פרויקט: מעקב פריסת יחידות צה"ל

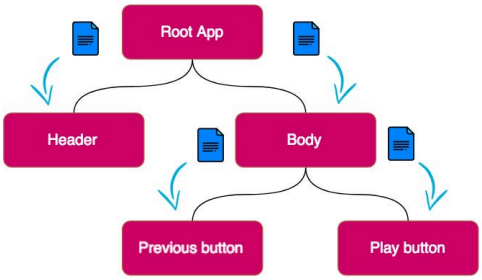
שימוש ב-ContextAPI, react-router-dom, conditional rendering, conditional formatting, useState

תיאור הפרויקט

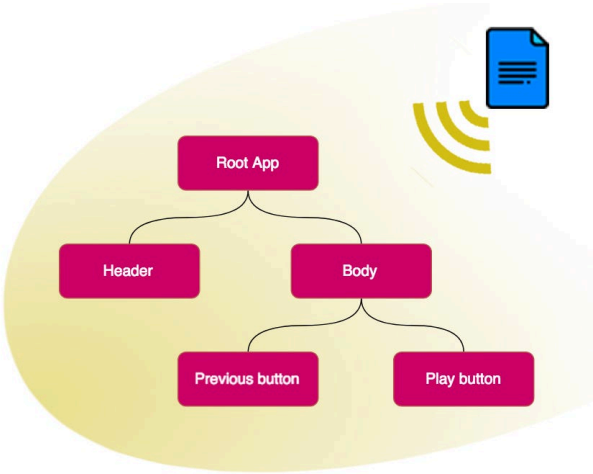
אפליקציית מעקב פריסת יחידות צה"ל היא יישום שנועד לעקוב אחר מצב הפריסה של יחידות וחטיבות שונות בצבא ההגנה לישראל, מאורגנות לקטגוריות כמו יחידות עילית, יחידות מיוחדות, יחידות קומנדו, וחטיבות לחימה. האפליקציה משתמשת במבנה נתונים כדי לשתף את מצבי היחידות, כך שניתן לעדכן ולהציג את מצב הפריסה של כל יחידה בממשק בצורה חלקה.

חלק א: בניית הפרויקט באמצעות ContextAPI

מה זה ContextAPI?



Passing prop to each level



The data in Context available to every component

ה-Context API מספק דרך ליצור משתנים גלובליים שנגישים לכל רכיב באפליקציה, ללא קשר לעומק המיקום שלו בעץ. זה שימושי במיוחד כאשר יש צורך לשתף נתונים (כמו מידע על משתמש, הגדרות עיצוב, או העדפות שפה) עם מספר רכיבים מבלי להעביר אותם בצורה ידנית דרך כל רכיב ורכיב באמצעות props.

ה-API כולל שני רכיבים עיקריים:

1. **Context Provider**: רכיב זה מספק את הנתונים לכל רכיב המנוי על הקונטקסט. ה-Provider עוטף חלקים של האפליקציה בהם הנתונים אמורים להיות נגישים.

2. **Context Consumer** (או **הוק useContext**): רכיב זה ניגש לנתונים מהקונטקסט בתוך רכיב מסוים.

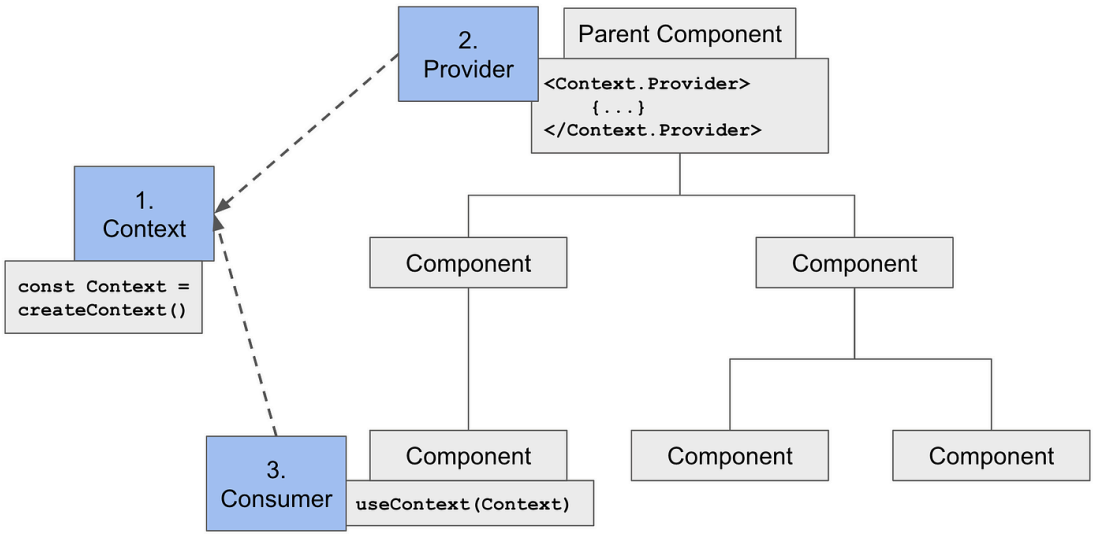
שלבים טיפוסיים ליצירת קונטקסט כוללים:

1. יצירת הקונטקסט באמצעות `React.createContext()`.
2. עטיפת עץ הרכיבים (או חלקים ממנו) עם ה-Provider והעברת הנתונים הרצויים.
3. שימוש ב-`useContext` hook בתוך רכיבים כדי לגשת לנתוני הקונטקסט.

