של בעיות במסד
- מומלץ: להטמיע bcrypt להצפנת סיסמאות לפני שמירתן במסד
- מומלץ: להשתמש ב־JWT לצורך ניהול הרשאות וגישה מאובטחת
- שימוש ב־localStorage נשמר כפתרון זמני לצורך שמירת מזהה המשתמש

8. הפקת לקחים

- למדתי איך לבנות מערכת Client-Server עם אינטראקציה מלאה
- שימוש ב־MySQL איפשר לי להבין איך לבנות ולתחזק מסד נתונים רלציוני
- עבודה עם fetch ו־Express העניקה לי הבנה עמוקה ב־REST API
- הבנתי את חשיבות האבטחה – במיוחד כשמדובר בסיסמאות ונתונים רגישים
- בעתיד אשפר את המערכת עם אימותים חזקים יותר וחוויית משתמש משופרת

9. סיכום

היא מערכת בסיסית אך שלמה לניהול רשימות קניות מותאמות אישית. היא משלבת טכנולוגיות פופולריות כמו HTML, JS, Node.js ו־MySQL ליצירת אפליקציית Web תפקודית. הפרויקט פותח את הדלת להרחבות עתידיות כמו קטגוריות, תצוגות מתקדמות, או חיבור לחשבון Google. העבודה עליו חידדה את מיומנויותיי במבנה פרויקט, עבודה בצוותים והקפדה על אבטחת משתמשים.