

ללמוד דיאקט בעברית

רן בר-זיק



הקריה האקדמית אונו
Ono Academic College
החוג למדעי המחשב

ללמוד ריאקט בעברית

רן בר-זיק

מהדורה: 2.0.0



Really Good



elementor

HoneyBook

כל הזכויות שמורות © רן בר-זיק, 2021.

ספר זה הוא יצירה המוגנת בזכויות יוצרים. אתה קיבלת רישיון לא-בלעדי, לא-ייחודי, אישי, בלתי ניתן להעברה (למעט על פי דין), ובלתי ניתן להסבה לעשות שימוש אישי בספר זה לצרכים לימודיים בלבד.

אסור לך להעתיק את הספר, לשכפל אותו, ליצור יצירות נגזרות ממנו או לפרסם אותו בכל צורה אחרת.

מותר לך לצטט קטעים קצרים מהספר במסגרת הגנת שימוש הוגן, כלומר פסקה או שתיים, כאשר אתה מפנה למקור ומזכיר את רן בר-זיק כמחבר הספר.

הדוגמאות המובאות בספר זה הן בבעלות של רן בר-זיק, ואסור לך להשתמש בהן בתוך תוכנות שתפתח. אם אתה רוצה להכניס אותן לפרויקט שלך, שלח מייל ונדבר על זה.

עריכה לשונית: יעל ניר

הגהה: חנן קפלן

עיצוב הספר והכריכה: טל סולומון ורדי (tsv.co.il)

הפקה: כריכה – סוכנות לסופרים

www.kricha.co.il



תוכן העניינים

10	על "ללמוד ריאקט בעברית"
11	על המחבר
12	על העורכים הטכניים
12	דורון זבלבסקי
12	גיל פינק
13	דורון קילדי
14	על החברות התומכות
14	Really Good
14	אלמנטור
16	HoneyBook
17	על ריאקט
19	דרך הלימוד
19	על המונחים בעברית
20	סביבת העבודה הבסיסית
23	סביבת עבודה בסיסית ב-codepen
26	אפליקציית ריאקט
32	בניית קומפוננט פונקציה בסביבת העבודה הבסיסית
40	בנייה של סביבת ריאקט אמיתית ומורכבת יותר באמצעות Create React App
41	התקנת Node.js על המחשב שלכם
41	התקנה על חלונות
44	התקנה על מק
45	התקנה על לינוקס
45	עבודה עם טרמינל
45	הפעלת הטרמינל
47	ניווט בטרמינל
48	מציאת מיקומים בטרמינל דרך חלונות
49	טרמינל ב-Visual Studio Code
49	בדיקת גרסת Node.js דרך הטרמינל
50	עבודה עם Create React App
55	קשיים ותקלות

57	כתיבת קומפוננטה ראשונה ב-React App Create
67	Export\ Import
71	ייבוא מנתיבים אחרים
72	אין חובה להשתמש בסיומת הקובץ ts
72	ייצוא של כמה משתנים
73	ייבוא של תמונות, CSS ומשאבים אחרים
76	JSX
83	רשימות ב-JSX
91	קומפוננטה עם תכונות (props)
105	דיבאג
110	סטייט
112	ריאקט ורנדור
113	סטייט
123	תכנון מבנה הקומפוננטות
129	אירועים ועדכון קומפוננטות
130	אירועי DOM
131	אירועים סינתטיים/ ריאקטיים
152	אלמנט fragment
156	useEffects
162	קומפוננטה ללא שם
165	עיצוב קומפוננטות
167	CSS בסיסי
168	סלקטור
170	תכונות
174	קלאס ריאקטי
184	מעגל החיים בקלאס
184	componentDidMount
184	componentDidUpdate (prevProps, prevState, snapshot)
184	componentWillUnmount
184	shouldComponentUpdate (nextProps, nextState)
185	getDerivedStateFromError (error)
185	componentDidCatch (error, info)

195.....	שימוש בקומפוננטות ממקורות אחרים
200.....	ייבוא כמה קומפוננטות
202.....	מודולים חסרים
215.....	<i>HOC: Higher Order Component</i>
228.....	ראוטינג
251.....	קונטקסט
266.....	חיבור לשרת עם סרוויסים
274.....	מבוא לבדיקות עם <i>jest</i>
276.....	חלק ראשון – <i>import</i>
276.....	חלק שני – כתיבת מסגרת הבדיקה
276.....	חלק שלישי – הרצת הקומפוננטה
276.....	חלק רביעי – הבדיקה
279.....	בדיקות לקומפוננטה עם <i>props</i>
280.....	פונקציית בדיקה – <i>assert</i>
280.....	<i>not</i>
280.....	<i>toBeTruthy</i>
281.....	<i>toBeFalsy</i>
281.....	<i>toContain</i>
286.....	יצירת בילד והעלאת האפליקציה לסביבה חיה
292.....	סיום – ומה עכשיו?
293.....	התחברות לקהילת הפיתוח
293.....	מפגשים ומיטאפים
293.....	האתר <i>Stackoverflow</i>
294.....	תרומת קוד בגיטהאב
295.....	נספח: <i>PropType</i>
295.....	התקנת <i>PropTypes</i>
296.....	השימוש ב- <i>PropTypes</i>
300.....	ערכים שאפשר לקבוע
302.....	נספח: שינויים מהמהדורה הקודמת (מהדורה 1.1.1)
302.....	נספח: שינויים ממהדורה 1.0.0

על "ללמוד ריאקט בעברית"

ריאקט היא ספרייה פופולרית מאוד לפיתוח אתרי אינטרנט, אפליקציות ווב, ובכלל כל סוג של ממשק אינטרנטי. באמצעות ריאקט והאקוסיסטם הנרחב והעשיר שלה, כולל קומפוננטות רבות המלוות אותה, כל מתכנת יכול ליצור אתר או אפליקציה מורכבים במהירות רבה ובאיכות יוצאת דופן. לריאקט יש גם מעטפת (כמו אלקטרון או קורדובה) המאפשרת לממשק שפותח לזרום להיות מותאם בקלות גם לתוכנות מחשב של ממש. העושר של האקוסיסטם הפיתוחי של ריאקט וחוזק הקהילה שלה בארץ ובעולם מאפשרים לקבל גם המון-המון מידע וסיוע בכל תקלה ובעיה. אם מתכנת נתקל בבעיה בפיתוח, סביר להניח שהוא ימצא סיוע ומידע בפורומים ובקבוצות מתכנתים. יש גם שפע של מיטאפים וקבוצות דיון המוקדשים למתכנתים העובדים בריאקט ונוסף על כך, הביקוש למתכנתים המכירים את ריאקט הוא גבוה. כל הדברים האלו הופכים את ריאקט לאידיאלית לכניסה לעולם פיתוח צד הלקוח.

הספר מניח ידע מקיף בג'אווהסקריפט, טייפסקריפט וידע בסיסי ב-HTML וב-CSS. אם אינכם מכירים ג'אווהסקריפט וטייפסקריפט, אפשר ללמוד את שני הנושאים בספר "ללמוד ג'אווהסקריפט בעברית". יש שם פרק המסביר על HTML בסיסי ועל CSS בסיסי. מי שסיים לקרוא את הספר ההוא ותרגל כהלכה, אמור להחזיק בידע מספק על מנת להתחיל ללמוד ריאקט. בספר זה נלמד על ריאקט ממש מהבסיס – הכרת מונחים בסיסיים ויצירת סביבת עבודה – ונגיע עד חומרים מתקדמים כמו high order components ו-state hooks ובשפת טייפסקריפט המודרנית. הספר מעודכן לגרסת ריאקט 18.

על המחבר

רן בר-זיק הוא מפתח תוכנה במגוון שפות ופלטפורמות מאז 1996 ועובד כמפתח בכיר במרכזי פיתוח של חברות רב-לאומיות, מ-HPE ועד Verizon, שם הוא מפתח בטכניקות מתקדמות הן בצד הלקוח, הן בצד השרת, ושם דגש על בניית תשתית פיתוח נכונה, על שימוש ב-CI\CD וכמובן על אבטחת מידע.

נוסף על עבודתו כמפתח במשרה מלאה, רן הוא עיתונאי ב"דה מרקר" במדור המחשבים, שם הוא מסקר נושאים הקשורים לטכנולוגיה ולאבטחת מידע וכותב על אינטרנט ורשתות.

משנת 2008 מפעיל רן את האתר "אינטרנט ישראל" (internet-israel.com), שהוא אתר טכני המכיל מדריכים, מאמרים והסברים על תכנות בעברית ומתעדכן לפחות פעם בשבוע.

רן הוא מחבר הספר "ללמוד ג'אווהסקריפט בעברית", "ללמוד Node.js בעברית", "ללמוד MySQL בעברית" והספר "ללמוד פיתוח ווב מעשי בעברית" ומלמד בקריה האקדמית אונו.

רן נשוי ליעל ואב לארבעה ילדים: עומרי, כפיר, דניאל ומיכל. רץ למרחקים ארוכים וחובב טולקין מושבע.

על העורכים הטכניים

דורון זבלבסקי

הקריירה של דורון התחילה דווקא בצד השרת, בחברות אבטחת מידע שבהן עבד כמהנדס תוכנה, מוביל טכני ומנהל מקצועי. את המעבר לעולם הפרונטאנד ביצע בשנת 2014 כאשר הצטרף ל-Applitoools, שם הקים את קבוצת פיתוח הווב, וכיום הוא מוביל אותה. במסגרת חיפושיו וניסיונותו להחיל עקרונות נכונים של הנדסת תוכנה על קוד פרונטאנד הוא התוודע לריאקט ואימץ אותה בחום. דורון הקים את קהילת ריאקט בישראל, כולל קבוצת פייסבוק ומיטאפ פופולרי, ואף יזם את כנס ReactNext הראשון בישראל והיה מפיק משותף שלו. נוסף על כך הוא מרצה על ריאקט, על בדיקות אוטומטיות ועל בניית צוותים מנצחים ותורם מזמנו בשמחה למתחילים המבקשים סיוע בתחום במסגרת מפגשים אישיים וקבוצתיים.

גיל פינק

גיל פינק הוא מומחה לפיתוח מערכות ווב, Web Technologies Google Developer Expert, Microsoft Developer Technologies MVP והמייסד של חברת sparXys. כיום הוא מייעץ לחברות ולארגונים שונים, שם הוא מסייע בפיתוח פתרונות מבוססי אינטרנט ו-SPAs. הוא עורך הרצאות וסדנאות ליחידים ולחברות המעוניינים להתמחות בתשתיות בארכיטקטורה ובפיתוח של מערכות ווב. הוא גם מחבר של כמה קורסים רשמיים של מיקרוסופט (Microsoft Official Course MOC), מחבר משותף של הספר "Pro Single Page Application Development" (Apress) ושותף בארגון הכנס הבינלאומי AngularUP. לפרטים נוספים על גיל: <http://www.gilfink.net>

דורון קילזי

דורון חי ונושם פיתוח לרשת זה כעשור.

את דרכו החל ביחידה טכנולוגית מובחרת בחיל המודיעין, שבה שירת כשש שנים. במהלך שירותו הקים ופיתח מערכות מבצעיות מורכבות והיה אחראי על הטמעה של טכנולוגיות חדשות. בארבע השנים האחרונות דורון מפתח full-stack ב-Verizon Media, אחת מחברות האינטרנט הגדולות בעולם. בימים אלו הוא עוסק, בין היתר, במעבר של החברה מטכנולוגיות ותיקות כגון Angular.js לריאקט ובפיתוח של מערכות מתקדמות בעולם ה-ad tech. הדחף של דורון ללמוד ולהשתפר גורם לו להמשיך להתמקצע ולהתאהב בכל יום מחדש בעולם ה-js.

על החברות התומכות

Really Good

Really Good היא בוטיק פיתוח Front End שעובדת עם סטארטאפים וחברות טכנולוגיה מאז הקמתה ב-2012 על ידי שחר טל ורוני אורבך. אנחנו נהנים לבנות אפליקציות מורכבות עם UX מוקפד במגוון טכנולוגיות ללקוחות מעניינים שחויית המשתמש חשובה להם, ושומרים על איזון בריא בין עבודה לחיים.

אנחנו מגייסים מפתחי Front End מנוסים וממש טובים עם תשומת לב לפרטים הקטנים.