יתרונות ה-Context API

1. **מונע Prop Drilling**:
   - Prop drilling הוא תהליך של העברת נתונים דרך מספר שכבות של רכיבים כדי להגיע לרכיב שממוקם עמוק בעץ וזקוק לנתונים. פעולה זו יכולה להקשות על קריאות וניהול הקוד. ה-Context API מסיר את הצורך ב-prop drilling, כיוון שהרכיבים יכולים לגשת לנתונים ישירות מהקונטקסט.
2. **ניהול מצב גלובלי**:
   - עבור נתונים שצריך לשתף בין רכיבים שונים באפליקציה (כגון פרטי משתמש, עיצוב, או העדפות שפה), ה-Context API מספק פתרון פשוט. זהו לעיתים קרובות פתרון פשוט יותר משימוש בספריות לניהול מצב (state) מורכבות יותר, כמו Redux, באפליקציות בינוניות.
3. **קל יותר לניהול ולהרחבה**:
   - כאשר משתמשים ב-Context API בצורה נכונה, קל יותר לנהל מצב באפליקציה, במיוחד בפרויקטים עם מספר רב של שכבות רכיבים. ה-Context מאפשר לארגן ולרכז את הנתונים, מה שמוביל לקוד שקל יותר לתחזוקה.
4. **שימוש חוזר והפרדת אחריות**:
   - כאשר משתמשים ב-Context לניהול נתונים שצריכים להיות משותפים, ניתן להשאיר את הרכיבים ממוקדים בפונקציונליות שלהם בלבד. זה מגביר את המודולריות והשימוש החוזר של הרכיבים כיוון שהם פחות תלויים בנתונים שעוברים דרך props.

איך להשתמש בפרויקט REACT?



מבנה התיקיות

```
JavaScript
IDF-Deployment-Project/
├── public/
├── src/
│   ├── components/
│   │   ├── UnitList.tsx
│   │   ├── UnitStatus.tsx
│   │   ├── ChangeStatus.tsx
│   │   ├── UnitDetails.tsx
│   │   ├── MissionCompleted.tsx
│   ├── context/
│   │   ├── DeploymentContext.tsx
│   ├── styles/
│   │   ├── UnitList.module.css
│   │   ├── UnitDetails.module.css
│   ├── App.tsx
│   ├── index.tsx
│   └── types/
│       ├── deployment.d.ts
├── package.json
└── README.md
```

שלבים ללימוד באמצעות הפרויקט

שלב 1: ContextAPI - הגדרת ה-Context לניהול מצב הפריסה

1. **הסבר על הרעיון**:
   - ניהול המצב מאפשר לנו לשתף את מצב היחידות על פני כל רכיבי האפליקציה ללא צורך בהעברת נתונים דרך רכיבים ביניים. נשמש בהקשר לניהול מצב הפריסה של יחידות צה"ל כמו גולני, צנחנים וגבעתי. גישה מרוכזת זו מאפשרת עדכון וגישה נוחה לנתוני הפריסה מכל רכיבי האפליקציה.

דוגמה (DeploymentContext.tsx):

```
JavaScript
import React, { createContext, useState } from 'react';
```

```
// הגדרת סוג המידע עבור הקשר הפריסה
type DeploymentContextType = {
    units: { [key: string]: string };
    setUnitStatus: (unit: string, status: string) => void;
};

// יצירת ההקשר
const DeploymentContext = createContext<DeploymentContextType |
undefined>(undefined);

const DeploymentProvider: React.FC = ({ children }) => {
    // הגדרת מצב התחלתי
    const [units, setUnits] = useState({ Golani: 'Idle', Paratroopers: 'Idle',
Givati: 'Idle' });

    // פונקציה לשינוי מצב היחידות
    const setUnitStatus = (unit: string, status: string) => {
        // לעדכון המצב `setUnits`-רמז: השתמשו ב
    };

    return (
        <DeploymentContext.Provider value={{ units, setUnitStatus }}>
            {children}
        </DeploymentContext.Provider>
    );
};

export { DeploymentContext, DeploymentProvider };
```

תרגילים לשלב 1

1. **השלימו את הפונקציה `setUnitStatus`** שתעדכן את מצב הפריסה של יחידה.
   - **רמז**: השתמשו ב-`setUnits` לעדכן את מצב היחידה באמצעות מיזוג מצב קודם והוספת היחידה החדשה.
2. **הוסיפו יחידות נוספות** במצב ההתחלתי כמו "תותחנים" ו"זרוע הים".
   - **רמז**: הוסיפו את השמות ליחידות במצב ההתחלתי.
3. **הוסיפו סוג נתונים עבור `DeploymentContextType`** בקובץ `deployment.d.ts`.
   - **רמז**: הגדירו סוג עבור `units` כאובייקט עם מפתחות המייצגים שמות יחידות וערכים המייצגים את מצב היחידה.

שלב 2: הגדרת רכיב האפליקציה המרכזי

1. **מבנה האפליקציה**:
   - האפליקציה מורכבת מרשימת יחידות, אפשרויות לשינוי מצב כל יחידה, והודעה שמוצגת כאשר כל היחידות הגיעו למצב "הושלם".

דוגמה (App.tsx):

```javascript
import React from 'react';
import { DeploymentProvider } from './context/DeploymentContext';
// יבאו את :רמז `UnitList` ו-`MissionCompleted` מהרכיבים

const App: React.FC = () => {
    return (
        <DeploymentProvider>
            <div style={{ padding: '20px', fontFamily: 'Arial, sans-serif' }}>
                <h1>מעקב פריסת יחידות צה"ל</h1>
                {/* רמז: הוסיפו כאן את רכיב `UnitList` ואת `MissionCompleted`
*/}
            </div>
        </DeploymentProvider>
    );
};

export default App;
```

2.

תרגילים לשלב 2

1. **יבאו והציגו את `UnitList` ו-`MissionCompleted`** בתוך רכיב ה-`DeploymentProvider` ב-`App.tsx`.
   - **רמז**: מקמו את הרכיבים `UnitList` ו-`MissionCompleted` מתחת לכותרת.
2. **התאימו את הכותרת** על ידי הוספת סגנונות מותאמים אישית.
   - **רמז**: נסו לשנות את גודל הפונט, הצבע והמיקום.
3. **הוסיפו כפתור רענון** להחזרת כל היחידות למצב "ממתין".
   - **רמז**: הוסיפו כפתור ב-`App.tsx` שמשנה את מצב כל היחידות בחזרה ל"ממתין".

