

## תודה רבה על הרכישה של הספר!

עבדתי מאוד קשה על הספר זהה: שעות רבות של כתיבה, הגאה, תיקונים ומעבר על תוכרי הערינה. יותר מ-1800 אנשים תמכו בספר זהה ואיפשרו לו לצאת לאור.

הספר אינו מוגן במערכת ניהול זכויות. כלומר ניתן לקרוא אותו בכל התקן ללא הגבלה. אם מתחשך לקרוא גם מהקינדל, גם מהמחשב וגם מהטלפון הנייד אין בעיה להעתיק את הקובץ שלוש פעמים ולשים אותו בתוך כל התקן. מתוך תקווה שהרוכש והתום לא ינצל את האמון שנתתי בו להעתקה סיטונאית של הספר לאנשים אחרים והפצה שלו. אני מאמין שרוב האנשים הוגנים.

העתק זהה נמכר ל:

yaniv.harpaz@gmail.com

בנוסף לדף זה - הקובץ מסומן בטבעת אצבע דיגיטלית - כלומר בתוך דפי הספר נჩבים אים פרטי הרוכש באופן שקוף למשתמש. כדאי מאוד להמנע מהעתקה של הספר לאלו שלא רכשו אותו באופן חוקי. אם ברצונכם להעביר את הספר למשהו אחר במתנה - העבירו לו את הפרטים שלכם באתר ומיחקו את העותק שנמצא ברשותכם.

תודה וקריאה נעימה!



yaniv.harpaz@gmail.com

# פיתוח מעשי בנברית

נן בר-זיך



הקריה האקדמית אונו

Ono Academic College

החוג למדעי המחשב

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

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

מותר לצלט קטעים קצרים מהספר במסגרת הגנת שימוש הוגן, כלומר פסקה או שתים, כאשר מפנים למקור ומציראים את רן בר-זיק כמחבר הספר.

הדוגמאות המובאות בספר זה הן בבעלותו של רן בר-זיק, אסורה להשתמש בהן בתוקן תוכנות שתפתחו. אם תרצו להכנסי אותן לפרויקט שלכם, שלחו מייל ונדבר על זה.

# פיתוח מעשי בעברית

## רן בר-זיק

מהדורה: 1.0.1

# תוכן העניינים

<b>על הספר .....</b>	<b>9</b>
<b>על המחבר .....</b>	<b>10</b>
<b>על העורכים הטכניים .....</b>	<b>11</b>
gil pinck .....	11
navor zvada .....	11
roni orben .....	11
shachar tal .....	12
הדר ספיק .....	12
<b>על החברות התומכות .....</b>	<b>13</b>
טורי .....	13
ויקו .....	13
<b>הקדמה .....</b>	<b>13</b>
ידע מקדים שנדרש לקריאה בספר .....	14
<b>מערכת ניהול גרסאות Git .....</b>	<b>17</b>
התקנה .....	18
לינוקס .....	18
מתק .....	18
חלונות .....	19
חלונות: הסבר על תוכנתה .....	29
קובץ קונפיגורציה .....	31
תחילת העבודה בטרמינל .....	31
יצירת ריפודיטורי מקומי .....	32
קומיט: הכנסת קבצים ושינויים לגרסה .....	33
שחזור קומייט .....	40
מציאת הבדלים ביי הקומייטים .....	40
שימוש ב- <code>git ref</code> מוקוצר .....	42
בראנץ' .....	42
פקודה מקוצרת ליצירת בראנץ' ומעבר אליו .....	46
חיבור בין שני בראנץ'ים באמצעות <code>merge</code> .....	47
שם הבראנץ' של ברירת המחדל .....	48
מחיקת בראנץ' .....	48
יצירת בראנץ'ים מבראנץ'ים .....	48
קונפליקטים .....	50
פתרון קונפליקטים והבנת התחביר של הקונפליקט .....	54
git rebase .....	55

59.....	<b>קובץ gitignore</b>
60.....	למה חשוב להשתמש ב-.gitignore
61.....	<b>git detached</b>
62.....	<b>git bisect</b>
65.....	<b>עבודה עם שרת גיט מרוחק</b>
66.....	פתרונות חשבון בGITLAB
68.....	משיכת ריפורטורי
71.....	דחיפת קוד
79.....	דחיפת למועד המרוחק
80.....	פול ריקווט: מיזוג בראנץ' בrifozitouri מרוחק
87.....	Git flow
89.....	דרכי המרג'
90.....	<b>git tag</b>
91.....	<b>Release</b>
92.....	<b>קובץ README ו-Down</b>
94.....	<b>תרומה לקוד פתוח</b>
95.....	fork (fork)
97.....	עדכון הריפורטורי
99.....	בקוד פתוח git flow
100.....	<b>Github GUI</b>
106.....	<b>VSCode</b>
107.....	<b>Git blame</b>
108.....	<b>GitHub Actions</b>
109.....	<b>תהליכי CI</b>
110.....	<b>תהליכי CD</b>
110.....	<b>תרגול גיט</b>
111.....	יצירת ריפורטורי עם בראנץ' main
112.....	יצירת initial commit
113.....	יצירת בראנץ'
115.....	מיזוג בראנץ'
115.....	יצירת בראנץ' main
117.....	יצירת בראנץ' נוספת main
117.....	פתרון
119.....	פתרון קונפליקטים
124.....	<b>סיכום</b>
126.....	<b>פרויקטים לדוגמה</b>
126.....	למה זה חשוב?
126.....	איך לומדים מהפרויקטים לדוגמה שיש בספר?
128.....	הכנסת פרויקט הדמו אל קורות החיים
130.....	<b>ידג'ט ג'אווהסקריפט המציג תמונה ממ פופולרי</b>
130.....	הגדרת הצורך
130.....	הבנייה הוועיגט
131.....	תכנון
131.....	יצירת התשתיות

132	הפעלת serve המועד לפיתוח סטטי.....
132	<b>יצירת קרייה לשרת .....</b>
135	<b>הכנסת המידע ל-DOM .....</b>
139	<b>סיכון .....</b>
142	<b>אפליקציית ריאקט לבדיקת משתמש בגייטהאב .....</b>
142	הגדרת הצורך .....
142	הבנת האפליקציה .....
143	תכנון .....
144	יצירת התשתיות – <b>create react app .....</b>
144	יצירת הקונטינר .....
145	יצירת טופס החיפוש .....
147	יצירת ה- <b>service .....</b>
150	<b>קומפוננטת List .....</b>
151	לחבר את הכל ייחד .....
154	<b>סיכון .....</b>
156	<b>אפליקציה מורכבת לניהול מעקבים על אתרי אינטרנט .....</b>
156	הגדרת הצורך .....
156	הבנת האפליקציה .....
157	תכנון .....
158	<b>סביבת העבודה והפיתוח .....</b>
158	הקמת מסד נתונים .....
160	יצירת התשתיות של js .....
166	התקנת eslint .....
167	התקנת nodemon .....
167	<b>בנייה התצוגה .....</b>
167	יצירת הדף הראשון – של טופס החיפוש .....
168	יצירת דף הצגת התוצאות .....
173	<b>בנייה התשתיות של תקשורת עם מסד הנתונים .....</b>
175	ابتחת מידע .....
177	<b>קריאה ממסד הנתונים .....</b>
179	<b>כתיבת למסד הנתונים .....</b>
183	<b> כתיבת הקוד שמבצע סריקה .....</b>
183	סריקה .....
184	חיפוש במידע המגיע מהסሪקה .....
185	קריאה וסריקה והכנסה לאובייקט .....
185	<b> לחבר את הכל .....</b>
189	<b>יצוא מסד הנתונים לגיט .....</b>
192	<b>סיכון .....</b>
194	<b>דיפול .....</b>
195	<b>אפליקציה סטטית או לא סטטית .....</b>
196	<b>אפליקציה סטטית .....</b>

<b>דיפלוי של אפליקציה סטטית באמצעות שירות אחסון שיתופי</b>	<b>198</b>
<b>דיפלוי של אפליקציה סטטית באמצעות GitHub pages</b>	<b>200</b>
התקנת gh-pages	201
<b>דיפלוי של אפליקציה סטטית באמזון</b>	<b>202</b>
יצירת חשבון באמזון	203
כניסה ל-amplify	203
<b>אפליקציה דינמית</b>	<b>208</b>
<b>דיפלוי באמזון</b>	<b>209</b>
יצירת המكونה	210
התקנת Node.js	215
התקנת MySQL	215
הצגת האפליקציה	218
<b>דיפלוי ב-heroku</b>	<b>219</b>
פתיחה של חשבונן והגדרות ראשוניות	219
יצירת האפליקציה	221
הדיפלוי של הקוד	225
חיבור מסד הנתונים לאפליקציה	232
Łosiscom	234
<b>תרומה לקוד פתוח</b>	<b>236</b>
מהו קוד פתוח	237
מי? אני?	238
מה עדיף – פרויקט גמר או קוד פתוח?	238
איתור פרויקט שכדי לתרום לו	239
<b>דוגמאות לתרומה לקוד פתוח</b>	<b>242</b>
תרומה ל-Verdaccio	243
תרומה ל-dockly	243
<b>הכנסת התרומה לקוד הפתוח אל קורות החיים</b>	<b>244</b>
פרסום פרויל הגיטהאב שלכם	244
פרסום התרומה לקוד הפתוח	245
הכנסת התרומה לקורות החיים	245
חשיבות תרומת הקוד גם למפתחים נוספים יותר	246
<b>סיכום – מה עושים עכשווי?</b>	<b>248</b>

## על הספר

הספר "פיתוח ווב מעשי" נועד ללמד אנשים שיש להם ידע בתכנות כיצד לפתח באופן מעשי. יש הבדל משמעותי בין הידע הנדרש לתכנות לבין הידע הנדרש לפיתוח תוכנה ממשית באופן מקצועי. הידע הנדרש להשתלבות בתחום ההיינט והפיתוח כולל מידע רב שאיןו תכנות – למשל שליטה בתוכנת גיט לניהול גרסאות, דיפולימנט והעלאת התוצרים לשרתים או לסביבות ענן, תכנון פרויקטים מאפס וכן תרומה לקוד הפתוח ושילוב התרומה זו ופרויקטים לדוגמה בקורות החיים.

מטרת הספר היא לגשר על פערו הידוע בין מי שיודע לתכנת לבין מי שעבד כמתכנת מקצועי; בין מי שישים תואר, תוכנית הכשרה או לומוד עצמי לבין מי שמצילח להשתלב בתחום ההיינט כمفחה או לתרום קוד. בספר, שהוא המשך ישיר לספר הליימוד הקודמים של רן בר-זיק – "לلمוד ג'אווהSCRIPט בעברית", "לلمוד Node.js בעברית", "לلمוד ריאקט בעברית" ו"לلمוד MySQL בעברית" – לומדים כיצד אפשר לתרום את הידע בתכנות לידע מקצועי של ממש, לרבות כיצד לתרגם את הידע המעשי לקורות חיים.

## על המחבר

רן בר-זיק מפתח תוכנה משנת 1996 ב מגוון שפות ופלטפורמות ועובד כארქיטקט תוכנה ומפתח בכיר במרכז פיתוח של חברות רב-לאומיות, מ-HP ועד Verizon וסולוטון, שם הוא מפתח בטכניקות מתקדמות, הן מצד הלוקה והן מצד השרת, ושם דגש על בניית תשתיות פיתוח נכונה, על שימוש ב-CDI וקמובן על אבטחת מידע.

נוסף על עבודתו כמפתח במשרה מלאה, רן הוא עיתונאי ב"הארץ" 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 והסבירים על תכונות בעברית ומתעדכן לפחות פעם בשבוע.

倫 是 一個 有 許 多 孩子 的 父親。他 是 一個 喜 愛 技術 和 网絡 的 人，特 别 是 在 Internet-israel.com 上。他 也 是 一個 善 良 的 父親，和 他的 孩子 一起 玩耍 和 学习。

ספריו הקודמים: "לلمוד ג'אווהSCRIPט בעברית", "לلمוד Node.js בעברית", "للمוד jQuery בעברית", "للمוד MySQL בעברית" והספר הקצר "للمוד jQuery בעברית".

# על העורכים הטכניים

## gil fink

gil fink הוא מומחה לפיתוח מערכות ווב, Web Technologies Google Developer Expert, Microsoft Developer Technologies MVP והמייסד של חברת sparXsys. כיום הוא מייעץ לחברות ולארגונים שונים, שם הוא מסייע בפיתוח פתרונות מבוססי אינטרנט ו-SPAs. הוא עורך הרצאות וסדנאות ליחידים ולחברות המעונייניות להתחמות בתשתיות, בארכיטקטורה ובפיתוח מערכות ווב. הוא גם מחבר של כמה קורסים רשמיים של מיקרוסופט (Microsoft Official Course – MOC), מחבר משותף של הספר "Single Page Application Development" (בհוצאת Apress) ושותף בארגון הכנס הבינלאומי AngularUP.

לפרטים נוספים על gil: <http://www.gilfink.net>

## נאור זבדה

נאור החל את דרכו המקצועית כמפתח מערכות ווב בבית תוכנה, בפיתוח מוצרים משלב האפיון ועד מסירת המוצר ללקוח. במהלך הקריירה עבד במגוון חברות וצבר ניסיון רב בתחום פיתוח הכוללים ארכיטקטורה, תכנון והקמה של מערכות מורכבות המיעילות מאות אלפי משתמשים. כיום עובד כמפתח בכיר בחברת Security, חלק מצוות SaaS ואחראי על פיתוח מנגנוןים שמשפרים את חווית הלפקות. בנוסף על כך הוא עוסק בבניית כלים לייעול תהליכיים וכלי פיתוח עבור המפתחים בחברה.

לנאור תואר B.Sc. במדעי המחשב מטעם המכללה למנהל. חלק מתרומותיו להילת המפתחים בארץ הוא פועל כמנטור למפתחים צעירים בתחום דרכם.

## רוני אורבן

בסיסת חובב, התחליל לפתח תוכנות ומשחקים בתורו לצד בשנות ה-90 ומazel 2001 עובד עם שחר טל בפיתוח Web.

## שחר טל

גיימר וחובב קפה שמשחק עם קוד מאז קיבל את ה-XT שלו בגיל ארבע. בקאנדייסט שמנר את נשמתו לפירונטאנד ב-2001 בעקבות השותף שלו, רוני אורבן, ומazel לא הסתכל לאחור.

## הדר ספיבק

כותב קוד מגיל 14, מפתח fullstack עם התמחות ב-ML בשנים האחרונות. בוגר ייחידת מצפ"ן/ממר"ם, לאחר שירות של שבע שנים ביחידת התוכנה המבצעית של אגף התקשוב. היה ה-CTO co-founder & Fixel, שנמנר לחברת Logiq האמריקאית, וכיום הוא ה-VP R&D שלה. בעל חיבת מוגמת ל-apis REST.

# על החברות התומכות

## טורי

Torii - הינה פלטפורמת ניהול אוטומטי של תוכנות ענן, שנוצרה עם החzon לבנות תוכנה שמנהלת תוכנות אחרות.

הטכנולוגיה של Torii מאפשרת איסוף ועיבוד נתונים ממספר רב של מקורות מידע, והציג תובנות עסקיות העוזרות לחסוך כסף, לאבטח את המידע הארגוני וליעיל תהליכיים בצורה אוטומטית.

בעזרת ממשק code-api המותם בפלטפורמה, ללקוחות החברה יכולים ליצור בקלות אוטומציה לניהול תוכנות הענן.

ארquitektורת Serverless 100% Torii – לשונת עליה, מאפשרת לצוותי הפיתוח פיתוח מהיר מאוד, תוך כדי התמודדות עם כמויות מידע גדולות אשר היא מעבדת במהירות מכל תוכנות הענן הארגוניות.

מטה חברת Torii ממוקם בניו יורק, ומרכז הפיתוח בישראל.

## ויקס

Wix היא פלטפורמה גלובלית מוביילית לייצור, ניהול וצמיחה של עסקים אונליין. Wix Engineering היא חטיבת R&D של Wix, והוא מהווה כ-50% מכוח העבודה שלנו היום. חטיבת הפיתוח שמה דגש רב על Engineering Culture, הגילדות המקצועית, הפיתוח האישי-מקצועי של המפתחים והפתחות - והוא מתמקד במתודולוגיות פיתוח מוביילות במטרה להתמודד ולפתור את הבעיות המאתגרות ביותר בסkill הגבוה ביותר באינטרנט כיום. חלק מתרבות הפיתוח שלנו לפחות 20% זמן העבודה מוקדש להכשרה המפתחים והפתחות, תוך עידוד וסיווע לחלוק את הידע והניסיונות הקיימים והנלמד עם הקהילה וلتרום לפרויקט קוד פתוח. Wix Engineering הוא המקום ללמידה אין' עובדים בסkill, אין' מתמודדים ומ畢ינים לעומק אתגרי פיתוח מורכבים ואיך מקודדים באופן נכון ונכון.

## הקדמה

הספר זהה נולד כתוצאה מתרסcole. מתווך מה שאני עושה והנראות הציבורית שלי, אני מקבל לא מעט מיילים ופניות של אנשים לעזרה. מייל אחד ממש נגע ללבבי. שלח אותו אדם בעל תואר ראשון, שלם והשקייע בלימודים, אך לא הצליח להשתלב בתחום. לא משנה איך למדתם – לימודי תואר ראשון במדעי המחשב באוניברסיטה, או במכיללה, בבודקאמפ, בבית ספר לתכנות או לימודי עצמי – האתגר הקשה ביותר הוא למצוא את העבודה הראשונה. ובעולם שבו כל עסק פוטנציאלי רוצה אנשים עם ניסיון, נהיה קשה יותר ויותר להשיג את הניסיון הזה. יש קיצורי דרך, כמו שירות מיוחדת טכנולוגית או שירותי התמחות בחברות גדולות, אבל מה עם השאר? איך אפשר להשיג ניסיון כשאף אחד לא מוכן לתת לכם סיינו? זו הייתה כוורתה המיל שקיבלת מאדם שלא הצליח להשתלב בתחום, אף על פי שלם והשקייע.

קיבلت את המיל לבדוק כשהבן הגדל שלי מצא, ללא תואר, עבודה כمفצת בחברה רב-לאומית גדולה. מה ההבדל בין שני המקרים? לבן שלי היה אבא, שהדריך אותו והסביר לו איך רוכשים ניסיון ואייך משמרים אותו. איך תורמים לקוד פתוח, איך יוצרים פרויקט לדוגמה, לומדים מהם וממנפים את הידע הזה לקורות החיים. בעצם, איך רוכשים את הניסיון לפני העבודה הראשונה. לאדם השני לא היה אבא צזה. וזה הרגיז אותי. כי לעיתים כל מה שצורך זה הכוונה מקצועית נcona. כך הספר הזה נולד.

כתבתי חמישה ספרי תכנות בעברית שמגדים לתוכנת. אף-על-פי אנשים רכשו אותם, משתמשים בהם בבתי ספר ובקורסים שונים. אבל ליום תכנות איננו מקצוע. אפשר לדעת לתוכנת היטב, אבל תכנות כמקצוע הוא לעובד עם גיט, לבנות פרויקט מקצה לקצה, ל懂得 לפתח אותו על שרת. הספר הזה נוסע לבדוק למטרת הזו – להסביר לאלה שיעודים לתוכנת איך מגשרים על הפער בין ליום תכנות לבין מקצוע של ממש.

## ידע מקדים שנדרש לקריאה בספר

הספר הזה הוא המשך ישיר לספרים "למוד ג'אווה סקריפט בעברית", "למוד jsNode", "למוד ריאקט בעברית" ו"למוד MySQL בעברית". אני יוצא מנקודת הנחה שהקורא קרא את הספרים האלה ותרgal את מה שכתוב בהם. שהוא לא ייבהל ממוניים כמו "הרץ בטרמינל את הפקודה `itoh wkh`" ויבין את משמעותה. **שידע** איך שרת אינטרנט עובד ושמימוש שרת זהה ב-`Node.js` או עם `create react app`. זה לא אומר שאנשים

שקראו רק את "למוד ג'אווהסקריפט בעברית" לא יפיקו ממנה ערך, אבל אני ממליץ בחום רב לקרוא את כל ארבעת הספרים לפני הקריאה בספר זהה.

החומר הנדרש הוא:

1. **שליטה בטרמינל** (لينוקס/מcker או חלונות) בrama מספקת לנישוט במיקומי התיקיות ולהקלחת פקודות.
2. **שליטה בעורך קוד (IDE)**. הדוגמאות המובאות בספר מסתמכות על **Visual Studio Code**.
3. הבנה טובה בג'אווהסקריפט מודרני – קלומר ES2017 ותכנות אסינכרוני באמצעות **await** ו**async** וקלאסים בסיסיים.
4. הבנה בהתקנת מודולים של **npm**, בפקודות **npm init** ו**npm install** וכן במבנה **package.json**.
5. הבנה איך שרת אינטרנט עובד והבנה בסיסית באקספרס ומימוש בסיסי שלו.
6. הבנה בקומפוננטות של **React** בrama הבסיסית ובקומponentות של **create react app**.

פרק 1

# מערכת ניהול גרסאות GIT



## מערכת ניהול גרסאות GIT

עד כה, כאשר למדנו תוכנות, כתבנו את התוכנה בקובץ. אבל יש בעיה מהותית בכתיבה קוד בקובץ פשוט ובשמירתו. אם הקובץ נמחק בגלל תקלת כלשהו במחשב – כל העבודה שעשינו הלה. אם אנו רוצים לחלוק את הקובץ הזה עם מישחו אחר או לעבוד על הקובץ במקביל יחד עימו – מדובר מסורבל לשלווח לו אותו ואז להכניס את השינויים שלו לתוך הקובץ. הבעיהגדלה כאשר יש לנו כמה וכמה קבצים, נמקובל במערכות קוד גדולות יותר שעובדים עליון כמה וכמה מתכנתים. גם אם יש תקלת, למשל כזו שנוצרה בغال קוד שהוספנו לאחרונה, יש צורך לבצע חזרה למצב הקודם (זה נקרא *back roll*) כדי להחזיר את המצב לקדמותו.

זו הסיבה שבגללה כל המתכנתים שעובדים בסביבה מודרנית משתמשים במערכות לניהול קוד. יש לא מעט מערכות כאלה, אבל המערכת הפופולרית ביותר, שנחיפה ממש לסטנדרט, היא מערכת GIT (Git). GIT נוצרה על ידי לינוס טורබאלדס, ממציא הלינוקס, והיא מערכת מבוזרת לניהול גרסאות שפותרת היטב את כל בעיות האלו.

מערכת לניהול גרסאות היא מערכת המאפשרת לנו לנהל גרסאות שונות של אותו קודבי מידע וולעקו אחריהם. הקבצים יכולים להיות קובצי תוכנה, תמונות או כל קוד אחר, ומערכת לניהול גרסאות מאפשרת לנו ליצור, למחוק או לעורך גרסאות שונות של אותו מידע.

GIT החליפה דה פקטו מערכות רבות לניהול גרסאות, כמו *subversion*, *mercurial* ו-*TFS*. השימוש בה הוא חלק חשוב מArsenal הכלים שיש למפתחים ונדרש מהם להכיר אותה. בחלק הראשון של הספר זהה אנו נלמד איך עובדים איתה.

העבודה עם GIT נעשית באמצעותCLI – בטל ממשק גרפי ( – GUI) או באמצעות הטרמינל (CLI – Terminal). כאן נלמד להשתמש בה עם CLI. השימוש ב-CLI – או עבודה מהטרמינל – קשה קצת יותר למידה מאשר כלים עם ממשק גרפי כיון שהוא מושך מעט ידע באנגלית, ההסבר הוא על מהות ולאו דווקא על תפריטים, כמו כן חלק גדול מהאנשים מסתדרים טוב יותר עם ממשק גרפי. אבל דרך ה-CLI קל יותר ללמידה היבט,ומי ששולט ב-CLI מבין את העקרונות האמיטיים של GIT ושולט בה הרבה יותר טוב, לדעתי.

בתחילת הלימוד של GIT נראה תלוש, אפילו מיותר, במיוחד מנקודת מבטו מתקנות בתחילת דרכם של הוטים לנכוב קוד שעשוה דברים. אבל הלימוד של GIT והשליטה בה חיוניים כיוון שאמור – מדובר בסטנדרט ואינו ביום חברה שלא משתמש בגיט או בכלי ניהול גרסאות דומה. יש הבדל בין לדעת לתוכנת לבין להיות מקצועית ולהבין גם דברים מעבר לתוכנות, וגיט היא אחד ההבדלים המרכזיים בין השניים. מטרת החלק הזה היא לforge על הפער ולאפשר לכם לרכוש ניסיון עוד לפני העבודה הראשונה. נוסף על כן, אי-אפשר לתרום לקוד פתוח או לפתח פרויקט לדוגמה לתצוגה ללא שימוש בגיט. כלומר אפשר, אבל זה נחשב הרבה-הרבה פחות מקובל והרבה פחות מקצועית. אולי זה לא מלהיב כמו למוד שפת תוכנות, אבל זה חשוב לא פחות, ואולי אפילו יותר. גם אנשים לא טכניים משתמשים בגיט. ישנן מדיניות, כמו גרמניה למשל, שבהן ספר החוקים מופיע בגיט ואזרחים מן השורה או חברי פרלמנט (בהתאם לחוק ולשלב שבו הוא נמצא) יוכולים להציג שינויים.

לכן ידע בגיט הוא חיוני בכל הנוגע לעולם המודרני בכלל, ולפיתוח בפרט.

## התקנה

GIT היא תוכנה שמותקנת על מערכת הפעלה המקומית כמו כל תוכנה אחרת. ההתקנה שלה פשוטה מאוד ומילויוני מפתחים מתקנים אותה במחשבים שלהם מדי יום.

## لينוקס

ההתקנה בלינוקס נעשית באמצעות הפקודה:

```
sudo apt install git-all
```

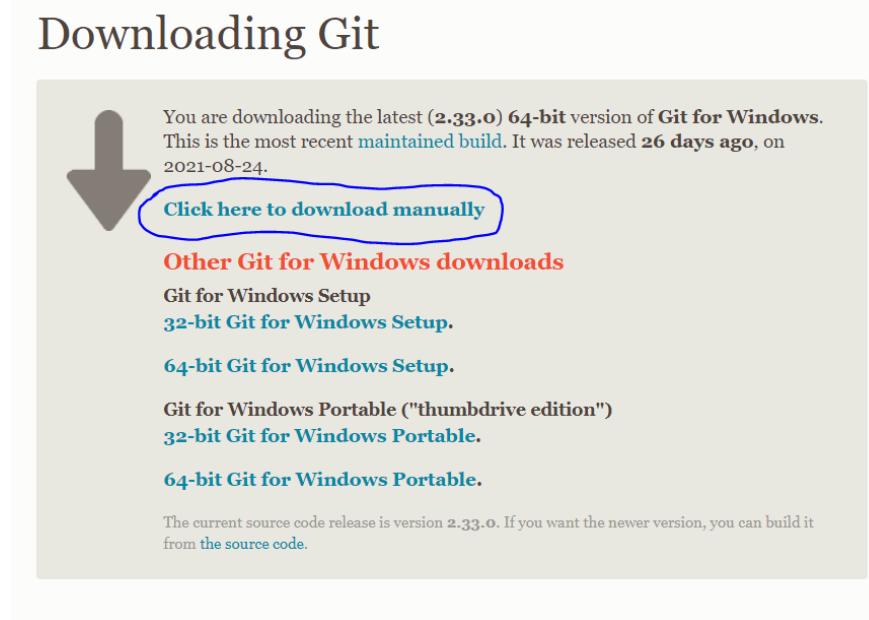
## מק

במק GIT באה באופן דיפולטיבי עם Xcode. אפשר להתקין את Xcode, שהוא חבילת תוכנות רשמית של Mac, באמצעות חנות האפליקציות של Mac.

## חלונות

בחלונות מזכיר בתוכנה קטנה שיש להפעיל ומורידים אותה מפה:  
<https://git-scm.com/download/win>

אם התוכנה אינה יורדת אוטומטית כשנכנים לדף, מומלץ ללחוץ על [download manually](#).

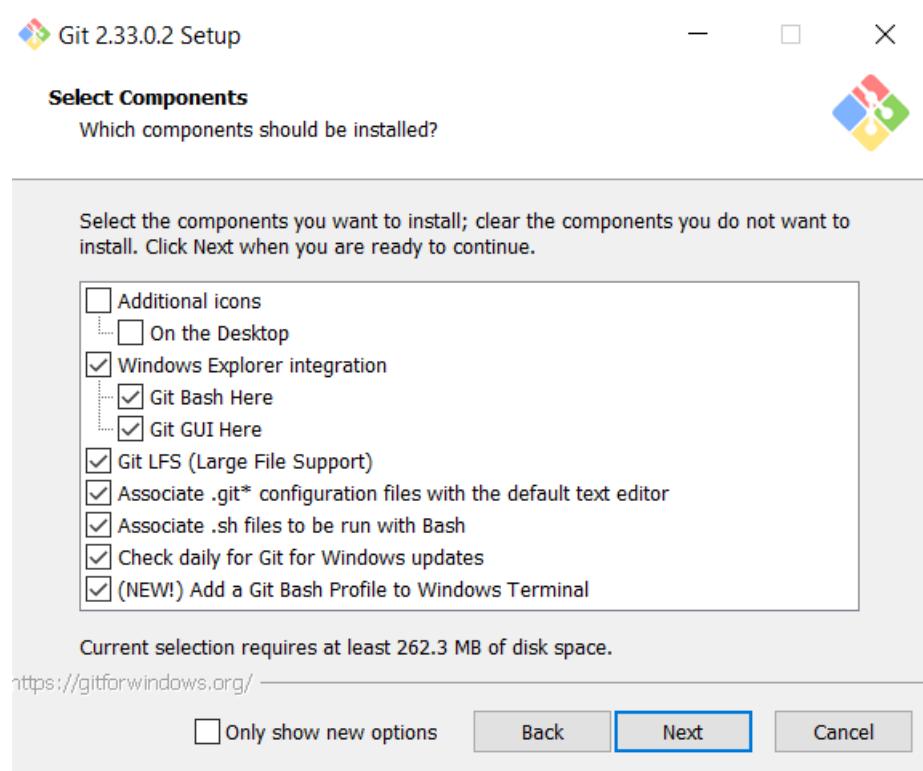


אחרי ההורדיה יש לפתח את הקובץ. ייפתחו מסכי התקנה שבתוכם אפשר ללחוץ על [Next](#), למעט מסך אחד שארחיב על חשבותו מאוחר יותר.