ReallyGood.co.il

אלמנטור

אלמנטור מפתחת פלטפורמת קוד פתוח לבניית אתרים שמשנה את הדרך בה בונים אתרי אינטרנט בשוק המקצועי. אלמנטור מעניק למעצבים את החופש ליצור עמודי אינטרנט ללא צורך בקוד ולמפתחים את החירות לדחוף את הגבולות, לרענן ולהרחיב את המערכת בצורה קלה ומהירה באמצעות API ידידותי למפתחים, ובכך לחסוך זמן פיתוח ולהיות יעילים ורווחיים.

עם מיליוני אתרים הפעילים על אלמנטור וצמיחה חודשית מדהימה, התגבשה סביב הפלטפורמה קהילה חזקה המונה מאות אלפי חברים, מפתחים, משווקים ומעצבים, המקיימים מיטאפים בכל רחבי העולם. מידי יום האלמנטוריסטים מייצרים וצורכים אלפי שעות של הדרכות, סרטי השראה ובלוגים מעמיקים, ומפתחים תורמים קוד ורעיונות באמצעות GitHub. האקוסיסטם המקצועי של אלמנטור מתפתח ללא הפסקה והוא אוצר המוסיף ומעשיר את היכולות של כל יוצר אינטרנט.

באלמנטור אנחנו משתמשים בטכנולוגיות קוד פתוח מתקדמות לפיתוח כלי אינטרנט חדשניים ומהירים. אם גם אתם רוצים להיות חלק מהטכנולוגיה שמשנה את חווית האינטרנט בעולם ויש לכם את הידע כדי לבנות עולם יפה יותר אנחנו מחפשים אתכם, מעצבי UX&UI, מפתחי Full

Stack, מהנדסי Big Data ו DevOps עם מומחיות בניהול Kubernetes על פלטפורמות הענן של AWS & GCP.

אתר החברה: [/https://elementor.com](https://elementor.com)

עמוד המשרות: <https://careers.elementor.com/>

HoneyBook

חברת HoneyBook מפתחת פלטפורמה לניהול פיננסי ועסקי עבור עצמאיים ועסקים קטנים. החברה מאפשרת ללקוחותיה לנהל את כל הלידים בצורה אפקטיבית יותר, ניהול כל התקשרות מול לקוחות הקצה שלהם, ניהול כספים והעברת תשלומים, חתימת חוזים, תזמון פגישות, ניהול משימות, אוטומציה וכו'.

הפלטפורמה עוזרת יום-יום לעשרות אלפי אנשים בארה"ב להתנהל בצורה אפקטיבית ומקצועית יותר, כך שהם סוגרים יותר עסקאות בפחות זמן ומאמץ. את הזמן הפנוי שלהם הם יכולים להשקיע בהגדלת העסק, מציאת עוד לקוחות ובמשפחה שלהם.

בהאניבוק הלקוח הוא המרכז ואיתו גם הברנדינג שלו. חשוב לנו לוודא שאנחנו מאפשרים לו להראות הכי טוב שהוא יכול בתקשורת מול הלקוחות שלו. עם טכנולוגיות מתקדמות ודגש על עיצוב, אנחנו מאפשרים לו בקלות לבנות חוזים, הצעות מחיר ואימיילים שנראים טוב ומאפשרים תקשורת מהירה ויעילה עם הלקוחות שלו.

אנחנו מתמודדים עם אתגרים טכנולוגיים, עיצוביים ופיננסיים. כאשר בכל אתגר אנחנו שמים את הלקוח במרכז על מנת להגיע להחלטה נכונה ומהירה. בואו תצטרפו לחברה מצליחה שרוצה לשנות את הדרך בה עצמאים עושים עסקים. חברה שמאפשר ללקוחותיה להתפרנס מהחלום שלהם. וכבר הזכרנו שהופענו ברשימת מקומות העבודה האטרקטיביים ביותר לשנת 2018 ו 2019? גם בישראל וגם בסן פרנסיסקו (אם תהיתם).

אז למה לעבוד בהאניבוק? כי התרבות עבודה פה מדהימה. כי כיף להגיע כל בוקר לעבוד עם אנשים מוכשרים כל כך. כי כל יום שומעים מאות פידבקים מדהימים מלקוחות ששינינו להם את החיים. כי האתגר הטכנולוגי דוחף אותנו כל יום לבנות דברים חדשים ולשפר את מה שכבר בנינו.

עמוד המשרות שלנו: <https://www.honeybook.com/careers>

על ריאקט

צד הלקוח הוא הכינוי לקבצים ששרת האינטרנט שולח אל המשתמש והם-הם בעצם אתר האינטרנט. בדרך כלל מדובר בקובץ אחד או יותר של HTML ו-CSS, בקובצי תמונות ובקובצי ג'אווהסקריפט. הדפדפן יודע לקרוא את כל הקבצים האלו ולבנות מהם תמונה שלמה של אתר, ה-HTML קובע את מבנה האתר, ה-CSS והתמונות קובעים את העיצוב שלו וקובצי הג'אווהסקריפט את התנהגותו. בתחילת ימי הרשת כך נראו אתרי אינטרנט; בתחילה ג'אווהסקריפט שימשה לאנימציות או לאינדיקציות שונות בדף ובהמשך לתפקידים מתוחכמים יותר כמו שליחת בקשות באמצעות AJAX. ובכל זאת, באתרי האינטרנט הראשוניים, כל אינטראקציית ניווט כלשהי – לחיצה על כפתור בתפריט או שיגור טופס – שיגרה בקשה לשרת וגרמה לטעינה מחודשת שלו בידי הדפדפן ולקבלת סט חדש של קובצי HTML, CSS וג'אווהסקריפט.

במהלך הזמן החלו להתפתח ספריות ג'אווהסקריפט, כמו jQuery. ספריית jQuery הייתה ספריית עזר שסייעה למתכנתי ג'אווהסקריפט ליצור אפקטים ואנימציות בקלות רבה יותר. קוד הג'אווהסקריפט בצד הלקוח הפך להיות משמעותי יותר ויותר. בשנת 2008 יצאה לשוק ספריית שנקראת Backbone.js. זו הייתה ספרייה ששינתה לחלוטין את הדרך שבה אנו מסתכלים על צד הלקוח: במקום קוד שמגדיר רק התנהגות – אפליקציה שלמה שיושבת בצד הלקוח ומתנהגת כמו אפליקציית צד שרת, כולל אפשרויות ניווט בעמודים, הבאת מידע מ-API של שרתים ועוד שימושים רבים. היתרונות של שימוש בספריות גדולות כאלו, או יותר נכון פריימוורקים המגדירים את ההתנהגות בצד הלקוח, היו רבים – מקלות פיתוח ועד חוויית שימוש יוצאת דופן עבור הלקוח. הפריימוורקים האלו אפשרו ליצור Single Page Application – SPA. כלומר, כשגולשים באתר ובמעבר בין דפים אין טעינה מחדש מהשרת אלא מעבר חלק בין דף לדף באמצעות רכיב מבוסס ג'אווהסקריפט שנקרא ראוטר (ועליו נלמד בהמשך הספר). Backbone.js הייתה הראשונה, אבל מהר מאוד הגיעו ספריות נוספות כמו Ember.js וכמובן אנגולר. ריאקט הגיעה אחרי אנגולר וגרסתה הראשונה יצאה ביוני 2013. מאז היא תפסה תאוצה משמעותית והפכה לאחת מהספריות הגדולות והנפוצות בעולם.

ריאקט (React) היא ספרייה מבוססת ג'אווהסקריפט המיועדת לצד הלקוח. היא מאפשרת לנו ליצור אתרים שלמים ומערכות שלמות בקלות רבה ובדרך מודרנית ופשוטה. באמצעות ריאקט אנו

יכולים ליצור דפי אינטרנט או אפליקציות שרצות על טלפונים ניידים ואפילו על מחשבים בקלות רבה.

במקור, המפתחת של ריאקט היא חברת פייסבוק, שעד היום היא התומכת הראשית שלה והמפתחים שלה מובילים את פיתוח הליבה של ריאקט ומתווים את הדרך. ריאקט מפותחת ברישיון קוד פתוח מלא (החל מגרסה 16) וקוד המקור שלה נמצא בגיטהאב. אפשר להשתמש בה לכל שימוש בצורה חופשית. אחד היתרונות הגדולים בריאקט הוא עושר הקומפוננטות שמשמשות בה, מה שאומר שאפשר ליצור בקלות אפליקציות מורכבות באמצעות שפע הקומפוננטות שמפתחים אחרים פיתחו – דבר המאפשר לכל צוות פיתוח שבוחר בריאקט גמישות ועבודה מהירה מאוד. גם סביבת הפיתוח של ריאקט וכלי הבדיקות שלה, החיוניים לפיתוח בקנה מידה גדול, הם מצוינים ועמידים מאוד. יש לה כמובן גם חסרונות – החיסרון העיקרי הוא שריאקט לא קובעת עבור המפתח את הרכיבים שאיתם הוא יכול לעבוד, דבר שעלול להוביל לבלבול או לקבלת החלטות לא נכונות, אבל יש מפתחים שיראו בזה יתרון. כך או כך, נכון לזמן כתיבת הספר, רוב המתכנתים שצריכים לפתח אתר כלשהו בצד הלקוח בוחרים בריאקט.

בשנים האחרונות, טייפסקריפט היא הבחירה הראשונה בכל הנוגע לשימוש בריאקט. היכולת של טייפסקריפט למנוע תקלות שונות הקשורות לסוגי מידע וקלות השימוש בה גרמה לטייפסקריפט להיות נפוצה מאד. זו הסיבה שבספר אנו לומדים ריאקט עם טייפסקריפט, למרות שניתן לכתוב ריאקט עם ג'אווהסקריפט בלבד.

הקונספציה של ריאקט ודרך העבודה בה לא שונות מהותית מספריות אחרות כמו אנגולר או Vue. אז אם אינכם מכירים אף פריימוורק או ספרייה של ג'אווהסקריפט, ריאקט היא מקום מצוין להתחיל בו את המסע לעולם המופלא של פיתוח צד לקוח. זאת אף שריאקט שונה בכמה דרכים מהותיות, שאותן נלמד בהמשך, מספריות אחרות כמו אנגולר.

דרך הלימוד

הניסיון שלי מלמד שכל דבר חדש בתכנות לומדים דרך הידיים. אני ממליץ מאוד להעתיק כל דוגמה וכל קטע קוד בספר, להדביק אותם ב-IDE החביב עליכם – כמו Visual Studio Code למשל, לשחק בהם ולבדוק איך הם עובדים. בסוף כל פרק יש תרגילים – אין לי די מילים כדי להבהיר עד כמה חשוב לפתור אותם ולשבור עליהם את הראש לפני שמציצים בפתרונות ובהסברים. כדאי מאוד לא לוותר ולא להרפות ולנסות שוב ושוב עד שמבינים את הפתרון.

יכול להיות שלמרות ההסברים ולמרות הדוגמאות לא תבינו נושא מסוים או שלא תבינו אותו עד הסוף, לעומק. זה קורה לטובים ולמבריקים ביותר. הפתרון? חיפוש בגוגל – במיוחד באנגלית. כיוון שריאקט היא כל כך פופולרית, יש סיכוי סביר ביותר שמישהו כבר נתקל בבעיה הזו וכתב עליה משהו. אתרים כמו StackOverflow והפורומים השונים מכילים שפע של מידע ותשובות לשאלות שונות. נוסף על כך, בפייסבוק יש לא מעט קבוצות מקצועיות בעברית שישמחו לסייע לכם – בפרק הסיכום של הספר יש כמה קישורים רלוונטיים.

כמובן שדרך טובה מאד ללמוד היא באמצעות בינה מלאכותית, כמו למשל צ'אט GPT. הבינה המלאכותית הזו זמינה באתר <https://chat.openai.com> וניתן, באמצעות תיבת השיח שם, לשאול שם שאלות או להדביק קוד שלם או שגיאות שונות ולשאול את הבינה המלאכותית מה מקורן ואיך לפתור אותן.

למרות הפיתוי העז, אני ממליץ לא להעזר יותר מדי בבינה מלאכותית לכתיבת קוד מורכב אלא להתאמץ ולכתוב את הקוד בעצמכם. בעבודה היומיומית עם בינה מלאכותית יש חשיבות רבה לידע בקוד המאפשר גם לבדוק את הקוד המוצע על ידי הבינה המלאכותית וגם להנחות את הבינה המלאכותית בביצוע משימות שונות. ניתן לרכוש את הידע הזה רק בעבודה עצמית.