שלב 3: יצירת רכיבי `UnitList` ו-`UnitStatus`

1. **הסבר על רכיבים**:
   - `UnitList` מציג רשימה של כל היחידות והמצב הנוכחי שלהן. `UnitStatus` הוא רכיב משנה שמציג את שם היחידה ומצבה.

דוגמה (UnitList.tsx):

```
JavaScript
import React, { useContext } from 'react';
import { DeploymentContext } from '../context/DeploymentContext';
// יבאו את :רמז `UnitStatus` ו-`ChangeStatus`

const UnitList: React.FC = () => {
    const deploymentContext = useContext(DeploymentContext);

    if (!deploymentContext) {
        throw new Error("UnitList must be used within a DeploymentProvider");
    }

    const { units } = deploymentContext;

    return (
        <div>
            <h2>רשימת יחידות</h2>
            {/* מיפוי על היחידות להצגת רכיבי :רמז `UnitStatus` ו-`ChangeStatus`
*/}
        </div>
    );
};

export default UnitList;
```

דוגמה (UnitStatus.tsx):

```
JavaScript
import React, { useContext } from 'react';
import { DeploymentContext } from '../context/DeploymentContext';

type UnitStatusProps = {
    unitName: string;
};
```

```
const UnitStatus: React.FC<UnitStatusProps> = ({ unitName }) => {
    const deploymentContext = useContext(DeploymentContext);

    if (!deploymentContext) {
        throw new Error("UnitStatus must be used within a DeploymentProvider");
    }

    const { units } = deploymentContext;

    return <p>{unitName}: {/* רמז: הציגו את המצב הנוכחי של היחידה כאן */}</p>;
};

export default UnitStatus;
```

תרגילים לשלב 3

1. **מלאו את פונקציית המיפוי** ברכיב `UnitList.tsx` כך שתציג את `UnitStatus` ו-`ChangeStatus` לכל יחידה.
   - **רמז**: השתמשו ב-`Object.keys(units).map` כדי למפות על היחידות ולהעביר את `unitName` לרכיב `UnitStatus`.
2. **הוסיפו צבעים שונים למצבי היחידות** ברכיב `UnitStatus.tsx`.
   - **רמז**: השתמשו בסגנונות CSS מותנים לפי המצב של כל יחידה.

שלב 4: יצירת רכיבי `ChangeStatus` ו-`MissionCompleted`

1. **הסבר על רכיבים**:
   - `ChangeStatus` הוא כפתור שמאפשר לשנות את מצב היחידה ל"פרוס".
   - `MissionCompleted` מציג הודעה כאשר כל היחידות הושלמו.

דוגמה (ChangeStatus.tsx):

```javascript
import React, { useContext } from 'react';
import { DeploymentContext } from '../context/DeploymentContext';

type ChangeStatusProps = {
```

```
    unitName: string;
};

const ChangeStatus: React.FC<ChangeStatusProps> = ({ unitName }) => {
    const deploymentContext = useContext(DeploymentContext);

    if (!deploymentContext) {
        throw new Error("ChangeStatus must be used within a
DeploymentProvider");
    }

    const { setUnitStatus } = deploymentContext;

    return (
        <button onClick={() => setUnitStatus(unitName, /* רמז: הכניסו כאן את
המצב "Deployed" */)}>
            פריסת {unitName}
        </button>
    );
};

export default ChangeStatus;
```

דוגמה (MissionCompleted.tsx):

```
JavaScript
import React, { useContext } from 'react';
import { DeploymentContext } from '../context/DeploymentContext';

const MissionCompleted: React.FC = () => {
    const deploymentContext = useContext(DeploymentContext);

    if (!deploymentContext) {
        throw new Error("MissionCompleted must be used within a
DeploymentProvider");
    }

    const { units } = deploymentContext;
    const allComplete = Object.values(units).every(status => /* רמז: בדקו אם כל
היחידות במצב "Complete" */);
```

```
    return (
        <div>
            {allComplete && <h2>כל היחידות הושלמו בהצלחה!</h2>}
        </div>
    );
};

export default MissionCompleted;
```

תרגילים לשלב 4

1. **התאימו את `onClick` ברכיב `ChangeStatus.tsx`** כך שיאפשר שינוי בין כל המצבים.
    - **רמז**: הגדירו מערך של מצבים ועדכנו את מצב היחידה לפי האינדקס הנוכחי.
2. **הוסיפו את רכיב `MissionCompleted`** כך שיופיע לאחר השלמת כל היחידות.
    - **רמז**: ודאו שמצב כל היחידות "הושלם" לפני הצגת ההודעה.
3. **עצבו את הודעת הסיום** כך שתבלוט יותר.
    - **רמז**: השתמשו ב-CSS או סגנונות מוטמעים כדי להבליט את ההודעה.

חלק ב: יצירת שרת Node עם מידע על יחידות צה"ל

הרחבת תרגיל: יצירת שרת Node.js מבוסס TypeScript המספק API לנתוני יחידות צה"ל ושימוש ב-React עם Context לקבלת הנתונים

בתרגיל זה תיצרו שרת בסיסי עם Node.js ו-TypeScript שישרת את הנתונים על יחידות צה"ל באמצעות API. הנתונים ייטענו לקובץ JSON המוטען לזיכרון בעת עליית השרת. נשתמש ב-React Context כדי לטעון את הנתונים מהשרת ולהפוך אותם לזמינים בכל רכיבי האפליקציה.

שלב 1: התקנת סביבת Node.js עם TypeScript

1. ודאו שיש לכם Node.js מותקן. אם לא, הורידו והתקינו אותו מהאתר הרשמי של Node.js.