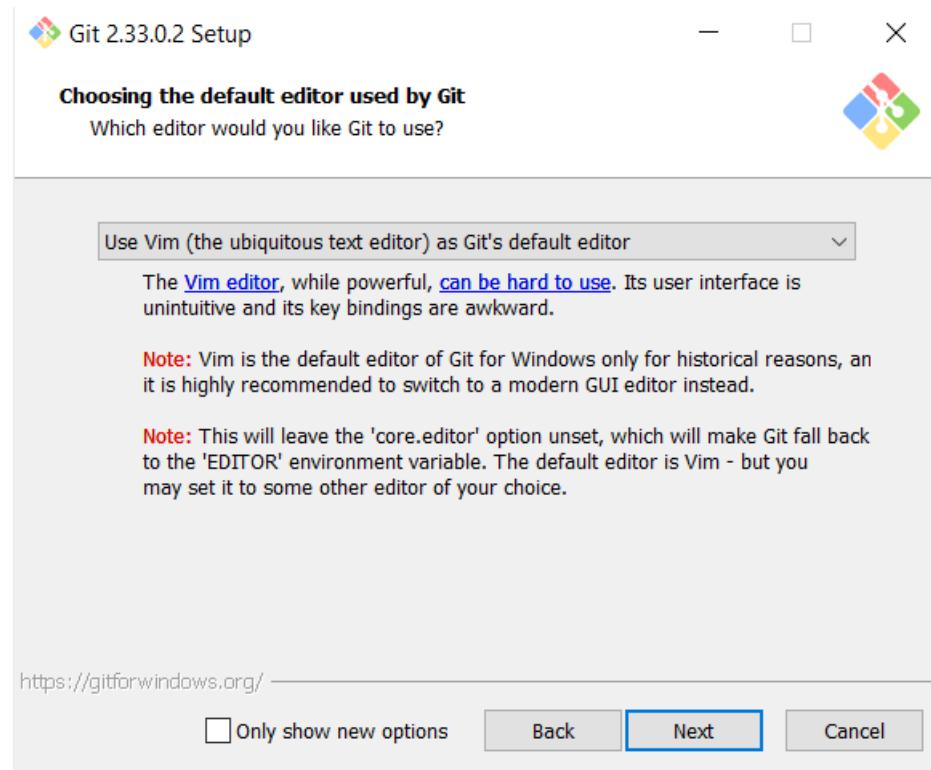
צריך לזכור שטבען של תוכנות לשתנות. אם יש שינוי במסך ההתקנה בין הגרסה שיש לבין הגרסה שיש בספר זה – אין סיבה להילחץ. ברוב המקרים צריך רק ללחוץ **Next**.



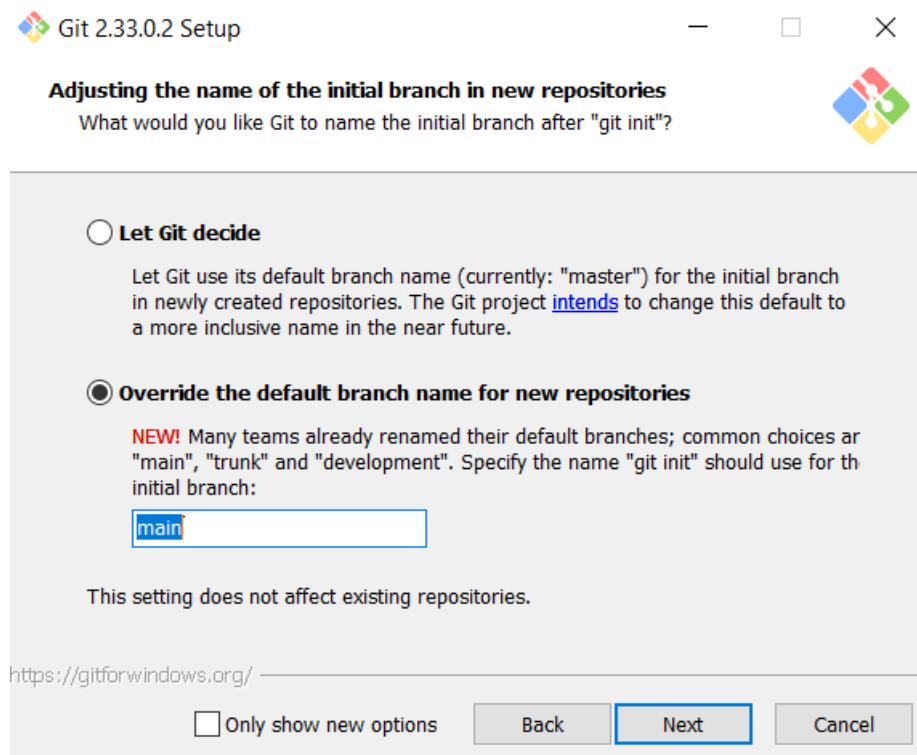
המסך ההתחלתי עם תנאי הרישיון – רישיון GNU בקוד פתוח. אפשר ללחוץ על Next.



זהו מסך שבו בוחרים את האינטגרציה בין חלונות לgit. מומלץ לסמן את הכל:

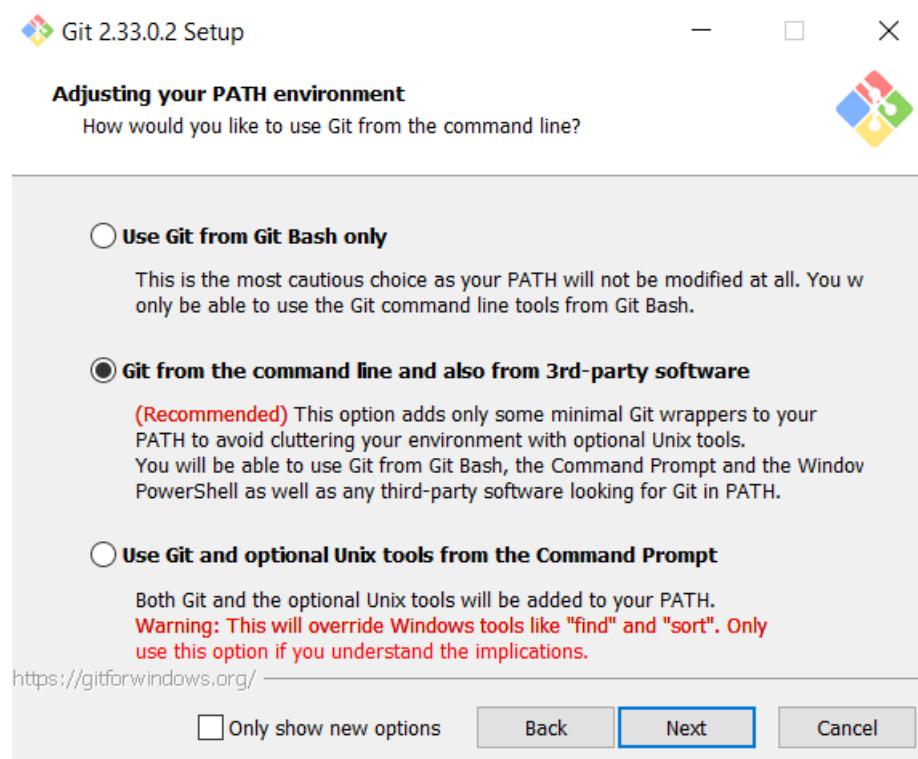
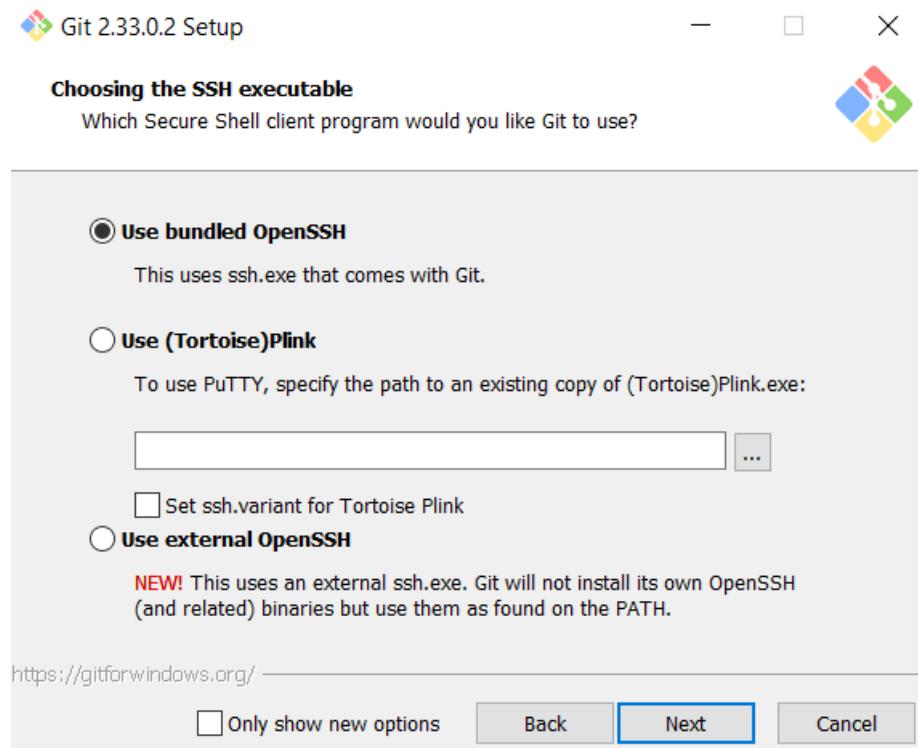


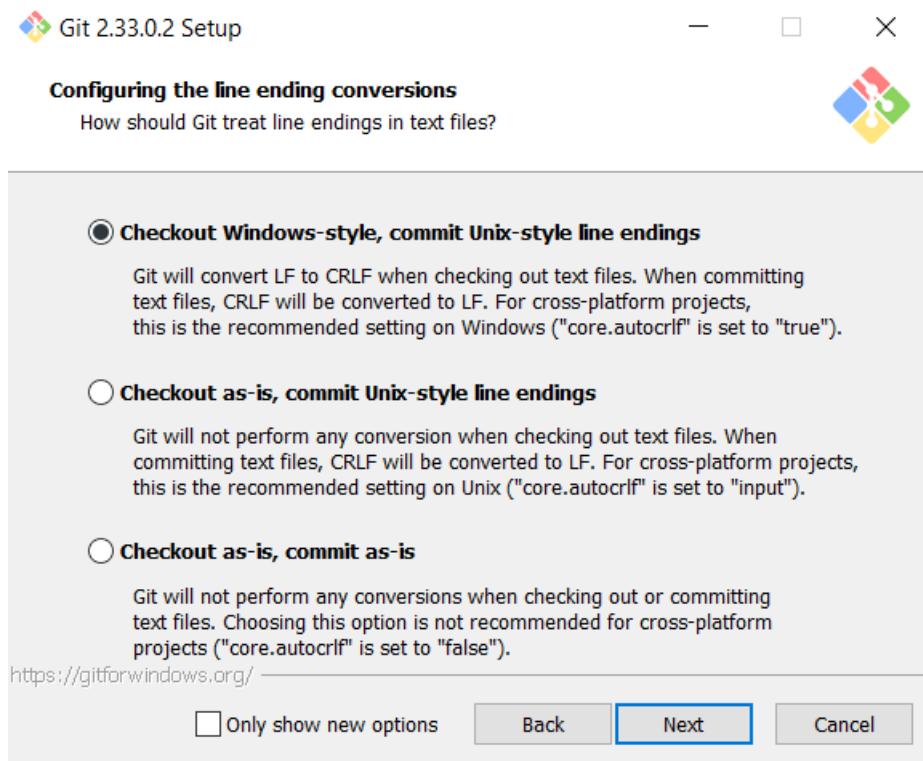
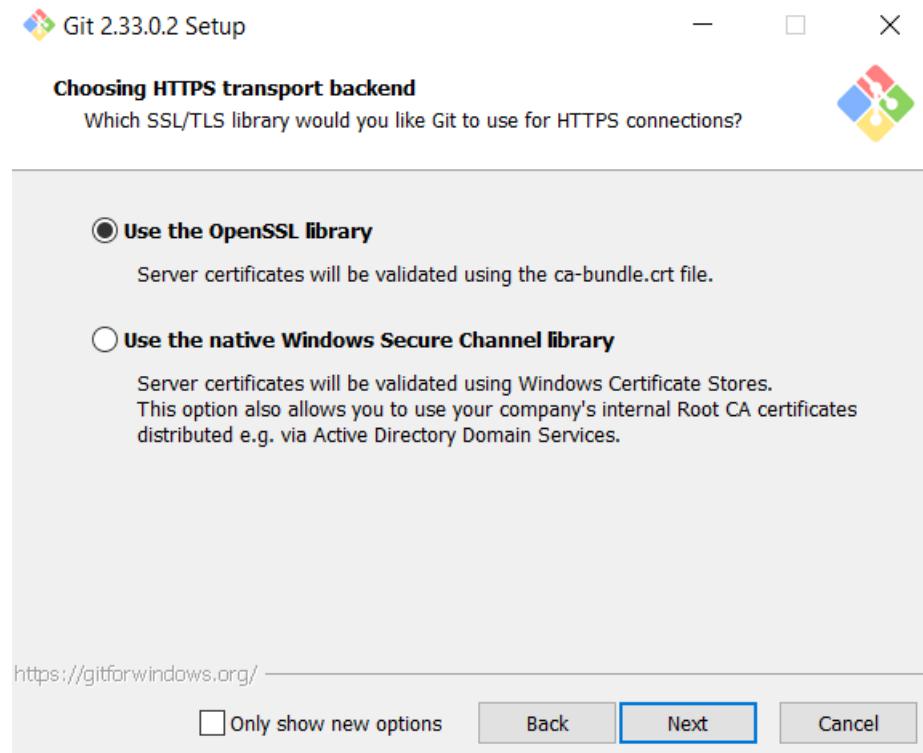
כאן מומלץ לבחור את Nano או כל עורך טקסט אחר, אבל אפשר להשאיר גם את Vim.

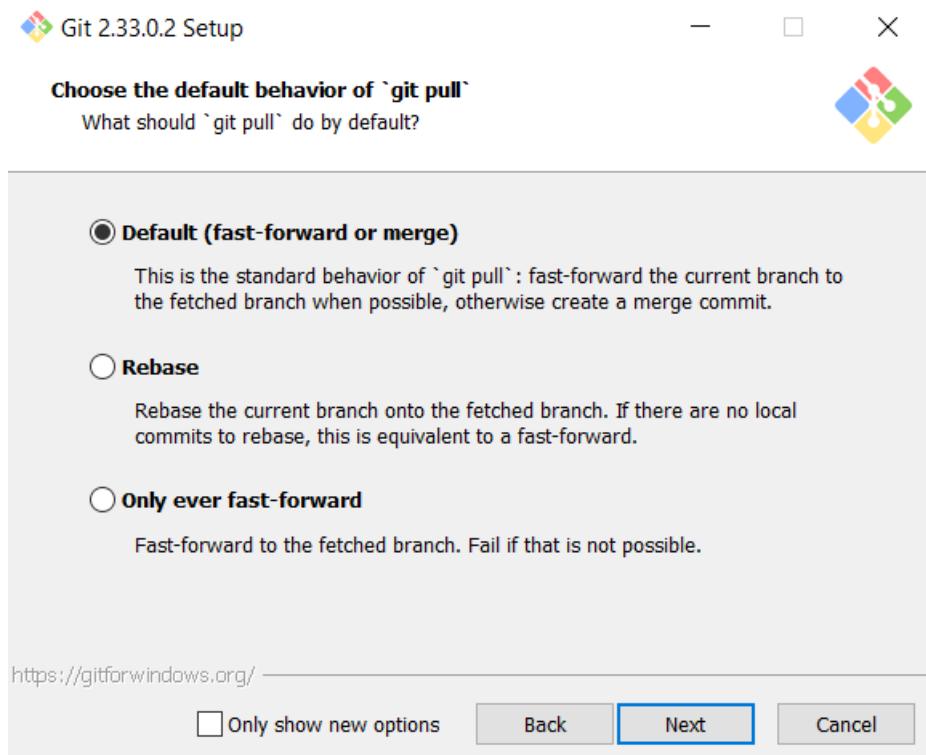
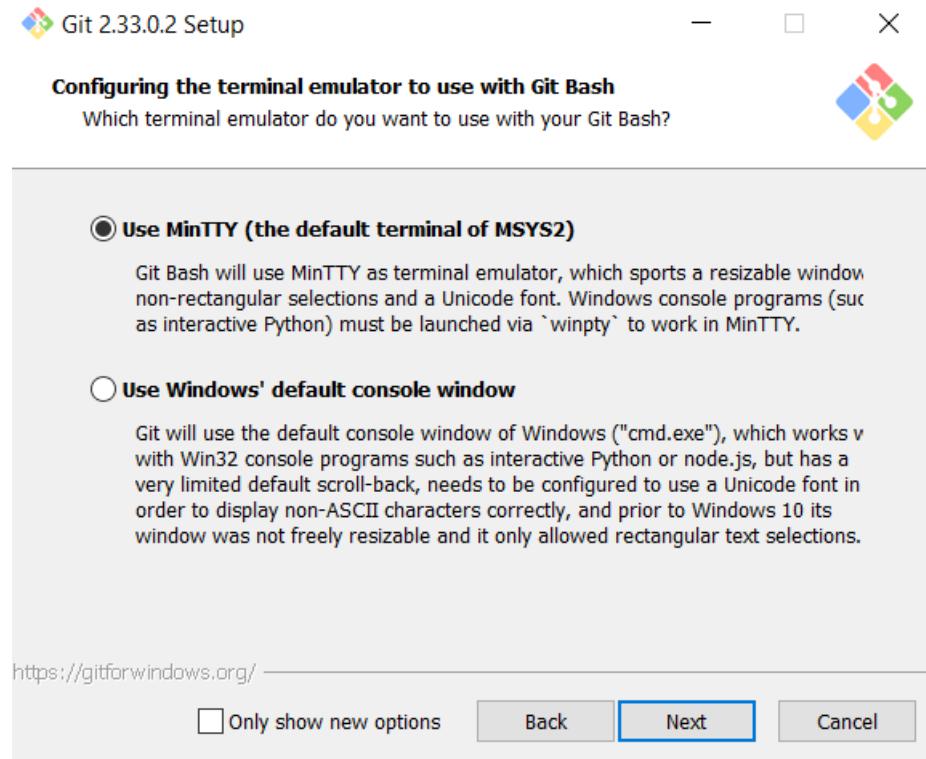


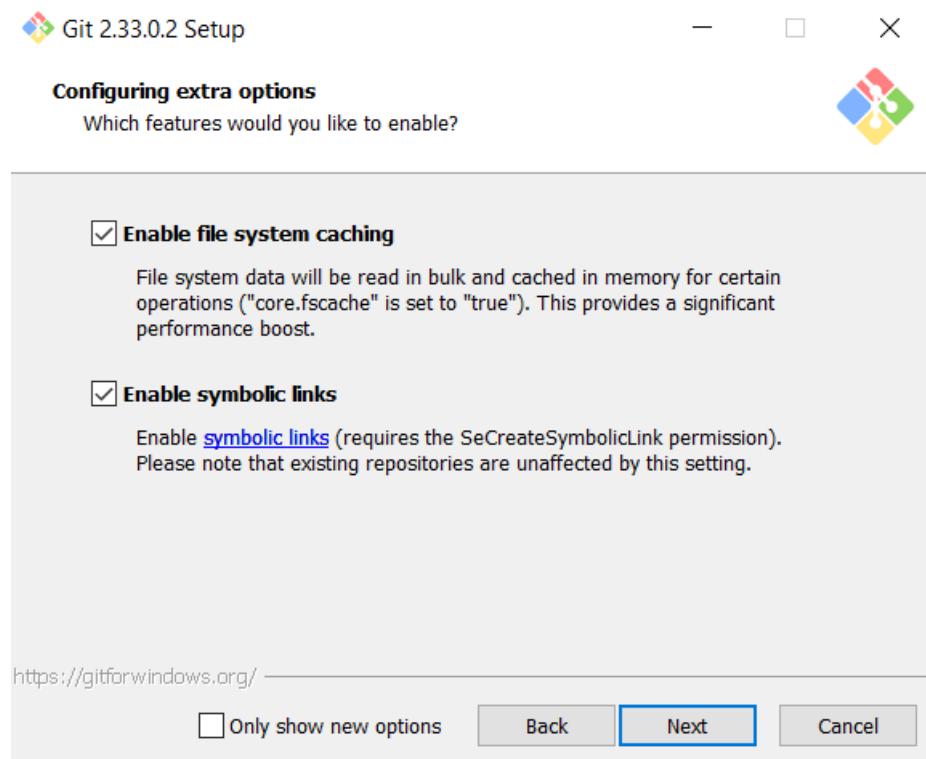
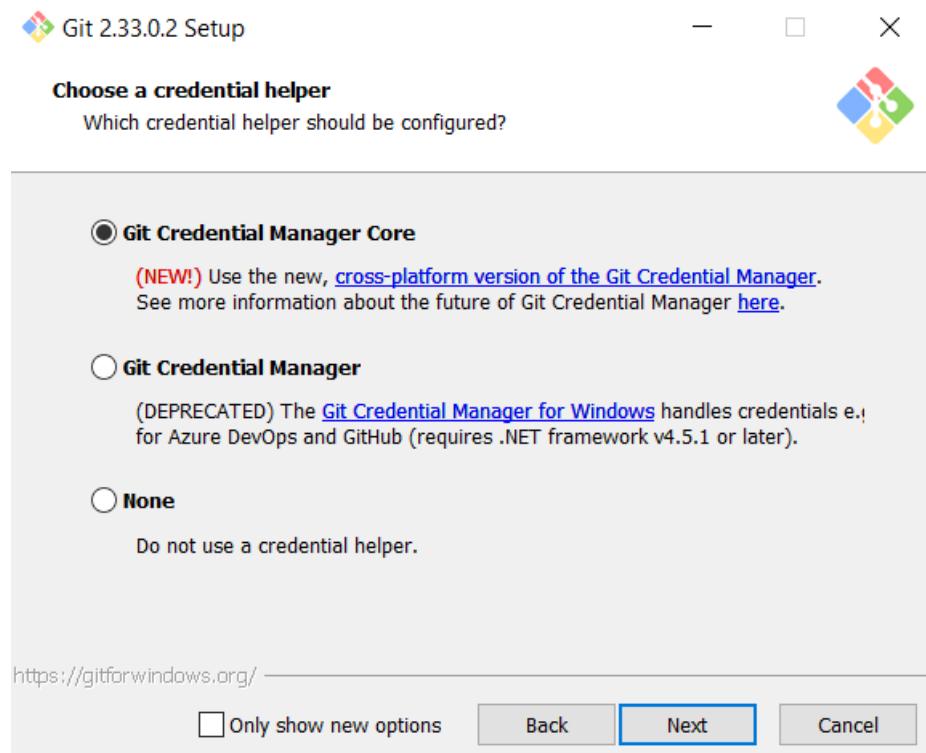
זה חשוב. יתכן שבגרסאות הבאות של GIT המסק הזה לא יהיה. מדובר בהגדרת ברירת המחדל של שם הבראנץ'. GIT הודיעה שהיא משנה אותו ל-'main' בגרסאות הבאות ומומלץ לשנות את השם ל-'main'. אסביר על המונחים האלה בהמשך.

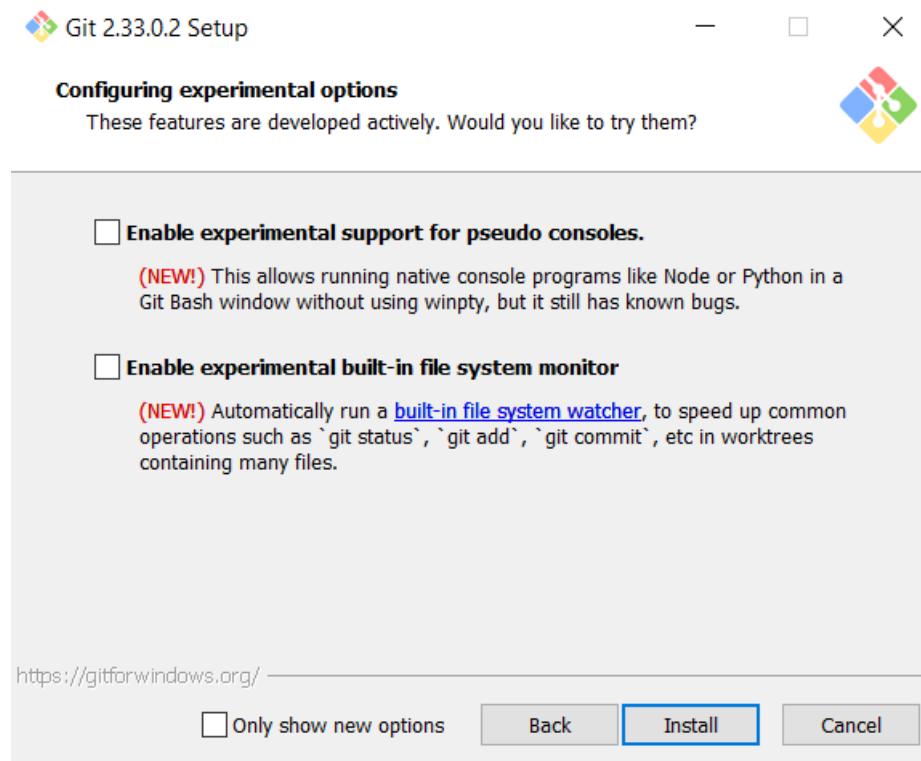
במשך הבא ובאליה שאחריו מומלץ פשוט ללחוץ על **Next**.



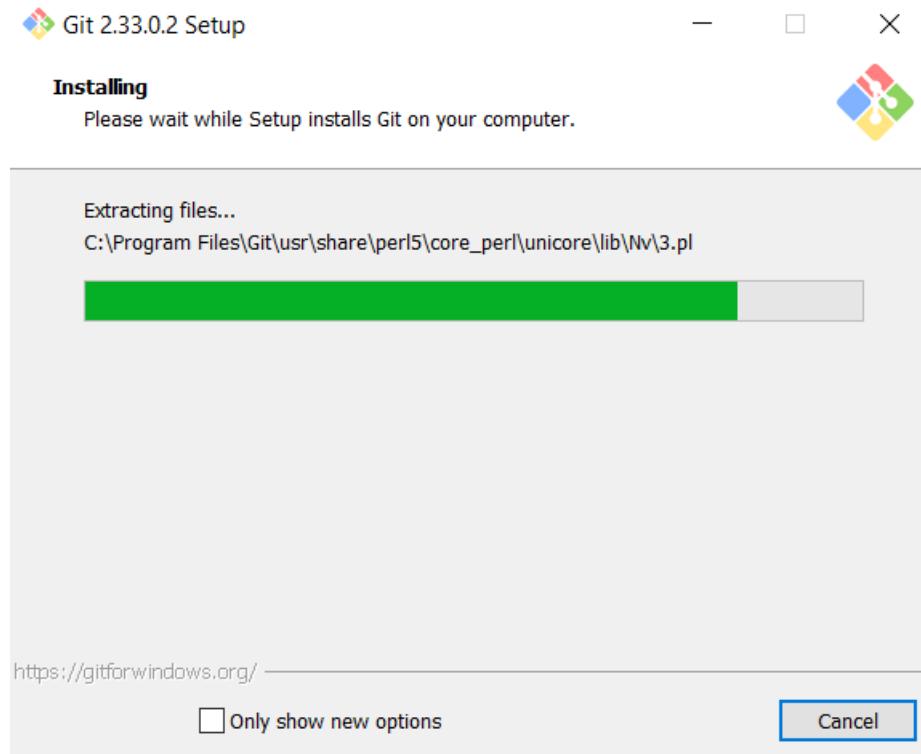




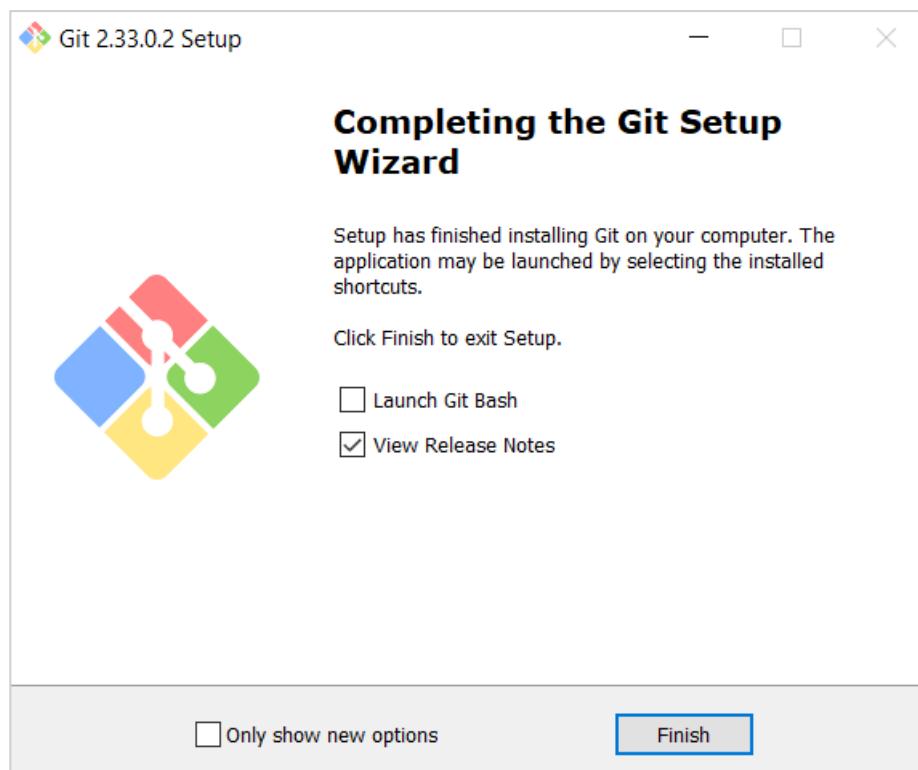




אחרי כל השאלות, שלרובן באמת לא צריך להתפנות מעבר למסכים הראשונים, מגיעים לשלב ההתקנה.