על המונחים בעברית

אני כותב בעברית על טכנולוגיה ועל תכנות כבר יותר מעשור, והדילמה באילו מונחים בעברית להשתמש מלווה אותי תמיד. מצד אחד, האקדמיה ללשון העברית מספקת לנו מונחים רבים בעברית. מצד שני, בתעשיית ההייטק, שממנה אני מגיע, איש לא משתמש ברבים מהמונחים האלו. אם תגיעו לריאיון עבודה ותגידו: "במפגש המתכנתים האחרון שמעתי על דרך חדשה לבצע

הידור שבודק הזחות במנשק מבוסס הבטחות", סביר להניח שלא תקבלו את העבודה. אבל אם תגידו, "במיטאפ האחרון שמעתי על דרך חדשה לבצע קמפול שבודק אינדנטציה ב-API מבוסס פרומיסים" – יבינו על מה אתם מדברים. זו הסיבה שלא תמצאו בספר מילים כמו "הידור", "מחלקה" או "מרשתת", אלא "קמפול", "קלאס" ו"אינטרנט". המונחים שבהם השתמשתי הם המונחים שבהם משתמשים בתעשייה בפועל. בכל מקום שבו אני משתמש במונח לראשונה, אני מספק גם את הגרסה שלו באנגלית כדי שתוכלו להכניס אותו לחיפושים שלכם בגוגל או בצ'אט GPT.

חשוב לציין שאיני בז כלל לאקדמיה ללשון ושחלק מהמונחים שלה אכן נכנסו לשפה המדוברת במרכזי הטכנולוגיה השונים (למשל: קובץ או מסד נתונים), אבל בכל מקום שהייתה בידיי הבחירה בין להיות מובן לבין לעמוד בכללי הלשון, העדפתי להיות מובן.

סביבת העבודה הבסיסית

זה הפרק החשוב ביותר, כיוון שאי-אפשר ללמוד קוד בלי ללכלך את הידיים בקוד משלכם. קריטי לקרוא את הפרק הזה וליצור סביבת עבודה בסיסית על המחשב שלכם. אם אתם נתקלים בקושי כאן – אל תוותרו ואל תתיאשו. התקנת סביבת עבודה היא החלק הקשה ביותר בלימוד טכנולוגיה חדשה. נסו שוב ושוב – בצעו ריסטרט למחשב, נסו ממחשב אחר, שדרגו את מערכת ההפעלה, נסו מדפדפן אחר שאינו הדפדפן הרגיל שלכם – כל טכניקה וטכניקה. פשוט אי-אפשר לדלג על השלב הזה. נסו להעזר בכלי בינה מלאכותית ולהזין לתוכם את השגיאות, אם יש כאלו, כדי לקבל סיוע. זה החלק החשוב ביותר.

ריאקט מורכבת מכמה חלקים – כולם בקוד ג'אווהסקריפט. החלק הראשון הוא הספרייה עצמה: ריאקט. מדובר בקובץ ג'אווהסקריפט מרכזי המכיל את כל הפונקציות של הספרייה. הקובץ מכיל מודולים של ג'אווהסקריפט וגם משתנים גלובליים ואחרים – ובלעדיו אי-אפשר להשתמש בריאקט. כשאנו בונים סביבת עבודה בסיסית, אנו נזקקים לריאקט. הגיוני, לא?

החלק השני הוא react-dom. גם הוא חלק מהספרייה וגם הוא כתוב בג'אווהסקריפט. הוא מכיל את הפונקציות הקושרות בין ריאקט ל-DOM. ראשי התיבות של DOM הם Document Object Model, ואני ארחיב לגביו בהמשך. כרגע צריך פשוט לזכור שמדובר בעוד חלק של הפריימוורק שאנו צריכים איתנו כשאנחנו מפתחים עבור אתרי אינטרנט.

החלק השלישי הוא babel. מה זה? מדובר בספריית ג'אווהסקריפט שימושית ופופולרית מאוד. יש לה כמה תפקידים חשובים. במקור babel סייעה למתכנתים שרצו להתאים את קוד הג'אווהסקריפט שלהם לדפדפנים ישנים שלא תמכו בפיצ'רים החדשים של השפה, למשל דפדפנים שלא ידעו מה זה let או const. הספרייה הזו לקחה את כל הקוד המודרני של ג'אווהסקריפט והעבירה אותו תהליך, שבמסגרתו הוא הפך לקוד שתואם גם דפדפנים ישנים. למשל, היא המירה את let ל-var, כך שדפדפן ישן יוכל לעבוד איתו. התהליך הזה נקרא "טרנספירציה" – זו המילה במדעי המחשב שמשמעה לקחת קוד שכתוב בשפה מסוימת ולתרגם אותו לקוד שכתוב בשפה אחרת. במקרה הזה לקחת קוד שכתוב בג'אווהסקריפט מודרנית ולהעביר אותו לקוד שכתוב בג'אווהסקריפט מגרסאות קדומות.

במקרה שלנו, ל-babel יש תפקיד משמעותי בהמרת ה-JSX שלנו, שהוא הסינטקס שבו אנו כותבים בריאקט קומפוננטות לקוד HTML שהדפדפן יודע לעבוד איתו. על JSX נלמד בהרחבה בהמשך.

אלו החלקים הבסיסיים שחייבים להיות בסביבת העבודה שלנו. נוסף על הקוד שלנו, אנו צריכים ליצור דף HTML ריק לחלוטין שקורא באמצעות ה-src לשלושת הקבצים האלו וגם לקובץ שבו אנו כותבים את הקוד שלנו.

אנו יכולים להוריד את הקבצים האלו מהאתר של ריאקט או להשתמש בהם ישירות מ-CDN. ראשי התיבות של CDN הם Content Delivery Network – זהו כינוי לשרתים גדולים שנמצאים בכל מקום בעולם. חלק מהם מציעים שירות פתוח לציבור של אחסון קבצים של ספריות גדולות ומובן שריאקט, הפריימוורק הפופולרי בעולם, נמצאת ביניהן. ה-CDN שהדוקומנטציה של ריאקט משתמשת בו הוא unpkg ואנו נשתמש בו. unpkg מציעה את הקבצים בקישורים הבאים:

קובץ הפריימוורק של ריאקט:

<https://unpkg.com/react@18/umd/react.development.js>

שימו לב: הספר מלמד על גרסה 18 של ריאקט ולכן קיים המספר הזה ב-URL.

קובץ react-dom:

<https://unpkg.com/react-dom@18/umd/react-dom.development.js>

קובץ ה-babel:

<https://unpkg.com/@babel/standalone/babel.min.js>

כך קובץ ה-HTML שלנו נראה:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>React development environment</title>
  <script crossorigin
src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="content"></div>
</body>
<script type="text/babel">
  console.log('Here will be React code');
</script>
</html>
```

בתחתית הקובץ נמצא הקוד שלנו. אנו מסמנים ל-babel שהוא קוד שצריך לעבור טרנספילציה באמצעות הצמדה של:

```
type="text/babel"
```

לתגית ה-script. בין התגית הפותחת של ה-script לתגית הסוגרת שלו, אנו יכולים לכתוב את קוד הג'אווהסקריפט שלנו. כרגע יש שם את השורה הבאה:

```
console.log('Here will be React code');
```

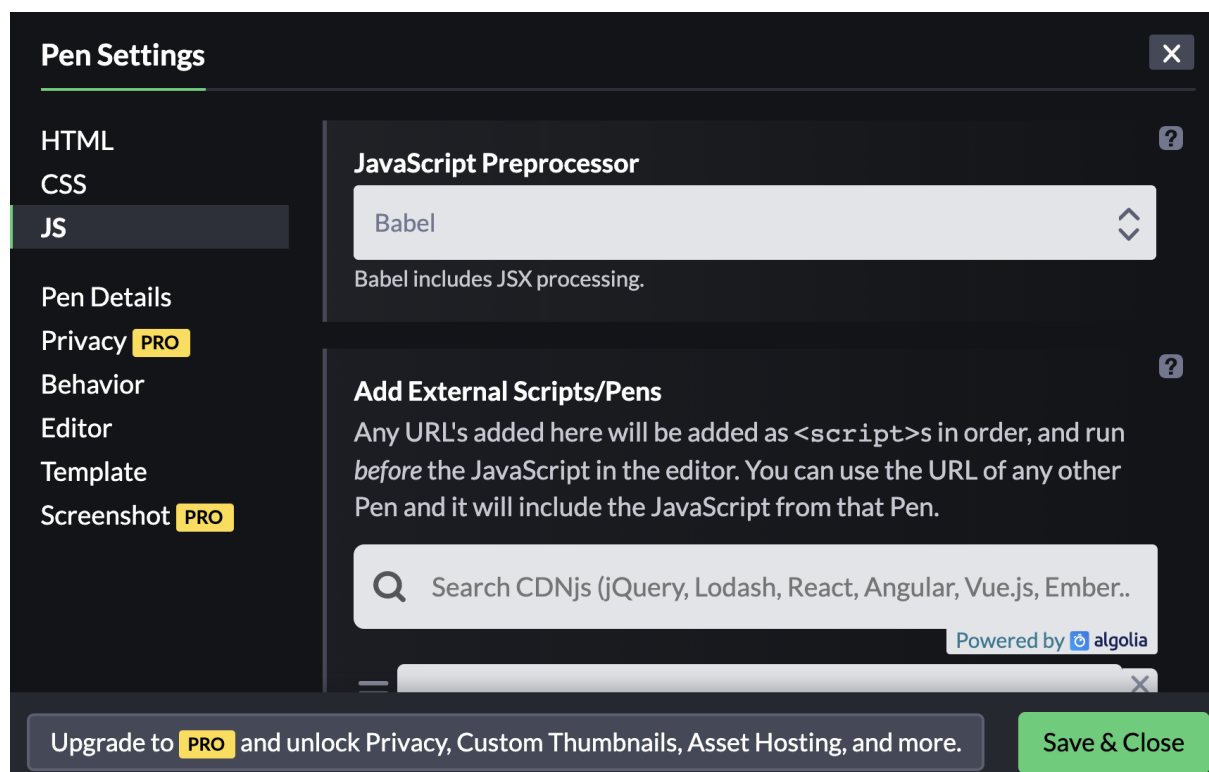
באמצעות עורך הקוד החביב עליכם (כמו Visual Studio Code החינמי), צרו את קובץ ה-HTML הזה ושמרו אותו במחשב המקומי שלכם. פתחו את קובץ ה-HTML בדפדפן באמצעות כניסה לתיקייה שבה הקובץ נשמר ולחיצה על "פתח עם דפדפן" וידאו שהוא עולה ושבקונסולה של כלי המפתחים מופיע Here will be React code. אם כן – זהו, אנחנו מוכנים לעבודה ראשונית.

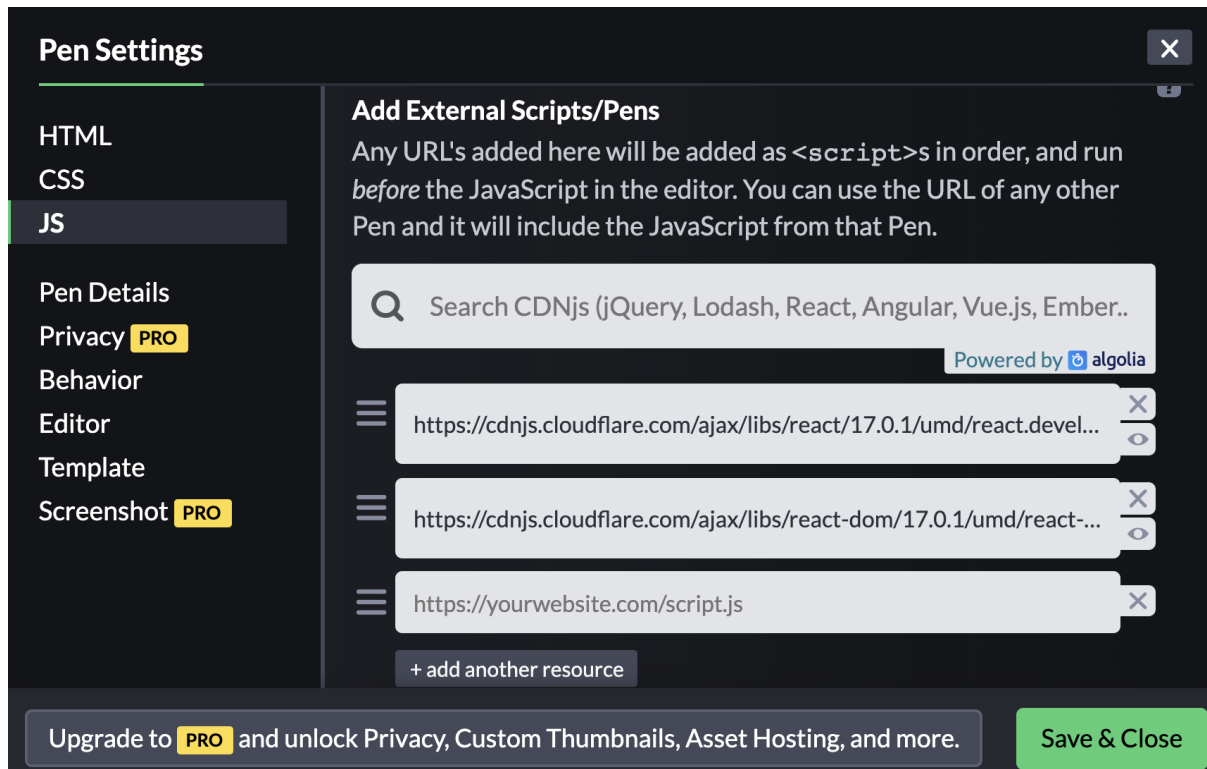
כדאי לשים לב שאנו עדיין לא משלבים טייפסקריפט בקוד, אנו נשלב אותו בסביבת העבודה המתקדמת. כרגע אנו רק לומדים איך ריאקט עובד ולא רוצים לערב טייפסקריפט.

סביבת עבודה בסיסית ב-codepen

אפשרות נוספת היא לבנות סביבת עבודה מרוחקת, שגם בה אפשר לכתוב ג'אווהסקריפט ולצפות בתוצאות באמצעות דפדפן בלבד. סביבת העבודה הזו זמינה בחינם בכמה וכמה כלים ואתרים, אבל האתר הכי פופולרי ומוצלח הוא codepen.io. מדובר באתר שמאפשר לכולם (אפילו בלי רישום, אף על פי שמומלץ להירשם כי זה מאפשר שמירה של הקוד שלכם) לכתוב קוד פשוט בג'אווהסקריפט, HTML ו-CSS. העבודה באתר פשוטה: נכנסים אל <https://codepen.io/pen/>, מתחילים לכתוב קוד ורואים את התוצאות.

כדי לעבוד עם codepen וריאקט חייבים להכניס את שלושת קובצי הג'אווהסקריפט שהזכרתי קודם: קובץ הפריימוורק של ריאקט, קובץ ה-react-dom וקובץ ה-babel. איך עושים את זה? בהגדרת כל "פרויקט" (שנקרא pen באותו אתר) הקפידו להכניס את babel כ-preprocessor וקישור ידני אל קובצי הג'אווהסקריפט מה-CDN. זה נראה כך:





עשיתי את זה עבורכם בפרויקט משלי ואתם יכולים, במקום לטפל בהגדרות בעצמכם, להיכנס אל הקישור הזה שבו הן מוכנות:

<https://codepen.io/barzik-the-vuer/pen/gOYMWoP>

צפו בקונסולה וראו את המסר React code .Here will be

חשוב לציין שזו סביבת פיתוח בסיסית מאוד של ריאקט ושהיא כמובן מוכוונת פיתוח בלבד ומיועדת ללימוד, אבל זה אמור להספיק לכתיבת אפליקציית הריאקט שלנו והקומפוננטה הראשונה.

פרק 1

אפליקציית ריאקט



אפליקציית ריאקט

כל הקוד של ריאקט אמור להיות בתוך אפליקציית ריאקט. זה סוג של מתחם שבו הקוד מבוסס הריאקט שלנו עובד. בעצם, מדובר באלמנט אב של DOM שמתחתיו ריאקט שולטת, יוצרת DOM משלה, שנקרא Virtual DOM, וחיה. יכולות להיות כמה אפליקציות ריאקט בדף HTML אחד. מובן שכל אחת מהן חיה מתחת לאלמנט DOM אחר. אפליקציית הריאקט היא בעצם אלמנט DOM שאנחנו מכריזים עליו כשלנו, ובו אנו יכולים ליצור את המרכיבים של האתר שלנו – שהם קומפוננטות הריאקט.

כדי ליצור אפליקציית ריאקט מתחת לאלמנט מסוים אנו חייבים פשוט... לבחור אותו. אנו נגדיר div, שזה אלמנט HTML פשוט ביותר עם id פשוט. להזכירכם – id הוא סלקטור ייחודי המגדיר אלמנטים ב-HHTML שאנו בוחרים לתת להם זהות ספציפית. בקובץ ה-HHTML שלנו כבר יש הגדרה של div כזה:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>React development environment</title>
  <script crossorigin
src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="content"></div>
</body>
<script type="text/babel"></script>
</html>
```