צרו תיקייה חדשה עבור הפרויקט והתחילו בפרויקט חדש עם הפקודה:

```
JavaScript
npm init -y
```

התקינו את TypeScript ואת Types עבור Node.js ו-Express:

```
JavaScript
npm install typescript ts-node @types/node @types/express express axios
```

צרו קובץ tsconfig.json עם ההגדרות הבאות:

```
JavaScript
{
  "compilerOptions": {
    "target": "ES6",
    "module": "commonjs",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "outDir": "./dist"
```

```
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules"]
}
```

שלב 2: יצירת קובץ JSON

צרו קובץ בשם `idfUnits.json` בתיקיית הפרויקט שלכם והעתיקו אליו את הנתונים.

```JavaScript
{
  "Elite Units": [
    {
      "name": "Sayeret Matkal",
      "description": "An elite special forces unit specializing in
counter-terrorism, reconnaissance, and intelligence gathering.",
      "image":
"https://www.idf.il/media/0l3a50pf/6.jpg?anchor=center&mode=crop&width=278&heig
ht=185"
    },
    {
      "name": "Shayetet 13",
      "description": "The naval commando unit of the IDF, focusing on maritime
operations, counter-terrorism, and sabotage.",
      "image":
"https://www.idf.il/media/vn2eoutk/tal_5860.jpg?crop=0.0502124150158684,0,0.170
92295042614858,0&cropmode=percentage&width=278&height=185"
    },
    {
      "name": "The Shaldag Unit",
      "description": "A special forces unit under the Israeli Air Force,
conducting deep reconnaissance and air force-related missions.",
      "image":
"https://www.idf.il/media/czib444x/50836.jpg?crop=0.22114676902378208,0,0,0&cro
pmode=percentage&width=278&height=185"
    },
    {
      "name": "Unit 669",
```

```
      "description": "The IDF's elite combat search and rescue unit,
responsible for extracting wounded soldiers and civilians from combat zones.",
      "image":
"https://www.idf.il/media/r5vgmohl/669-אודות-יחידה.jpg?crop=0.01939403019522878
8,0,0.20174133524678819,0&cropmode=percentage&width=278&height=185"
    }
  ],
  "Special Forces": [
    {
      "name": "Alpinist Unit",
      "description": "Specializes in mountain warfare and operations in snowy
terrains.",
      "image":
"https://www.idf.il/media/3kulgkda/אלפיניסטים-min.jpg?crop=0.055815757710530149
,0,0.16531960773148682,0&cropmode=percentage&width=278&height=185"
    },
    {
      "name": "Oketz Unit",
      "description": "The IDF's canine unit, trained for attack, detection, and
search and rescue missions.",
      "image":
"https://www.idf.il/media/agfdthn1/_dsc8783-min.jpg?anchor=center&mode=crop&wid
th=278&height=185"
    },
    {
      "name": "Yahalom Unit",
      "description": "An elite combat engineering unit specializing in
explosive ordnance disposal, tunnel warfare, and special engineering tasks.",
      "image":
"https://www.idf.il/media/nvhb0ktb/עותק-2יהלם.jpg?crop=0.27053136508493658,0.00
0000000000000076815430893,0.17847764392407253,0.000000000000000153630861786&cro
pmode=percentage&width=278&height=185"
    },
    {
      "name": "Sky Rider Unit",
      "description": "Operates tactical unmanned aerial vehicles (UAVs) for
reconnaissance and intelligence gathering.",
      "image": ""
    }
  ],
  "Commando Units": [
    {
      "name": "Duvdevan Unit",
```

```
      "description": "An elite unit specializing in undercover operations,
counter-terrorism, and targeted assassinations within urban areas.",
      "image":
"https://www.idf.il/media/mrkgardp/עותק-של-_yhv0563.jpg?crop=0.1678874929930818
9,0,0.053259276030700332,0&cropmode=percentage&width=278&height=185"
    },
    {
      "name": "Maglan Unit",
      "description": "Conducts deep reconnaissance and special operations
behind enemy lines, often using advanced weaponry and tactics.",
      "image":
"https://www.idf.il/media/2vkpnkla/עותק-של-dsc_0057.jpg?crop=0,0,0.221135365442
017,0&cropmode=percentage&width=278&height=185"
    },
    {
      "name": "Egoz Unit",
      "description": "Specializes in guerrilla warfare, anti-guerrilla warfare,
and complex reconnaissance missions, primarily in northern Israel.",
      "image":
"https://www.idf.il/media/bsbcyx21/אגוז-תרחט-קומנדו.jpg?crop=0.2099076818137282
4,0,0.011227683628288751,0&cropmode=percentage&width=278&height=185"
    }
  ],
  "Combat Brigades": [
    {
      "name": "Golani Brigade",
      "description": "An infantry brigade known for its toughness and
participation in major conflicts since its inception.",
      "image":
"https://www.idf.il/media/zeedcin2/סיום-תרח-ט.jpg?crop=0.13607646329053646,0,0.
085070305733245727,0&cropmode=percentage&width=278&height=185"
    },
    {
      "name": "Nahal Brigade",
      "description": "Combines military service with social and agricultural
development, focusing on infantry combat.",
      "image":
"https://www.idf.il/media/uhce0gbk/4-הגולן-ברמת-נחל-932-תרגד.jpg?crop=0.1538793
4135681897,0,0.067267427666963239,0&cropmode=percentage&width=278&height=185"
    },
    {
      "name": "Givati Brigade",
      "description": "An infantry brigade specializing in urban warfare and
operations in the southern regions of Israel.",
```

```
      "image":
"https://www.idf.il/media/jg2jalvr/copy-of-_yhv2693.jpg?crop=0.0894418438300096
,0,0.13170492519377264,0&cropmode=percentage&width=278&height=185"
    },
    {
      "name": "Paratroopers Brigade",
      "description": "An elite airborne infantry brigade trained for parachute
operations and rapid deployment.",
      "image":
"https://www.idf.il/media/anjoxaz0/באנר-שחזור-מתלה.jpg?crop=0.11185488644803025
,0,0.10929188257575198,0&cropmode=percentage&width=278&height=185"
    },
    {
      "name": "Kfir Brigade",
      "description": "Specializes in counter-terrorism and urban warfare,
primarily operating in the West Bank.",
      "image":
"https://www.idf.il/media/rhbdz2le/באנר-כפיר-הכירו.jpg?crop=0.11185488644803025
,0,0.10929188257575198,0&cropmode=percentage&width=278&height=185"
    },
    {
      "name": "Search And Rescue Brigade",
      "description": "Focuses on search and rescue operations during both
military and civilian emergencies.",
      "image":
"https://www.idf.il/media/tvzhxpc1/1-עותק-של.jpg?crop=0.14547445037506124,0,0.0
75672318648721,0&cropmode=percentage&width=278&height=185"
    }
  ]
}
```