בסוף יוצג בפנינו מסך הסיום.



כיוון שמדובר בתוכנה אולטרה-פופולרית, אם תיתקלו בתקלה כלשהי במהלך ההתקנה, חיפוש מהיר בגוגל עם הודעת השגיאה יסייע לכם להבין מה השتبש. אבל ברוב המקרים ההתקנה תעבור חילק.

## חולנות: הסבר על תוכנת `git bash`

חלק ממהליך ההתקנה בחולנות, מותקנת גם תוכנה קטנה בשם `git bash`. מדובר בתוכנה שהיא אלטרנטיבה ל-`cmd` (הטרמינל המקורי בחולנות) המוכר יותר ומומלץ להשתמש בה.

`bash` הם ראשי תיבות של `bourne again shell`, ומדובר בעצם בשפה של הטרמינל של לינוקס ומק. כיוון שהרטמים מבוססי לינוקס מאוד פופולריים בתעשייה ומחשיبي מק. פופולריים מאוד בקרב המפתחים, רוב הדוגמאות שיש ברשת ה-`bash`.

ההפעלה הבסיסית היא פשוטה – פשוט מATTRIM את התוכנה באמצעות החיפוש המקומי של cholnot. כתבים `git bash` וונכנסים אליה. קיבל חלון של טרמינל.

סימן ה-\$ בהתחלה הוא בעצם הסימן של תחילת ההקלדה. لكن בלי כמעט דוגמאות קוד ברשת ובספרים נראה את סימן ה-\$ שמסמן את תחילת השורה. הטרמינל עובד בדיקות כמו ה-prompt של חלונות רק עם פקודות של לינוקס. חלק מהפקודות זהות לאלו של חלונות. על מנת להתרשם, הנה טבלה קטנה של פקודות שונות:

תיאור הפקודה	הפקודה בחלונות	הפקודה בלינוקס
רשימת הקבצים והתיקיות שיש בתיקייה שבה אנו נמצאים	dir w	-al \ 
מעבר לתיקייה שלמעלה	.. cd	.. cd
מעבר לתיקייה	cd DIRNAME	cd DIRNAME

מי שמסתדר ב-cmd ידע להסתדר מעליה גם ב-bash git. אם יש צורך לעשות דבר-מה ב- bash git ואתם לא מכירם את הפקודה, חיפוש קל של מה שצריך לעשות יחד עם השם bash יבהיר לכם איך לעבוד.

בספר "לימוד Node.js בעברית" למדנו באופן מקיף על cmd ועל עבודה בטרמינל בリンוקס ובחלונות.

היתרון העצום ב-`git bash` הוא שיש תצוגה המציגת את הסטטוס שלכם בגייט, במיוחד הריפוזיטורי והבראנץ' (מונהחים שיווסבו בהמשך). לכן מומלץ להשתמש בה, אך זו אינה חובה והדוגמאות שיש בספר מופיעות דווקא ב-`cmd` כדי להראות שגירט היא תוכנה שרצה בכל סביבה, לאו דווקא ב-`git bash`. הדוגמאות כמפורט מתחילה לכל פלטפורמה שהיא.

**שימוש לב, הערה חשובה:** אם אתם משתמשים ב-`git bash`, אתם בעצם בסביבת לינוקס, גם אם המחשב שלכם שולכים בחלונות.

## קובץ קונפיגורציה

קובץ הקונפיגורציה נקרא `gitconfig` ואפשר ליצור אותו גלובלי. במחשבים חלונים הוא יימצא בתיקייה של שם המשתמש שלכם.

`C:\Users\username\.gitconfig`

במחשבים לינוקס ומック הוא יימצא בתיקיית המשתמש. בקובץ יהיו הגדרות שונות שאפשר לעורך, כמו שם המשתמש והמייל שלכם (חינוי לקומיטים, שעלהם למד בהמשך), והגדרות שונות שיש לעתים צורך לשנות. רוב המפתחים, למעט עדכון שם משתמש ומיל, לא יוכנסו לעולם לקובץ זהה – אבל הוא נמצא שם ולכך כדאי לפקות לדעת היכן הוא.

## תחילת העבודה בטרמינל

נפתח את הטרמינל (`git bash`, `cmd` או ה-IDE שלנו). ה-IDE, שעליו למדנו בספר "למידה גאווהסקריפט בעברית", הוא התוכנה שבה כתבים קוד, כמו `Visual Studio`. ראשי התיבות של IDE הם IDE (integrated development environment) ונקליד `--version`.

אם הכל תקין, נקבל את מספר הגרסה של גירט. אפשר להתחיל לעבוד.

```
C:\Users\barzik>git --version
git version 2.10.2.windows.1
C:\Users\barzik>
```

שים לב: צילומי המסך הם מ-`cmd` של חלונות, אך אין הבדל ממשוני בתצוגה בין הטרמינל של לינוקס, מק או `bash`.

## יצירת ריבזיטורי מקומי

הקוד בಗיט מאורגן על ידי מה שנקרא ריבזיטורי, או ריבו בקצרה (באנגלית: Repository). הריבזיטורי הוא התיקייה הראשית של הפרויקט שבה יש קבצים אחרים, או `repo`). התיקייה הראשית של הפרויקט שבה יש קבצים אחרים. אנו תיקיות אחרות שבנה יש קבצים וכך הלאה - כאמור, התיקייה הראשית של הפרויקט. אנו יכולים להגדיר כל תיקייה שאינה תת-תיקייה של פרויקט אחר כתיקייה ראשית. לשם האימון, ניצור תיקייה ראשית בשם `local` ונגדיר אותה כתיקייה ראשית של פרויקט – בעצם, כריבזיטורי שלנו.

כל הפעולות שאנו עושים על גבי המחשב שלנו נקראות "פעולות מקומיות". הפעולות הללו נשמרות אך ורק על המחשב המקומי שלנו. הגודלה של גיט היא שאפשר לעבוד באופן מקומי בלבד באופן מושלם – ליצור גרסאות שונות של הקוד, לשחזר ולבוחן את השינויים בין כל גרסה וגרסה. מובן שם מערכת הפעלה נמתקת, כל הריבזיטורי נמתקים. גם לא ניתן לשתח עם מפתחים נוספים את פרטיعمالנו. זו הסיבה שבגללה נלמד בהמשך איך לסנן את הריבזיטורי המקורי עם שירות מרוחק (גיטהאב) כדי להנוט מגיבוי גם יכולת שיתוף עם אחרים. היכולת לעבוד באופן מקומי בלבד היא הסיבה שagit נחשבת למערכת ניהול גרסאות מבוצרת שאפשר לעבוד אליה גם ללא שרת מרכזי. בדוגמה זו אנו מבצעים פעולות מקומיות בלבד.

שם התיקייה הראשית הוא השם שנadan לריבזיטורי המקומי. כדי ליצור ריבזיטורי מקומי נקליד ב-`cmd`:

```
git init
```

ונקיש על אנטר. מיד נקבל הודעה שנוצר ריבזיטורי ריק של גיט בתיקייה.

```
C:\local>git init
Initialized empty Git repository in C:/local/.git/
```

אם נפתח את סייר הקבצים ונסתכל בתיקייה, נראה שאחריו שהקלדנו `git init`, נוצרה תיקייה בשם `git`. (שים לב לנקודה לפני השם). כשים נקודה לפני שם התיקייה,

מדובר בתיקייה נסתרת. אם לא שינויתם את האפשרות בסיר הקבצים שלכם לצפייה בתיקיות נסתרות, לא תראו את התיקייה זו. ברוב המקרים אין שום צורך להויכנס לתיקייה, אך זו התיקייה שמכילה את המידע המקומי של גיט. אם תמחקו אותה, המידע על הריפוזיטורי יימחק.

כך למשל במקרה הזה, לאחר שהקלדנו `A/ dir` ניתן לראות תיקיית `.git` בתיקייה של הפרויקט.

```
C:\local>dir /A
Volume in drive C is OS
Volume Serial Number is 0C36-DF83

Directory of C:\local

07/30/2021  11:52 AM    <DIR>      .
07/30/2021  11:52 AM    <DIR>      ..
07/30/2021  11:52 AM    <DIR>      .git
              0 File(s)        0 bytes
              3 Dir(s)   16,243,769,344 bytes free
```

אפשר לראות שנוצר ריפורזיטורי חדש גם בלי לצפות בתיקייה הנסתרת. אם נקליד את הפקודה:

`git status`

נקלט הסטטוס שלנו ונראה שהוא שוכן על ענף (branch) בשם `main`. כרגע נקלט זאת בקלות. בהמשך נלמד איך עובדים עם ענפים שונים.

## קומיט: הכנסת קבצים ושינויים לגרסה

על מנת להתאמן בಗיט נפתח נפתח את ה-IDE שלנו. בספר זה אני משתמש ב-`Visual Studio Code`, שעליו לימדתי בספרים הקודמים. אם איןכם זוכרים איך להשתמש בו, מומלץ לעשות חזרה או לקרוא עליו מעט. נכוון אותו לתיקיית `local` שעליה הרצנו קודם את `git init`. הספרייה זו ריקה כמעט לחלוטין `git log`. שבה לא נוגעים כמובן, כי היא שייכת לגיט עצמה. ניצור קובץ בשם `index.js` ונכנסים בו את הקוד:

```
console.log('hello world');
```

נשמר את הקובץ. עכשו נטפל בו באמצעות git. נחזור לטרמינל ונקליד `git status`. נראה שהקובץ `index.js` שיצרנו זה עתה מופיע באדום כ-`untracked`. בעצם גיט מודיעה לנו שמדובר בקובץ שהוא לא עוקבת אחריו ושאינו מנוהל על ידי הריביזיטורי.

```
Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    index.js

nothing added to commit but untracked files present (use "git add" to track)
```

git לא מכינה אוטומטית שום קובץ לריביזיטורי ולא עושה שום פעולה ברקע בלי שניזום אותה, וזאת כדי להימנע מטעויות שלולות כתוצאה בגל תהליכי אוטומטיים, שאולי לא מבינים בהם את מטרת המתכנת. אנחנו כן רוצים להכניס את הקובץ לריביזיטורי ועל כן علينا לבצע שתי פעולות. הראשונה הוא להוסיף את הקובץ לששתנה אל "ה毋עדים להכנסה". הפעולה זו נקראת סטייג' (stage). השנייה היא לבצע את ההכנסה בפועל, ופעולה זו נקראת קומיט (commit).

זה בדיקן כמו בשידוך, ואני השדכנים. הפעולה הראשונית היא לבחור את המ毋עדים לשידוך (במקרה זה יש לנו רק מועמד אחד, `index.js`), זהה שלב הסטייג', והפעולה השנייה היא לחת את המועמד לשידוך אל הטקס, וזה שלב הקומיט.

לחלופין, זה כמו אפליקציית היכריות, טינדר למשל. אני בוחן את המ毋עדים, לכל מי שמוצא חן בעיניי אני מבצע סטייג', זהה בעצם הסופי ימינה – הבחירה במועמד. הפעולה השנייה היא הפגיעה עצמה, שמובילה לחתוונה באולמי בונבן – פעולה הקומיט.

כאמור, רשות הקבצים (או השינויים) המ毋עדים להכנסה אל הריביזיטורי נקראת סטייג'. אנו מכינים אותם באמצעות הפקודה `git add` ו-`git commit`. במקרה שלנו:

```
git add index.js
```

זו פעולה הסטייג'. אם נכתב `git status` אחריה, נראה שם הקובץ `index.js` אינו מופיע יותר באדום, אלא בירוק, כਮועמד לשלב הקומיט, השלב המרגש שבו הקובץ נכנס לריביזיטורי.

```
C:\local>git add index.js

C:\local>git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   index.js
```

אם אני מתחرت ורוצה להוציא את הקובץ מהסטיביג', אני יכול להקליד, בדיק נפי שמצוין בהודעה של `git status`, את הפקודה:

`git rm --cached`

ואת שם הקובץ על מנת להסירו מרשימה המועמדים.

```
C:\local>git rm --cached index.js
rm 'index.js'

C:\local>git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    index.js

nothing added to commit but untracked files present (use "git add" to track)
```

אבל אנו כן רוצים להכניס את `index.js` לרייפוזיטורי. אז לאחר שהוספנו את `index.js` יOIDIANO, באמצעות `git status`, שהקובץ בסטייג', נבצע את הקומיט המיויחל באמצעות פקודה `git commit`. הפקודה זו לוקחת את כל הקבצים שיש בסטייג' ומכניסה אותם לרייפוזיטורי. אנו נדרשים להכניס בכל קומיט גם הודעה, בדרך כלל על אופי השינויים. את זה עושים על ידי הוספת פלאג של `-m`. ולאחריו ההודעה שעתופה במירכאות, כמו בדוגמה הבאה:

```
git commit -m "This is a commit message"
```

```
C:\local>git add index.js

C:\local>git commit -m "This is a commit message"
[master (root-commit) 9bcd042] This is a commit message
 1 file changed, 1 insertion(+)
 create mode 100644 index.js
```

מיד אחרי שנקליד את הפקודה ונ קיש על אונטר, קיבל אישור ומידע על הפעולה. אם נקליד `git status` נראה שכותוב בפירוש:

```
nothing to commit, working tree clean
```

הקובץ נכנס לריפוזיטורי. מהנקודה הזאת בפועל הוא נרשם. זה בדיקן כמו שמירת משחק. כל עוד תיקיית `git`. תישמר, יוכל לחזור למצב הזה של הקובץ. אם הכנסנו כמה קבצים, תמיד יוכל לחזור אליהם בנקודות הזמן הללו. תמיד.

כדי לראות את כל הקומיטים שעשינו, אנו יכולים להשתמש בפקודה `log .git`.

```
C:\local>git log
commit 9bcd04260489b0a098b0b9e3e815ad0e85e96506
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Fri Jul 30 12:22:34 2021 +0300

    This is a commit message
```

הקלדת הפקודה והקשה על אונטר יציגו לנו את הקומיטים, במקרה הזה קומיט אחד. ליד כל קומיט כתובים הזמן שלו, ההערה שלו וכן מחרוזת טקסט מזהה ארוכה. המחרוזת הזאת, שנគרא `A` או `git ref hash`, `git SHA` או `git log`, היא ייחודית לכל קומיט ונלמד בהמשך איך עובדים איתה.

בוצע קומיט נוסף, הפעם קומיט שמורכב משני קבצים. ניצור את הקובץ `package.json` באמצעות הפקודה:

```
npm init
```

.package.json לא חשובים. מה שחשוב הוא שייהי לנו קובץ package.json.

```
C:\local>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (local)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\local\package.json:

{
  "name": "local",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes)
```

כמו כן נשנה את ה-`log` ל `hello world` שלנו מעט ונהפוך את `console.log` את הקבצים.

```
console.log('Hello World');
```

אם נקליד `git status`, נראה שיש לנו קובץ חדש בשם `package.json` שagit לא עוקבת אחריו וכן שינוי בקובץ `index.js`.

```
C:\local>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   index.js

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    package.json

no changes added to commit (use "git add" and/or "git commit -a")
```

אנו יכולים לבחור אחד מהקבצים לסטויג' שלנו או את כלם. אם נקליד:

`git add index.js package.json`

שני הקבצים ייכנסו לסטויג'.

**חשיבות:** אפשר להקליד:

`git add .`

כדי להוסיף את כל הקבצים שהשתנו לסטויג'. משמעותה הנקודתית אחרי ה-`add` היא כל הקבצים שהשתנו.

```
C:\local>git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   index.js
    new file:   package.json
```

נכenis את כל מה שיש בסטייג' לריפורזיטורי עם קומיט, בדיקן כמו בפעם הקודמת, עם הפקודה git commit וavanaugh. הפעם אין חשיבות למספר הקבצים וכמוהן אין צורך להכנס את שמותיהם.

```
git commit -m "This is a package.json change"
```

```
C:\local>git commit -m "This is a package.json change"
[master 9685862] This is a package.json change
 2 files changed, 12 insertions(+), 1 deletion(-)
 create mode 100644 package.json
```

גם פה קיבל ref ייחודי ואם נבחן את כל הקומיטים שעשינו (שניים עד כה!) עם git status, נראה אותו ואת המספרים שלהם.

```
C:\local>git log
commit 96858626429f0f3fee2adbb0d74d8b8d70c6b2d5
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Fri Jul 30 12:42:58 2021 +0300

    This is a package.json change

commit 9bcd04260489b0a098b0b9e3e815ad0e85e96506
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Fri Jul 30 12:22:34 2021 +0300

    This is a commit message
```

לסיום, עד כה למדנו איך ליצור git repository בתיקייה במחשב, איך מכניסים קבצים ושינויים בקבצים לסטייג' ואיך מבצעים קומיט ומכניסים את כל השינויים לסוג של "שמירת משחק".

## שחזור קומייט

כל ref git הוא כמו שמיירת משחק, ואפשר לנوع בין שמיירות המשחק לנקודת השמירה בקהלות. איך? באמצעות פקודת checkout git. ראיינו עם git log שכל קומייט יש מחרוזת טקסט ייחודית שמצויה אותו – הלא היא ה-ref git. אם רוצים להגיע אל נקודת השמירה הזו, פשוט מקלידים:

```
git checkout 9bcd04260489b0a098b0b9e3e815ad0e85e96506
```

או את מחרוזת הטקסט של כל קומייט שאנו רוצים לעבור אליו. כבmeta קסם, כל הקבצים, השינויים שהיו מאז והשמירות – הכל נעלם וモחלף بما שהיה קודם. אנחנו יכולים לנוע לכל קומייט שאנו רוצים – כמו חזרה בזמן. כדי לחזור לנוקודה الأخيرة בזמן נקליד:

```
git checkout main
```

ה-command הוא שם הבראנץ' שלנו והוא יביא אותנו לנוקודה الأخيرة – לקומיט האחרון שלנו או לשמירה الأخيرة שלנו.

אפשר לראות איך באמצעות יצרת קומייטים אנחנו יכולים לנוע על ציר הזמן. אפשר לחזור לנוקודה קודמת באותו הזמן ולהמשיך לפתח ממנה. זה נותן גמישות ממש רבה למתקנת. חשבו למשל על מתכנת שהכניס שגיאה בקוד וגילה את זה רק כשהקובד עלה לשרת – הוא פשוט יכול לחזור לקומייט הקודם ללא השגיאה ולהעלות לשרת.

## מציאת הבדלים בי/o הקומייטים

אפשר להשתמש בפקודה diff git על מנת למצוא הבדלים בין ref git, במקרה שלנו קומייטים. איך? פשוט בקומיט שבו נמצאים בו כתבים diff git ואת מחרוזת הטקסט של הקומייט:

```
git diff 9bcd04260489b0a098b0b9e3e815ad0e85e96506
```

```
C:\local>git diff 9bcd04260489b0a098b0b9e3e815ad0e85e96506
diff --git a/index.js b/index.js
index 5893f9d..73c0265 100644
--- a/index.js
+++ b/index.js
@@ -1 +1 @@
-console.log('hello world');
\ No newline at end of file
+console.log('Hello World');
\ No newline at end of file
diff --git a/package.json b/package.json
new file mode 100644
index 0000000..1d4a9e0
--- /dev/null
+++ b/package.json
@@ -0,0 +1,11 @@
+{
+  "name": "local",
+  "version": "1.0.0",
+  "description": "",
+  "main": "index.js",
+  "scripts": {
+    "test": "echo \\"$Error: no test specified\\" && exit 1"
+  },
+  "author": "",
+  "license": "ISC"
+}
```

או עם שני git ref. למשל:

```
git diff 9bcd04260489b0a098b0b9e3e815ad0e85e96506
96858626429f0f3fee2adbb0d74d8b8d70c6b2d5
```

אם בקומיט יש הרבה שינויים,>Kצת קשה לעקוב אחרי כולם בcmd, ולכ"ע לפעם נדאי להשתמש במשק גרפי כדי לראות את השינויים.

## שימוש ב-ref git מקוצר

אפשר לראות בפקודה לעיל שם משתמשים ב-ref git ארוך מדי, זה עלול לבלבל. git יכולה לעבוד גם עם שש הספרות הראשונות של ה-ref git. למשל, במקרה:

```
9bcd04260489b0a098b0b9e3e815ad0e85e96506
```

אפשר לנתחו:

```
9bcd042
```

זה כבר נראה אלגנטי יותר. לדוגמה:

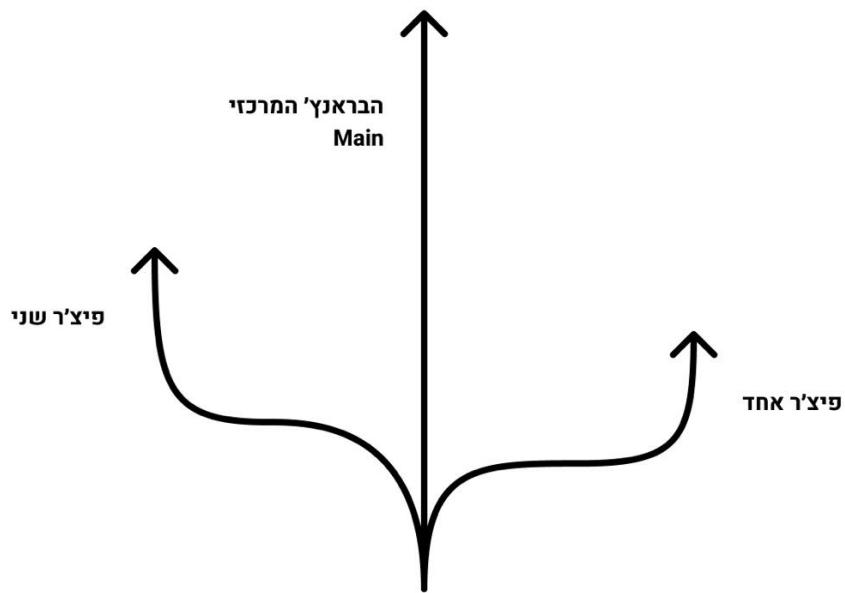
```
git diff 9bcd042 9685862
```

לסיום, ראיינו איך קומיטים אפשריים לנו לנوع בזמן בין קוד אחד לאחר ואיך אנחנו יכולים, כמו במשחק מחשב, לשמור נקודות בזמן וafilו לראות את ההבדלים ביניהן. בפרק הבא, על בראנצ'ים, נראה איך אנחנו יכולים לקחת את זה צעד אחד קדימה.

## בראנץ'

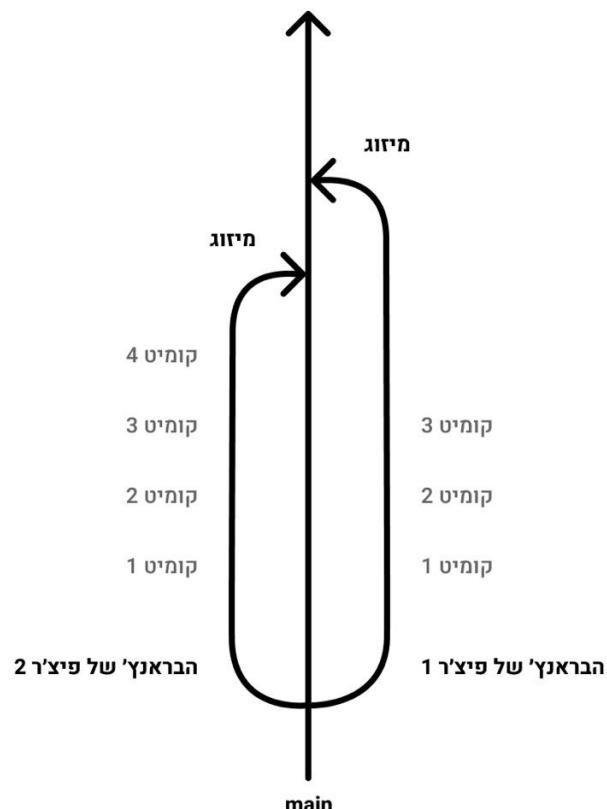
אף על פי שהתרגום של branch בגיט הוא "ענף", בתעשייה אומרים את השם הלועזי: בראנץ'. זו קצת "עברית אנגלית" מוזרה, אבל בספר הזה אני משתמש בשפת המפתחים ולא בשפה הנכונה. בראנץ' הוא אחד הקונספטים החשובים בגיט. באמצעות קומיטים אנו יכולים לנوع קדימה ואחוריה בזמן, ממש כמו במשחק מחשב. אבל עם בראנצ'ים אנחנו יכולים גם לנوع באופן אופקי.

מה זאת אומרת? בואו נניח שאנו מתכוון ויש מוצר שאנו בונה. מנהל המוצר פונה אליו וمبקש פיצ'ר, ואני מתייחל לעבוד עליו. אבל אז מנהל המוצר מבקש פיצ'ר אחר. מה אני יכול לעשות? אם אני העבודה עם קומיטים – אני עלול להתבלבל. לעיתים אני גם רוצה לצאת מאותו בסיס בלי לערבות קוד של פיצ'ר מסוים עם פיצ'ר אחר. כמובן, אני נדרש להתקדם בשתי דרכים.



בפועל, ביפוי זיטורי יהיה לי שלוש דרכי: הדרכ הראשונה של המוצר המרכזי, שהוא זה שנמצא בעצמו בשרת ואנשים משתמשים בו. אני מفضل ממנו קוד של פיצר אחד שלו אני יכול להכניס אליו קומיטים כרצוני, וקוד של פיצר שני, שגם אליו אני יכול להכניס קומיטים כרצוני. הפיצולים, שהם הבראנצ'ים, נפרדים זה מזה ואני עובד עליהם באופן נפרד. כשארצה וasisים – אמצע אותם עם הבראנץ' המרכזי, *the main* שלי.

**חשוב:** הבראנץ' המרכזי ברוב החברות הוא הבראנץ' המיצג את הגרסה שהלkopחות רואים, גרסת הפרודקشن (על פרודקشن נלמד בפרק על דיפלומנט בהמשך הספר).



זה עלול להיות מבלבל למי שלא מORGן בMONחחים האלה, אבל בגדול, בכל פיצ'ר שהוא - משינוי קטן כמו תיקון באג ועד הכנסת שינוי ממשוני – אנו משתמשים בבראנצ'ים. אנו משתמשים בבראנצ'ים גם לכמה GRסאות. כך למשל יכול להיות לנו בראנץ' פרודקشن, בראנץ' של הגרסה הבאה ובראנץ' של הגרסה הבאה-הבאה.

בואו נדגים, לפנוי שגעמיך יותר. יוצרים בראנצ'ים עם פקודת `git branch`. הקלדה של הפוקודה הזו, ללא תוספות, מראה לנו את כל הבראנצ'ים הקיימים. אם יש לנו רק `main`, זה מה שנראה:

```
C:\local>git branch
* main
```

יצירת בראנץ' חדש נעשית עם פקודת `git branch` ואז שם הבראנץ'. למשל:

```
git branch add-end-point-to-start
```

שם הבראנץ' יכול להיות כל שם שהוא באנגלית ולא רוחים (מקף גם אפשרי), אבל מקובל בתעשייה לתת שם שמתאר את הבראנץ'.

מייד אחרי שאנו יוצרים את הבראנץ', אנחנו יכולים לראות את כל שמות הבראנצ'ים בקלות עם פקודת `git branch`. הבראנץ' שאנו נמצאים בו מסומן בכוכבית ובצבע ירוק.

```
C:\local>git branch
  add-end-point-to-start
* main
```

כדי לעבור מבראנץ' לבראנץ' משתמשים, בדיק כמו בкомיט, בפקודה `git checkout` ו-`git checkout add-end-point-to-start`:

```
C:\local>git checkout add-end-point-to-start
Switched to branch 'add-end-point-to-start'

C:\local>git branch
* add-end-point-to-start
  main
```

עכשו אנו יכולים לעשות קומיט בבראנץ' החדש, והקומיטים בעצם נפרדים. לכל בראנץ' יש את הקומיטים שלו. CAN למשל הכנסתי שינוי – הוסףו ל-`package.json` שינוי קוד עשייתי לו קומיט.

```
C:\local>git status
On branch add-end-point-to-start
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   package.json

no changes added to commit (use "git add" and/or "git commit -a")

C:\local>git add package.json

C:\local>git commit -m "add start script"
[add-end-point-to-start 724a04d] add start script
  1 file changed, 1 insertion(+)
```

אם נקליד את הפקודה `git log`, נראה את הקומיט זהה:

```
C:\local>git log
commit 724a04d67b935848e8daa4197723ed50337f5036
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Jul 31 15:13:09 2021 +0300

    add start script

commit 96858626429f0f3fee2adbb0d74d8b8d70c6b2d5
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Fri Jul 30 12:42:58 2021 +0300

    This is a package.json change

commit 9bcd04260489b0a098b0b9e3e815ad0e85e96506
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Fri Jul 30 12:22:34 2021 +0300

    This is a commit message
```

אבל אם נחזור לבראנץ' `main`, לא נראה את הקומיט זהה. למה? כי הוא שייך לבראנץ' השני. אבל אנחנו כן נראה את הקומיטים שבוצעו בעבר בבראנץ' הנוכחי ובבראנץ' שייצנו ממנו.

## פקודה מקוצרת ליצירת בראנץ' ומעבר אליו

ראיינו קודם لكن שיצרנו בראנץ' באמצעות פקודה `git branch` ואז מעבר אל הבראנץ' החדש באמצעות `git checkout`. למשל:

```
git branch my-new-branch
git checkout my-new-branch
```

אבל אפשר לעשות את זה גם באופן מקוצר, והשיטה הזו פופולרית מאוד. משתמשים בפלאג-ה- כדי לסמן שרצוים ליצור בראנץ' חדש ואז מבצעים checkout. למשל, משתמש בפקודה הזו:

```
git checkout -b my-new-branch
```

כדי ליצור בראנץ' ולעבור אליו מיידית.

