**שיטה אחרת נהוגה בשוק:**

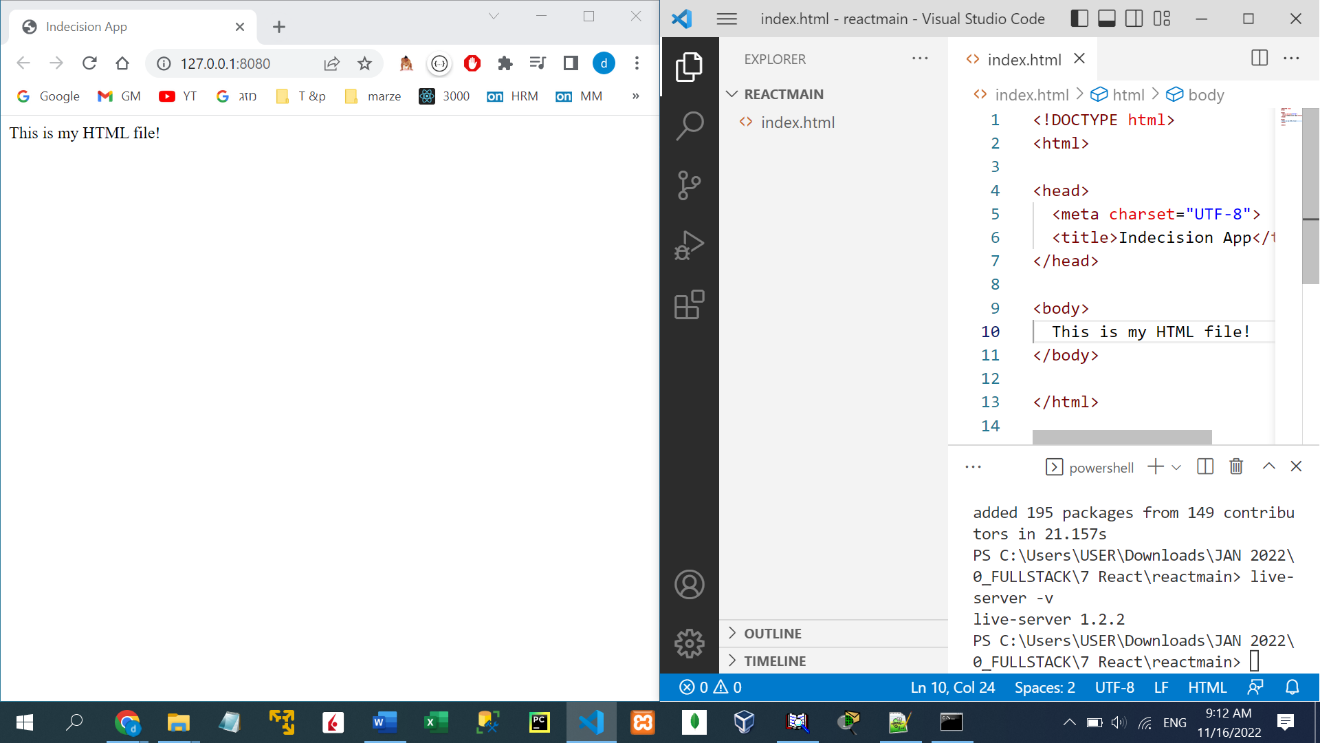
**רינדור אונליין**

**פרק 1 – הכנת הסביבה**

1. נגדיר ספריית עבודה (אצלי: reactmain)
2. נפתח **command** ונגיע לספרייה זו
3. נפתח VSC מהספרייה הזו ונפתח כאן טרמינל
4. מריצים מהטרמינל את הפקודה npm install -g yarn   
   (כלי המנהל את החבילה מאחורי הקלעים)
5. בדיקת תקינות: מריצים yarn --version (שימו לב – פעמיים מינוס)
   1. אם יש הודעת שגיאה של SecurityError מבצעים את הבא:
      1. מחפשים במחשב את PowerShell
      2. מריצים אותו כ- Administrator
      3. כותבים את הפקודה Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
6. מריצים מהטרמינל את npm install -g live-server (הדבר מאפשר לעשות עדכונים מידיים – כפי שאנו מכירים)
7. בדיקה על ידי live-server -v כרגיל
8. אם בהמשך משהו לא תקין – עושים כיבוי והדלקה למחשב
9. פותחים CMD ומגיעים לספריית העבודה של ריאקט  
   Text

   Description automatically generated

**פרק 2 – סביבת העבודה**

1. שמים את קובץ ה- html המאוד-פשוט ששמתי במודל - בתוך הספרייה הנכונה (כמובן דאגנו ש- VSC כבר פתוח)
2. מריצים מחלון ה- CMD את הפקודה live-server ורואים שנפתח חלון דפדפן עם פריסת הטקסט. (כדאי ליצור קיצור דרך ל- cmd מהספרייה!)
3. נשים לב ל- URL שממנו האתר רץ
4. נשים לב שבכל שינוי ב- HTML שנרשום – האתר מתעדכן
5. נשים לב שהשורש של השרת הוא המיקום של חלון ה- CMD.
6. **תרגיל כיתתי**: הוסיפו ספרייה (למשל public ), העבירו את קובץ ה- HTML לשם והריצו את ה- live-server בהתאמה הנדרשת. (תוך כדי התהליך כדאי לעיין בהורדעות המופיעות על המסך
7. נכבה את ה- live-server כי נחליף קבצים בספריית העבודה