שלב 3: יצירת שרת Express ב-TypeScript לטעינת JSON לזיכרון

בשלב זה ניצור קובץ `server.ts` שבו נגדיר את השרת. הנתונים ייטענו לזיכרון בעת עליית השרת ויהפכו לזמינים עבור כל בקשת API.

1. צרו תיקייה בשם `src` ובתוכה קובץ `server.ts`.
2. כתבו את הקוד הבא, תוך השמטת חלקים להשלמה.

דוגמה חלקית (`src/server.ts`):

JavaScript

```javascript
import express, { Request, Response } from 'express';
import fs from 'fs';
import path from 'path';

const app = express();
const PORT = 3000;

interface Unit {
    name: string;
    description: string;
    image: string;
}

interface UnitsData {
    [category: string]: Unit[];
}

// משתנה לנתונים שמאוחסנים בזיכרון
let unitsData: UnitsData | null = null;

// בעת עליית השרת טוען את הנתונים מה-JSON
fs.readFile(path.join(__dirname, '../idfUnits.json'), 'utf8', (err, data) => {
    if (err) {
        console.error('Error loading data:', err);
        return;
    }
    unitsData = JSON.parse(data);
    console.log('Data loaded successfully');
});

app.get('/api/units', (req: Request, res: Response) => {
    // קיים והחזירו תגובה מתאימה unitsData רמז: בדקו אם
    if (!unitsData) {
        return res.status(/* השלימו את סטטוס הקוד */).send(/* הוסיפו הודעת
שגיאה */);
    }
    res.json(unitsData);
});

// הפעלת השרת
app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

שלב 4: שימוש ב-React Context לטעינת הנתונים עם axios

בשלב זה נגדיר Context באפליקציה ב-React לטעינת הנתונים ב-axios, כך שהנתונים יהיו זמינים לכל רכיב באפליקציה.

התקינו את axios:

```
JavaScript
npm install axios
```

1.
2. צרו תיקייה חדשה בשם `context` עם קובץ `DeploymentContext.tsx`.
3. כתבו את הקוד הבא, תוך השמטת חלקים להשלמה.

דוגמה חלקית (`DeploymentContext.tsx`):

```
JavaScript
import React, { createContext, useState, useEffect, ReactNode } from 'react';
import axios from 'axios';

interface Unit {
    name: string;
    description: string;
    image: string;
}

interface UnitsData {
    [category: string]: Unit[];
}

interface DeploymentContextType {
    units: UnitsData | null;
}
```

```
const DeploymentContext = createContext<DeploymentContextType |
undefined>(undefined);

export const DeploymentProvider: React.FC<{ children: ReactNode }> = ({
children }) => {
    const [units, setUnits] = useState<UnitsData | null>(null);

    useEffect(() => {
        const fetchUnits = async () => {
            try {
                const response = await axios.get(/* הכניסו כאן את כתובת ה-API
*/);
                // עם המידע שהתקבל units רמז: עדכנו את מצב
                setUnits(/* עדכנו את מצב units */);
            } catch (error) {
                console.error("Error fetching units:", error);
            }
        };
        fetchUnits();
    }, []); // רמז: השתמשו במערך ריק כדי לוודא שהבקשה מתבצעת פעם אחת בלבד

    return (
        <DeploymentContext.Provider value={{ units }}>
            {children}
        </DeploymentContext.Provider>
    );
};

export const useDeployment = () => {
    const context = React.useContext(DeploymentContext);
    if (!context) {
        throw new Error("useDeployment must be used within a
DeploymentProvider");
    }
    return context;
};
```

שלב 5: שימוש ב-Context ב-Routes של האפליקציה

כעת נשתמש ב-`DeploymentProvider` כדי לספק את הנתונים לכל הרכיבים.

1. ערכו את הקובץ `App.tsx` לשימוש ב-`DeploymentProvider`.

דוגמה חלקית (`App.tsx`):

```javascript
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { DeploymentProvider } from './context/DeploymentContext';
import Home from './pages/Home';
import UnitsPage from './pages/UnitsPage';

const App: React.FC = () => {
    return (
        <DeploymentProvider>
            <Router>
                {/* שונים עבור הדפים Routes רמז: הוסיפו כאן */}
            </Router>
        </DeploymentProvider>
    );
};

export default App;
```

שלב 6: שימוש בנתונים בדף `UnitsPage`

בשלב זה נטען את הנתונים מה-Context לתוך רכיב יחידות.

דוגמה חלקית (`UnitsPage.tsx`):