## חיבור בין שני בראנץ'ים באמצעות merge

ראינו שבראנץ' אפשר לנו לבצע הפרדה בין קומיטים שונים וגרסאות שונות. אחרי שאנו יוצאים מהבראנץ' הראשי שלנו לבראנץ' אחר, אנחנו יכולים להוסיף לו קומיטים新增. אבל מגיע שלב מסוים שבו רוצים לאחד בחזרה בין הבראנץ' שיצאנו ממנו והבראנץ' החדש. בדיק נמו כביש שמתפצל מכביש ראשי, עובר לו בשדות ואז חוזר אל הכביש הראשי – כך גם בניהול גרסאות. הוצאנו מהגרסה הראשית בראנץ' שמשמש להוספת תוכנה כלשהי למוצר. אנחנו מוסיפים עוד ועוד קומיטים (כלומר שמירות), אבל מתישנו אנו מרווחים מההתוצאה ורוצים למזג בחזרה את הבראנץ' עם הבראנץ' הראשי. לפעולה הזו קוראים מיזוג (merge, מרג'). מפתחים קוראים לה "למרג'".

איך מבצעים את המיזוג? נכנסים לבראנץ' אליו רוצים למזג ומשתמשים בפקודת merge באופן פשוט: git merge *שם הבראנץ'* *שם הבראנץ' שלנו*. בדוגמה הזו, למשל, אנו נמצאים בבראנץ' main ומיזגים אליו את בראנץ'-add:end-point-to-start

```
git merge add-end-point-to-start
```

```
C:\local>git merge add-end-point-to-start
Updating 968586d..724a04d
Fast-forward
 package.json | 1 +
 1 file changed, 1 insertion(+)
```

מיד אחרי המיזוג, כל הקומיטים שהיו בבראנץ' השני ייכנסו לתוך הבראנץ' אליו מיזגנו אותו, וכך ייצורנו אותו על הבראנץ' הזה ממש.

המרג' עובד באופן אוטומטי ועובד היטב גם אם היו שינויים באותו קובץ בבראנס' אחר. אם השינויים נעשו ממש באותו שורות, יוצר קונפליקט שאותו יהיה צריך לפתור. על קונפליקט ועל דרכי נוספות לבצע מיזוג נדון בפרק הבא.

## שם הבראנס' של בריית המחדל

כאשר יוצרים ריפוזיטורי מרוחק עם `git init`, שם הבראנס' של בריית המחדל יהיה `main`. בעבר הוא נקרא `master`, אך הוחלף על ידי גיט. אם אתם נתקלים במאמרם שבו כתוב `master`, הכוונה היא ל-`main`.

## מחיקת בראנס'

אחרי המיזוג אפשר למחוק את הבראנס' הקודם באמצעות פקודה `branch` והפלאג-`-d` באופן הבא:

```
git branch add-end-point-to-start -d
```

```
C:\local>git branch add-end-point-to-start -d
Deleted branch add-end-point-to-start (was 724a04d).
```

זהו צעד בלתי הפיך, אז כדאי להיזהר איתו ולבצע אותו רק לאחר שמסייםים בודדות בעבודה על הבראנס'.

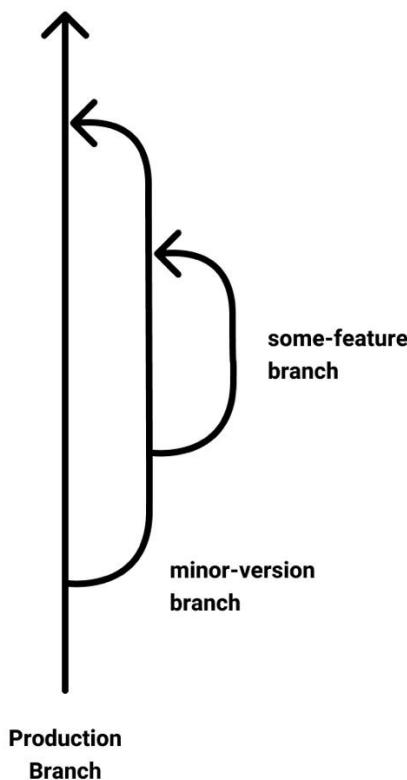
חשוב: פלאג – במקרה זה `-d` – הוא חלק מהותי מכל פקודה שהיא. בדרך כלל יש סימן מינוס לפני הפקודה, וכן כמעט כל פלאג שימושי ב-CLI זמין בגרסת ארוכה עם שני מקפירים ושם מלא ובגרסת מקוצרת עם מקף אחד, למשל, אפשר למחוק באמצעות `-d` אבל גם `--delete` יעבד.

## יצירת בראנס'ים מבראנס'ים

כדי להוסיף על השמחה אפשר ליצור בראנס'ים מבראנס'ים אחרים שיצרנו. כך למשל בחלק מהחברות יש כמה בראנס'ים של המוצר: בראנס' מסווג פרודקשן, בראנס' מסווג major-version לגרסה המיג'ור הבאה ובראנס' מסווג minor-version לגרסה המינור הבאה (למדו על גרסאות מיג'ור, מינור ופאץ' בספר "למוד Node.js בעברית").

כשנפתח רצה להכנס קוד לモצר, הוא צריך לבחור לאיזה בראנץ' להוסיף את הקוד, לעבור אליו באמצעות git branch ואז ליצר ממנו בראנץ' נוסף.

זה נשמע מסובך, אז אולי הדיאגרמה הבאה תסייע:



דרך הפעולה במוצר מבוסס בראנץ'ים היא לתחזק כמה בראנץ'ים שמתמזגים ביניהם כל הזמן. כך יש לנו בראנץ' מרכזי שנקרא main (או כאמור master בגרסהות ישנות של גיט) שמייצג, בשאייפה, את מה שיש בשרת הפודקשן ופועל ממש עכשויו בשרת, וממנו אנו מוצאים את הbraanç' next version. אל הbraanç' זהה מוכנסים קוד באמצאות braanç'ים שיוצאים מיחוזה next-version, מוסיפים אליום קומיטים ואז ממזגים את braanç'ים האלה בחזרה עם next-version.

ברגע שכל הקוד שאנו רוצים בגרסה הבאה נמצא בbraanç' next-version, אנו ממזגים את next-version עם main, לוקחים את main וטוענים אותו אל השרת. אם הכל עובד ואנו מרצו, מוחקים את next-version ויוצרים אותו שוב מיחוזה main – עכשויו הוא זהה ל-main ומוכן לקבל פיצרים חדשים וכך הלאה. כך בעצם תמיד יש שני בראנץ'ים – main וmain ומכאן

המרכזי ו-`next-version` שנוצר ממנו, נכנס אליו קוד חדש ואז בנקודת זמן מסוימת מתמזג בחזרה עם `main`. כמו מגל החיים בסרט "מלך האריות", רק בלי אריות, או חיים.

## קונפליקטים

כל עוד כל מפתח עובד על הקבצים שלו, הכל מצוין, אבל מה קורה כאשר שני מפתחים עובדים על אותה שורה בדיק בלי לדעת שאחד מהם שינה אותה? מי מנצח?

הסיטואציה הזו מתרכשת כאשר שני מפתחים שונים יוצאים מאותו בראנץ', כל אחד לבראנץ' משלהו, ושניהם משנים את אותה השורה. האחד מבצע מרג' לבראנץ' המרכזי. מן הסתם השינוי מתתקבל. ואז השני מבצע מרג' לבראנץ' המרני, וכיון שההיסטוריה של הבראנץ' שונה – יש לנו קונפליקט: מה לקבל? את השינוי של המפתח הראשון או של השני?

נדגים זאת ב簡單 דוגמה פשוטה. ניצור בראנץ' `branch-1` וונשנה את ה-`index.js` לטקסט נלשהו, כמו `Hello World`. נבצע קומיט. נחזיר ל-`main` (זה חשוב, לא ניצור בראנץ' נוסף `branch-1`), ניצור בראנץ' בשם `branch-2` וונשנה את `Hello World` לטקסט שונה למארי כמו `Hello World`.

```
C:\local>git checkout -b branch-1
Switched to a new branch 'branch-1'

C:\local>git add .

C:\local>git commit -m"change to branch 1"
[branch-1 77797d9] change to branch 1
 1 file changed, 1 insertion(+), 1 deletion(-)

C:\local>git checkout main
Switched to branch 'main'

C:\local>git checkout -b branch-2
Switched to a new branch 'branch-2'

C:\local>git add .

C:\local>git commit -m"change to branch 2"
[branch-2 6fa5acc] change to branch 2
 1 file changed, 1 insertion(+), 1 deletion(-)

C:\local>git checkout main
Switched to branch 'main'
```

זה שינוי שנעשה על ידינו, אבל בבחירה יכול להיווצר על ידי שני מתכנתים שעובדים על אותו קוד בדיק. נחזיר ל-.main וنمרג' את-1 אל-.main.

```
C:\local>git merge branch-1
Updating 724a04d..77797d9
Fast-forward
  index.js | 2 +-
  1 file changed, 1 insertion(+), 1 deletion(-)
```

הכל מצוין ועובד היטב, אבל העניינים יתחלו להסתרך כשאנסה לMarg' את-2-branch. כמובן, מדובר בשינוי שדורס את השינוי של-.branch-1. מי מנצח? גיט לא יכולה להחליט עבורנו.

```
C:\local>git merge branch-2
Auto-merging index.js
CONFLICT (content): Merge conflict in index.js
Automatic merge failed; fix conflicts and then commit the result.
```

אנו נראה שבניגוד להודעת המיזוג הנעימה הקודמת, כאן יש לנו התרעה ממשמעותית על קונפליקט. אם נבדוק את הסטטוס באמצעות `git status`, נראה שגם פה יש סטטוס אחר:

```
C:\local>git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

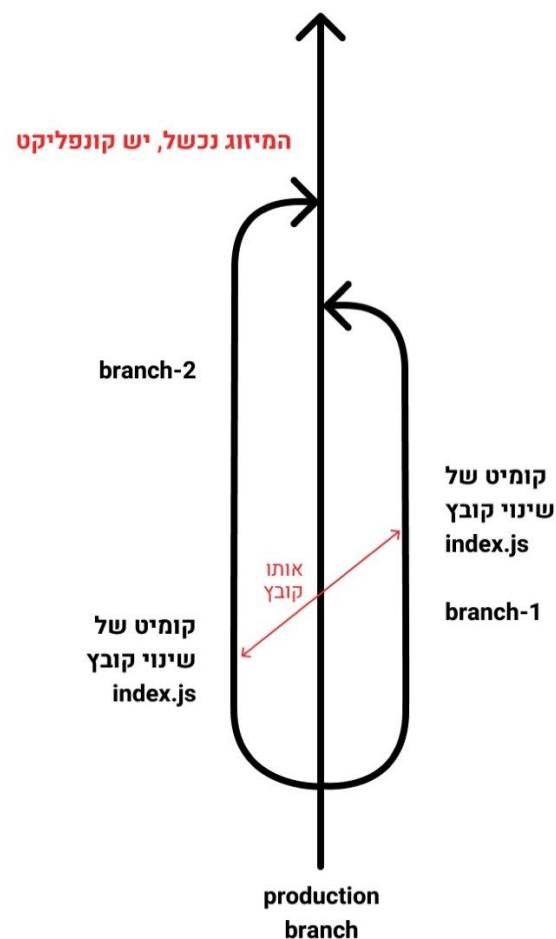
    both modified:  index.js

no changes added to commit (use "git add" and/or "git commit -a")
```

אם נפתח את הקובץ `index.js` ב-IDE, נראה שגיט שינתה את תוכנו, כך שאפשר לראות את הקונפליקט:

```
1  <<<<< HEAD (Current Change)
2  console.log('Hello World A');
3  =====
4  console.log('Hello World B');
5  >>>>> branch-2 (Incoming Change)
```

ישוב – זה קרה כי המפתחים שינו אותה שורה בבדיקה באותו קובץ.



עכשו מי שמבצע את המרג' צריך להכריע מי צודק ומהו הקוד הנוכחי, למחוק את מה שנוטר ולבצע קומיט חדש ישירות לבראנץ', או לחלופין להחזיר את המצב לקדמותו באמצעות התהליכיים הבאים:

חזרה לאחרו של הניסיון הכשל של המרג' שנעשה באמצעות `git reset` בתוספת המזהה של הקומיט האחרון ואז `git checkout`. לניקוי כל השינויים.

השלב הבא הוא לחזור לבראנץ' שבו נמצא הקונפליקט, למזג אליו את הבראנץ' של המפתח השני, לפטור את הקונפליקט ואז למזג אותו עם הבראנץ' המרכזי.

קונפליקטים הם חלק מהעבודה השוטפת בGIT. אפשר לצמצם את מספרם באמצעות עבודה במחזוריים (שנקראים גם איטרציות) קרים. למשל, לצאת לבראנץ' מוצמצם ולהשתדל לא להשאיר אותם אצלכם הרבה זמן, אלא למזג אותם מיד כשאפשר.

## פתרון קונפליקטים והבנת התחברות של הקונפליקט

ישנם כמה כלים לפתורן קונפליקטים, אך הכליל הטוב ביותר הוא עורך הטקסט שלנו והשינויים שגיט מכניסה. בואו נסתכל על קונפליקט זה:

```

1 <<<<< HEAD (Current Change)
2 console.log('Hello World A');
3 =====
4 console.log('Hello World B');
5 >>>>> branch-2 (Incoming Change)
6

```

כל מה שיש בין הטקסט:

>>>>>HEAD

:7

=====

הוא בעצם מה שקיים בגרסתו הנוכחית, ככלומר בבראנס' הנוכחית. ה-HEAD הוא הינו של גיט לעדכון האחרון, מה שיש בקצתה של הקומיט האחרון, המצב הנוכחי.

כל מה שיש אחרי הטקסט:

<<<<<YOUR-BRANCH-NAME

אליה השינויים שאנו רוצים להכניס, המכונים גם incoming. אנו יכולים לבחון את שני הטקסטים, להחליט מה אנחנו רוצים שייכנס, ל כתוב את הקוד ולמחוק את השורות שגיט המכניסה (כโลמר ה===== וה->>>>> (אחרי שווידאו) שיש לנו את הגרסה המאוחדת שפותרת את הקונפליקט. יש IDEs שמאפשרים לעשות את השינויים האלה בצורה נוחה יותר, הצגה של כל ה-diff ובחירה של הדרכים להתקדם.

## git rebase

ביצוע מרג' הוא דרך אחת לביצוע איחוד בין בראנץ'ים. דרך נוספת היא ביצוע `rebase`. מדובר בטכניקה שונה לביצוע אותו דבר – חיבור בין שני בראנץ'ים שונים שנפרדו בנקודת איחוד בהיסטוריה, נקודת ההיפרדות, ושהמשתמש רוצה לאחד ביניהם. המיזוג נעשה עד כה, כפי שלמדנו, באמצעות כניסה אל הbraanck' שהוא רוצים לאחד אליו והקלדה של `git merge BRANCHNAME`. ה-`BRANCHNAME` הוא שם הbraanck' שאוינו רוצים למזג.

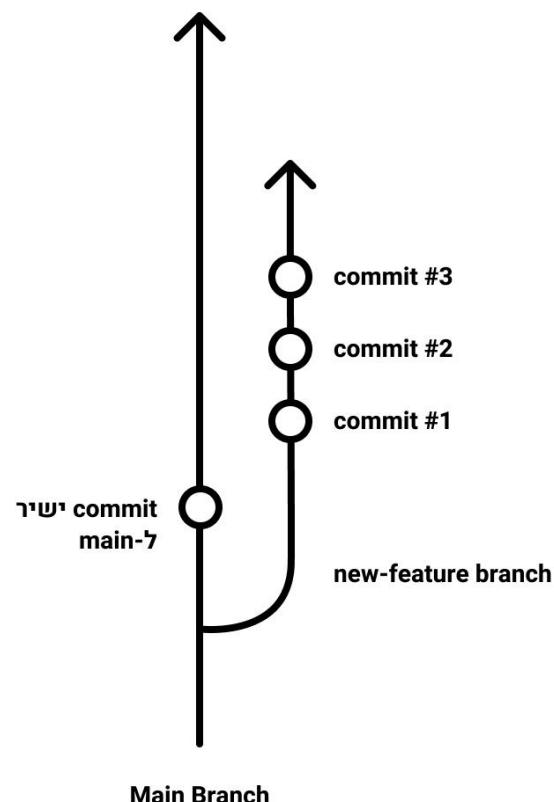
אחרי המיזוג, אם נסתכל בלוג, נראה שהMargin' זכה לקומיט משלו. אם מדובר בbraanck', פעיל בכל הזמן ממזגים אליו, הלוג יהיה מאוד לא מובן למפתחים. הבلغן יהיה גדול אפילו יותר אם יש קונפליקטים.

אתה הדריכם להתמודד עם העניין היא `git rebase`, שגם היא ממזג בין שני בראנץ'ים אבל בטכניקה שונה. כאן כל הקומיטים נראים כאלו הם נעשו ישרות לbraanck' שלו אין ממזגים. גיט בעצם מבצעת "חזרה בזמן" ומכניסה את הקומיטים לפי סדר הזמן שלהם. אם יש קונפליקטים, צריך לסדר אותם לפי הסדר. זה יכול להקל את עניין הקונפליקטים, כי במקרה פעם צריך לסדר קונפליקט אחד.

הבה נדגים איך נראהelog עם Margin' לעומת איך שהוא נראה עם `rebase`. נניח שיש פרויקט שיוצא מbraanck' `main` לbraanck' בשם `new-feature`.

המפתח יוצר קומיט ראשון, ואחריו קומיט שני ואז שלישי. לכל קומיט יש את ה-`hash` המזהה שלו. במקביל, מפתח אחר, או אפילו אותו מפתח, מבצע קומיט לbraanck' `main`.

כלומר, זה המצב הנוכחי:



עכשו המפתח רוצה למזג את new-feature עם main. הוא עובר לבראנץ' main ומקליד את הפקודה הבאה:

```
git merge new-feature
```

הלוגו ייראה כך:

```

commit 92a8ad7d521b424f36457272c51bdada61139248 (HEAD -> main)
Merge: 6e5565a b5261ed
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 10:50:42 2021 +0300

    Merge branch 'new-feature'

commit 6e5565a31224b5820034814761d4c882d03c1f28
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 10:50:13 2021 +0300

    commit to main branch

commit b5261ed902f677b1236a8cc1b4b9133d6fad5f98 (new-feature)
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 10:49:30 2021 +0300

    commit in new feature branch #3

commit d243ff61a50525189c79482efcf8632f114f9ab
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 10:49:19 2021 +0300

    commit in new feature branch #2

commit ad94266d653f9ffb1088a659308f567f726fd88
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 10:49:07 2021 +0300

    commit in new feature branch #1

commit 87672002795e0dd361fe125a44fd45de29f4cb64
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 10:48:13 2021 +0300

    initial commit

```

או ב\_TBLה, כך:

הפעולה	סדר כרונולוגי
קומייט ראשון ב-new-feature-branch	1
קומייט שני ב-new-feature-branch	2
קומייט ראשון ב-new-feature-branch	3
קומייט ל-main	4
קומייט של מרג' מ-new-feature-branch	5

הבעיה היא בשלב 5. יש לנו קומיט נוסף על כל מרג', וכל הקומיטים של הבראןץ' החדש נכנסים לאו דווקא למקום שבו אנו רוצחים אותם – הם נכנסים לפני קומיט מס' 4 שנכתב בבראןץ' `main` לפני שבוצע מיזוג ל-`new-feature`. עם `git rebase` אפשר להימנע מכך ובעצם לנוקות את ההיסטוריה שלנו. הפקודה נראה如下:

```
git rebase new-feature
```

אם אין קונפליקטים, הפעולה מתבצעת بكلות אך הלוג נראה שונה – ללא הקומיט של המרג'.

```
barzik@Ran-ASUS-Laptop MINGW64 /c/local/repository (main)
$ git log
commit c833f2ec62f1cb925c72fb2ca7567c9973709c85 (HEAD -> main)
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 11:19:34 2021 +0300

    commit to main branch

commit 7cd9a46078cd5b58470ace81de68f894b23412b3 (new-feature)
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 11:18:59 2021 +0300

    commit in new feature branch #3

commit 2fed3ed68b0ab4591d335ba47f408c67bbed7185
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 11:18:41 2021 +0300

    commit in new feature branch #2

commit 0c75b542b486844aa1a4dc04c546c2028c49b1e
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 11:18:28 2021 +0300

    commit in new feature branch #1

commit b44a3e5a526fce0a53b7be7e864d9e949634a1a5
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 4 11:17:50 2021 +0300

    initial commit
```

הקומיטים מופיעים בבראןץ' החדש בדיק, אבל בדיק, כפי שהם נכנסו לבראןץ' הקודם. ככלו ביצענו קומיט ישירות ל-`main`:

אם יש קונפליקטים, אפשר להשתמש בפקודה:

```
git rebase -i new-feature
```

ニיכנס למוד אינטראקטיבי, כלומר מוד שבו נישאל בנוגע להתקדמות ונצרך לישב את הקונפליקטים לכל קומיט בנפרד, ממש כאילו חזרנו בזמן ויצרנו את הקומיט שנייה אחרי שהקונפליקט נוצר. נעשה כך עבור כל קומיט בנפרד. זה נראה מאיים מעט, אבל כל מאד לעשות את זה. במקרה לקבל עירימה של קונפליקטים, פותרים אותם קומיט אחריו קומיט.

הבעיה היא שـ`git rebase` משנה את ההיסטוריה של הבראנץ' ולא מומלץ לשימוש בפקודה בבראנץ'ים שעוד אנשים תורמים להם, כי אז עלולים לגרום לשגיאות בלתי צפויות. ככלומר, אפשר להשתמש בـ`git rebase` על מנת למציג את `main` עם הבראנץ' שלנו (כדי לוודא שאין קונפליקטים לפני שמציגים את הבראנץ' שלנו בחזרה עם `main`), אבל לא מומלץ לעשות ההפך.

בחלק גדול מצוותי הפיתוח שאני מכיר לא נהוג להשתמש בـ`git rebase` כיוון שבתפיסה פיתוח מודרנית, מסתכלים פחות על קומיטים ויוטר על פול ריקוסטים, עליהם נלמד בהמשך.

## קובץ `gitignore`

לעתים יש ברייפוזיטורי שלנו קבצים או תיקיות שאין לנו רצים להכניס לgit. הדוגמה המוכרת ביותר היא קובצי `node_modules` שנוצרים לאחר שמקינים מודולים של `Node.js` (גם בפרויקטים מבוססי `js`). גם `create react app`, אבל הכוונה גם לקובצי סביבה, קבצים שנוצרים אוטומטית בתהליך בילד וקבצים אחרים שאין לנו רצים שייכנסו חלק מניהול הגרסאות שלנו. בדוק עבור זה יש לנו קובצי `gitignore` יש להקליד נקודה לפני השם, כה:

`gitignore`.

לא סימת של קובץ – רק נקודה ו-`gitignore`. מדובר בקובץ שהוא קובץ טקסט פשוט. אפשר לפתח אותו באמצעות `pad` או בעזרת ה-IDE שלנו. אנו יוצרים קובץ זה ומציבים אותו בתיקייה הראשית של הפרויקט. בכל שורה אפשר לכתוב שם של קובץ או של תיקייה ואת הקבצים שמתאים לשם זהה. GIT לא יראה אותם, הם כאילו לא היו קיימים מבחינתו.

למשל, קובץ `gitignore` שיש בו את השורה זו:

`node_modules/`

ימנע מgit להכנס את התקינה `node_modules` ומה שיש בה אל הריפוזיטורי. הוא פשוט יתעלם ממנו.

אם נכנס שורה עם סולמית לפניה, זה ייחשב להערה. למשל:

```
#Dependency directories
node_modules/
jspm_packages/
#Snowpack dependency directory (https://snowpack.dev/)
web_modules/
```

זה בעצם מפרט מדוע הכנסנו את התקיות האלו ל-`.gitignore`. יש דוגמאות רבות לקובץ `gitignore`. מומלצים ודרך כלל השלב הראשון בכל פרויקט, עוד לפני הקładת שורת הקוד הראשונה, הוא ליצור קובץ `gitignore`. באתר הבא יש דוגמאות רבות לקבצים כאלה בשפות שונות:

<https://github.com/github/gitignore>

### למה חשוב להשתמש ב-`.gitignore`.

חשוב להשתמש ב-`.gitignore` כי הריפוזיטורי אמור להכיל את הקוד שלנו בלבד ולא תוספות אחרות. אם נכנס קוד שאינו שלנו לתוכה הריפוזיטורי, ובמיוחד תופעות של קוד שאינו שייך לנו ונוצר אוטומטית – יהיה לנו קשה לבדוק את השינויים בין גרסאות שונות וכל שינוי קטן בקוד יכול לכלול המון שינויים שנתקשה לעקוב אחריהם. זו הסיבה לכל החשוב ממש: רק קוד שאנו כתבimos נכנס לתוכה הריפוזיטורי. קוד, קבצים או מידע שנוצרו אוטומטית מתהליכים שונים, לוגים, קבצים שה-IDE יוצר, קבצים שיורדים אוטומטית מתהליכי התקנה וכו' – לא נמצאים בריפוזיטורי. זהו כלל ברזל שמקפידים עליו בכל צוות פיתוח ובכל חברה.

## git detached

אם נשחרר קומיט מסוים באמצעות git checkout או git hash, נקבל הודעה שאנו נמצאים במצב detached. בפרק על שחרור קומיט הסברתי שכאשר משחזרים קומיט זהה, אפשר לבדוק את הקבצים ולהזור באמצעות main.git checkout.

```
C:\local\example-repository>git checkout 2710741a459a739805364cceb92e8af5670be173
Note: switching to '2710741a459a739805364cceb92e8af5670be173'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 2710741 Some Changes #2
```

אך מצב detached מאפשר לנו לעשות מעט יותר אם אנו רוצים בכך. ראשית, מה הוא מצב detached? הוא מתרחש כאשר אנו חוזרים בזמן לקומיט מסוים, ואז אנחנו בעצם "מנוטקים" מהזמן הנוכחי שלנו. אם נעשה git status, נראה שיש ציון של הקומיט שבו אנו נמצאים וציון של detached באדום:

```
C:\local\example-repository>git status
HEAD detached at 2710741
nothing to commit, working tree clean
```

אם רק רוצים לבדוק את הקבצים – אין בעיה, אבל מה קורה אם רוצים לבצע שחרור? זה יכול להתרחש אם למשל כתבנו קוד שיוצר תקלת לפני כמו קומיטים ואנו רוצים להחזיר את המצב לקדמותו, לפחות חלקית. מה עושים? במקרה זהה נרצה לצאת אל בראנץ' אחר מהקומיט ואחר כך לבצע מיזוג בחזרה אל הבראנץ' שלנו. אין עושים את זה? כאשר אנחנו במצב detached, אנו יוצרים מהקומיט בראנץ' שעומד ברשות עצמו באמצעות:

```
git switch -c my-new-branch-from-old-commit
```

שם הבראנץ' יכול להיות כרצוננו, כמובן. בסופו של דבר יש בראנצ' שיצא מהקומיט הזה ואני יכול לעשות אותו כרצוני, כמובן. וכך יוצאת ממנה בעבר. הקומיטים החדשניים שקיים ב-`main` לא יהיו שם.

## git bisect

כלי נחדר ו שימושי מאוד שמאירה את הכוח של גיט בפתרון בעיות הוא `git bisect`. מדובר בכלל שמאפשר לנו לעשות "אריה במדבר" באופן אוטומטי וכך למצוא גרסאות בקלות.

יכול להיות שהמונה "גרסיה" (Regression) לא מוכר לכם. מה זו גרסיה? מדובר בתקלה בקוד שלנו שפעם עבד היטב. בגודל, יש שני סוגים תקלות. התקלה הראשונה היא תקלה רגילה, באג. למשל, אם יש לנו פונקציה שמוסיפה מע"מ וקורסת כאשר היא מקבלת קלט שלילי – זה באג רגיל שמתרכש כי לא חשבנו שמשהו יכנס מספר שלילי לפונקציה. הפתרון של הבאג הוא פשוט לנצל את הקלט השלילי – הדפסת שגיאה או כל דבר אחר.

גרסיה היא באג ש חוזר על עצמו או תקלה שמתרחשת במקום שלא הייתה בו קודם תקלה. למשל, פונקציה שמוסיפה מע"מ שעבده היטב חדשנים ושנים ופתחות מפסיקה לעובוד, או אם אחרי שתיקנו את תקלת המספר השלילי בפונקציית המע"מ, אחרי שלושה חדשנים התקלה הזו חוזרת – זה נקרא גרסיה.

אחד הדרכים לפתורן גרסיה היא חיפוש בינארי, או בשמו הפופולרי שיטת "אריה במדבר". השיטה זו פשוטה. נניח שיש אריה בודד במדבר עצום. איך מוצאים אותו? מחלקים את המדבר לשניים. מחפשים אותו בחצי המדבר שבוחרים באופן אקראי. הוא שם? לא? אז אפשר להסיק בוודאות שהוא נמצא בחצי המדבר השני. לוקחים את חצי המדבר השני, מחלקים אותו לשניים (כלומר שני רביעים של המדבר המקורי), בוחרים באופן אקראי את אחד החלקים ומחפשים בו את הארייה. לא מצאנו? אפשר לומר שהוא נמצא בחלק השני. לוקחים את החלק שנשאר ומחלקים אותו לשניים, וכך הלאה, ממשיכים לחלק את השטח עד שמדוברים את הארייה.

כך `git bisect` עובדת. נניח שיש לנו אתר ויום אחד מדווחים לנו שדף מסוים לא עובד. אנחנו יודעים שהוא עבד בעבר, ולפיכך מדובר ברגרסיה. איך מוצאים את המקום שבו בעצם נוצרה הרגרסיה? עם `git bisect`. מתרימים קומיט כלשהו בעבר ובודקים שם התקלה לא קיימת – במקרה שלנו, שהדף זהה עובד. ואז מכניסים את `git bisect` לתמונה. אומרים לה:

במצב הנוכחי יש לנו תקלה, התקלה לא קיימת בקומיט עם ה-`ref` זהה.

git `bisect` עושים באופן אוטומטי `checkout` לקומיט כלשהו באמצע הדרך בין הקומיט הנוכחי, `HEAD` שלנו, לבין הקומיט שנתנו ושואלת אותנו אם התקלה קיימת. אנחנו מרכיבים את הקוד ועונים לה אם כן או לא. היא ממשיכה לחלק את המדבר וושה `checkout` לקומיט אחר ושואלת שוב – האם התקלה קיימת? כן או לא? אנו עונים לה, וכך הלאה, עד שמקבלים את הקומיט המדווח שבו נוצרה התקלה, ואז באמת אפשר לראות מה השתנה בקוד מהנקודה שבה הכל עבד באופן מאוד מוגבל ומדויק.