הינה ההגדרה – div שה-id שלו הוא content:

```
<div id="content"></div>
```

זה המקום שבו תוצב אפליקציית הריאקט שלנו. אנו צריכים רק לזכור שה-id הוא content. את אפליקציית הריאקט אנו יוצרים באמצעות:

ReactDOM.createRoot

המתודה הזו באה עם האובייקט הגלובלי ReactDOM. האובייקט הגלובלי הזה בא בזכות קובץ ה-react-dom.js, שדאגנו שיהיה בסביבת הפיתוח שלנו. המתודה מקבלת ארגומנט אחד שהוא האלמנט שאנו בוחרים כדי להריץ בו את ה-JSX של האפליקציה ומחזירה אלמנט ריאקטי שמייצג את האפליקציה. בו יש את מתודת render שמקבלת את ה-JSX.

הציבו את הקוד הזה בין תגיות הסקריפט שלכם:

```
<script type="text/babel">
  const rootElement = document.getElementById('content');
  const root = ReactDOM.createRoot(rootElement);
  root.render(
    <h1>Hello World!</h1>
  );
</script>
```

שמרו וצפו בתוצאה. אתם תראו Hello World! בדפדפן אם הכול תקין. אם לא הכול תקין ואתם לא רואים Hello World! הפסיקו לקרוא, פתחו את הקונסולה, צפו בשגיאות וחפשו אותן בגוגל כדי לתקן, ודאו שהעתקתם את הקוד כשורה והמשיכו רק כאשר אתם יודעים שסביבת הקוד שלכם עובדת.

הבה נעבור על הארגומנטים השונים של ReactDOM. נתחיל מהשני, ה-target. אני יוצר רפרנס לאלמנט שבו אני רוצה להציב את התוכן שלי באמצעות:

```
document.getElementById('content');
```

אני מעביר אותו לקבוע rootElement. אתם אמורים להכיר את זה אם יש לכם ידע בג'אווהסקריפט. אם לא, אנא חזרו על החומר של ג'אווהסקריפט ו-HTML (שנמצא גם בספר שלי "ללמוד ג'אווהסקריפט בעברית"). את הקבוע rootElement אני מעביר כארגומנט אל הפונקציה createRoot של ReactDOM שמחזירה לי אלמנט שנכנס אל הקבוע root. מעכשיו בעצם root הוא אפליקצית הריאקט שלי ואני יכול לקרוא למתודת render ולהכניס לתוכה ארגומנט של JSX.

עכשיו נדבר על הארגומנט הזה, שמורכב שיש בו כרגע רק את המידע הזה:

```
<h1>Hello world!</h1>
```

הוא לא מורכב מדי, בסך הכול תגית HTML. אבל שימו לב למשהו מעניין – ה-HTML הזה נמצא בתוך קוד ג'אווהסקריפט! איך זה יכול להיות? הרי אם תשתמשו בקוד HTML ללא מירכאות בקוד ג'אווהסקריפט, הקוד ידפיס שגיאה ויפסיק לפעול. ג'אווהסקריפט לא מכירה HTML ולא יודעת לעבוד איתו. איך יכול להיות שאני משתמש ב-h1 וב-HTML בג'אווהסקריפט ללא מירכאות והקוד לא נופל?

הסיבה היא JSX. זוכרים את המרכיב השלישי בסביבת הפיתוח שלנו, ה-babel? הוא אחראי למצוא את כל התגיות של ה-HTML שיש בקוד הג'אווהסקריפט ולהמיר אותן למשהו שג'אווהסקריפט יודעת להתמודד איתו. הקוד הזה, שכרגע יש בו HTML פשוט בלבד, הוא JSX – ראשי תיבות של JavaScript XML – והוא אחד מהפיצ'רים החזקים שיש לריאקט. אנו נלמד עליו לעומק בהמשך ונראה איך מהמצב שיש לנו אפליקציית ריאקט אנו מגיעים למצב שבו אנו מפתחים קומפוננטות.

JSX הוא ג'אווהסקריפט לכל דבר ועניין. אני יכול לשים אותו במשתנה לצורך העניין:

```
const rootElement = document.getElementById('content');
const root = ReactDOM.createRoot(rootElement);
const value = <h1>Hello World!</h1>;
root.render(value);
```

אפשר גם לשים אותו, כפי שנראה בהמשך, בלולאות ובמקומות אחרים.

תרגיל

צרו HTML שמכיל אפליקציית ריאקט במחשב המקומי שלכם. אפליקציית הריאקט תהיה ב-div שייקרא my-react-app.

פתרון

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>React development environment</title>
  <script crossorigin
src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="my-react-app"></div>
</body>
<script type="text/babel">
  const rootElement = document.getElementById('my-react-app');
  const root = ReactDOM.createRoot(rootElement);
  root.render(
    <h1>Hello World!</h1>
  );
</script>
</html>
```

ראשית ניצור קובץ HTML פשוט שבו יש קריאה לשלושת הקבצים שאנו צריכים לאפליקציית ריאקט ושעליהם הסברנו בתחילת הפרק.

הצעד הנוסף הוא ליצור div שיש לו id מסוים שאנו רוצים להכניס אליו את אפליקציית הריאקט. במקרה הזה:

```
<div id="my-react-app"></div>
```

השלב הבא הוא ליצור את אפליקציית הריאקט באמצעות:

ReactDOM.createRoot

המתודה הזו מקבלת ארגומנט שהוא האלמנט שהאפליקציה תהיה בו. אני מקבל אותו באמצעות `getElementById`, שהוא מתודת ג'אווהסקריפט שנמצאת בדפדפן באופן טבעי (בלי קשר לריאקט), ומעביר אותו באמצעות משתנה:

```
const rootElement = document.getElementById('my-react-app');
const root = ReactDOM.createRoot(rootElement);
```

המתודה מחזירה את ה-`root` של האפליקציה. אובייקט מיוחד שיש בו מתודות של אפליקצית ריאקט.

כשיש לי את ה-`root` של האפליקציה, אני יכול להריץ בו קוד `JSX`. אני יכול לעשות כן עם מתודת `render` שמקבלת ארגומנט אחד: `JSX` פשוט שאינו שונה מ-`HTML`, מלבד העובדה הפשוטה שהוא בתוך קובץ ג'אווהסקריפט:

```
root.render(
  <h1>Hello World!</h1>
);
```

שמירת ה-`HTML` ופתיחה שלו באמצעות הדפדפן תציג לי `Hello World!`. או כל טקסט אחר.

פרק 2

בניית קומפוננטות פונקציה בסביבת העבודה הבסיסית



בניית קומפוננטת פונקציה בסביבת העבודה הבסיסית

אחרי שלמדנו לבנות את האפליקציה ואפילו למדנו על JSX בסיסי, הגיע הזמן ללמוד על המבנה הבסיסי של ריאקט – קומפוננטות. אחת המהפכות הגדולות בפיתוח צד לקוח שריאקט הביאה היא הקומפוננטות. מדובר ביחידות תוכנה קטנות שאפשר להשתמש בהן שוב ושוב במקומות שונים באתר שלנו. המתכנתים בריאקט יוצרים למשל קומפוננטה אחת של טבלה הניתנת למיון ויכולים להשתמש בה בכל מקום באפליקציה או באתר שלהם. אם הם צריכים לשנות את הטבלה, הם משנים קומפוננטה אחת בלבד. בכל אפליקציית ריאקט יכולות להיות אינספור קומפוננטות. למתכנתים חדשים קל לחשוב על קומפוננטות בתור פונקציות.

בתרגיל בפרק הקודם יצרנו אפליקציית ריאקט שבה כתוב "Hello World!" או כל טקסט אחר. אם אני רוצה לכתוב כמה פעמים "I am learning React!", אני יכול לשכפל את ה-h2 ב-JSX באופן הבא:

```
const rootElement = document.getElementById('my-react-app');
const root = ReactDOM.createRoot(rootElement);
root.render(
  <div>
    <h2>I am learning React!</h2>
    <h2>I am learning React!</h2>
    <h2>I am learning React!</h2>
  </div>
);
```

שימו לב שאני צריך לעטוף את השכפולים שלי ב-div אחד מקיף, כיוון שפונקציית render דורשת ממני אלמנט אב אחד. בתוך אלמנט האב אני יכול לשכפל כמה פעמים שאני רוצה את מה שבא לי, אבל זו לא הדרך הריאקטית. הדרך הריאקטית היא ליצור קומפוננטה.

הבה ניצור קומפוננטה שהיא זו שתדפיס עבורנו את:

```
<h2>I am learning React!</h2>
```

יש שני סוגי קומפוננטות – סוג אחד הוא קומפוננטות מבוססות קלאס, שעליהן נלמד בהמשך הספר. הסוג השני והנפוץ יותר הוא קומפוננטה של פונקציה, והיא פשוטה למדי. מגדירים אותה באמצעות פונקציה (זו לא הפתעה גדולה, נכון?). כרגע זה די פשוט. הקומפוננטה נראית כך:

```
function Greeting() {
  return <h2>I am learning React!</h2>;
}
```

כדאי לשים לב לכמה דברים – ראשית, שם הפונקציה מתחיל באות גדולה. זו קונבנציה של ריאקט שמחייבת כל קומפוננטה שהיא. שנית, הפונקציה מחזירה JSX. במקרה הזה מדובר ב-HTML פשוט. זה מאפשר לנו להבדיל בקלות בין פונקציה רגילה לבין פונקציה שיוצרת קומפוננטה. איך נשתמש בקומפוננטה הזו? בדיוק כמו HTML. כך:

```
<Greeting />
```

כאלמנט שסוגר את עצמו. או כך:

```
<Greeting></Greeting>
```

אף על פי שהקונבנציה החד-משמעית היא לכתוב אלמנט שסוגר את עצמו.

ואיך הקוד המלא שלנו ייראה? כך:

```
function Greeting() {
  return <h2>I am learning React!</h2>;
}

const rootElement = document.getElementById('my-react-app');
const root = ReactDOM.createRoot(rootElement);
root.render(
  <div>
    <Greeting />
  </div>
);
```

אני יכול לשכפל את הקומפוננטה כרצוני, כמובן. הקוד המלא יראה כך:

```
function Greeting() {
  return <h2>I am learning React!</h2>;
}

const rootElement = document.getElementById('my-react-app');
const root = ReactDOM.createRoot(rootElement);
root.render(
  <div>
    <Greeting />
    <Greeting />
    <Greeting />
    <Greeting />
    <Greeting />
    <Greeting />
  </div>
);
```

היתרון הגדול? אם אני רוצה לשנות את הכיתוב בקומפוננטה או את האלמנט או להוסיף לקומפוננטה, אני יכול לעשות את זה במקום אחד בלבד, ובבת אחת השינוי הזה ישפיע על כל שימוש ושימוש בקומפוננטה בתוך האפליקציה שלי.

בתוך הקומפוננטה אני יכול להשתמש בעוד קומפוננטות, וזה חשוב מאוד. הבה ניצור קומפוננטה נוספת שבה נשתמש בתוך קומפוננטת Greeting. משהו בסגנון הזה:

```
function Hello() {
  return <span>Hello,</span>
}

function Hello() {
  return <span>Hello,</span>
}

function Greeting() {
  return <h2><Hello />I am learning React!</h2>;
}

const rootElement = document.getElementById('my-react-app');
const root = ReactDOM.createRoot(rootElement);
root.render(
  <div>
    <Greeting />
  </div>
);
```

אולי הקוד הזה נראה לכם מסובך, אבל הוא ממש לא! ראשית, יש לנו קומפוננטה שמחזירה לנו Hello . היא נראית כך:

```
function Hello() {
  return <span>Hello,</span>
}
```

ניתן להשתמש בקומפוננטה הזו בכל מקום – היישר באפליקציה או בתוך כל קומפוננטה אחרת. במקרה הזה מי שמשתמש בה הוא קומפוננטת Greeting. איך היא משתמשת בה? בדיוק כמו ב-HTML. שם הקומפוננטה כתגית HTML:

```
function Greeting() {
  return <h2><Hello />I am learning React!</h2>;
}
```

אני יכול להשתמש בקומפוננטה הזו בכל מקום ב-JSX ואפילו כמה וכמה פעמים, אבל פה הסתפקתי רק בפעם אחת.

אני יכול להשתמש, כמובן, בקומפוננטת Greeting כמה פעמים שאני רוצה. בכל פעם שאני משתמש בה, אני אראה על המסך Hello, I am learning React!. אם ארצה לשנות את הברכה, אוכל לעשות זאת בקלות דרך שינוי במקום אחד. זה מה שיפה בקומפוננטות ריאקטיות – אפשר לבצע בהן שימוש חוזר (המונח המקובל הוא reuse) בכל מקום. קומפוננטות שמכילות עוד קומפוננטות ובתוכן יש עוד קומפוננטות וכך הלאה; הכול מתנקז בסופו של דבר לאפליקציית ריאקט אחת שמכילה בתוכה אינספור קומפוננטות.

תרגיל:

צרו אפליקציית ריאקט שבתוכה יש קומפוננטה אחת שנקראת Root. Root בקומפוננטת Root יש שתי קומפוננטות נוספות, אחת שנקראת Inigo ומחזירה JSX שהוא Hello, my name is Inigo Montoya, והשנייה שנקראת Greeting ומחזירה JSX שהוא Prepare to die! die! בהפעלת האפליקציה אני אראה שכתוב: Hello, my name is Inigo Montoya, Prepare to die!

פתרון:

```
function Inigo() {
  return <span>Hello, my name is Inigo Montoya</span>
}

function Greeting() {
  return <span>prepare to die!</span>
}

function Root() {
  return <span><Inigo />, <Greeting /></span>
}

const rootElement = document.getElementById('my-react-app');
const root = ReactDOM.createRoot(rootElement);
root.render(
  <div>
    <Root />
  </div>
);
```

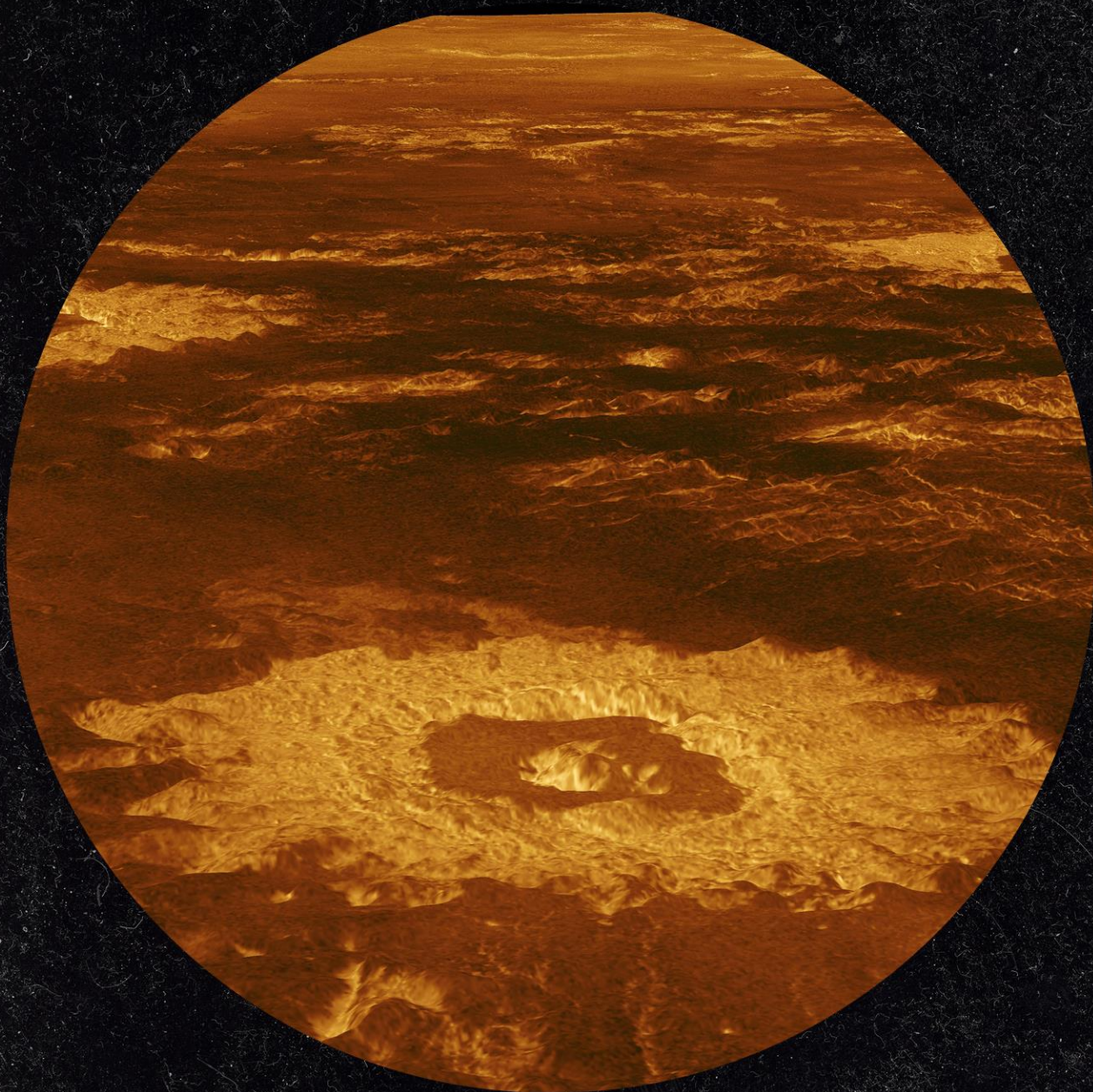
יצרתי שלוש קומפוננטות פשוטות. הראשונה והשנייה Inigo ו-Greeting מחזירות JSX פשוט. כדאי לשים לב שב-JSX אני חייב לשים אלמנט אב אחד, במקרה הזה בחרתי ב-span. הקומפוננטות האלו הן קומפוננטות מסוג פונקציה. כדאי לשים לב ששמן חייב להתחיל באות גדולה. מהרגע שהגדרתי אותן, אני יכול להשתמש בהן בכל קומפוננטה אחרת בדיוק כמו כל אלמנט HTML אחר.

הקומפוננטה השלישית היא קומפוננטת Root והיא מכילה את שתי הקומפוננטות Inigo ו-Greeting. גם פה, חשוב לציין, יש אלמנט אב אחד. גם הוא span.

כל מה שנותר לי לעשות הוא להציב את הקומפוננטה השלישית, Root, באפליקציה שלי.

פרק 3

בנייה של סביבת דיאקט אמיתית ומורכבת יותר באמצעות CREATE REACT APP



בנייה של סביבת ריאקט אמיתית

ומורכבת יותר באמצעות

Create React App

בפרקים הקודמים למדנו להקים סביבת עבודה פשוטה וראינו איך יוצרים אתרים או אפליקציות ווב באמצעות אפליקציית ריאקט וקומפוננטות פשוטות. סביבת העבודה הזו היא פשוטה מאוד ומלמדת אותנו שריאקט זה לא קסם ולא וודו – אבל רובם המוחץ של המתכנתים לא משתמשים בסביבת ריאקט כזו. היא פשוטה ופרימיטיבית מדי. כמה נוח היה אילו היו בסביבה שלנו שרת מובנה, במקום לטעון את הקובץ דרך מערכת הקבצים המקומית, או hot reload, שמרפרש אוטומטית את האפליקציה בסביבת הפיתוח בכל פעם שאנחנו עושים שינוי, או דיבאגר מובנה... כל אלו ועוד קיימים ממש מן הקופסה בריאקט.

יחד עם ריאקט יש פרויקט שנקרא Create React App, המפותח ומתוחזק על ידי הצוות של ריאקט. הפרויקט הזה הוא Bootstrapper. כלומר הוא פרויקט של ריאקט עם המון תוספות שהצוות של ריאקט חשב שהן טובות וחיוניות לפרויקט חדש. זו בעצם סביבת פיתוח מלאה הכוללת שרת שאפשר להפעיל בקלות מכל מחשב. סביבת הפיתוח הזו מבוססת על Node.js, אך לא נדרש ידע משמעותי ב-Node.js על מנת לעבוד בה.

בנוסף, ניתן להשתמש בטייפסקריפט בקלות ב-Create React App ולכתוב בה והאפליקציה תדאג להמיר את טייפסקריפט לג'אווהסקריפט יחד עם ה-JSX. זה מאפשר לי להשתמש בטייפים ומאד מקובל.

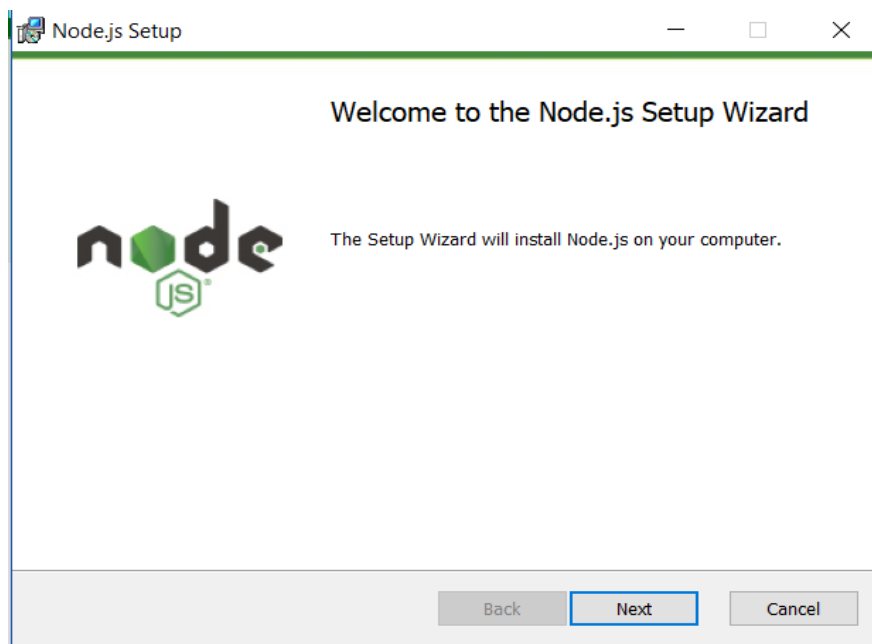
אנו לא יכולים להישאר בעולם ה-Hello World, ולימוד בניית סביבה אמיתית עם Create React App הוא חלק בלתי נפרד מהלימוד של ריאקט. הפרק הזה הוא אחד הפרקים הכי חשובים והכי מורכבים שיש בספר וחשוב מאוד לא להירתע ולא להיבהל. אם יש תקלות, כדאי מאוד לפתור אותן. כאמור, העולם של ריאקט הוא עשיר מאוד ומישהו אחר כבר חווה כל תקלה שתחוו בדרך (אם תחוו). חיפוש מהיר של התקלה בגוגל יסייע לכם מאוד.

התקנת Node.js על המחשב שלכם

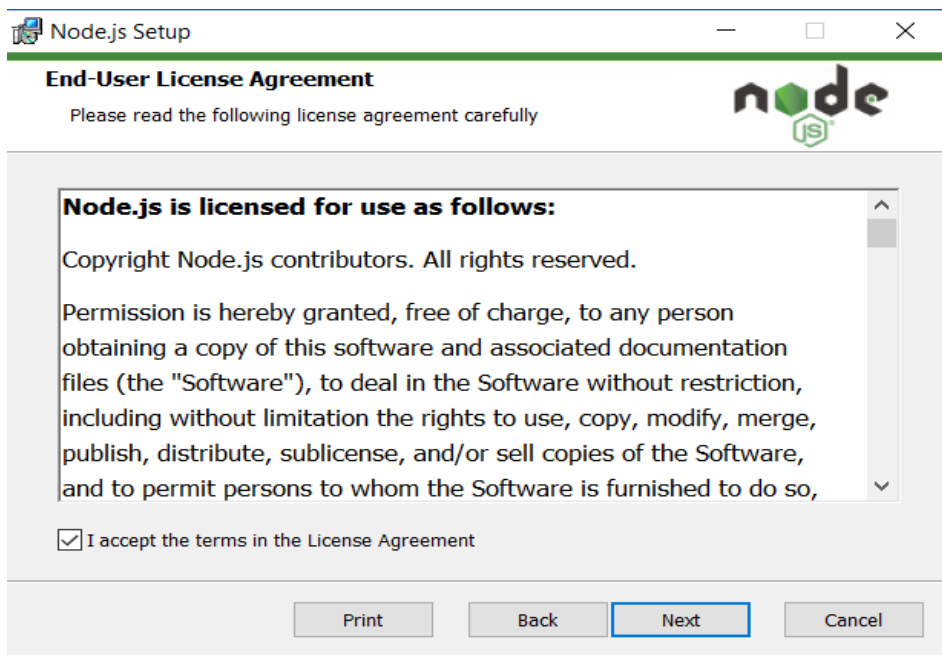
ראשית, אתם חייבים להתקין את Node.js על המחשב המקומי שאתם עובדים בו. Node.js הוא סביבה המאפשרת להריץ ג'אווהסקריפט על גבי מערכת ההפעלה, במקרה הזה מערכת ההפעלה שלכם. ג'אווהסקריפט, למי שלא יודע, היא שפה גמישה מאוד ואפשר להפעיל אותה לא רק בסביבת הדפדפן, כמו שאנו עושים בריאקט, אלא גם בסביבת מערכת הפעלה/ שרתים – ואת זה עושים באמצעות Node.js. הפלטפורמה הזו ניתנת להתקנה בכל מערכת הפעלה שהיא ובקלות. בחרו את מערכת ההפעלה שלכם והתקינו את Node.js לפי ההוראות.

התקנה על חלונות

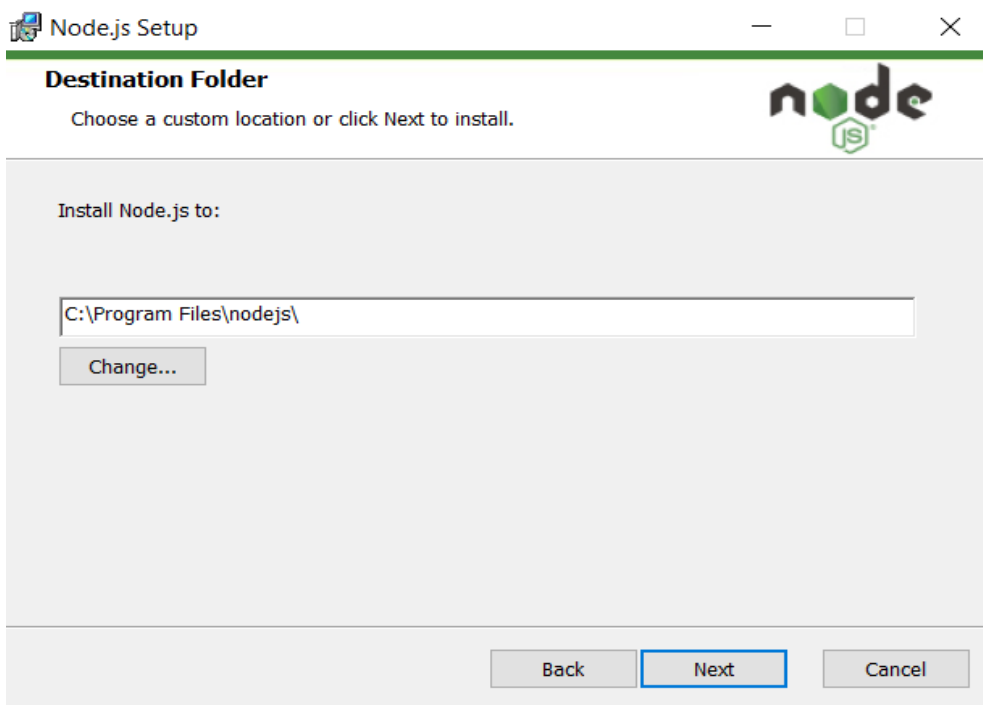
ההתקנה של Node.js על חלונות היא פשוטה. נקליד בגוגל Node.js Download או ניכנס אל: <https://nodejs.org/en/download/> אנו נבחר בגרסת LTS – ראשי תיבות של "גרסה לטווח ארוך", ונבחר במערכת ההפעלה שלנו – אם מדובר בחלונות, יש לנו installer נוח. מורידים, לוחצים על התוכנה שיוורדת:



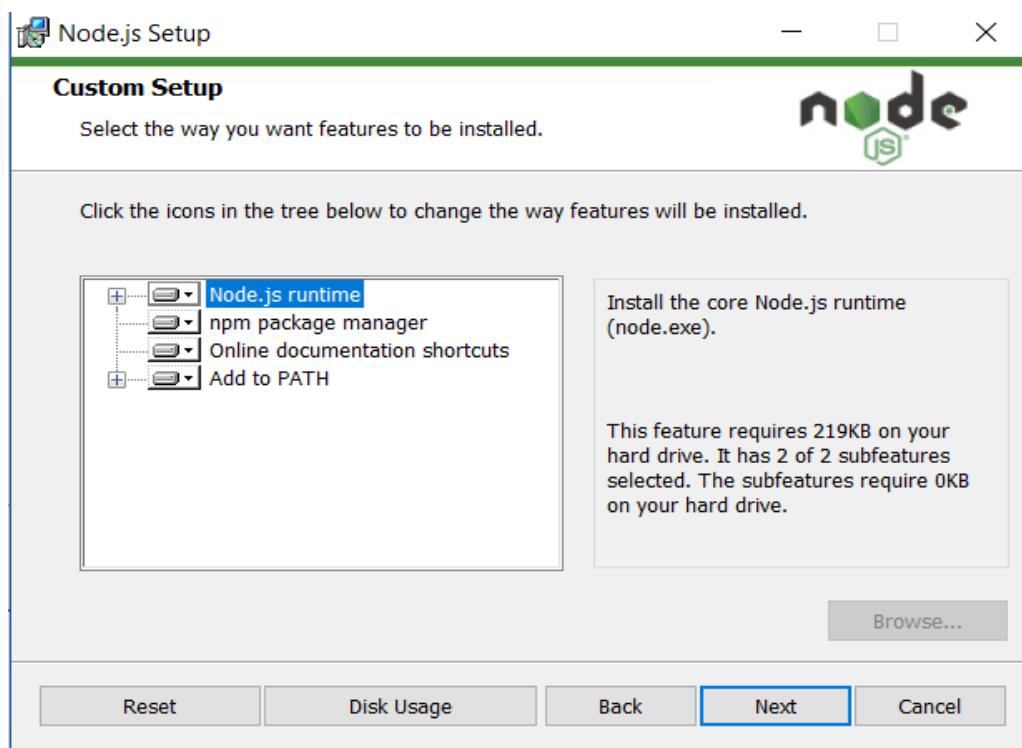
מקבלים את התנאים:



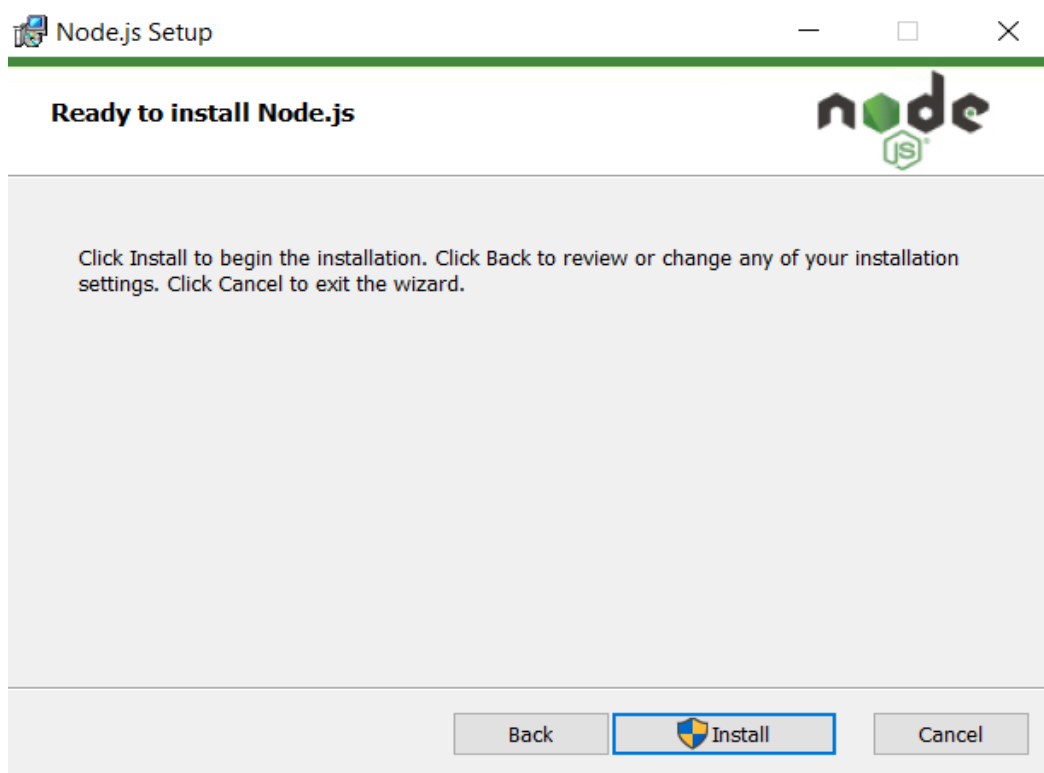
לוחצים על Next:



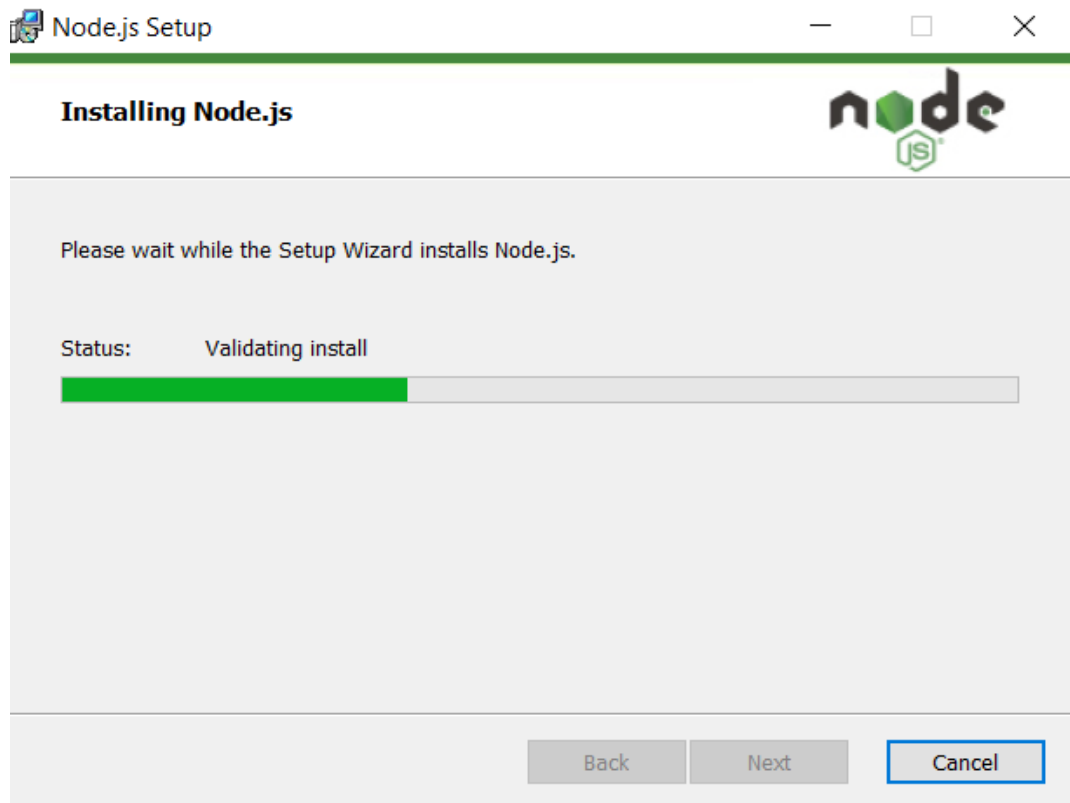
ושוב על Next:



לחיצה על Install לבסוף תתקין את התוכנה:



כל מה שנותר הוא לחכות לסוף ההתקנה:



התקנה על מק

ההתקנה של Node.js על מק פשוטה מאוד. נקליד בגוגל Node.js Download או ניכנס אל: <https://nodejs.org/en/download/>

אנו נבחר בגרסת LTS – ראשי תיבות של "גרסה לטווח ארוך", ונבחר במק – יִרְד קובץ dmg שאותו אפשר להתקין כמו כל תוכנה אחרת בהנחה שהמחשב שלכם הוא לא מחשב ארגוני שמונע התקנות מהאינטרנט. ההתקנה היא פשוטה ביותר.

אם אתם משתמשים ב-Zsh או ב-Oh My Zsh אני ממליץ להתקין את Node.js בעזרת homebrew באמצעות הפקודה `brew install node` (אם homebrew מותקנת אצלכם, וכדאי שהיא תהיה מותקנת).
כך או אחרת, לאחר ההתקנה, כניסה לטרמינל והקלדה של `node -v` יראו לכם את מספר הגרסה.

התקנה על לינוקס

אם אתם משתמשים בדביאן, בדרך כלל, ברוב ההפצות, `sudo apt-get install node` יטפל בהתקנה, אך ייתכן שתתקינו גרסה ישנה של Node.js, וזה עלול להיות בעייתי. למרות הפיתוי, קראו לפני ההתקנה את המדריך המלא לכל ההפצות של לינוקס, שמסביר על ההתקנות: <https://nodejs.org/en/download/package-manager/>

אני יוצא מנקודת הנחה שמשתמשים בלינוקס הם מיומנים בהרבה ממשתמשי חלונות ויודעים להתקין חבילת תוכנה ללא הסברים נוספים. כך או אחרת – לאחר ההתקנה, כניסה לטרמינל והקלדה של `node -v` יראו לכם את מספר הגרסה בדיוק כמו במק.

עבודה עם טרמינל

כמו כל אפליקציית Node.js, גם Create React App מצריכה אותנו לעבוד עם טרמינל. טרמינל הוא הממשק שבו אני מקליד פקודות למערכת ההפעלה. כל מתכנת עובד עם טרמינל כאשר הוא מתחבר מרחוק לשרתים שונים וזה ידע שימושי למדי. אנו נלמד כעת איך להתחבר לטרמינל ואיך לעבוד איתו בחלונות. אני לא מסביר על טרמינל במק או בלינוקס כי אני יוצא מנקודת הנחה שמי שיש לו את מערכות ההפעלה האלו יודע איך להשתמש באופן בסיסי בטרמינל.

הפעלת הטרמינל

בחלונות הגישה לטרמינל פשוטה למדי. בחלונות 10 וב-11 לוחצים על הזכוכית המגדלת, מקלידים `cmd` ואז לוחצים על אנטר.



מיד מקבלים מסך שחור. לוותיקים בינינו הוא יזכיר את מסך ה-DOS הישן של לפני 20 שנה.

Command Prompt