**פרק 3 – שלב ביניים להבנת הרקע**

1. בקובץ ה- index.html הוספנו ארבע שורות:
   1. שורת div עם id מתאים שקובץ ה- js העתידי "ייתפס אליו"
   2. הפניה לספריית און-ליין (ובגלל זה אין צורך בהתקנת כל ספריית ה- node-modules כמו שהיה עד כה, אלא רק חלק ממנה – בהמשך ההרצאה) המכילה קלאסים עם ריאקט מתאים
   3. כנ"ל שורה לגבי reactDOM כדי שיעבוד תקין תחת דפדפן (ולא למשל: אנדרואיד וכו)
   4. הפנייה לקובץ JS שהוא יריץ את האפליקציה שלנו. למען הבהירות השים אותו כמובן בתוך ספרייה משל עצמה.  
      **יותר לא ניגע בקובץ HTML זה!**
2. פתחו ספרייה בשם src ותשימו שם את app.js מהמודל. נפתח את קובץ ה- app.js.   
   זהו שלב ביניים!!!! בהמשך ניפרד משלב זה – רק להבין את הרקע:נעשה כאן שימוש בספריות הנ"ל, ואנו רואים שיש כאן הרבה עבודה עבור תוצאה קטנה..... בפרק הבא נפתור את זה!  
   - התחביר הוא JS טהור  
   - מתחברים ל- id של קובץ ה- HTML שנקרא app  
   - מריצים שם את ה- JS של קובץ זה
3. נריץ בחלון ה- CMD את ה- live-server public (שימו לב - הפנייה היא לקובץ ה- html ולא לקובץ ה- js – בשונה ממה שראינו עד כה).

נרצה לכתוב קוד פשוט, בשפת JSX, שהיא הרבה יותר נוחה – ערבוב של HTML ו- JS,

כתיבת ה- JS הטהור מקשה על הפיתוח

אנו זקוקים לשירותי תרגום... בשביל זה יש לנו את **Babel**

**פרק 4 – יישום react עם תרגום Babel**

1. **הכנת הסביבה:** נריץ בטרמינל של VSC את הפקודה:  
   npm install -g babel-cli@6.24.1
2. את בדיקת התקינות נוכל להריץ מחלון ה- CMD ונרשום babel -- help (פעמיים מינוס) ונראה המון פקודות, ובעיקר רואים שההתקנה תקינה
3. נרצה להקים את ה- package.json המוכר שלנו, והפעם נעשה את זה מחלון ה- CMD ונרשום yarn init, ונענה עם enter על כל השאלות תשובה ריקה
4. נוודא שאכן התווסף לנו הקובץ הנ"ל
5. מאותו חלון זה נעשה גם את ההתקנות הדרושות של babel ונרשום:  
   yarn add babel-preset-react@6.24.1 babel-preset-env@1.5.2
6. נראה את ההשפעה של פעולה זו:
   * הותקנה הספרייה המוכרת של node-modules (עם בערך שמינית ממספר הקבצים שהיו לנו עד כה)
   * נראה את התוספת לקובץ package.json. הותקנו dependencies – דהיינו שהפרוייקט יודע לאן לפנות כדי להריץ באופן תקין.
   * התווסף קובץ yarn.lock, דומה בתפקידו ל package.json המוכר לנו. אין לגעת בקובץ זה, הוא יתעדכן אוטומטית.
7. כעת ניצור ספרייה בשם src (אח של ספריית public) ושם ניצור קבוץ **חדש** בשם app.js. בו נכתוב קוד, כמו שאנו רגילים. הקובץ app.js **שכבר היה** **קיים**, יתעדכן אוטומטית על יד שירותי התרגום של Babel.

**בשיעור** נעתיק את כל כל תוכן ספרייה 4 אל תוך reactmain (בספרייה שיצרתם להרצת הפרוייקט) תוך דריסה של התוכן האחר. כמובן ש- node-modules נשאיר כמות שהוא!

**מה שנעשה כעת ובהמשך הוא**

* **כתיבת JSX בקובץ src\app.js**
* **על ידי פעולה של babel הקובץ שלנו יתורגם לשפה מובנת לדפדפן. קוד זה יימצא ב- public\scripts\app.js אבל זה לא הכי מעניין אותנו....**
* **נותר לנו לראות איך הדפדפן מציג את תוצאות השינוי.**

1. נריץ ב- CMD את הפקודה:  
   babel src\app.js --out-file=public/scripts/app.js --presets=env,react   
   (שימו לב - פעמיים שני מינוסים)
   * מה שכתוב פה הוא:
     1. תפעיל את המתרגם על הקובץ הראשון,
     2. תפיק פלט של התרגום אל תוך הקובץ השני,
     3. ותשתמש בהגדרות-תרגום הבאות (במקרה זה: env, react)
   * **הערה**: הפקודות ארוכות, אבל ניתן להשתמש כמובן בחיצים כדי לחזור לפקודות קודמות, ועוד מעט תראו פתרון יותר פשוט
2. נוודא שכתבנו נכון ושאין הודעות שגיאה.
3. ניתן לוודא שאכן הקובץ public\scripts\app.js נכתב ברגע זה
4. נוסיף בהרצת babel שבחלון ה- CMD את –watch ונריץ. נראה שהפקודה **לא** מסתיימת.
5. מה שזה עושה שהכל נהיה live, דהיינו אם נעשה שינוי ב- app.js שבספריית src ונשמור, נראה מיד שינוי בהתאם ב- public\scripts\app.js. , וכן בדפדפן. תנסו את זה....
6. בשיעורים - נכתוב במקום אחר תוכן של app.js ונשים אותו במקום הנכון, ונשמור

**כדי להקל עליכם הכנתי לכם שני קבצי bat שעושים לכם את כל ההרצות ברקע. שימו אותם בספריית העבודה שלכם והריצו אותם בזה אחר זה**

**פרק 5 – תרגול ראשוני**

נראה את התוכן. הריצו אותו.

**משימה כיתתית**: כיתבו משהו (רעיון...: קורות חיים !! -😉)