איך מפעילים את פקודת הקסם זו? באמצעות `git checkout` מתרימים קומיט כלשהו בעבר שבו הכל עבד, חוזרים ל-`HEAD`, ככלומר לגרסה הנוכחית, ואז מקלידים:

```
git bisect start
```

כדי להתחיל את התהלין. Unless נציגו בפני GIT מהי הגרסה הרעה – ככלומר זו שיש בה את התקלה. אם אנו נמצאים על הגרסה זו, נכתבו:

```
git bisect bad
```

Unless נציגו בפני GIT מהו הגרסה הטובה – ככלומר זו שאין בה את התקלה. נקליד `git bisect good` ואז את ה-`ref`. למשל:

```
git bisect good 5c82fc
```

ואז `git bisect` תיכנס לפעולה, התוכנה תעבור מיד לקומיט אחר, תציג את מספר הצעדים המשוער שיש לה עד למציאת התקלה ותשאל אם הגרסה זו תקינה. אם כן, נקליד `git bisect good`. אם לא, נקליד `git bisect bad`. וכך הלאה, עד ש-`git bisect` נמצא תאריך במדדיק את הגרסה שבה הכל השتبש ותעביר אליה במצב `detached`.

```
C:\local\example-repository>git bisect start

C:\local\example-repository>git bisect bad

C:\local\example-repository>git bisect good 5c82fc
Bisecting: 3 revisions left to test after this (roughly 2 steps)
[2710741a459a739805364cceb92e8af5670be173] Some Changes #2

C:\local\example-repository>git bisect bad
Bisecting: 1 revision left to test after this (roughly 1 step)
[05fe9b0fe90939d50ec06668a9e09117a02e3510] Some Changes #1

C:\local\example-repository>git bisect good
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[486dbf97aaee5a1b7fcdfdbe041d38ce6651e183] Some Changes #1

C:\local\example-repository>git bisect good
2710741a459a739805364cceb92e8af5670be173 is the first bad commit
commit 2710741a459a739805364cceb92e8af5670be173
Author: Ran Bar-Zik <ran@bar-zik.com>
Date:   Sat Sep 11 09:45:33 2021 +0300

        Some Changes #2

 README.md | 3 +-+

```

זו הסיבה, אגב, שמקפידים על קומיטים קצריים ממש וautomated (מלשון אוטום, יחידה קטנה) כדי שנוכל לאתר תקלות בקלות. הקומיטים הקצריים מאפשרים לנו להבין ב מהירות יחסית מה השתנה זהה מסוייע לנו להבין מה השتبש.

מערכות קוד מודרניות הן מורכבות, ולעתים השינויים שאנו עושים בהן הם מורכבים. לכן כדאי לכתוב קומיטים קצריים – כדי שם שהוא ישتبש, נוכל להבין באופן מהיר יותר מה קרה.

כאשר מקלידים git bisect, עוברים למצב אינטראקטיבי. זה יכול להלחיץ קצר ולסמן, כי פתאום אם ננסה לעשות קומיט או פעולה אחרת, נקבל התראה. בכל רגע נתון אפשר לצאת ממצב git bisect באמצעות:

```
git bisect reset
```

git bisect היא שיטה נחדרת למציאת גרסאות בשיטת "אריה במדבר". במבט ראשון היא נראה כמו קסם, אבל הוא פשוט מחליפה פעולות ידניות ומאוד-מאוד נוחה. נכוון, זה נראה קשה בהתחלה, אבל למידה של git bisect יכולה לשדרג מאוד את היכולות המקצועיות שלכם.

## עבודה עם שרת גיט מרוחק

בפרקים הקודמים למדנו לעבוד מול גיט מקומי, כזה שנמצא על המחשב שלנו ובריפוזיטורי המקומי שלו. כל השינויים נשמרים בתיקיית git והמקומית במחשב. אם נמחק אותה – כל הקומייטים, הבראנצ'ים והמרג'ים שייצרנו ייעלמו. במקרה כזה גם העבודה היא מקומית; אי-אפשר לפתוח או לשתף פעולה עם מתכנתים אחרים. נכון, נדרש לעבוד מקומי, אבל מתיישה צריך לנכון את העבודה שלנו עם שרת מרוחק. זה גם הכוח של גיט – עבודה בקלהות לצד המקומי אבל גם הסתמכנות מהירה עם השרת המרוחק, ודרךו עבודה עם מתכנתים אחרים.

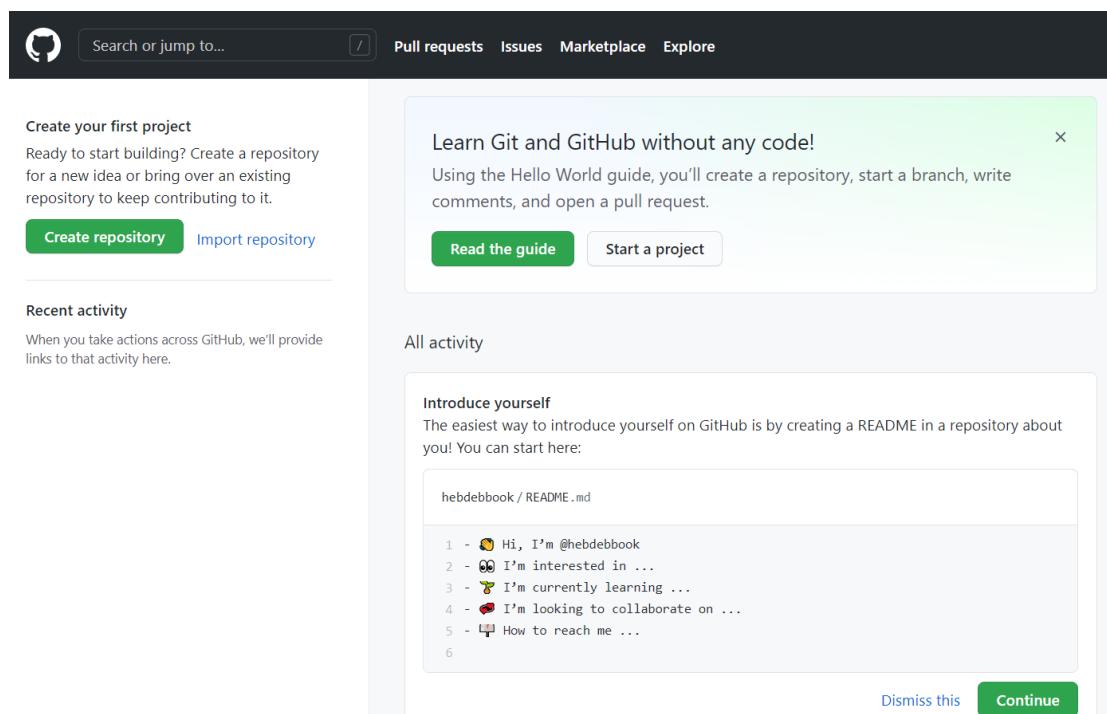
שרת גיט מרוחק הוא שרת שלא יושב במחשב שלנו אלא במחשב אחר. הגיט שהתקנו במחשב שלנו נקרא גיט קלינט. הגיט המותקן בשרת הוא גיט אחר, שנקרא שרת גיט (git server). זו תוכנה כמו כל תוכנה אחרת במחשב, שרצה ברקע ומאזינה לפורט 9418. פורט הוא פשוט דרך כניסה אל המחשב, ואין במידע זהה חשיבות גדולה (הסביר מפורט על פורטים תמצאו בספר "למוד Node.js בעברית"). תוכנת הגיט במחשב שלנו יכולה להתחבר אל הפורט הזה בשרת מרוחק.

אפשר להתקין שרת גיט (כלומר את התוכנה) על כל מחשב. ברגע שהתקינו אותו, כל אחד בעולם שיש לו את כתובת המחשב שלו מותקן שרת הגיט יכול להתחבר אליו. המחשב הזה יכול להיות מחשב אחר בראשת המקומית (אפיקו וספברי פיי – מחשב קטן וזול), שרת בראשת חיצונית גלויה או שירות SaaS (Software as a Service, כלומר תוכנה כשירות). מה זה אומר? מדובר בשרת מרוחק שמופעל על ידי חברת חיצונית שמשכירה את השרת שלה לחברות אחרות יחד עם מסק גרפי נוח לתפעול. ככלומר, בمكان להתקין על מחשב בית או עסק שרת גיט ולדאוג לניהל אותו ואת הרשאות שלו – פשוט משלימים לחברת שתנהל את הכל וגם תיתן לנו מסק גרפי נוח לניהול. שירותים מפורטים לגיט ממש מסוג זה, המאפשר גם חשבונות חינמיים לאנשים פרטיים, הוא גיטהאב, שנחפק ממש לסטנדרט בעולם הפיתוח. גיטהאב הייתה חברת עצמאית פופולרית ומצליחה שנרכשה על ידי מיקרוסופט. אף על פי שהיא סטנדרט, יש לה מתחרים כמו גיטלאב או ביטבאקט. לחלק מהחברות יש פיצ'רים שונים, ממשקים גרפיים שונים ויכולות אינטגרציה אחרות עם שירותים שונים – אבל כולן-כולן משתמשות באותה מערכת גיט והשימוש בכלל זהה. בפרק הזה נדגים עם גיטהאב הפופולרי, אבל המידע הזה רלוונטי לכל חברת שהיא.

## פתרונות חשבון בGITLAB

האתר של גיטהאב יושב בכתובת <https://github.com> וכל אדם יוכל ליצור בו חשבון ללא עלות באמצעות בחירה בcpfator ה-`sign up`. מומלץ לבחור שם משתמש ויחודי וסימנה מאובטחת יחד עם אימות דו-שלבי. אני הכנסו מייל אמיתי, כיוון שתצטרנו לאמת אותו.

אחרי אימת המail, תיכנסו בעצמכם לחשבון שלכם:



בחשבון אפשר למלא פרטים נוספים כדי שימושים אחרים יכירו אתכם, אבל אנו נרצה ליצור ריפורטורי מרוחק. ריפורטורי הוא בדיקת כמו פקודת `git init`. הוא יוצר את אותה תיקית `.git` אבל לא במחשב שלכם אלא בשרת המרוחק. בממשק יש כפתור שנקרא `Create repository`.

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

---

Owner *	Repository name *
 <b>hebdebbook</b> 	/ <input type="text"/>
Great repository names are short and memorable. Need inspiration? How about <a href="#">friendly-train</a> ?	
Description (optional)	
<input type="text"/>	
<input checked="" type="radio"/>  <b>Public</b> Anyone on the internet can see this repository. You choose who can commit.	
<input type="radio"/>  <b>Private</b> You choose who can see and commit to this repository.	
<b>Initialize this repository with:</b> Skip this step if you're importing an existing repository.	
<input type="checkbox"/> <b>Add a README file</b> This is where you can write a long description for your project. <a href="#">Learn more.</a>	
<input type="checkbox"/> <b>Add .gitignore</b> Choose which files not to track from a list of templates. <a href="#">Learn more.</a>	
<input type="checkbox"/> <b>Choose a license</b> A license tells others what they can and can't do with your code. <a href="#">Learn more.</a>	
<input style="background-color: #4CAF50; color: white; padding: 5px; border-radius: 5px; border: none; width: 100%;" type="button" value="Create repository"/>	

נצרך לבחור שם לריבזיטורי. כאשר יצרנו ריבזיטורי מקומי במחשב שלנו עם פקודה `init`, שם הריבזיטורי יהיה שם התקינה. בગיטהאב צריך לקבוע את השם. השם מורכב מתחלית שהיא שם המשתמש שלנו – למשל `barzik` או `hebdevbook` – ואז שם הריבזיטורי.

אפשר לבחור אם הריבזיטורי יהיה ציבורי (כלומר שכל אחד יוכל לראות אותו) או פרטי (שרק אתם תוכלו לראות אותו). אפשר להוסף קבצים בסיסיים כמו `README`, קובץ `gitignore` או רישיון שימוש שמקובל להוסף לפרויקט תוכנה.

אחרי יצירת הריבזיטורי, נועבר אל הריבזיטורי עצמו, שכרגע הוא ריק. עכשו אפשר להתחיל.

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#) <https://github.com/hebdebbook/example-repository.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# example-repository" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/hebdebbook/example-repository.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/hebdebbook/example-repository.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Firefox, Chrome, Opera, Edge, Internet Explorer, Microsoft Word icons

אפשר לראות בחלק הכתובת בדף את כתובות הריביזיטורי. אם היא פומבית, כל אחד בעולם יוכל להקליד אותה ולראות את הריביזיטורי. במקרה שלנו הכתובת היא:  
<https://github.com/hebdevbook/example-repository>

אם תיכנסו לפרויקטים אחרים בגיטהאב, תראו שלכולם יש את אותו המבנה וכולם זהים לכתובת זו. למשל:  
<https://github.com/facebook/react>

שם המשמש או הארגוןפה הוא `facebook`, שהוא המפתחת של `React`.  
 אצל הריביזיטורי המרוחק כרגע קיים. הריביזיטורי המרוחק הוא בדיקן כמו הריביזיטורי שעבדנו עליו עד עכשיו – יש לו בראנצ'ים, קומיטים, מרג'ים – הכל. ההבדל העיקרי בין  
 לבין מה שיש לנו במחשב הוא שהוא פשוט במחשב אחר.

## משינית ריביזיטורי

הדרך הפешטה ביותר לעבוד עם ריביזיטורי מרוחק היא ליצור אותו במחשב. ברגע שיש לנו ריביזיטורי פומבי, כל אחד בעולם יכול להעתיק אותו למחשב המקומי – הפקודה של

העתיקת הריביזיטורי היא `git clone` ואותו שם הריביזיטורי המרוחק. בדוגמה שלנו ניכנס אל הטרמינל, ללא צורך בפתיחת תיקיה, ונבצע `clone git`. אם נקליד את הפוקודה זו:

```
git clone https://github.com/hebdevbook/example-repository.git
```

נראה שנוצרה לנו תיקייה בשם `example-repository` במקום שהkładנו את הפוקודה זו. ניכנס אל התיקייה באמצעות:

```
cd example-repository
```

ואם נקליד `status`, נראה שיש לנו את העותק של הריביזיטורי המרוחק.

```
C:\local>git clone https://github.com/hebdevbook/example-repository.git
Cloning into 'example-repository'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

C:\local>cd example-repository

C:\local\example-repository>git status
On branch main
Your branch is up-to-date with 'origin/main'.
nothing to commit, working tree clean
```

**שימוש לב:** אפשר להתחבר לגיטהב בשני פרוטוקולים, פרוטוקול HTTPS ופרוטוקול SSH. פרוטוקול HTTPS הוא פרוטוקול שפותח משתמשים בו, אך אפשר לבצע אותו הורדה ללא צורך ברישום. אנו נלמד בהמשך על פרוטוקול SSH.

אחרי שהורדנו באמצעות `git clone` את הריביזיטורי המרוחק, אנו עומדים בפני מצב חדש: כל העת יהיו לנו שני ריביזיטורי – מקומי ורחוק, ואנו נבחר מתי לՏנכרן אותם. הם לא חייבים להיות תואמים, והבחירה לՏנכרן היא בידיינו.

ברגע שיש לנו גיט מקומי ורחוק – אנו מנהלים את הקישור בעזרת הפוקודה `git remote`, שאחריותה לניהול הקשר בין הריביזיטורי המקומי לריביזיטורי המרוחקים יותר. לכל ריביזיטורי מקומי יכול להיות ריביזיטורי מרוחק אחד או יותר.

כך למשל אנו בודקים אם הריביזיטורי המקומי מסונכרן עם הריביזיטורי המרוחק:

```
git remote -v
```

התוצאה היא שיש לנו שני קישורים – אחד לדחיפה (push) ואחד למשיכה (fetch).

```
C:\local\example-repository>git remote -v
origin  https://github.com/hebdevbook/example-repository.git (fetch)
origin  https://github.com/hebdevbook/example-repository.git (push)
```

שניהם נקראים `origin`. כמובן, יש לנו ריבזיטורי מרוחק אחד, בשם `origin`, שהכתובת שלו היא:

<https://github.com/hebdevbook/example-repository.git>

ואפשר להשתמש בו גם למשיכה וגם לדחיפה.

הבה נבדוק את המשיכה. אם נקליד את הפקודה `git fetch all`, שמוסכת ענפים מרוחקים, נקבל חיווי של `origin Fetching`, מה שאומר שיצרנו קשר עם הריבזיטורי המרוחק ומשכנו את כל הענפים. אם מתכנת אחר יוצר ענפים כלשהם בריבזיטורי המרוחק עם פקודת `git fetch all`, אנו נמשוך אותם ונוכל לעבור אליהם באמצעות `git checkout` בריבזיטורי המקומי כדי לנצלו אנו בעצמן יצרנו אותם.

אם אנו נמצאים בבראנץ' מסוים שכבר קיים, שמיishaו שינה את הקוד שלו בריבזיטורי המרוחק, ורוצים למשוך את הקוד החדש אליו, נקליד:

`git pull origin`

מה שagit תעשה במקרה זה הוא לכת לבראאנץ' בריבזיטורי המרוחק, ששמו זהה לבראנץ' שבו הקלדנו את הפקודה, ולמשוך את כל הקוד ממנו אל הבראנץ' בריבזיטורי המקומי. אם יש לנו קונפליקטים, נצטרך לסדר אותם, בדיקן כמו כל מרג' רגיל. אם אין כל שינוי, נקבל חיווי שאין שינוי.

```
C:\local\example-repository>git pull origin
Already up-to-date.
```

אנו יכולים למזג בראאנץ' מרוחק לבראאנץ' שלנו באמצעות ציון שם הבראנץ'. למשל:

`git pull origin BRANCHNAME`

גם כאן יהיה לנו מרג' רגיל, עם קונפליקטים או בלי קונפליקטים. זהו החלק המבלבל ביותר בכל מה שקשרו לGIT ולהתנהלות מול GITהאב – הסנכרון הזה. צורך לזכור שפשוט מדובר במשיכת קוד מהריפורזיטורי מרוחק, שיכולים לתרום לו מאות אנשים, לריפורזיטורי המקומי. ואף שמדובר בשני ריפורזיטורי זחים, יכולים להיות כל הזמן שינויים ביניהם, בגלל שינויים שעשינו בriforzitori הרחוק השתנה או בriforzitori המקומי.

**שימוש לב:** origin הוא הכינוי של השירות המרוחק. בהמשך נלמד על עבודה עם כמה שירותים מרוחקים.

## דחיפת קוד

משיכת הקוד,riforzitori פומבי, יכולה להיעשות בכל מקום ומכל מקום בעולם, אבל מובן שהכנסת קוד לריפורזיטורי אפשרית רק על ידי מי שמורשה לעשות כן. במקרה שלנו אלה בעלי הריפורזיטורי. הגיוני, אף שמדובר בקוד פתוח, למנוע מכל אדם אקרים להכניס קוד. לקרוא? כל אחד יכול. לנתח? רק בעלי הריפורזיטורי.

אפשר לבצע הזרחות בשני פרוטוקולים. האחד הוא HTTPS והשני הוא SSH. קודם השתמשנו ב-SSH, כשהיצענו clones git, אך הפרוטוקול המקובל הוא פרוטוקול SSH (Secure Shell). מדובר בפרוטוקול מאוד מקובל המשמשים בו המונע, לחיבור לשרתים למשל או לתקשורת מול שירותים, כמו GITהאב, גם הוא שירות.

על מנת לעבוד עם SSH ולהזרחות מול GITהאב, משתמשים באמצעות הזרחות שנקרה מפתח איסימטרי.

הזרחות איסימטרית היא דרך הזרחות מקובלת שכדי להכיר, אך לא חובה להכיר לעומק. אתם יכולים פשוט לעקוב אחר ההוראות.

באמצעי ההזרחות האיסימטרי יוצרים מפתח ציבורי ומספר מפתח פרטי. המפתח נקרא מפתח קריפטוגרפי - ולא ניכנס כאן לתיאוריה שמאחורי המפתחות הקריפטוגרפיים או בכלל להסביר מהי קריפטוגרפיה, וגם אין צורך להבין זאת על מנת להשתמש במפתחות. המפתח הציבורי והמפתח פרטי ישערנו הם קבצים טקסטואליים לכל דבר, נשמרים במחשב המקומי שלנו. בעזרתו המפתח הפרטי אנו יכולים לבצע הצפנה, ובעזרתו המפתח הציבורי אין שום בעיה

לשלוח לכל אחד או להעלות לאתר גיטהאב. המפתח הפרטי נותר אצלנו ואנו מורים לתוכנת ה-SSH, התוכנה שאיתה אנו מתחברים לשרתים, ובמקרה זהה לגיט, המותקנת בכל מחשב, להשתמש בו לכל חיבור. כך נוכל לבצע פעולות מול גיטהאב בלי להקליד סיסמה בכלל פעם.

על מנת ליצור את המפתח הציבורי והפרטי יש להיכנס לטרמינל בחלונות, במק או בלינוקס ולהקליד:

```
ssh-keygen -t ed25519 -C "support@hebdevbook.com"
```

עם כתובת המייל שאיתה נרשמתם לגיטהאב במקום המייל בשורת הפקודה. אנו יוצרים מפתח. במהלך ייצורו נדרש לאשר אותו ולהקליד סיסמה נוספת אם רוצים. בשלב זה פשוט נקיש על אונטר ונמשיך להלאה, ללא ייצור סיסמה נוספת.

```
C:\local\example-repository>ssh-keygen -t ed25519 -C "support@hebdevbook.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (C:\Users\barzik/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\barzik/.ssh/id_ed25519.
Your public key has been saved in C:\Users\barzik/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:ueMleV5zRIhACun11dD+FJStOoJLNxXhbJoIq3mHjGo support@hebdevbook.com
The key's randomart image is:
+--[ED25519 256]--+
|   .. o .+.o.o |
|   .... .++..o .|
|   . .o. ....=..o |
|   . o.o +o.o |
|   . S.o. +. |
|   = .oo+ o.. |
|   + +.*oooo.. |
|   E. . o.* . o |
|   .. . . |
+---[SHA256]-----+
```

בסיום התהליך, בתיקייה של המשתמש שלנו ששם ssh נראה שני קבצים:

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

Directory of C:\Users\barzik\.ssh

08/28/2021  12:11 PM    <DIR>      .
08/28/2021  12:11 PM    <DIR>      ..
09/28/2020  10:28 AM           62 config
08/28/2021  12:11 PM          419 id_ed25519
08/28/2021  12:11 PM          105 id_ed25519.pub
08/28/2021  11:49 AM          2,956 known_hosts
09/26/2020  06:38 PM          1,811 known_hosts.old
                           5 File(s)        5,353 bytes
                           2 Dir(s)   15,868,743,680 bytes free
```

האחד, שהסימת שלו היא `sk`, הוא המפתח הציבורי. השני הוא המפתח הפרטי; הוא  
ישמר בסוד ולעולם אין לשთף אותו.

עכשו עליינו לבצע את רישום המפתח הפרטי ל-SSH, כך שהוא ידע לעבוד איתו. CAN יש  
הבדל בין חלונות 10 למק או לינוקס.

במק וلينוקס זה פשוט:

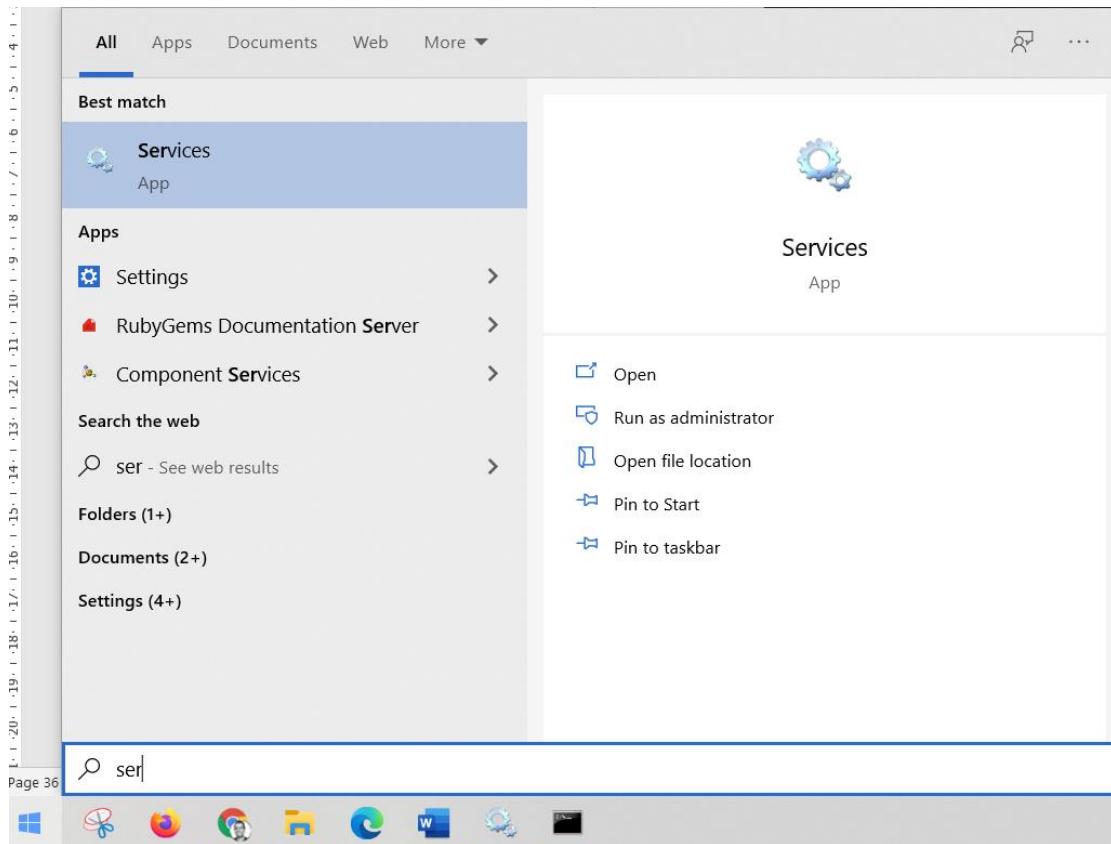
```
eval "$(ssh-agent -s)"
```

:1@CAN:

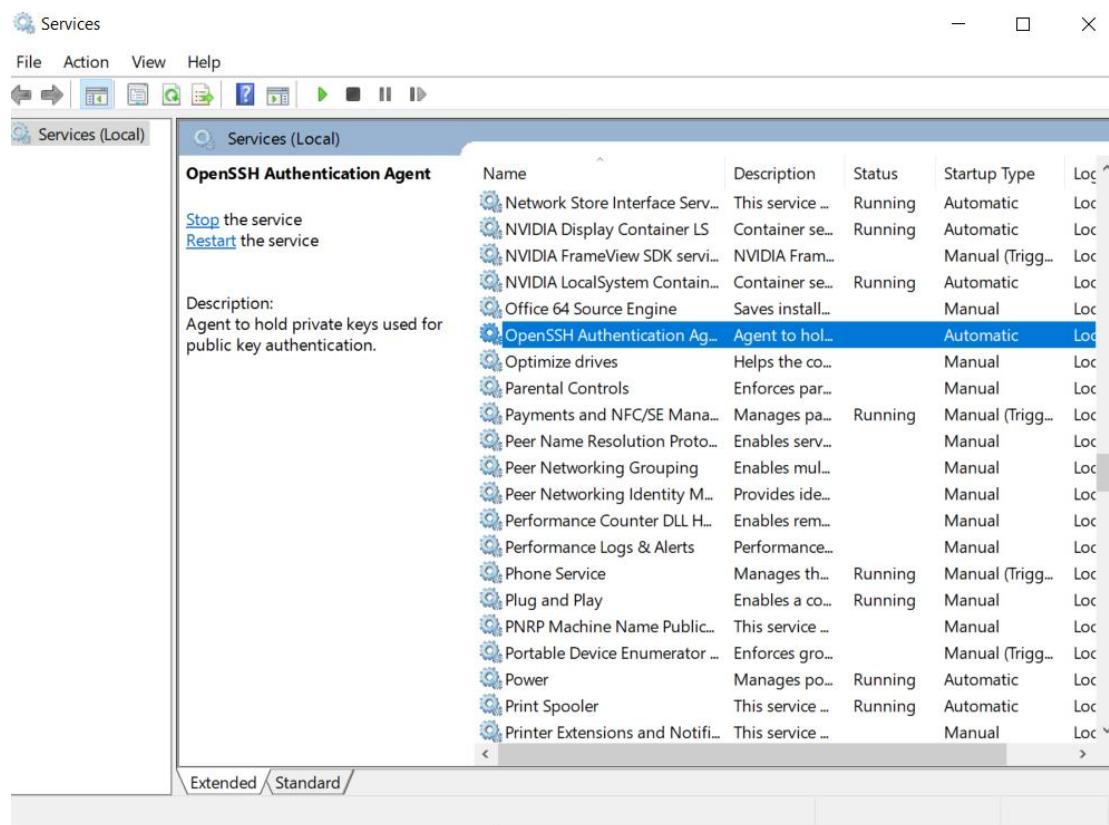
```
ssh-add ~/.ssh/id_ed25519
```

או שם המפתח שלנו.

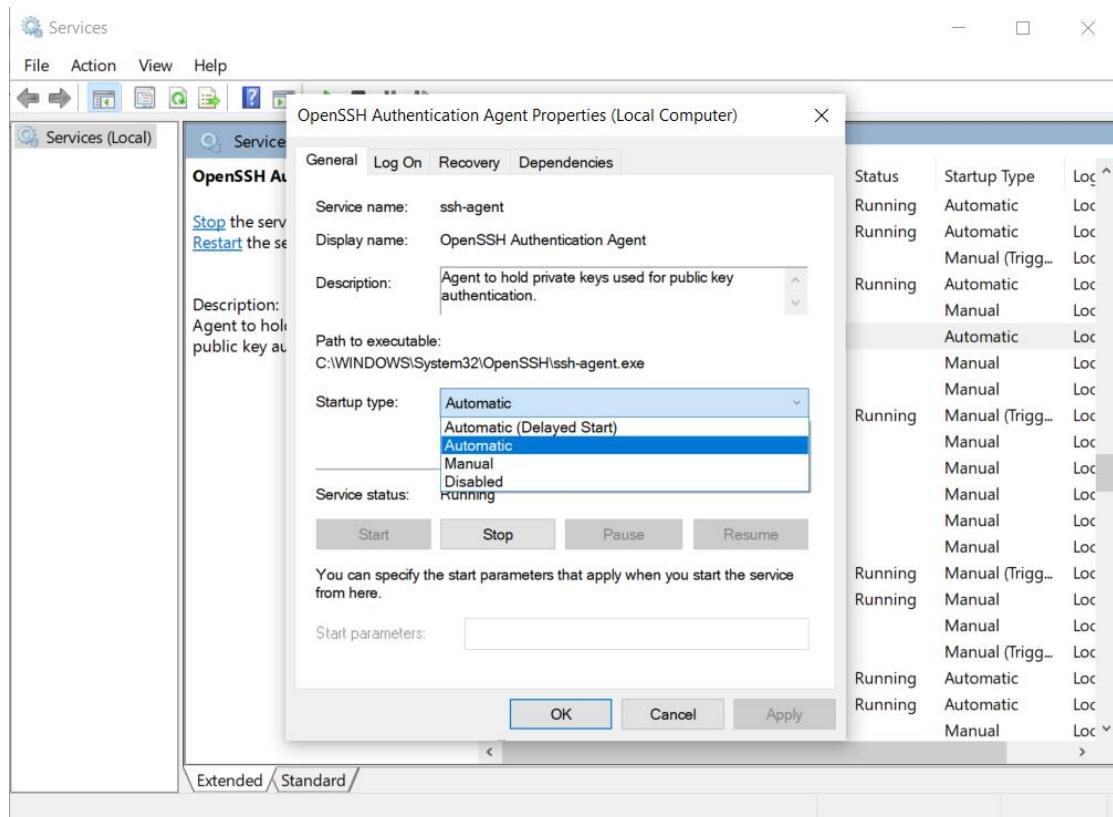
בחלונות 10 זה מעט יותר מסובך. ראשית עלינו להיכנס אל תוכנת services, ככלומר  
לחוץ על איקון החלונות ולהתחיל להקליד services:



התוכנה מאפשרת לנו להגיע לרשימת השירותים המלאה שיש במערכת הפעלה. נחפש ברשימה הארכוה את OpenSSH Authentication Agent.



לחיצה כפולה עליו תציג את האפשרויות שלו. נוודא שב-**type** **Startup** כתוב **Automatic**.



נסמוך, נסגור את כל חלונות הטרמינל ונפתח אותם מחדש. כעת כל מה שעשינו לעשות הוא להקליד:

`ssh-agent`

:תא

`ssh-add C:\Users\barzik\.ssh\id_ed25519`

עם החלפה של הנטייה בנתיב שלנו נמובן.