```javascript
import React from 'react';
import { useDeployment } from '../context/DeploymentContext';

const UnitsPage: React.FC = () => {
    const { units } = useDeployment();

    return (
        <div>
```

```
            <h2>רשימת יחידות</h2>
            {units && Object.keys(units).map(category => (
                <div key={category}>
                    <h3>{category}</h3>
                    <ul>
                        {/* רמז: מיפוי של כל יחידה לפי קטגוריה */}
                        {units[category].map(unit => (
                            <li key={unit.name}>
                                {/* רמז: הצגת שם היחידה, תיאור, ותמונה אם קיימת
*/}
                            </li>
                        ))}
                    </ul>
                </div>
            ))}
        </div>
    );
};

export default UnitsPage;
```

הסבר על שימוש ב-`useEffect` לטעינת נתונים

השימוש ב-`useEffect` עם מערך ריק כפרמטר שני מבטיח שהבקשה תתבצע פעם אחת בלבד בעת טעינת הרכיב, מה שמונע שליחת בקשות מיותרות.

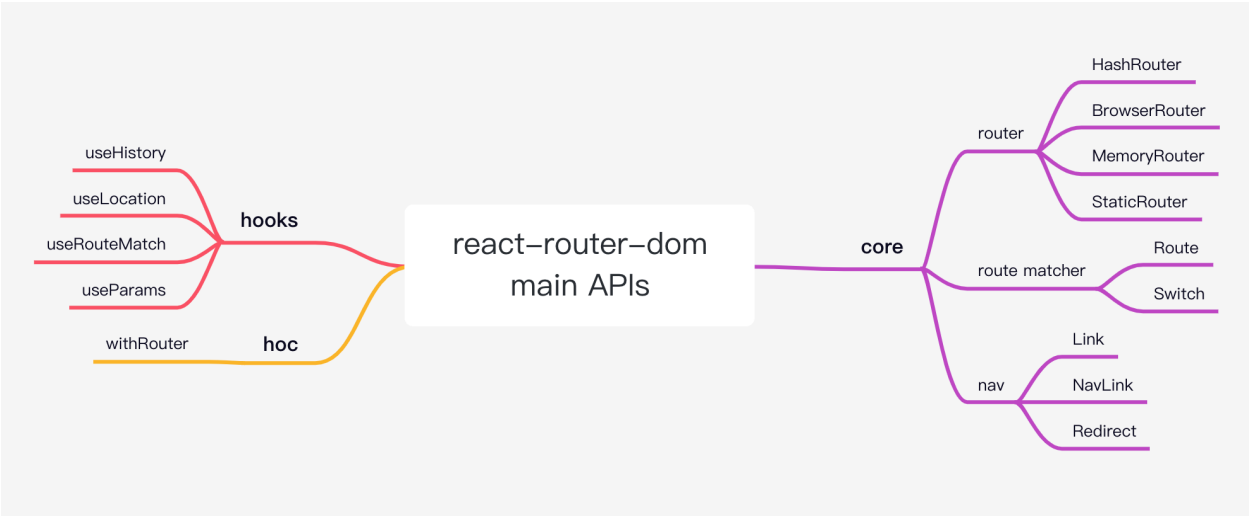
חלק ג: הוספת תמיכה ב-React-Router-Dom

הסבר על react-router-dom לעומת רינדור מותנה (Conditional Rendering)

מה זה `react-router-dom`? `react-router-dom` היא ספרייה לניווט ביישומי רשת שנבנו עם React.
במקום להסתמך על רינדור מותנה (Conditional Rendering) עבור כל דף או מסך באפליקציה,
`react-router-dom` מאפשר לנו להגדיר "מסלולים" (Routes) עבור כל דף. כל מסלול מותאם לכתובת
URL ספציפית, כך שניתן לגשת בקלות לכל דף ישירות דרך הכתובת או באמצעות קישורים פנימיים.

```
import React from 'react';
import {
  BrowserRouter,
  Routes,
  Route,
} from 'react-router-dom';
import Page1 from './pages/page1.js';
import Page2 from './pages/page2.js';
import Page3 from './pages/page3.js';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route index element={<Page1 />}/>
        <Route path="page2" element={<Page2 />}/>
        <Route path="page3/:id" element={<Page3 />}/>
      </Routes>
    </BrowserRouter>
  );
}
```



הבדלים עיקריים בין react-router-dom לרינדור מותנה:

1. **רינדור מותנה**: מראה רכיב מסוים בהתבסס על תנאים מסוימים. למשל, ניתן להשתמש ברינדור מותנה כדי להציג או להסתיר תוכן מסוים כאשר המשתמש מחובר או כאשר משתנה כלשהו משתנה. אך גישה זו אינה יעילה כאשר יש צורך בניווט בין דפים או מצבים שונים באפליקציה מורכבת.
2. **`react-router-dom`**: מאפשר לטעון דפים שונים על סמך הנתיב של הכתובת URL. כאשר משתמש לוחץ על קישור או ניגש לכתובת מסוימת, `react-router-dom` מנתב את המשתמש לדף הנכון ללא צורך בטעינה מחדש של הדף. כך ניתן ליצור חוויית משתמש חלקה יותר וקל יותר לארגון הניווט באפליקציה.

מבנה התיקיות

```JavaScript
IDF-Deployment-Project/
├── public/
├── src/
|   ├── components/
|   |   ├── UnitList.tsx
|   |   ├── UnitStatus.tsx
|   |   ├── ChangeStatus.tsx
|   |   ├── UnitDetails.tsx
|   |   ├── MissionCompleted.tsx
|   |   ├── NavBar.tsx
|   ├── pages/
|   |   ├── Home.tsx
|   |   ├── UnitsPage.tsx
|   |   ├── AboutUnit.tsx
```

```
|   ├── context/
|   |   ├── DeploymentContext.tsx
|   ├── styles/
|   |   ├── UnitList.module.css
|   |   ├── UnitDetails.module.css
|   ├── App.tsx
|   ├── index.tsx
|   └── types/
|       ├── deployment.d.ts
├── package.json
└── README.md
```

שלבים להוספת ניווט עם react-router-dom

שלב 1: התקנת `react-router-dom`

ראשית, התקינו את הספרייה על ידי הפקודה:

```JavaScript
npm install react-router-dom
```

שלב 2: הגדרת רכיב `BrowserRouter` ב-`App.tsx`

1. **הגדרת `BrowserRouter`**:
   - `BrowserRouter` הוא רכיב ברמה עליונה המאפשר ניהול מסלולים (Routes) בתוך האפליקציה.

דוגמה (App.tsx):

```JavaScript
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { DeploymentProvider } from './context/DeploymentContext';
```

```
import Home from './pages/Home';
import UnitsPage from './pages/UnitsPage';
import AboutUnit from './pages/AboutUnit';
import NavBar from './components/NavBar';

const App: React.FC = () => {
    return (
        <DeploymentProvider>
            <Router>
                <NavBar />
                <div style={{ padding: '20px', fontFamily: 'Arial, sans-serif'
}}>
                    <Routes>
                        {/* רמז: הוסיפו כאן מסלולים לכל דף - דף הבית, רשימת
יחידות, דף יחידה */}
                    </Routes>
                </div>
            </Router>
        </DeploymentProvider>
    );
};

export default App;
```

תרגילים לשלב 2

1. **הוסיפו מסלול לדף הבית** עם הנתיב `/`.
   - **רמז**: השתמשו ברכיב `Route` והפנו את הנתיב `/` לדף `Home`.
2. **הוסיפו מסלול לדף רשימת היחידות** עם הנתיב `units/`.
   - **רמז**: הפנו את הנתיב `units/` לרכיב `UnitsPage`.
3. **הוסיפו מסלול לדף יחידה מסוימת** עם הנתיב `units/:unitName/`.
   - **רמז**: השתמשו בפרמטר דינמי `unitName` בנתיב והפנו אותו לדף `AboutUnit`.

שלב 3: יצירת רכיב `NavBar` עם `NavLink`

1. **הסבר על רכיב `NavBar`**:
   - רכיב `NavBar` ישמש לניווט בין הדפים. כל קישור יהיה `NavLink` כך שיתווסף סגנון פעיל לקישור הנוכחי.

דוגמה (NavBar.tsx):

```javascript
import React from 'react';
import { NavLink } from 'react-router-dom';
import './NavBar.module.css';

const NavBar: React.FC = () => {
    return (
        <nav>
            <ul>
                <li>
                    <NavLink to="/" /* רמז: הוסיפו כאן סגנון פעיל */>
                        בית
                    </NavLink>
                </li>
                <li>
                    <NavLink to="/units" /* רמז: הוסיפו כאן סגנון פעיל */>
                        רשימת יחידות
                    </NavLink>
                </li>
            </ul>
        </nav>
    );
};

export default NavBar;
```

תרגילים לשלב 3

1. **הגדירו סגנון פעיל עבור `NavLink`** בדף הבית.
   - **רמז**: השתמשו ב-`activeClassName` או `isActive` להתאמת הסגנון כאשר הנתיב תואם לנתיב הנוכחי.
2. **הוסיפו קישור פעיל ל"רשימת יחידות"** ברכיב `NavBar`.
   - **רמז**: הגדירו את הקישור `NavLink` לנתיב `units/` עם סגנון פעיל.

שלב 4: הגדרת הדפים

1. **דף הבית**: הצגת תיאור קצר של האפליקציה.
2. **רשימת יחידות**: הצגת רשימת יחידות כמו `UnitList`, המוביל לכל יחידה.
3. **דף יחידה בודדת**: מציג מידע מפורט על יחידה מסוימת, בהתאם לפרמטר `unitName`.

רכיב `Home` - דף הבית

דוגמה (`Home.tsx`):

```javascript
import React from 'react';

const Home: React.FC = () => {
    return (
        <div>
            <h1>ברוכים הבאים למעקב פריסת יחידות צה"ל</h1>
            <p>כאן תוכלו לעקוב אחר מצב הפריסה של יחידות צה"ל השונות.</p>
        </div>
    );
};

export default Home;
```

רכיב `UnitsPage` - רשימת יחידות

דוגמה (`UnitsPage.tsx`):

```javascript
import React from 'react';
import UnitList from '../components/UnitList';

const UnitsPage: React.FC = () => {
    return (
        <div>
            <h2>רשימת יחידות</h2>
            {/* הוסיפו את רכיב `UnitList` :רמז */}
        </div>
    );
};
```

```
export default UnitsPage;
```

רכיב `AboutUnit` - דף יחידה בודדת

דוגמה (`AboutUnit.tsx`):

```JavaScript
import React from 'react';
import { useParams } from 'react-router-dom';
import UnitStatus from '../components/UnitStatus';

const AboutUnit: React.FC = () => {
    const { unitName } = useParams<{ unitName: string }>();

    return (
        <div>
            <h2>פרטים על יחידת {unitName}</h2>
            {/* להצגת מידע על היחידה `UnitStatus` רמז: השתמשו ברכיב */}
        </div>
    );
};

export default AboutUnit;
```

עדכונים לרכיבים עם `Link` לדף יחידה בודדת

כדי לאפשר למשתמש לעבור לדף יחידה מסוימת מתוך רשימת היחידות, נוסיף קישורי `Link` ברכיב `UnitList`.

דוגמה (`UnitList.tsx`):