```
Microsoft Windows [Version 10.0.17134.950]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\barzik>
```

זה הטרמינל של חלונות. לפני כמה עשורים, כך נראו מחשבים. זה המחשב שלכם, אבל הממשק הוא לא גרפי אלא טקסטואלי. יכול להיות שזה ייראה לכם מיושן ולא רלוונטי, בטח ובטח לעומת הממשקים האחרים שאתם מכירים, אבל ככה רוב המתכנתים עובדים – מול הטרמינל של חלונות, של מק או של לינוקס. חשוב מאוד להבין איך עובדים איתו.

כשתיכנסו לטרמינל, מצד שמאל תוכלו לראות את המיקום שלכם. אצלי המיקום הוא:
C:\Users\barzik

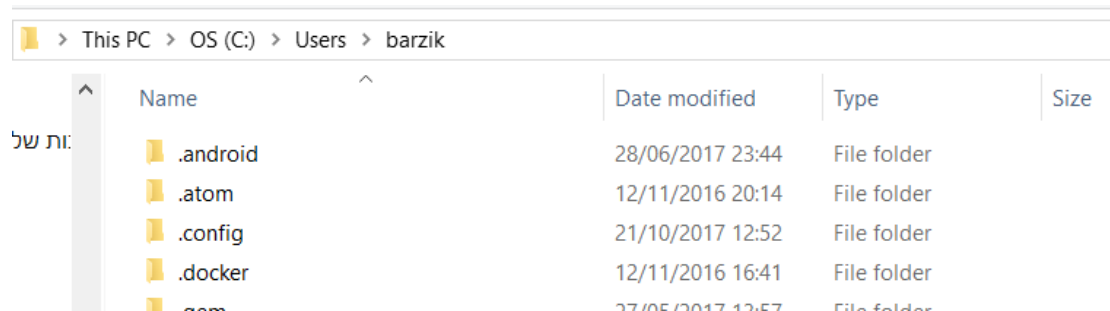
אצלכם המיקום יהיה שונה בהתאם לשם המשתמש שלכם. הבה ננסה להקליד פקודה. הקלידו dir והקישו על אנטר. אנטר בטרמינל הוא "שידור". אם תעשו את זה כמו שצריך, תראו משהו כזה:

```
C:\Users\barzik>dir
Volume in drive C is OS
Volume Serial Number is 0C36-DF83

Directory of C:\Users\barzik

08/17/2019  11:07 AM    <DIR>          .
08/17/2019  11:07 AM    <DIR>          ..
06/28/2017  11:44 PM    <DIR>          .android
11/12/2016  09:14 PM    <DIR>          .atom
07/28/2019  07:30 PM    3,842 .bash_history
10/21/2017  12:52 PM    <DIR>          .config
```

זאת בעצם כל רשימת הקבצים בתיקייה שלכם. אם תפתחו את סייר הקבצים המובנה בחלונות ותחפשו את התיקייה, תראו שמה שהפקודה dir נותנת זהה לתצוגה שלכם.



Name	Date modified	Type	Size
.android	28/06/2017 23:44	File folder	
.atom	12/11/2016 20:14	File folder	
.config	21/10/2017 12:52	File folder	
.docker	12/11/2016 16:41	File folder	
.npm	27/05/2017 12:57	File folder	

ניווט בטרמינל

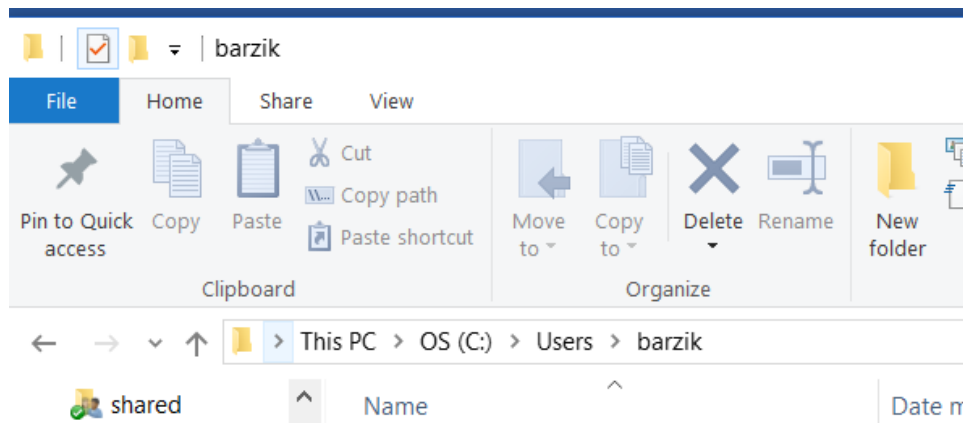
אתם יכולים לנווט בטרמינל ולהגיע לתיקיות אחרות. למשל, אם יש תיקיית Documents במיקום שלכם, אפשר להגיע אליה. נסו להקליד למשל: `cd Documents`. תגיעו לתיקיית Documents שיש תחת השם שלכם. אם תקלידו `dir` ותצפו בתוכן התיקייה, תראו שהיא זהה לתיקיית My Documents מסייר הקבצים. אפשר "לעלות" לתיקייה אחת למעלה באמצעות `cd ..`. נסו לעלות שוב ושוב עד שתגיעו לתיקייה הראשית, הלוא היא `C:`. אנו יכולים לנווט שוב בחזרה באמצעות `cd` ושם התיקייה. אפשר גם להקליד ישירות את הנתיב:

```
C:\Users\barzik\Documents>cd ..
C:\Users\barzik>cd ..
C:\Users>cd ..
C:\>cd Users\barzik\Documents
C:\Users\barzik\Documents>
```

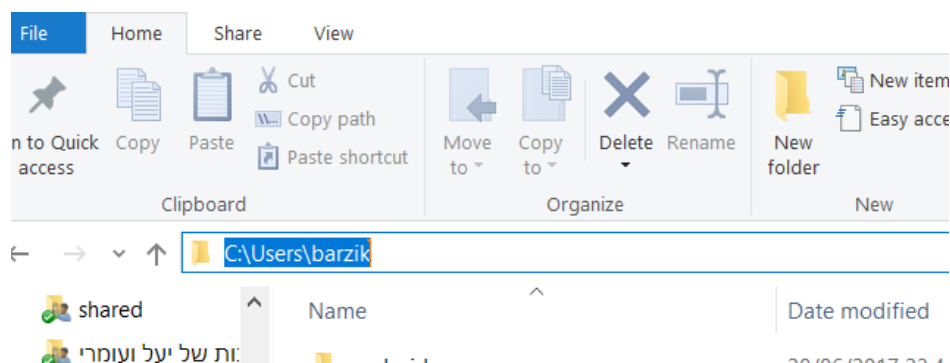
טיפ קטן – אפשר להקליד `tab` כדי לבצע השלמה.

מציאת מיקומים בטרמינל דרך חלונות

אם אתם רוצים להגיע לתיקייה מסוימת ולא בטוחים מה מיקומה, מצאו אותה בסייר הקבצים הגרפי ולחצו על הכותרת. למשל, התיקייה הזו:



אם אני אלחץ על המיקום, אקבל את מיקום של התיקייה בנתיב מסודר שבו אני יכול להשתמש בטרמינל:



אני יכול להעתיק את הנתיב אל הטרמינל. להקליד:

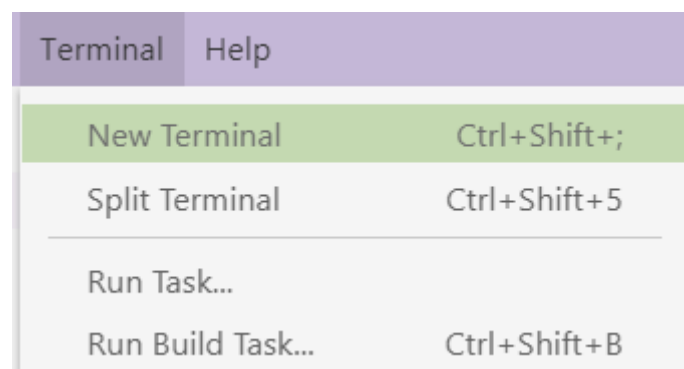
```
cd C:\Users\barzik
```

ואז ללחוץ על אנטר ולהגיע אל היעד המבוקש.

טרמינל ב-Visual Studio Code

אפשר לעבוד עם הטרמינל דרך ה-IDE החינמי Visual Studio Code, שמומלץ לכל מפתח ג'אווהסקריפט. העורך הזה, מבית מיקרוסופט, הוא חינמי, לא מכביד על המחשב וגם קל להסרה. אם אתם לא משתמשים בו, אני ממליץ בחום שתתקינו אותו באמצעות כניסה לאתר הרשמי שלו: <https://code.visualstudio.com/>

אם הוא מותקן אצלכם, פתחו את תיקיית הפרויקט שלכם, לחצו על לשונית הטרמינל למעלה ובחרו ב-New Terminal:



ב-IDE שלכם יופיע חלון של טרמינל שנפתח כבר במיקום הפרויקט שלכם. אפשר גם ליצור תיקיות בטרמינל, למחוק תיקיות, לשנות ולערוך קבצים ולעשות פעולות נוספות. רבים מהמתכנתים עובדים דרך הטרמינל של Visual Studio Code.

בדיקת גרסת Node.js דרך הטרמינל

הפעולה המרכזית שאנו נעשה בטרמינל היא בדיקה שהתקנת ה-Node.js שעשינו תקינה. אנו נקליד:

```
node -v
```

אם נקבל שגיאה, סימן שההתקנה לא הייתה תקינה. נסו להפעיל מחדש את המחשב או להתקין מחדש את Node.js.

אם הכול תקין, תראו את הגרסה של Node.js שהותקנה אצלכם:

```
C:\Users\barzik\Documents>node -v
v10.15.3
```

אם התקנתם את Node.js וכשאתם מקלידים `node -v` בטרמינל אתם מקבלים את הגרסה, אפשר להתקדם לשלב הבא – עבודה עם Create React App.

עבודה עם Create React App

פתחו את הטרמינל ונווטו באמצעות `cd` לתיקייה שאתם רוצים שהפרויקט שלכם יהיה בה. אצלי למשל כל האפליקציות נמצאות בתיקיית `local`. יצרתי תיקיית `local` תחת המשתמש שלי ונכנסתי אליה באמצעות:

```
cd C:\Users\barzik\local
```

לחלופין, פתחו את Visual Studio Code, צרו באמצעותו פרויקט במיקום מסוים ובאמצעות לחיצה על לשונית Terminal ו-New Terminal ייפתח לכם בתחתית התוכנה חלון טרמינל שנמצא במקום של הפרויקט שלכם.

כשאני נמצא בתיקייה שבה אני רוצה להקים את פרויקט הריאקט הראשון שלי, אני מקליד בטרמינל:

```
npx create-react-app my-app --template typescript
```

`npx` היא פקודת הפעלה של Node.js.

`create-react-app` הוא שם התוכנה שאנו מפעילים, Create React App.

`my-app` הוא שם האפליקציה שלנו. כשנפתח תוכנה אמיתית אנו נקרא לה מן הסתם בשם משמעותי יותר. כרגע נבחר ב-`my-app` – האפליקציה שלי.

`--template typescript` הוא הוראה ליצור את הפרויקט עם טייפסקריפט מובנה.

אם יש לכם Node.js במערכת והכול תקין, ההתקנה תעבור בקלות. היא נמשכת כמה דקות טובות, אין מה לחשוש. במהלך הדקות האלו התוכנה מורידה את כל המרכיבים מהרשת על מנת ליצור אצלכם במחשב סביבת עבודה מלאה של ריאקט.

```

Creating a new React app in C:\Users\barzik\local\my-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...

> core-js@2.6.9 postinstall C:\Users\barzik\local\my-app\node_modules\babel-runtime\node_modules\core-js
> node scripts/postinstall || echo "ignore"

> core-js@3.1.4 postinstall C:\Users\barzik\local\my-app\node_modules\core-js
> node scripts/postinstall || echo "ignore"

+ react-dom@16.9.0
+ react@16.9.0
+ react-scripts@3.1.1
added 1455 packages from 685 contributors and audited 903603 packages in 178.887s
found 0 vulnerabilities

Initialized a git repository.

Success! Created my-app at C:\Users\barzik\local\my-app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd my-app
  npm start

Happy hacking!

C:\Users\barzik\local>cd my-app
C:\Users\barzik\local\my-app>

```

זה הפלט שאמורים לראות. בסיום ההתקנה אתם אמורים להיות מסוגלים להקליד שוב בטרמינל. נוצרה לכם תיקייה שנקראת my-app. היכנסו אליה באמצעות:

```
cd my-app
```

והפעילו את המערכת החדשה שלכם באמצעות:

```
npm start
```

מדובר בפקודה של Node.js שפשוט מפעילה סקריפט שנקרא start. אם הכול תקין, הדבר הראשון שתשימו לב אליו הוא שנפתח לכם חלון של דפדפן שמכיל את האפליקציה שלכם! זה לא קסם. מה שתהליך ה-Node.js עושה הוא לייצר שרת על המחשב שלכם מאחורי הקלעים, לטעון את רכיבי הריאקט ואז לפתוח דפדפן שמכוון אל השרת המקומי שלכם יש מאין. אם תסתכלו על כתובת הדפדפן תוכלו לראות שמדובר בכתובת של localhost:3000. הכתובת הזו מחולקת לשני חלקים. localhost הוא בעצם הכתובת של המחשב שלכם ו-3000 הוא הפורט (נתב הגישה). כיוון שהאתר הוא פנימי, אנו לא צריכים ליצור לו דומיין. בשלב מאוחר יותר נלמד איך להעלות את האתר שלנו מהשרת הפנימי אל שרת חיצוני. כרגע זו פשוט סביבת לימוד ופיתוח מצוינת.

אם תפתחו את Visual Studio Code (או כל עורך טקסט אחר) תוכלו לראות שכבר יש מבנה בסיסי לאפליקציה. בתיקיית public אנו נראה שקובץ ה-HTML קיים והוא כבר מכיל:

```
<div id="root"></div>
```

אפליקציית הריאקט גם היא כבר מוגדרת ב-index.tsx וכוללת הפניה לקומפוננטת ריאקט פשוטה:

```
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App />, document.getElementById('root'));

// If you want your app to work offline and load faster, you can
change
// unregister() to register() below. Note this comes with some
pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

איזו קומפוננטה? App. איך מתבצעת ההפניה? באמצעות import, שנרחיב עליו בהמשך. באיזה קובץ נמצאת App? רואים את זה ב-import. בקובץ App.tsx:

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.tsx</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

אם נשנה משהו בקומפוננטה, למשל נוסיף שורה כמו:

<p>

שלום עולם

</p>

ונשמור, נוכל לראות שבמטה קסם, גם האפליקציה שלנו השתנתה ומופיע בה "שלום עולם".



Edit `src/App.tsx` and save to reload.

שלום עולם

[Learn React](#)

זה קורה כי השרת מאזין לכל הקבצים בתיקייה ואם אחד מהם משתנה הוא טוען את עצמו מחדש. נשמע מסובך ופלאי, אם כי זה לא מסובך להבנה וכל מתכנת שמכיר Node.js יכול לייצר כזה דבר. אבל אנחנו לא צריכים להתאמץ – Create React App יכולה לעשות את זה עבורנו. ממש נפלא. בסופו של התהליך יש לנו אפליקציה בסיסית שעובדת ואנחנו יכולים ליצור לה קומפוננטה ראשונה.

שמות הסיומות של קבצים המכילים קומפוננטות של ריאקט הן tsx ולא ts. זו וריאציה של JSX רק עם טייפסקריפט. בפרויקטים שאין בהם טייפסקריפט, הסיומות הן js. אנו כמובן עובדים עם טייפסקריפט אז בכל קובץ שיש בו קומפוננטה של ריאקט או קוד ריאקטי אנו נשמור אותו בסיומת tsx. שם פורמט הנתונים הוא JSX, אך הסיומת היא tsx. אם יש לנו קבצים שיש בהם רק טייפסקריפט ללא קומפוננטות של ריאקט, הסיומת שלהם תהיה ts ולא tsx. אנו נרחיב על כך בהמשך, אבל כרגע כלל האצבע הוא סיומת tsx לקומפוננטות של ריאקט, סיומת ts לקבצים שבהם אין קומפוננטות כאלו.

קשיים ותקלות

אם נתקלתם בקשיים בתהליך ההתקנה של Create React App, אל דאגה! יש לפעמים בעיות שנובעות משינויים במערכת ההפעלה, אנטי-וירוס שפתאום מתערב או בעיה אחרת שנובעת מכך שמערכות ההפעלה שלנו שונות. אבל כדאי מאוד לא לוותר. אם יש הודעת שגיאה – פשוט חפשו אותה בגוגל או בידקו עם בינה מלאכותית כמו צ'אט GPT. Create React App היא כל כך פופולרית, עד שיש תיעוד רב מאוד לגביה ברשת וגם תיעוד של כל תקלה אחרת. קיבלתם שגיאה אדומה ומפחידה? הדביקו אותה בחלונית השאלה של צ'אט GPT עם השאלה: "How to solve", חפשו אותה בגוגל, התייעצו לגביה בקבוצת פייסבוק או באתר stackoverflow. אל תדלגו על בניית סביבת העבודה הזו כי היא הבסיס העיקרי לעבודה עם ריאקט.