```javascript
import React, { useContext } from 'react';
import { DeploymentContext } from '../context/DeploymentContext';
import { Link } from 'react-router-dom';

const UnitList: React.FC = () => {
    const deploymentContext = useContext(DeploymentContext);

    if (!deploymentContext) {
        throw new Error("UnitList must be used within a DeploymentProvider");
    }

    const { units } = deploymentContext;

    return (
        <div>
            <h2>רשימת יחידות</h2>
            {Object.keys(units).map((unit) => (
                <div key={unit}>
                    <Link to={`/units/${unit}`}>
                        {unit}
                    </Link>
                </div>
            ))}
        </div>
    );
};

export default UnitList;
```

תרגילים לרכיב `UnitList`

1. **הוסיפו `Link` לדף יחידה** לכל שם יחידה.
   - **רמז**: השתמשו בנתיב `/units/{unitName}` עבור כל יחידה.
2. **עצבו את הקישורים** כך שיבליטו את שם היחידה.
   - **רמז**: הוסיפו סגנונות לקישורים ברשימה באמצעות CSS מותאם.

חלק ד: תרגילים להרחבה לשרת

תרגילי הרחבה בצד השרת

1. **הוסיפו נקודת קצה חדשה** לקבלת יחידה ספציפית לפי שם:
   - **רמז**: השתמשו בפרמטר דינמי `name:` כדי לסנן את היחידות.
2. **אפשרו סינון לפי סוג היחידה**:
   - **רמז**: הוסיפו שאילתה (query parameter) עבור סוג היחידה וסננו את התוצאה.
3.
4.
   - **רמז**: התקינו `cors` ושילבו אותו בשרת.
5. **הוסיפו לוגיקה לחיפוש מילות מפתח בתיאור היחידה**:
   - **רמז**: חפשו במחרוזת `description` בכל יחידה והתאימו לפי מילות מפתח מהמשתמש.
6. **הוסיפו מונה בקשות** לכל בקשת API:
   - **רמז**: השתמשו במשתנה גלובלי לעקוב אחר מספר הבקשות לכל נתיב.

חלק ה: תרגילים להרחבה של react-router-dom

תרגילים נוספים להרחבה עם `react-router-dom`

1. **הוסיפו קישור חזרה לדף הבית** בכל דף.
   - **רמז**: השתמשו ב-`Link` עם נתיב `/` בכל דף משנה לחזרה.
2. **השתמשו ב`NavLink` עבור הקישורים הראשיים** כדי להדגיש את הקישור הפעיל בדף הנוכחי.
   - **רמז**: השתמשו ב-`activeClassName` כדי להחיל סגנון לקישור כאשר הוא בדף הנוכחי.
3. **הוסיפו מסך 404** שיוצג כאשר הנתיב לא נמצא.
   - **רמז**: הוסיפו מסלול `*` עם רכיב שמציג הודעת שגיאה כאשר אין התאמה לנתיב.
4. **אפשרו ניווט דרך ההיסטוריה בדפדפן**.
   - **רמז**: השתמשו ב-`useHistory` או `useNavigate` לניווט בין דפים באופן פרוגרמטי.
5. **עצבו את הניווט כך שייראה ככפתורי טאב**.
   - **רמז**: השתמשו ב-`NavLink` עם סגנונות ייחודיים לכפתורי טאב לניווט בין הדפים.

חלק ו: תרגילים נוספים להרחבה

להלן 20 תרגילים נוספים להרחבת היכולות של הפרויקט, עם רמזים להכוונה:

1. **הוסיפו שורת חיפוש** לסינון יחידות לפי שם.
   - **רמז**: הגדירו משתנה מצב שיאחסן את ערך החיפוש והשתמשו בו לסינון היחידות המוצגות.
2. **מיינו את היחידות** לפי שם או מצב.
   - **רמז**: הוסיפו לחצני מיון ושנו את הסדר בהתאם.
3. **הציגו יחידות הנמצאות במשימה** בפאנל צדדי.
   - **רמז**: הציגו רק יחידות במצב "במשימה".
4. **אפשרו שינוי מצב יחידות קבוצתיות** בלחיצת כפתור.

- **רמז**: השתמשו בצ'קבוקסים למיון והחליפו מצבים קבוצתיים.

5. **הוסיפו מצב כהה** לאפליקציה.
   - **רמז**: הוסיפו משתנה מצב למצב כהה ושנו את הצבעים בהתאם.
6. **יומן שינויים** עם תיעוד שינויים במצבי היחידות.
   - **רמז**: הוסיפו מערך ותעדו שינויים ותאריכים.
7. **הוסיפו פרטי יחידה נוספים** כמו מיקום וסוג משימה.
   - **רמז**: הוסיפו שדות נוספים לאובייקט היחידה והציגו אותם.
8. **מפתח צבעים** להמחשת מצבים שונים.
   - **רמז**: השתמשו בצבעים שונים עבור כל מצב בפריסה.
9. **הוסיפו תיאור זמן עדכון** לכל יחידה.
   - **רמז**: הוסיפו משתנה עבור זמן עדכון.
10. **עדכון אוטומטי למצב יחידות** כל פרק זמן קבוע.
    - **רמז**: השתמשו במנגנון `interval` לקביעת עדכונים.
11. **הוסיפו תרשים עוגה** להצגת אחוזי היחידות.
    - **רמז**: הציגו אחוזים לכל מצב במבנה עוגה.
12. **מערכת הודעות** לשינויי מצבים.
    - **רמז**: הציגו הודעות קופצות עבור שינויים במצבים.
13. **אפשרות חזרה לשינוי האחרון**.
    - **רמז**: השתמשו במערך כדי לשמור היסטוריה קצרה.
14. **מסך משימות שהושלמו**.
    - **רמז**: מסך נפרד שמציג יחידות במצב "הושלם".
15. **גרירה ושחרור** לשינוי סדר היחידות.
    - **רמז**: הוסיפו פונקציונליות לגרירה ושחרור לרשימת היחידות.
16. **התאמה אישית למידע מוצג**.
    - **רמז**: אפשרו למשתמש לבחור אילו שדות להציג.
17. **תצוגת מסך מלא** של האפליקציה.
    - **רמז**: התאימו את כל הרכיבים כך שיתאימו למסך מלא.
18. **אפקטים קוליים לפעולות שונות**.
    - **רמז**: השמיעו צליל בעת ביצוע כל פעולה.
19. **סרגל סטטוס תחתון** עם נתוני מצב כלליים.
    - **רמז**: הציגו את המידע לפי אחוזים ומספרי יחידות.
20. **השוואת מצב יחידות לפי תאריכים**.
    - **רמז**: השתמשו במבנה נתונים למעקב השוואתי לפי תאריכים.