```
C:\Users\barzik>ssh-agent
C:\Users\barzik>ssh-add C:\Users\barzik\.ssh\id_ed25519
Identity added: C:\Users\barzik\.ssh\id_ed25519 (support@hebdevbook.com)
```

מהנקודה זו – כל תקשורת עם ה-SSH תבצע אוטנטיקציה עם המפתח הפרטי שלנו. אבל צריך שהצד השני יוכל את המפתח הציבורי. נתחבר לחשבונו בגייטהאב ונגיע אל דרך תפריט ה-`Settings` של גיטהאב:

The screenshot shows the 'Account settings' sidebar with 'SSH and GPG keys' selected. The main area has two sections: 'SSH keys' and 'GPG keys'. Both sections indicate no keys are associated with the account and provide links to generate keys. A 'Vigilant mode' section is also shown.

נבחר ב-'key New SSH key'. ייפתח לנו מסך שבו נוכל לכתוב תיאור קצר של המפתח ויהיה מקום להדיביך את המפתח הציבורי שלנו, שכאמור נמצא בקובץ בתיקיית `.ssh` עם הסימטת `pub`. את המפתח זהה יצרנו קודם.

כדי למצוא את המפתח הציבורי השתמש בפקודת `type` בחלונות:

```
type C:\Users\barzik\.ssh\id_ed25519.pub
```

או `cat` במק ובלינוקס:

```
cat ~/.ssh/ id_ed25519.pub
```

נעתיק את התוצאה שתתקבל:

```
C:\Users\barzik\.ssh>type id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINzoQDNk0UrhUVNHHsCm1Ew3oXmpq1LYQW0eL1/w7ltB support@hebdevbook.com
```

ונדביך במקום המועד, מהטקסט ssh-ed25519 עד המיל ועד בכלל.

## [SSH keys / Add new](#)

Title

Hebdevbook PC key

Key

```
ssh-ed25519 AAAAC3NzaC1IzDI1NTE5AAAAINzoQDNkOURhUVNHHsCm1Ew3oXmpqlLYQWOeLi/w7ltB
support@hebdevbook.com|
```

[Add SSH key](#)

נלחץ על Add SSH key זהה. ביצענו את תהליך ההזדהות. על מנת לוודא שהcoil תקין  
נשתמש בפקודה:

```
ssh -T git@github.com
```

מדובר בפקודת ניסיון חיבור. אם הכל תקין, נקבל פלט בסגנון:

```
Hi hebdevbook! You've successfully authenticated, but GitHub
does not provide shell access.
```

כמובן עם שם המשתמש שלנו. זה מראה שהcoil מצוין ותקין ושאפשר להתחיל לעבוד.  
את ה-clone או כבר אפשר לבצע עם SSH וגם מומלץ לעשות זאת.

חיבור HTTPS נחשב לטוב ומקובל יותר מחיבור HTTPS ומומלץ מאוד לא להתנצל  
ולהשתמש בו. מעבר לכך שהוא קל יותר ומאובטח יותר, גם לא מעט שירותים משתמשים  
בו, והוא אף מראה על רצינות ועל מקצועיות.

**אם יש תקלה – פעלו בהתאם לצעדים הבאים:**

1. ודאו שהמפתח שהעתקתם והדקתם לחשבון הגיטהאב שלכם הוא המפתח

הציבורי ללא תוספות או שינויים.

2. ודאו שבאמת מדובר בפתח הציבורי שモופיע בתיקיית ssh.
3. משתמשי חלונות – מומלץ להשתמש ב-`git-bash`, שזה תחליף `cmd`.

אחרי שהתחברנו עם SSH, נבצע `git clone` לרייפוזיטורי שלנו, הפעם עם SSH.

```
git@github.com:hebdevbook/example-repository.git
```

```
C:\local>git clone git@github.com:hebdevbook/example-repository.git
Cloning into 'example-repository'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 eceiving objects: 33% (1/3)
Receiving objects: 100% (3/3), done.

C:\local>ssh -T git@github.com
Hi hebdevbook! You've successfully authenticated, but GitHub does not provide shell access.
```

## דחיפה לקוד המרוחק

אחרי שביצענו ווידאנו את ההזדהות, אנו יכולים לבצע דחיפה של כל דבר לקוד של הריפוזיטורי שלנו, החל בבראנץ' חדש לחלוטין על הקוד שיש בו ועד שינויים בבראנץ' שכבר קיים. הפקודה לדחיפת קוד לרייפוזיטורי מרוחק נקראת `git push`. כדי לדחוף את הקוד לבראנץ' חדש מקלידים:

```
git push --set-upstream origin some-feature-branch
```

הפקודה יוצרת בשורת המרוחק את הבראנץ'. זהה המשמעות של הפלאג `set-upstream`. אנו חייבים להכניס את שם הבראנץ' שלנו כדי ליצור אותו ברייפוזיטורי המרוחק ולהכניס אותו אחרי שהוא נוצר.

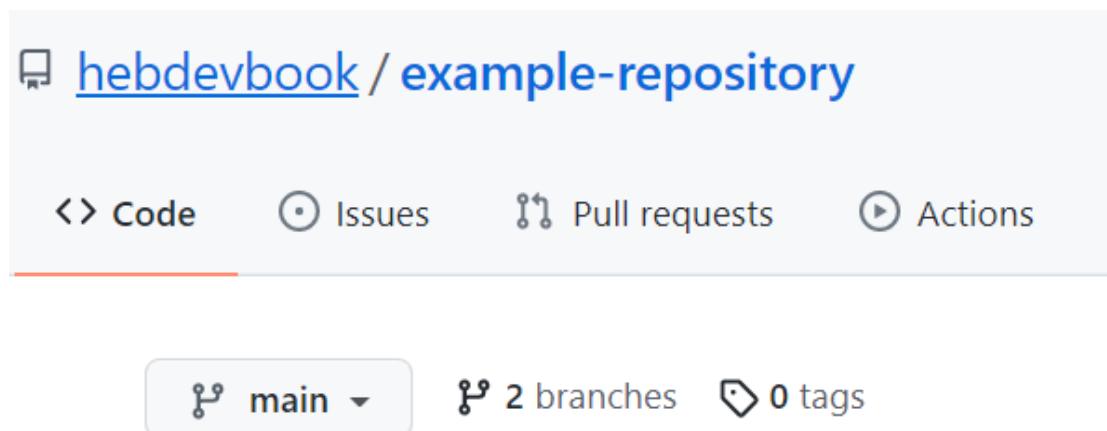
הפקודה הזו חשובה מאוד, כיון שהיא גם יוצרת את הבראנץ' ברייפוזיטורי המרוחק, ואנו חייבים להקליד אותה עם שם הבראנץ', לאחרת נקבל שגיאה.

אחרי שייצרנו את הבראנץ', אם רוצים להוסיף קוד נוסף, אין שום בעיה לעשות כן. פשוט כתבים את השינויים, עושים להם קומיט, ואז שוב דוחפים לרייפוזיטורי המרוחק עם `push`. אבל הפעם, כיון שהבראנץ' קיים שם, אפשר להסתפק רק ב-`push origin`

```
C:\local\example-repository>git push origin
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 516 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:hebdevbook/example-repository.git
  c3e73fd..66b02a6  main -> main
```

אם נעשו שינויים בבראנס', שאנו רוצים לדחוף אליו, הדחיפה תיכשל ונקבל הודעת שגיאה. הפתרון הוא לבצע סyncרון, קודם כל `git pull origin`, לפטור את הקונפליקטים, אם ישם, ואז `push`.

אם אחרי יצירת הבראנס', ניכנס אל הריפוזיטורי שלנו בגייטהאב, נראה שאכן נוצר לנו בראנס'. איך? לוחצים על `branches`:



אפשר לבדוק את הקוד באופן גרפי, לראות את הקומיטים וההיסטוריה שלו ולהשווות את השינויים בין לבין בראנס'ים אחרים, כמו `main`.

## פול ריקווסט: מיזוג בראנס' בrifozitro מרחוק

אם רוצים למזג שני בראנס'ים, יש שתי דרכים לעשות זאת. הראשונה והפחות מומלצת היא פשוט לבצע מרג' בין שני הבראנס'ים באופן מקומי ואז לדחוף את הבראנס' שמייגנו אליו. כך, למשל, אם יש לנו שני בראנס'ים, האחד `main` והשני `new-feature`, נגיע ל-

היום בריפוזיטורי המקומי שלו, נודא שהוא מסונכרן עם `git pull origin` ואז נבצע `pull` לבראנץ' המרוחק שאנו רוצים למזג:

```
git pull origin new-feature
```

אם יש קונפליקטים, פותרים אותם. אבל אם אין, מה שיש לנו כרגע זה בראנץ' בשם `main` שהוא ממזג, אבל נמצא על הריפוזיטורי המקומי שלו. כדי למזג אותו נדחף את `the-main` לריפוזיטורי המרוחק בגיთהאב עם:

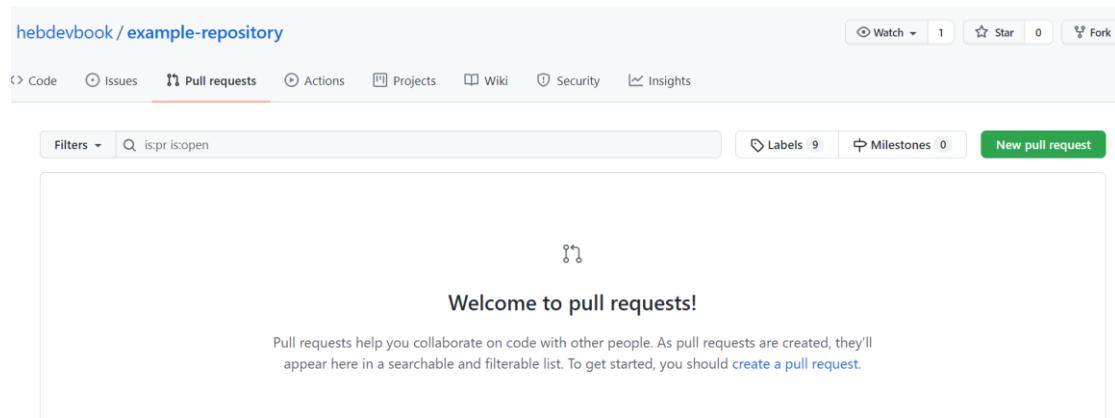
```
git push origin main
```

למה הטכניקה זו פחותה מומלצת? אמנם עם שני בראנץ'ים פרטיים שלנו זה בהחלט אפשרי וגם מקובל, אבל דחיפה לבראנץ' הראשי – בין שמדובר בחום וביין שבבראנץ' של הגרסה הבאה, למשל – פחותה מומלצת ומקובלת כיון שאז התיעוד שלנו על המיזוג לוקה בחרס. השיטה המקובלת למיזוג שני ענפים היא באמצעות פול ריקטוסט (`pull request`). לא מדובר בפקודה של גיט, אלא בפייצר של גיטהאב. כאשר יוצרים בראנץ' בריפוזיטורי המקומי שיוצא `main` ועובדים עליו, ואז מסיימים ודוחפים אותו עם `git push` לריפוזיטורי המרוחק בגיთהאב, גיטהאב מציעה מיד, במשך הזמן, לבצע פול ריקטוסט. הפול ריקטוסט הוא בעצם בקשה למיזוג שבה רואים את השינויים בין המצב הקודם למצב הנוכחי ואפשר לנכון בקשה מסודרת עם הסבר על מהות שינוי הקוד.

מודל הפול ריקטוסט חשוב מאוד בכל מה הקשור לפיתוח תוכנה מודרני, כי בשלב זהה מתכונת אחר יכול לבחון את השינויים ולאשר אותם ורק לאחר השינוי לבצע מריג' שנעשה באופן אוטומטי דרך המשך הגרפי של גיטהאב. הפול ריקטוסטים נכנסים לארכיוון ואפשר לבחון אותם שם או ליזור `revert`, פול ריקטוסט שמחזיר את המצב המקורי וקיים ואפשר לבחון אותו שם או ליזור `revert`.

אפשרו אם אנו יוצרים פול ריקטוסט גם בריפוזיטורי שرك אנחנו מפתחים עליהם, זה עדין מקרים רציניות גם מסדר את הקוד בצורה טובה יותר. מדובר בדרך מקובלת לעבוד על פרויקטי קוד פתוח, מכיוון שכך אפשר "להציג" שינויי קוד לפרויקטים שאנו נלאם הבעלים שלהם.

از איך מבצעים פול ריקטוסט? פשוט מאד – נכנסים למשך הגרפי של גיטהאב, בוחרים את הלשונית `Pull Requests` ולוחצים על `New Pull Request`.



ונגיעה לממשק שמאפשר לנו לבחור מה אנו רוצים למזוג ולאן. נבחר שברצוננו לבצע מרג' אל `main` באופן זהה:

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

base: `main` compare: `some-feature-branch` Able to merge. These branches can be automatically merged.

Choose a head ref

Find a branch

Branches Tags

`main` default `some-feature-branch`

Commits on Aug 28, 2021

`add eslint`

Showing 2 changed files with 862 additions and 0 deletions.

סימן החץ עוזר לנו להבין מה הכניס בצד השמאלי (את הבראנס' שמנכנים אליו) ומה בימני (את הבראנס' עם השינויים). אחרי הבחירה אפשר לראות את ההבדלים ומהו הקוד שמשתנה. נלחץ על אישור הפעולה ונגיע למסך שבו נוכל לכתוב פרטים על הפלול ריקווט:

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

The screenshot shows the GitHub interface for creating a pull request. At the top, it says "base: main" and "compare: some-feature-branch". A green checkmark indicates "Able to merge". The main area is titled "My pull request" and contains a text input field with the placeholder "This is a new code". Below the input field is a note: "Attach files by dragging & dropping, selecting or pasting them." At the bottom right is a green "Create pull request" button.

ⓘ Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

חשוב לזכור כי קיימת אפשרות לאפשר וטייאור מפורט ומסודר. במקרים מסוימים יש טמפלטים שבהם חיברים נכתבו מהו השינוי בדרך מסוימת ולצרף צילומי מסך והסבירים. אבל לא צריך להיבהל - אפשר לבצע ערכינה לפול ריקווסט אחרי השינוי. אחרי שיצרנו את הפול ריקווסט, אפשר לראות אותו בראשית הפול ריקווסט, תחת לשונית ה-`Pull Requests` בדף הראשי של גיטהב.

The screenshot shows a GitHub repository page for 'hebdevbook/example-repository'. The 'Pull requests' tab is selected, showing 1 open pull request. The pull request is titled 'My pull request' and was opened 21 seconds ago by 'barzik'. A 'ProTip!' section at the bottom right suggests using the 'is:pr is:open' filter to find open pull requests.

hebdevbook / example-repository

< Code Issues Pull requests 1 Actions Project

Lab

Now, GitHub will help p

Filters is:pr is:open

1 Open ✓ 0 Closed

My pull request  
#1 opened 21 seconds ago by barzik

### 💡 ProTip!

יוצר הפול ריקווסט ובעל הריפוזיטורי יכולם לעורק את השינויים, וכל אדם בעולם, אם הריפוזיטורי פומבי, יוכל להגיב בשאלות שונות. מי שיש לו הרשות יכול לבצע מרג', אם אין קונפליקטים, או לבטל את הפול ריקווסט לחלוטין (כਮובן עם נימוק).

The screenshot shows a GitHub pull request interface. At the top, it says "My pull request #1" and "barzik wants to merge 1 commit into `main` from `some-feature-branch`". The commit message is "This is a new code". Below the commit is a note "add eslint" and the ID "be01938". To the right, there's a sidebar with various status indicators: "Still in progress", "Assigned to", "No one", "Label", "None", "Project", "None", "Miles", "No more", and "Links" with a note "Success these". The main area shows a green box with a checkmark and the text "Continuous integration has not been set up. GitHub Actions and several other apps can be used to automatically catch bugs and enforce style." Below that is another green box with a checkmark and the text "This branch has no conflicts with the base branch. Merging can be performed automatically." At the bottom of the main area is a button "Merge pull request" and a note "You can also open this in GitHub Desktop or view command line instructions." Below the main area is a toolbar with "Write" and "Preview" buttons, and a text input field "Leave a comment".

המיזוג, כבירותת מיוחד, יכנס את כל הקומיטים של הבראנץ' אל הבראנץ' החדש.

בחברות הייטק מחייבים בדיקה של מתכנת אחר כדי לראות שהפול ריקווסט בסדר. לבדיקה הזו קוראים (code review), והוא הליך מקובל מאוד גם בצוותי פיתוח. בדרך כלל המתכנת השני מעיר ומאייר על הקוד, יש שאלות וশינויים לצריך לעשות – צריך לזכור שכל שינוי בבראנץ' ישנה את הפול ריקווסט באופן אוטומטי וויסוף לו את הקוד שרצינו. תהליך הפול ריקווסט, אם הוא מבוצע נכון, עשוי להועיל לארגון בקוד אינטואיטיבי יותר ועל פי הסטנדרטים שהארגון מצפה שיישמו על ידי המתכנתים.

The screenshot shows a GitHub pull request review interface. At the top right, there are statistics: +862 -0 and a green progress bar. Below that, it says "0 / 2 files viewed" and has a "Review changes" button.

The main area is titled "Finish your review". It has a toolbar with "Write" (selected), "Preview", and various rich text editing icons (H, B, I, etc.).

In the text input field, the user has typed: "What about package.json? You have to add it."

Below the text input, there is a placeholder: "Attach files by dragging & dropping, selecting or pasting them." with a "More" button.

At the bottom, there are three radio button options:

- Comment: Submit general feedback without explicit approval.
- Approve: Submit feedback and approve merging these changes.
- Request changes: Submit feedback that must be addressed before merging.

A "Submit review" button is located at the bottom left of the review area.

אחרי הביקורת, שהוא שלב שכמובן לא קיים בrifozitouri שرك אנחנו כותבים לו, אפשר לבצע מרג', והקוד בבראנץ' מתמזג לתוכן main. אפשר למחוק את הבראנץ' המרוחק והמקומי. בחלק מהמקרים מקובל לעשות זאת, אך כמו בכל דבר, מדובר בבחירה שלכם. הפול ריקווסט שמוגן מתועד כפול ריקווסט סגור. אפשר לבדוק אותו, את הבקשה ואת תהליך הבדיקה וכן לבצע revert – ככלומר להחזיר את השינויים לאחר ולבטל אותם.

זו הסיבה שבגללה גיטהאב וגייט פופולריות מאד – היכולת לבצע ניהול קוד בצורה טيبة כל כך, גם באופן מקומי וגם עם המיצירות לניהול הריפוזיטורי של גיטהאב.

## Git flow

כדי לסכם את הכל, נעבור שוב על דרך העבודה, שנקראת גם **Git flow**.

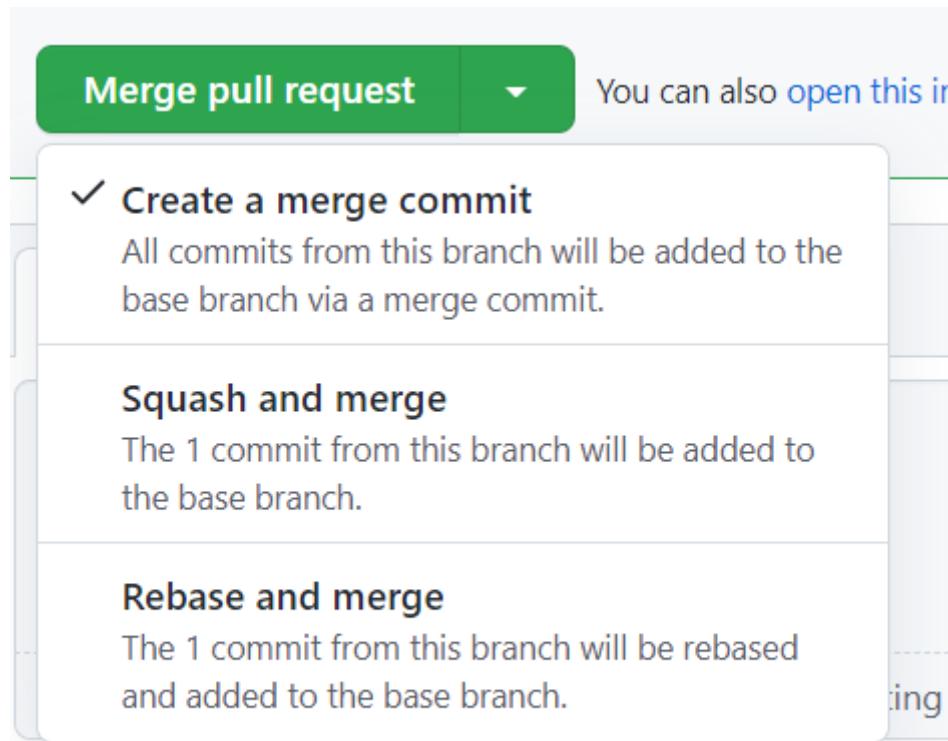
1. יצרת ריפוזיטורי מרוחק בגיטהאב.
2. יצרת עותק מקומי שלו עם `git clone`.
3. יצרת בראנץ' מה-`main` שבו נכניס את השינויים.
4. הכנסת השינויים המקומיות.
5. יצרת קומיט לבראנץ' המקומי.
6. הכנסת שינויים נוספים אם צריך.
7. יצרת קומיטים נוספים אם צריך.
8. דחיפת הקוד עם `git push` ויצירת בראנץ' ברייפוזיטורי המרוחק.
9. יצרת פול ריקווסט עם הסבר על השינויים.
10. אם יש קונפליקט עם ה-`main`, צריך לעשות `git pull origin main` כדי להתקדם מול `main` ולפתח את הקונפליקטים כרגע. אז `git push` עם `git push` על `main` המרוחק.
11. בשלב זהה עדין אפשר לעשות קומיטים בבראנץ' המקומי ולדוחף עם `git push` לבראנץ' המרוחק.

12. אם הפל ריקווסט מאושר (או אם היחידים שתרומותם קוד לריפוזיטורי),  
אפשר לבצע מרג' לפל ריקווסט.

כל הפעולות האלו נראות עכשו מרכיבות יחסית, בודאי אם אתם עדין לא שולטים  
במנחים. אבל כל מתכנת שהוא, בכל חברה מקומית או בינלאומית, מכיר אותו על בורין  
יחד עם הפקודות.

## דרכי המרג'

כאשר מבצעים מרג', אפשר לבחור את אופן הביצוע באמצעות לחיצה על החץ הקטן ליד כפתור ה-`merge`:



האפשרות הראשונה היא לבצע אותו כרגע, והקומייטים נכנסים כסדרם ללוג, יחד עם הקומייט של מרג'. אפשר גם לבצע `rebase`, כפי שLEARNT בפרקם הקודמים, אבל אפשר גם לבצע `Squash and merge`. הפעולה זו בעצם "דוחשת" את כל הקומייטים לקומייט אחד. כל ההודעות והקומייטים שהיו בבראנס' שאנו מmag'ים אליו בעצם נעלמים.

למה זה נעשה? כיוון שבשיטת הפיתוח הנהוגה בחלוקת מהמקומות מסתכלים יותר על הפול ריקווסט ולא על הקומייטים. ה-`revert` שעושים? נעשה לפול ריקווסט. הבדיקה אם משחו השتبש? גם היא נעשית לפול ריקווסט. הקומייטים הם עניין פרטי של המפתח.

ברירת המחדל היא מרג' רגיל, אך אפשר לשנות אותה בהגדרות של הריפוזיטורי בגיןהאָב.

## git tag

תיק גיט (git tag) וריליסים (git release) הם חלקים חשובים בעבודה בגיט, אותו git flow שהוזכר קודם לכן. עד כה למדנו שגיט שומרת מידע על קבצים ומטא-מידע על הקומיטים. תיק מאפשר פשטוט "לסמן" נקודות מסוימות בזמן, ובעצם ליצור גרסאות עם שם. בדיק כמו קומיט, רק בלי קשר לקבצים ועם שם שאנו נזכיר בוחריהם.

יצירת תיק נעשית באמצעות הפקודה:

```
git tag -a "TAGNAME"
```

שם התקג יכול להיות כל דבר, בודך כלל האות זה או זו נוספת לא חובה.

אפשר להוסיף הערה כלשהי לתיק באמצעות -m:

```
git tag -a "v1.0.0" -m"Optional message"
```

אך זו לא חובה.

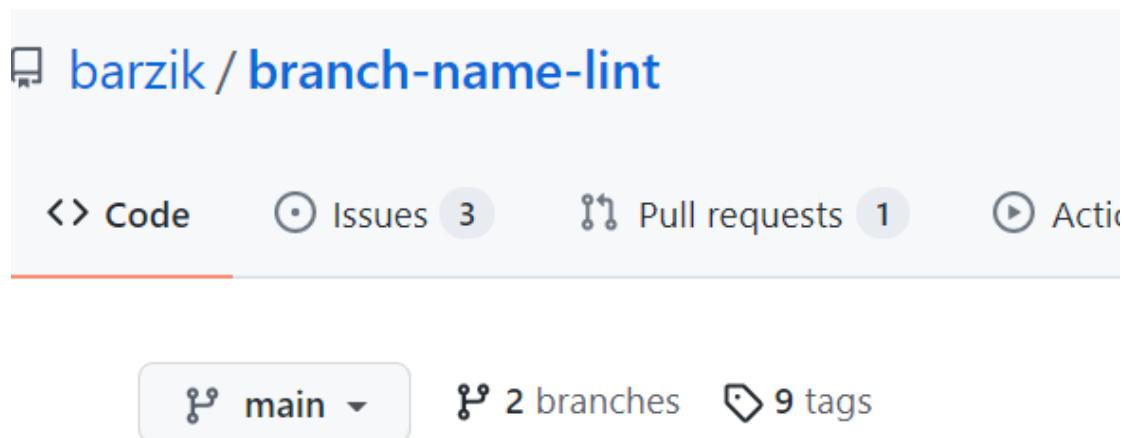
אפשר לצפות בתוצאות באמצעות הפקודה -l .git tag -l

```
C:\local\repository>git tag -a "v1.0.0" -m"Optional message"  
C:\local\repository>git tag -l  
v1.0.0
```

בדיק כמו קומיטים, התקגים נשמרים מקומית על המחשב ואפשר לדוחף אותם באמצעות:

```
git push origin -tags
```

אפשר לבדוק את רשימת התקגים שיש בריפוזיטורי המרוחק באמצעות הממשק בגיטהאב:



לחיצה על לשונית **Code** וזו על לשונית **tags** תציג את כל רשימת התגים שנוצרו.

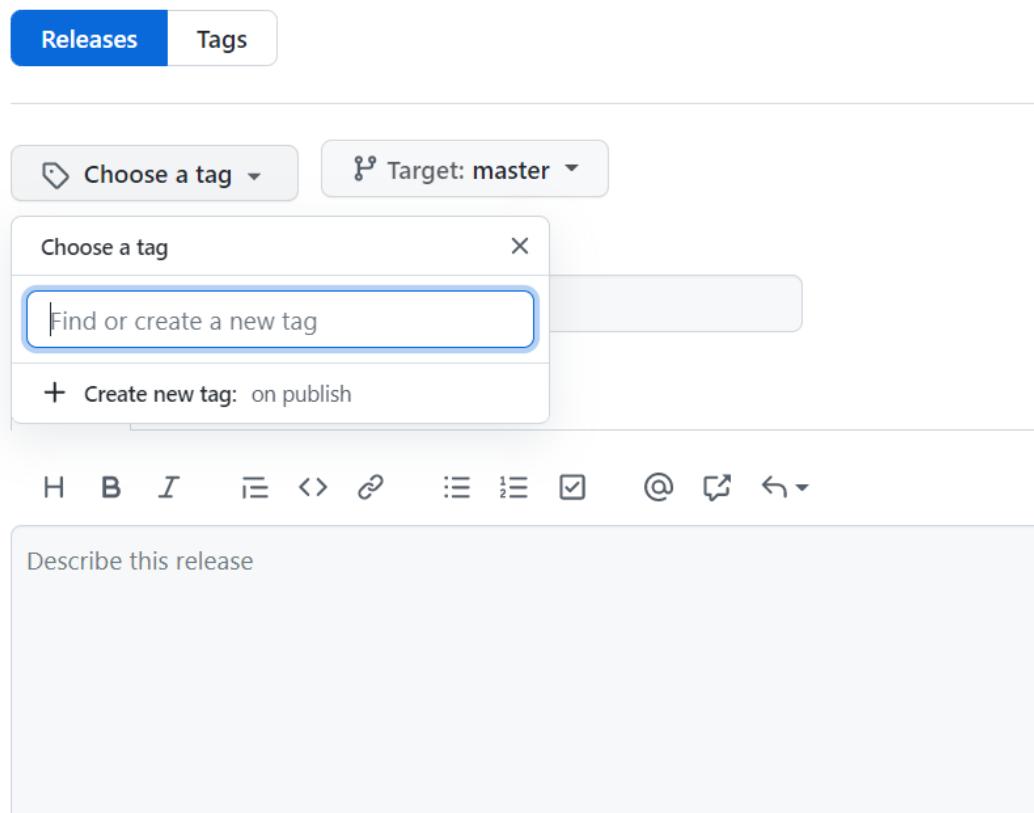
## Release

מדובר בתוכנה של גיטהאב המאפשרת לנו לסמן תגיות ספציפיות וליצור מהן "גרסאות" רשםיות. זה עוזר בסימון הקוד וכן בתהליכי Continuous Deployment שונים (תהליכי בדיקת קוד שנלמד עליהם מעט בהמשך). על מנת ליצור **Release** ניכנס לריפורזיטורי שלנו בגיטהאב ונחפש את [Create a new release](#).

## Releases

No releases published  
[Create a new release](#)

נלחץ עליו ונגיע למסך המאפשר לנו ליצור Tag וממנו ליצור **Release**, נוסף למידע ולהערות הקשורות ל-**Release**. מקובל מאד להוסיף כמה שיותר מידע בנוגע לגרסת החדשה, מה השתנה בה ופתרונות יישנים וחדים שיש בה.



גם-tag ו-gemRelease הם דרכיהם לסדר את הקוד ולהודיע פומבית על כך שיש גרסה חדשה שאפשר להוריד. פעמים רבות, יצרת הגרסה החדשה היא טריגר לפעולות אוטומטיות שלוקחות את הקוד הנוכחי ברגע היוצרות הגרסה ושולחות אותו ליעדים שונים. כך, למשל, במקרים של wkhtml (שעליהם למדנו גם בספרים "למוד Node.js בעברית" ו"למוד ריאקט בעברית"), יציאת Release חדש בגיטהאב גוררת יצרת גרסה חדשה גם ב-wkhtml. חברות הייטק רק המפתח הבכיר, או גורם בכיר יותר בארגון כגון מנהל הפיתוח אחראי לייצרת Release, וההחלטה בוגרתו לייצרטו חשובה, כיוון שיש לה משמעות מבחינת הלכוות והמשתמשים. גם בפרויקט קוד פתוח לוקח זמן לבצע Release.

## קובץ README ו-.md

קובץ README הוא חלק ממשמעותי בכל ריפוזיטורי ובמיוחד בריפוזיטורי של קוד פתוח. בקובץ README יהיו הנחיות להתקנה, הערות, הסברים וכל מידע שיכל לעזור למפתחים אחרים בעת וgam בעtid. גיטהאב עבדת עם קובצי README בפורמט Markdown, או בקיצור md. מדובר בפורמט טקסטואלי, קלומר אפשר לפתח אותו עם

כל עורך טקסט שהוא, מ-`notepad` ועד `VSCode`. אפשר לכתוב בתוכו בטקסט רגיל, אבל אפשר גם להוסיף סימנים המסיעים לעיצוב. למשל:

# כוורת ראשית

## כוורת משנית

#### כוורת משנית קטנה

טקסט עם \*הדגשה\* כלשהו

אפשר גם לעשות [ קישור](<http://some-site.com>)

או לכתוב 'code' `console.log('some code');` והוא יופיע קוד.

אפשר להוסיף לא מעט סימנים שיוצרים עיצוב. היתרון שלהם הוא שבעורק טקסט לא רואים את העיצוב, אבל גיטהאב (וגם מתחנותיה) מציגות את `md` עם העיצוב. כך למשל יראה קובץ `md` כאשר ניכנס אליו מגיטהאב:

## branch-name-lint



Validating and linting the git branch name. Create a config file or use the default configuration file. Use it in husky config file to make sure that your branch will not be rejected by some pesky Jenkins branch name conventions. You may use it as part of a CI process or just as an handy `npx` command.

### Install

```
$ npm install branch-name-lint
```

### CLI usage

```
$ npx branch-name-lint
```

הגרסה הטקסטואלית של תיראה כז:

```

1 # branch-name-lint ![Build Status](https://github.com/barzik/branch-name-
lint/workflows/Branch%20Lint%20Name%20CI/badge.svg) !![Known Vulnerabilities]
  (https://snyk.io/test/github/barzik/branch-name-lint/badge.svg)](https://snyk.io/test/github/barzik/branch-
name-lint) !![npm](https://img.shields.io/npm/dt/branch-name-lint)
2
3 Validating and linting the git branch name. Create a config file or use the default configuration file. Use it
in husky config file to make sure that your branch will not be rejected by some pesky Jenkins branch name
conventions. You may use it as part of a CI process or just as an handy `npx` command.
4
5 ## Install
6
7 ```
8 $ npm install branch-name-lint
9 ```
10
11 ## CLI usage
12
13 ```


```

גיטהאב מציגה אוטומטית כל קובץ `readme.md` שיש בתיקייה הראשית של הפרויקט כאשר נכנסים לדף הראשי של הריביזיטורי, ומכאן חשוב מאוד ליצור קובץ זה שמתאר את הפרויקט שלנו, בעיקר אם אנחנו בונים ספריית קוד פתוח, אך לא רק.

## תרומה לקוד פתוח

קוד פתוח הוא שם כללי לגישה שלמה להתייחסות לקוד ולמוציאר תוכנה. בעבר, חברות היוተה מפתחת מוצר תוכנה ומספקת אותו ללקוחות – בחינם או בתשלום. הלוקחות קיבלו את מוצר התוכנה זו As, כלומר בדיק כפי שהוא, בלי יכולת לשנות תוכנת וורד של מיקרוסופט היא דוגמה מעולה למוצר שהוא בקוד סגור. אנו מקבלים אותו כמשתמשים זהה. אם אנחנו רוצים לשנות תוכנה מסוימת בוורד, אי-אפשר. אין דרך לעשות זאת.

במוצר קוד פתוח המצב שונה. תנועת הקוד הפתוח, שהחלה כבר בשנות ה-70, דוגלת בשקיופות ובASP.NET קוד המקור. מה זאת אומרת? שאם וורד היה בקוד פתוח, היינו מקבלים את קוד המקור שלו, הקוד שאם מקמלים אותו (ממירים אותו לשפת תוכנה) התוכנה נוצרת, והיינו יכולים לשנות אותו לרצוננו. למשל, להוסף דרך לשמר את הקובץ כפורמט `md`, שעליו למדנו קודם. אם היינו רוצים, היינו יכולים להציג את שינויי הקוד האלה למיקרוסופט, שהיו מחייבים אם לאמץ אותם לטובת שאר המשתמשים.

חלק גדול מהקוד בעולם היום הוא פתוח. ריאקט כתובה בקוד פתוח, אקספרס של Node.js כתובה בקוד פתוח. רוב המודולים והתוכנות שנלמדו בספר זה (גמ Visual Studio Code וגיט עצמה) הם בקוד פתוח. זאת אומרת שכל מתכנת בעולם יכול לבחון את הקוד ולהציג תיקונים או תכונות חדשות, בין שהתוכנה בתשלום ובין שהיא בחינם. חלק גדול מהקוד הפתוח מנהל באתר גיטהאב.

תרומה לקוד פתוח היא חשובה מאוד לכל עולם הפיתוח, ולא מעט מתכנתים עובדים בהתקנות בפרויקטים רבים של קוד פתוח. תרומה לקוד פתוח עשויה לתרום הרבה לקורות החיים של מתכנתים – ועל כך נרחיב בהמשך הפרק.

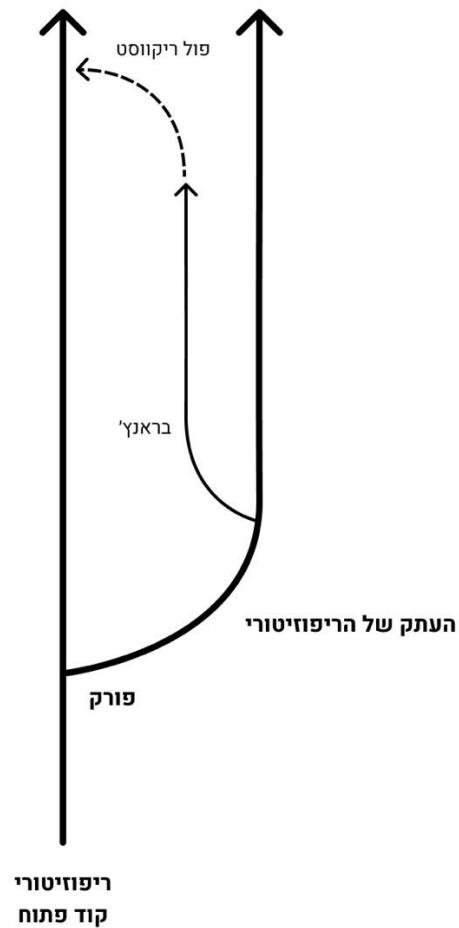
כאשר אנו ניגשים לתרום קוד פתוח, אנו עובדים עם גיט ויכולים בקלות לבצע `git clone` ולאחר מכן מוכמית את הפרויקט שלנו. אבל אם ניצור בראנץ', נכתב בו קוד ואז נדחף אותו עם `push`, לא נצליח. מדוע? כי אנו לא הבעלים של הריפוזיטורי המרוחק ורק בעלי זכות כתיבה לריפוזיטורי המרוחק יכולים ליצור בראנצ'ים. אחרת, זה היה פתח לביעות. דמיינו שאתם מנהלים ריפוזיטורי של קוד פתוח ופתאום מופתעים לגלוות בראנצ'ים חדשים שלא מיini אנשים יצרו, שחלקם עלולים להיות עם קוד זמני, וכך להיות שמשהו יוריד אותם (חשבו למשל על מישוה שפותח בראנץ' בשם `next-version` שמכיל קוד זמני). זו הסיבה שرك בעלי הריפוזיטורי יכולים ליצור בראנץ'.

از איך עובדים בכלל זאת? אם רוצים לתרום קוד לריפוזיטורי – אין אפשרות ליצור בראנץ' ולהציג תרומה קוד?

## פורק (Fork)

הפתרון הוא פורק (fork). מדובר במקרה שבו מבצעים העתקה של הריפוזיטורי המרוחק ממשתמש אחד אל המשמש שלנו, וכך יש לנו שני ריפוזיטורי מרוחקים שאנחנו עובדים מולם: האחד שלנו, שבו אנו יכולים ליצור בראנצ'ים ולדוחף אליהם קוד, והשני של המאגר המרכזי של הקוד הפתוח, שמןנו אפשר רק לקרוא. יוצרים בראנץ' על הריפוזיטורי שלנו ושולחים פול ריקווסט אל הריפוזיטורי המרוחק. עכשו יש על הריפוזיטורי המרוחק פול ריקווסט שלנו, שמנהל הריפוזיטורי המרוחק יכול לבחון ולאשר, אף על פי שלא יצרנו בראנץ' בrifozitro.

בתרשים זהה אפשר לראות המ/change של התהליין:

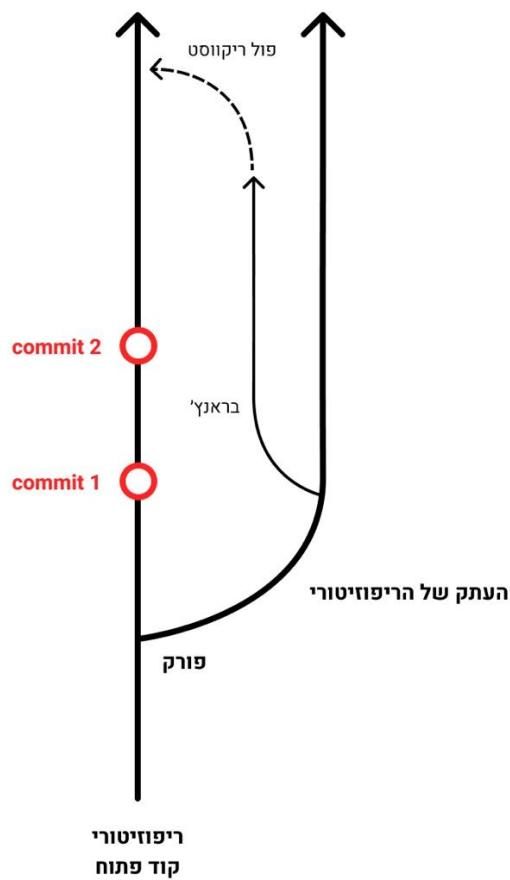


יש לנו ריפוזיטורי מרוחק שאנו רוצים לתרום לו. עושים לו פורק. הפורק הוא שלנו וחתת השם שלנו. למשל, אם עשיתי פורק ל-`facebook/react`, הריפוזיטורי שלי ייקרא `barzik/react`. הוא יהיה עותק מושלם של `facebook/react`. כל הקוד, ההיסטוריה הקומיטים והמידע יעברו אליו, ותהיה לי יכולת לעשות בו מה שאני רוצה. למשל, אני אוכל ליצור לו עותק מקומי עם `clone`, ליצור לו בראנץ', לעשות לו קומיטים. ברגע שהבראנץ' המקומי שלי יהיה מוכן, אני אדחוף אותו לריפוזיטורי המרוחק שלי, כרגע. oczywiście יש לי בראנץ' על הריפוזיטורי שלי ואני מבצע פול ריקוуст, אבל לא לריפוזיטורי שלי, אלא לריפוזיטורי המקורי. הבעלים שלו יוכל לקבל את הפול ריקוуст, לדחות אותו, להגיב לו או לאשר אותו.

פורק הוא בעצם הפעולה הראשונית שאנו עושים לריפוזיטורי כשאנו רוצים לתרום לו. הוא העתקה המושלמת שלו אלינו, לחשבון הגיטהאב הפרטי שלנו.

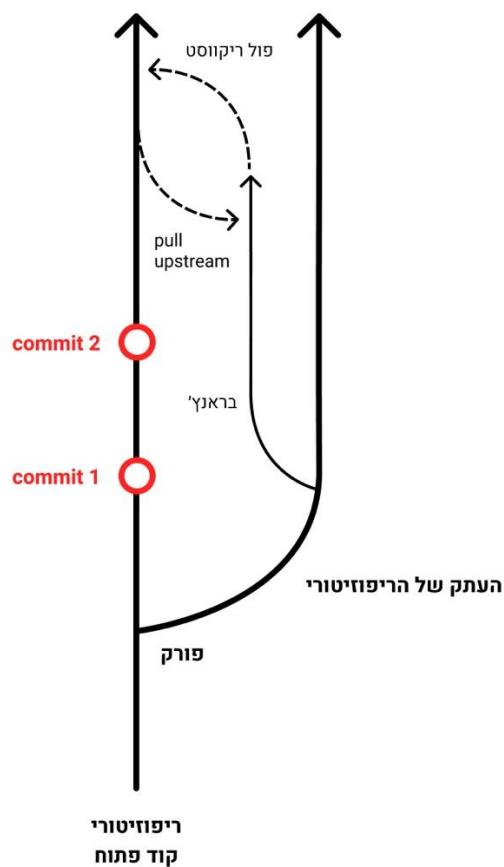
### עדכן הריפוזיטורי

הבעיה היא שהריפוזיטורי המקורי, המרוחק, ממשיך להתעדכן כל הזמן. ייתכן שהבעליים המקוריים הוציאו בראנץ' משלו ומייצג אותו, ייתכן他们会 פול ריקווסט של אחרים שתרמו. הינה המחשבה:

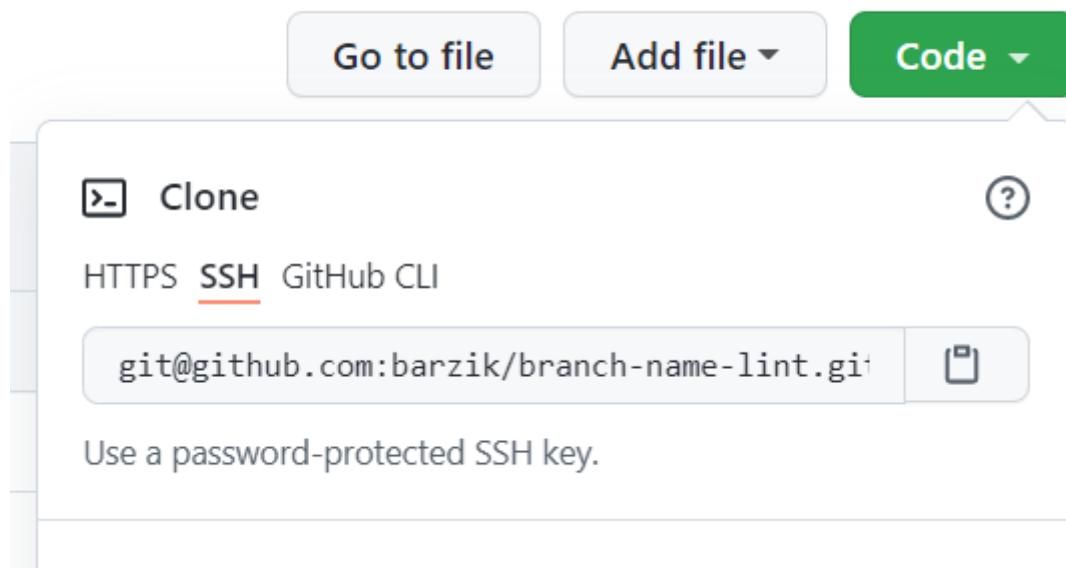


איך אנו מעדכנים את הריפוזיטורי שלנו בקומיטים שנוספו לריפוזיטורי שעשינו לנו פורק? אם היה מדובר ברייפוזיטורי שלנו, היינו יכולים לעשות `git pull origin git` כדי לעדכן את הריפוזיטורי המקומי משינויים שנעשו ברייפוזיטורי המרוחק, אבלפה הריפוזיטורי המרוחק הוא בשליטתנו ושלנו. אנחנו בעצם צריכים לעדכן ריפוזיטורי מרוחק (שלנו) בשינויים שנעשו ברייפוזיטורי מרוחק אחר (זה שמןנו עשינו פורק).

איך אנחנו מתעדכנים מריפוזיטורי מרוחק שהוא לא שלנו? בפרקים הקודמים הסבירתי ש-origin הוא הכינוי של השרת המרוחק. למדנו לעשות push/pull מהריפוזיטורי שלנו שיושב בשרת המרוחק בגייטהאב, שהכינוי שלו הוא `origin`, אבל אפשר להגדיר שרת נוספת עם שם אחר ואז למשוך ממנה את העדכונים. נוהג להגדיר את השרת הנוסף, שמננו רק מושכים (אי-אפשר לדחוף אליו, כי אין לנו זכויות אליו), כ-`upstream`:



אחריו SMB צעדים פורק ויוצרים clone למחשב המקומי שלנו, אפשר לראות עם git remote ה-`origin` ששרת ה-`origin` שלנו הוא הריפוזיטורי שלנו. ניתן גם לחבר לריפוזיטורי המרוחק עם git remote – צריך לבחור שם ( כאמור, מקובל לבחור בשם `upstream`) ואת כתובות הריפוזיטורי המרוחק. את הכתובת לוקחים מהריפוזיטורי המרוחק בגייטהאב, מהיכן שעושים clone. למשל:



ואז מקלידים את הפקודה `git remote` `git` כדי לבצע את ההגדה:

```
git remote set-url upstream
git@github.com:USERNAME/REPOSITORY.git
```

אם נקליד `-v` `git remote`, נראה שנוסף על ה-`origin` יש לנו `upstream`. במקרה,>Katz לפני שאנחנו עושים `push` לבראנס' שלנו ותיכוננים לפול ריקווט, כדאי לנו לעשות:

```
git pull upstream main
```

כדי לוודא שאנחנו מעודכנים.

## בקוד פתוח `git flow`

ה-`flow` `git` במקורה של עבודה בקוד קוד פתוח יהיה כך:

1. יוצרת פורק תחת השם שלנו מהריפוזיטורי שאנחנו רוצים לתרום לו קוד.
2. `git clone` לריפוזיטורי המרוחק שלנו אל הריפוזיטורי המקומי.
3. עבודה על הריפוזיטורי המקומי – פיתוח בראנס', קומיטים.
4. `git push origin MY-BRANCH-NAME`

5. כדי להתעדכן בשינויים של הריפוזיטורי המרוחק מגדרים upstream ועושים git pull upstream main (או כל שם אחר של בראנץ' שיצאנו ממנו).
6. בסיום הפעולה – git push origin main ויצירת פול ריקווסט.

זו שיטת עבודה של קוד פתוח ויש גם חברות מסחריות שמאמצות אותה.

## Github GUI

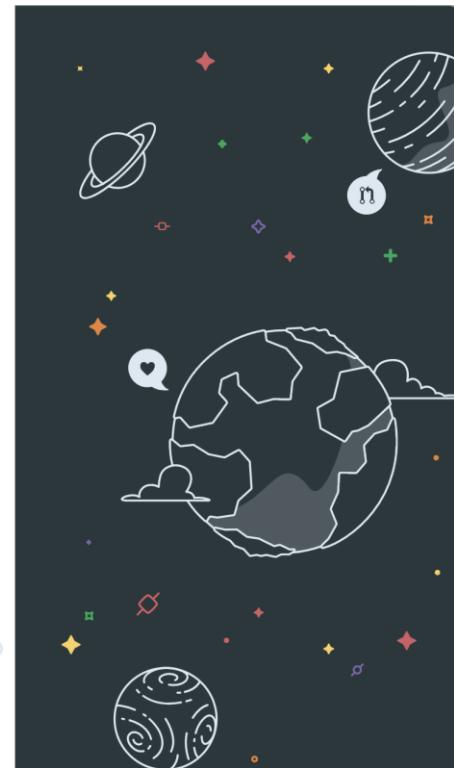
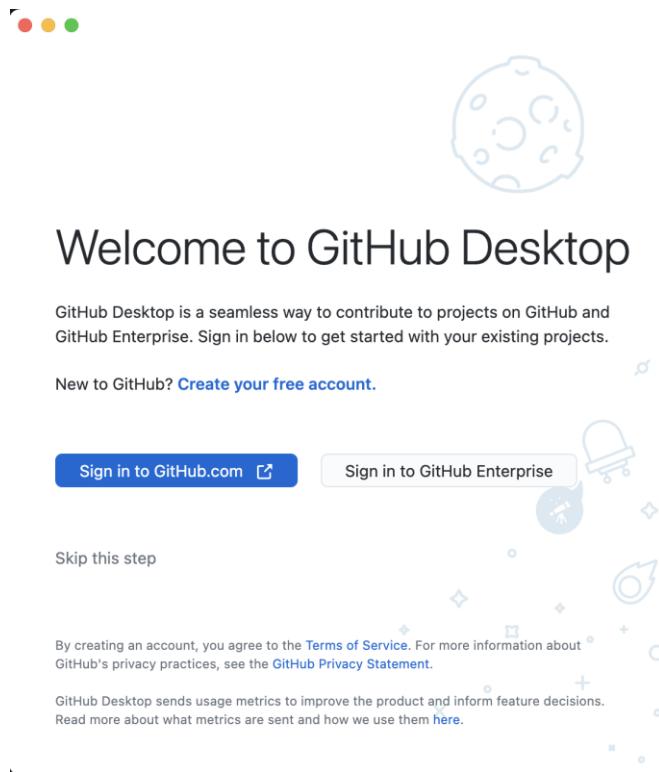
לגייטהאב יש גם כלי גרפי לעובדה עם גיט, שאמור לבוא כתחליףCLI או לפחות להקל חלק מהעבודה. מהניסיון שלי, הוא דזוקא הופך את כל ההנהלות מול גיט לקשה יותר, כי אפשר להפעיל אותו ללא הבנה ממשית בגיט. لكن כאן אנחנו לומדים על גיט בטרמינל ולא על התוכנה זו.

אבל יש צוותים שכן משתמשים בה, ואם אתם מתקשים באמת עם CLI, אפשר לנסות אותה.

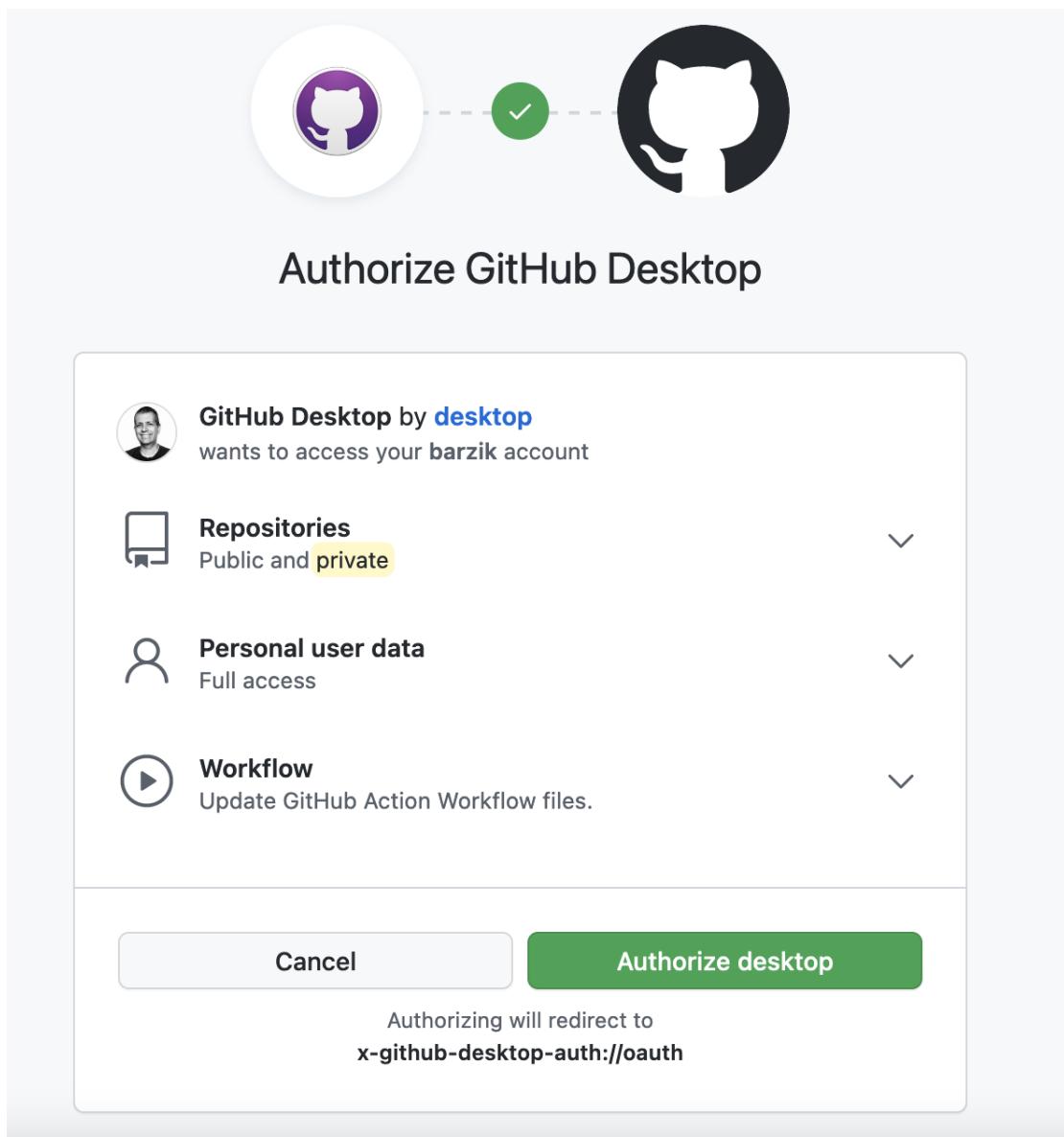
ההתקנה של Github GUI (Graphic User Interface) היא פשוטה בכל מערכת הפעלה. נכנסים לכתובת התוכנה, שהיא תוכנה רשמית של גיטהאב, פה:

<https://desktop.github.com>

מיד לאחר ההתקנה נתקבש להכניס את הפרטים שלנו, כמשתמש ארגוני (Enterprise) או כמשתמש רגיל. אנו נבחר במשתמש רגיל, שיש לו חשבון ב-Github.com.

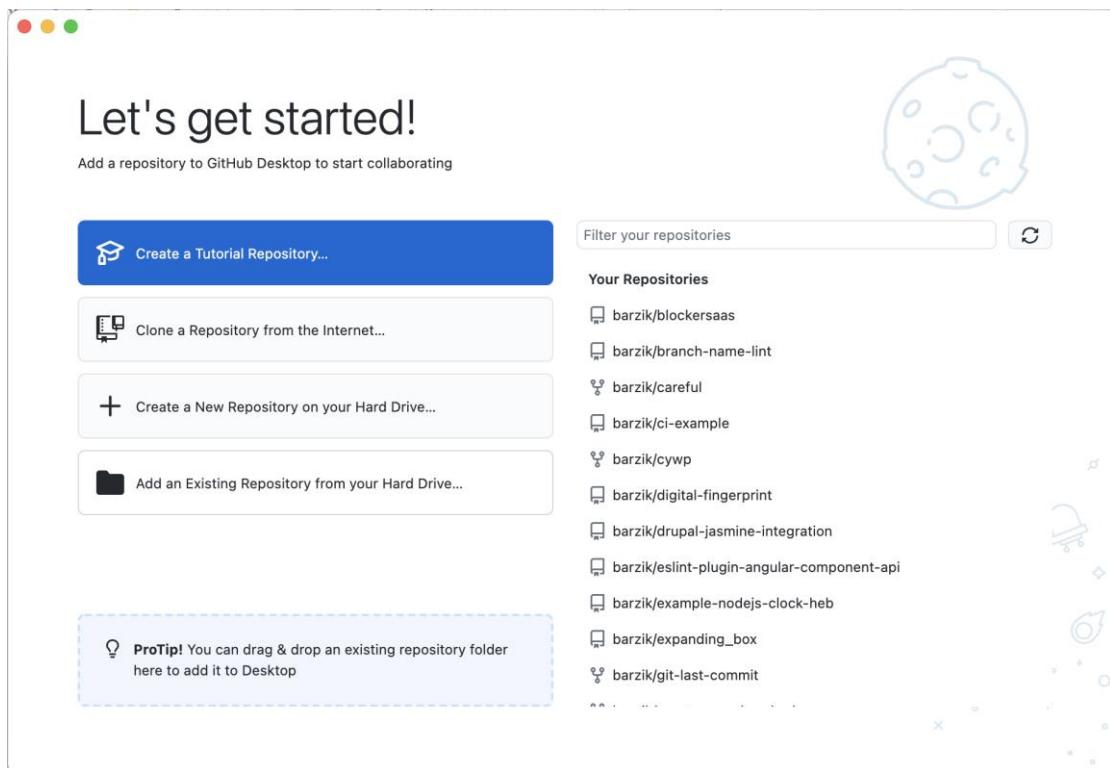


אחרי הלחיצה נעבור למסך ווב, שבו נוכל להכנס את שם המשתמש והסיסמה,  
ונאשר.



השלב הבא הוא לאשר גם את שילוב התוכנה. זה הכל. התוכנה מוכנה לעבודה.

המסך הראשי של התוכנה מציג את כל הריפורזיטורי שלנו ואת אלה שעשינו להם פורק – כל מה שנמצא תחת השם שלנו. למשל, שם המשתמש שלי בגייטהאב הוא barzik, והתוכנה תציג לי את כל הפרויקטים שלי ואת אלה שעשיתי להם פורק:



אני יכול לסמך אחד מהם ולעשות לו clone למחשב המקומי או, אם יש לי ריפורזיטורי מקומי, ליבא אותו באמצעות לחיצה על סימן הפלוס. אני יכול לבצע גם clone לפרויקט אחר.

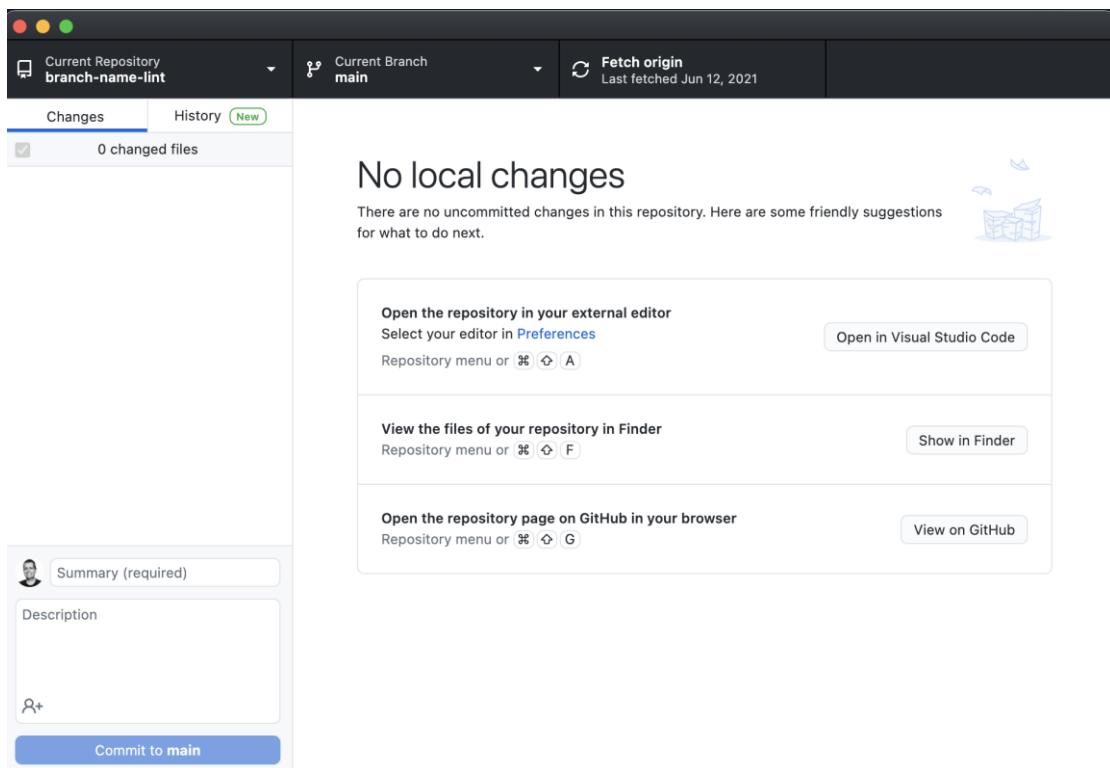
Filter your repositories

Your Repositories

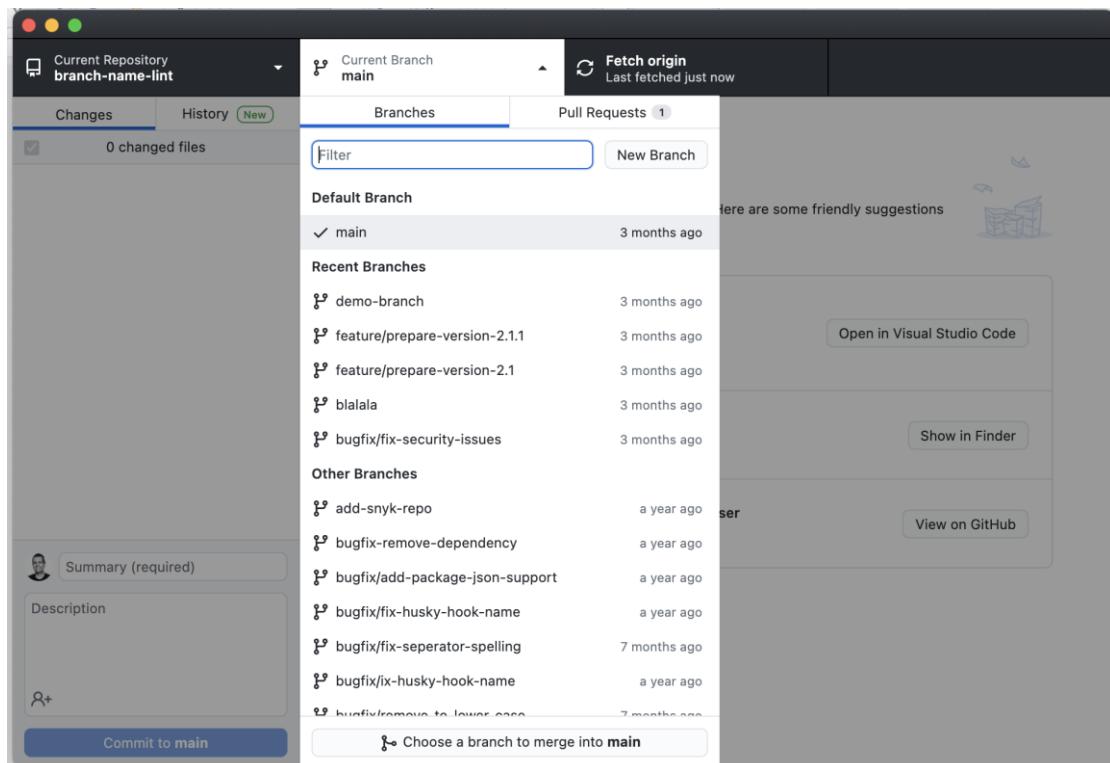
- [barzik/blockerssaas](#)
- [barzik/branch-name-lint](#)
- [barzik/careful](#)
- [barzik/ci-example](#)
- [barzik/cywp](#)
- [barzik/digital-fingerprint](#)
- [barzik/drupal-jasmine-integration](#)
- [barzik/eslint-plugin-angular-component-api](#)
- [barzik/example-nodejs-clock-heb](#)
- [barzik/expanding\\_box](#)

Clone **barzik/blockerssaas**

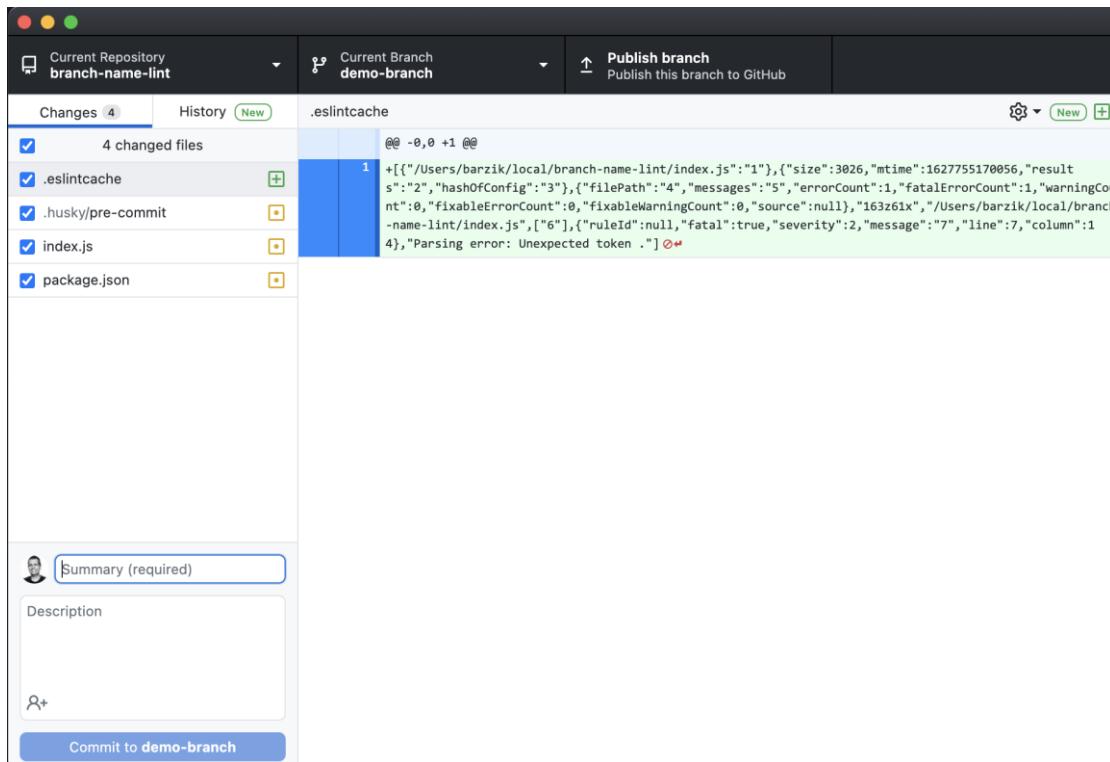
לא משנה אם עשית `clone` לפרויקט שלי או של מישהו אחר, ברגע שאפתח את הריביזיטורי המתאים שנמצא מקומית אליו, אני אראה על איזה בראנץ' אני, אוכל למשוך שינויים מרוחק וכמובן אראה את כל השינויים בין הבראנץ' המקומי לבראנץ' המרוחק.



אני אוכל לעבור לאיזה בראנץ' שארצה או ליצור בראנץ' חדש מקומי:



היתרון הגדול מואוד של GUI הוא שرؤים את השינויים בין הגרסה המקומית לגרסה המרוחקת בצורה נוחה מאוד, למשל git diff נוח מאוד.

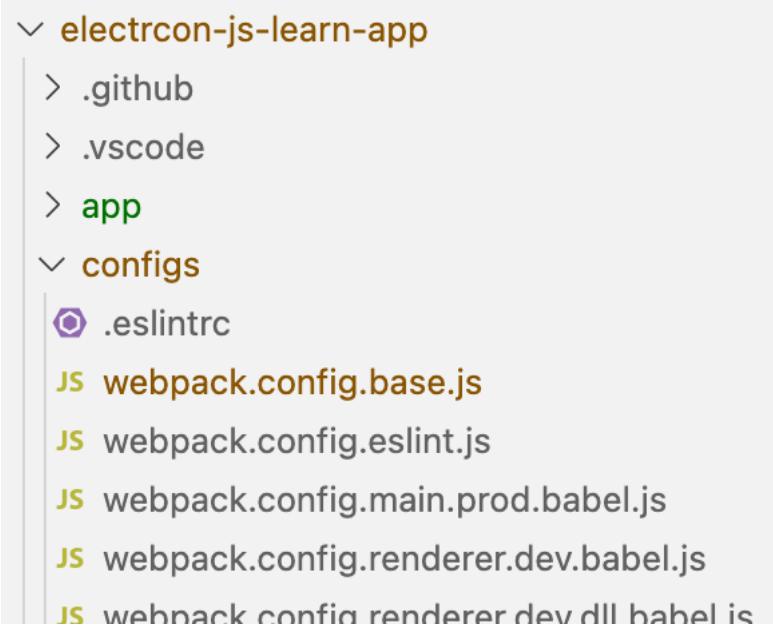


אפשר לעשות קומיט ו-push git לריפוזיטורי המרוחק ובעצם לעשות חלק גדול מהפעולות של CLI Git, אבל חשוב לציין שלא את כלן. מי שישולט ב-CLI יסתדר היטב עם Github GUI, אולי יהיה מההממשק הנוח, אבל מי שמסתדר עם GUI בלבד לא יוכל לעבוד עם CLI Github.

## תוספים לגיט ב-`git`

סביבת הפיתוח של VSCode (שבספרים הקודמים תוכלו ללמוד עליה רבות) עשירת בתוספים המאפשרים עבודה קלה ונוחה עם גיט, אבל גם סביבות פיתוח אחרות מכילות תוספים כאלה שמקילים את העבודה. אסביר מעט על התמיכה של VSCode בגיט כיוון שהוא IDE הפופולרי ביותר.

VSCode מכילה תמיכה בגיט כברירת מחדל. כאשר פותחים תיקייה שיש בה ריפורזיטורי, נראה שעכז התיקיות מכיל צבעים שונים:



הצבעים האלה מסמנים את סטטוס הקבצים: יירוק לקובץ חדש שלא נכנס לקומיט (יש צורך להוסיף אותו), חום לקובץ שיש בו שינויים. למשל, בתמונה הקודמת – התיקייה הירוקה היא תיקייה חדשה, והקובץ המסומן בחום (`webpack.config.base.js`) הוא קובץ שנעשה בו שינויים.

## Git blame

אם רוצים להרחיב את השימושות של גיט-IDE, אפשר להתקין תוספים שונים. המפורסם שבהם הוא GitLens. התקינה שלו נעשית כמו התקינה של כל Tosfot ב-VSCode, ומהרגע שהוא מותקן אפשר לראות בשינויים בכל קובץ שיש בריפוזיטורי המקומי שلنנו את ההיסטוריה של כל שורה ושורה ומילא אחרראי עליה. זה נहדר כאשר עובדים על פרויקט גדול שיש לו תורמים רבים (בחברת הייטק למשל) וצריך לראות מי עבד על הקובץ לפניו, במקרה שיש לנו שאלות אלו.

```

> OPEN EDITORS 1 UNSAVED
BRANCH-NAME-LINT
> .github
< .husky
> _ 
❖ .gitignore
pre-commit
> .nyc_output
> bin
> node_modules
❖ .editorconfig
❖ .eslintrc.js
❖ .gitattributes
❖ .gitignore
❖ .npmrc
CHANGELOG.md
JS cli.js
JS index.js M
JS license
JS npm-shrinkwrap.json
JS package.json M
readme.md
sample-configuration.json
JS test.js

{} package.json > {} lint-staged > {*}js,*json
42   "meow": "9.0.0"
43 },
44 "devDependencies": {
45   "ava": "^3.15.0",
46   "eslint": "7.28.0",
47   "eslint-config-airbnb-base": "^14.2.1",
48   "eslint-plugin-import": "^2.23.4",
49   "husky": "^6.0.0",
50   "lint-staged": "11.1.1",
51   "nyc": "^15.1.0",
52   "sinon": "^11.1.1"
53 },
54 "lint-staged": [
55   {"*.{js,json)": "eslint --cache --fix"
56 }
57 }

58

```

You, a minute ago (September 7th, 2021 1:13pm)  
Uncommitted changes  
Working Tree <<  
+ \*.js: "eslint --cache --fix"  
Changes Index → Working Tree | ⌂  
You, seconds ago \* Uncommitted changes

האפשרות הזו נקראת **git blame** ואפשר להפעיל אותה עם שורת הפקודה:

`git blame FILENAME`

הפלט יהיה הקומיט של כל שורה ושורה יחד עם הודעת הקומייט. השם **blame** (האשמה) הוא חצי הומוריסטי ונועד לرمוז שימושים בפקודה זו כדי לראות מי אחראי לשורה מסוימת ו"להאשים" אותו, אבל האמת היא שלעיתים משתמשים בשורה זו של ההערה כתיעוד צללים – ככלומר מקפידים על commit messages על תקינים ואז אפשר לקבל תיעוד על כל שורה ושורה עם GitLens יישורות בתחום ה-IDE. עם GitLens אפשר לעשות עוד כמה פעולות יישורות דרך ה-IDE. לא נלמד עליו כאן, אבל יש עליו פירוט מדויק באתר התוסף:

<https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens>

## GitHub Actions

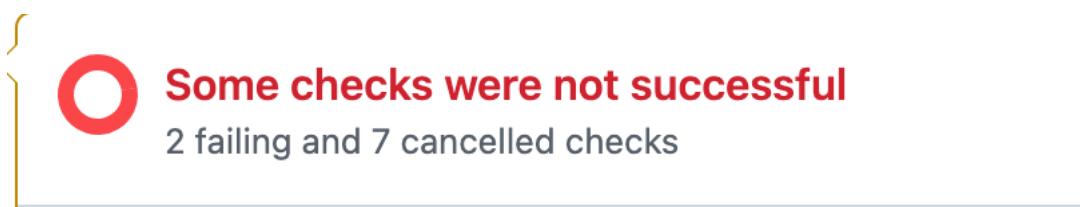
אחד הדברים החשובים בעבודה עם גיטהאב הוא היכולת לבצע שני תהליכיים – CI (Continuous Integration) ((Continuous Deployment

## תהליך CI

CI הוא הליך שבמסגרתו הקוד החדש שהוא שארנו משלבים (כלומר עושים לו אינטגרציה) בקוד היישן נבדק לפי סטנדרטים מסוימים. אילו סטנדרטים? למשל, בג'אוויסקריפט בודקים סטנדרטים עם eslint. בשפת PHP בודקים סטנדרטים עם PHP-CS-Fixer. הסטנדרטים האלה מודאים שהקוד שלנו בעל אינדנטציה של רווחים ולא של טאבים (או להפָק), שיש את התו ";" בסוף כל שורה, שאין log.console בקוד וסט עצום של כללים שונים, בהתאם להחלטת ארכיטקט התוכנה או המפתח הבכיר (בספר "למוד ג'אוויסקריפט בעברית" תמצאו שני נספחים הדנים באיכות קוד בג'אוויסקריפט ואיך לבדוק אותה. אבל כפי שיש בדיקת קוד סטטיסטית בג'אוויסקריפט, היא קיימת גם בשפות אחרות).

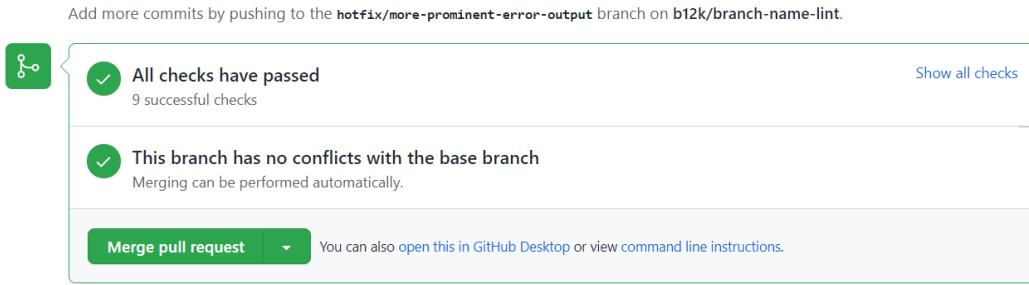
ישנן גם בדיקות נוספות. למשל, בדיקת אבטחה שבודקת מודולים וחבילות תוכנה, כמו מודולים של Node.js, כדי לראות אם הם מעודכנים. הבדיקה הזאת נעשית על מודולים חדשים שהוספנו. ישנן גם בדיקות מהירות, בדיקות סיבוכיות קוד ובדיקות רבות אחרות שנעשות על הקוד אחרי שהוספנו את התוספה שלנו. אם הבדיקות האלה עוברות בהצלחה, סימן שהתוספה שלנו עומדת בכללים שהוגדרו מראש ושארנו אמונים על שמירת איכות הקוד שאנו מיצרים.

בדיקות אלו נראות בבדיקות CI, ולא מעט פרויקטים של קוד פתוח משתמשים בהן בתהיליכים שונים של שילוב קוד. נראה אותן במשק גיטהאב בפול ריקווסט שאנו עושים לפרויקטים שהגדירו CI. כאשר נעשה פול ריקווסט, נראה שהבדיקות מתבצעות, ולאחר מכן נראה אם הן הצליחו או לא. אם הן לא הצליחו, אפשר לראות את ה成败ון.



זה חלק ממשמעוני מאד מתרומה לקוד פתוח – היכולת לעמוד בסטנדרטים של הפרויקט שתורמים לו קוד. דוקא מתוך הכשלון נוכל להבין מה השتبש, לתקן ואולי להבין איך מפעילים ובודקים את הקוד לפני הכשלון. גם בחברות מסחריות רבות יש תהליכי CI

שמיצר קוד תקין ואחד. זהה דוגמה מצוינת להבדל בין אדם שיודע תכונות לבין מתכנת אמיתי – מתכנת שהוא מוכזו יודע להבין מה השتبש בקוד שהוא כתב, על אילו סטנדרטים או תהליכי CI הוא עבר ואיך לתקן את הטעון תיקון (כל פרויקט מפעיל כלים אחרים לבדיקות קוד).



החיבור הזה נעשה באמצעות מגנון שנקרא Github actions. חשוב לדעת עליו כאשר תורמים קוד למוצר קוד פתוח או כשבודדים בצוות פיתוח בחברה. בסוף זה לא נלמד איך מתקנים אותו.

## תהליכי CD

בעוד תהליכי CI בודק את תקינות המערכת אחורי שהוספנו לה קוד, תהליכי CD פועל אחורי שהפול ריקווט שלנו מזג. התהליך הזה לוקח את הקוד המאוחד ועשה מה שבתעשייה קוראים לו דיפלי (Deploy), כלומר מפיץ את הקוד. איך? תלוי במוצר. אם מדובר במודול של `js`, יהיה `publish` עם גרסה חדשה. אם מדובר באתר, יבוצע עדכון של הקוד בשרת האתר. הכל תלוי במוצר. לעיתים ה-CD מתרחש רק בשעת יצירתת טג או `release` חדש. גם את ה-CD מבצעים עם GitHub actions, שלא נלמד בסוף זה.

## תרגול גיט

בכנות, תרגול גיט הוא תרגול טכני מאד שעלול להיראות תלוש מאד. התרגול האמיתי הוא להתחיל לעבוד עם גיט מיד בכל הפרויקטים שלכם ולתרום לפרויקטים בקוד פתוח. זהו בפועל התרגול הטוב ביותר, ובוצע אותו בחלקי הספר האחרים, אבל המימנות

הטכנית גם היא חשובה, ולפיכך יש כאן כמה תרגילים שיסייעו לכם לתרגל את המומנות הזו – אם כי הם עלולים להווראות תלושים מהמציאות.

לשם התרגול יש להתקין גיט לפי ההוראות. למשתמשי חלונות – יש להשתמש ב-`git bash`. מומלץ להכניס את השינויים באמצעות `VSCode` ולבוחן את עצםם כל הזמן באמצעות הפקודות:

```
git diff
git log
git branch
git status
```

שכלן נלמדו בפרקם הקודמים.

את כל התרגילים יש לבצע לפי הסדר, כיוון שהלכם נסמכים על תרגילים קודמים. אנו משתמשים דוקא בקובץ `md` כדי לתרגל עם טקסט שכולם מכיריהם.

### **יצירת ריבזיטורי עם בראנץ' `main`**

צרו ריבזיטורי בשם `hatikva` בתיקייה בעלת אותו שם.

**פתרון**

```
mkdir hatikva
cd hatikva
git init
```

```
C:\local>mkdir hatikva
C:\local>cd hatikva
C:\local\hatikva>git init
Initialized empty Git repository in C:/local/hatikva/.git/
```

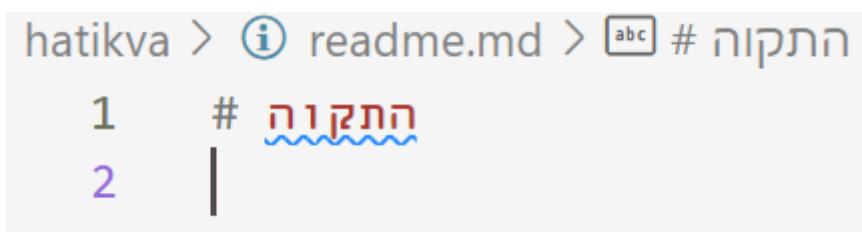
## יצירת initial commit

צרו קובץ בשם md והכניסו אליו את השורה:

```
# התקווה  
הכניסו את הקובץ עם קומיט והערה:  
.initial commit
```

### פתרון

יצירת הקובץ:



יצירת הקומיט עם:

```
git add readme.md  
git commit -m"Initial Commit"
```

```
C:\local\hatikva>git status  
On branch main  
  
No commits yet  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    README.md  
  
nothing added to commit but untracked files present (use "git add" to track)  
  
C:\local\hatikva>git add README.md  
  
C:\local\hatikva>git commit -m"Initial Commit"  
[main (root-commit) 1dad21b] Initial Commit  
  1 file changed, 1 insertion(+)  
  create mode 100644 README.md
```

## יצירת בראןץ'

צרו בראןץ' בשם first-verse והכנסו אל תוך קובץ readme.md את הטקסט הבא:  
 עוד לא אבדה תקותנו  
 התקווה הנושנה  
 משוב לארץ אבותינו  
 לעיר בה דוד חנה.  
 הכנסו אותו כקובץ לבראנץ' עם הערה .The first verse עם הקובץ:  
 פתרון

```
hatikva > ⓘ readme.md > abc # התקווה #
1      # התקווה
2
3      עוד לא אבדה תקותנו
4      התקווה הנושנה
5      משוב לארץ אבותינו
6      לעיר בה דוד חנה
7
```

הפקודות אחורי שמקלידים את הטקסט לקובץ:

```
git checkout -b first-verse
git add readme.md
git commit -m"The first verse"
```

```
C:\local\hatikva>git checkout -b first-verse
Switched to a new branch 'first-verse'

C:\local\hatikva>git branch
* first-verse
  main

C:\local\hatikva>git status
On branch first-verse
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   readme.md

no changes added to commit (use "git add" and/or "git commit -a")

C:\local\hatikva>git add readme.md

C:\local\hatikva>git commit -m"The first verse"
[first-verse 1290e4b] The first verse
  1 file changed, 5 insertions(+)
```

## מיזוג בראנץ'

mezgo את הbraanz' first-verse main ומחקו את הbraanz' first-verse לאחר המיזוג.

### פתרונות

חזרה לـmain:

```
git checkout main
```

ביצוע מרג':

```
git merge first-verse
```

מחיקת braanz' first-verse:

```
git branch -d first-verse
```

```
C:\local\hatikva>git checkout main
Switched to branch 'main'

C:\local\hatikva>git merge first-verse
Updating 1dad21b..1290e4b
Fast-forward
  README.md | 5 +++++
  1 file changed, 5 insertions(+)

C:\local\hatikva>git branch -d first-verse
Deleted branch first-verse (was 1290e4b).
```

## יצירת בראנץ' main

צרו בראנץ' נוסף של add-punctuation, החליפו את הבית הראשון בבית זהה:

עוד לא אָבְּזָה תִּקְוֹתֵנוּ

התקווה הנטשנה

משוב לארץ אָבוֹתֵינוּ

לעיר בה דוד חנה.

צרו קומיט עם הערה "Add punctuation to first verse" וחזרו לـ`main`.

### פתרונות

יצירת בראנץ' עם:

```
git checkout -b add-punctuation
```

החלפת הטקסט עם:

```
diff --git a/readme.md b/readme.md
index 1234567..8901234
--- a/readme.md
+++ b/readme.md
@@ -1 +1 @@
 1 # התקווה #
 2
 3 3 עד לא אבדה תקנותינו
 4 4 התקווה תפונשנה
 5 5 מושב לארץ אבותינו
 6 6 לעיר ביה דוד ענן
 7
```

שמירה ויצירת קומיט עם:

```
git commit -m"Add punctuation to first verse"
```

חזרה לـ`main` עם:

```
git checkout main
```

אפשר לוודא ב-IDE שהטקסט השתנה לגרסתו עם הניקוד.

```
C:\local\hatikva>git checkout -b add-punctuation
Switched to a new branch 'add-punctuation'

C:\local\hatikva>git add readme.md

C:\local\hatikva>git commit -m"Add punctuation to first verse"
[add-punctuation 4c5d2df] Add punctuation to first verse
 1 file changed, 4 insertions(+), 4 deletions(-)
```

## יצירת בראנץ' נוסף main מ-`main`

ניצור בראנץ' חדש `main` בשם `alternate-verses` ונכנס לתוכו את הטקסט הבא:

כל עוד בלב פנימה

נעפֵש יהוזי הומיה,

ולפאתמי מזרח, קדימה,

עין לציוון צופיה,

עד לא אבזהה תקנותנו,

התקווה בת שנות אלפים,

להיות עם חפשי בארץנו,

ארץ ציון וירושלים.

ניצור קומיט עם ההערה: New verses. נחזור ל-`main` ונבצע מרג' אליו.

## פתרונות

```
git checkout -b alternate-verses
```

הכנסת הטקסט:

hatikva > ⓘ readme.md > abc התקוה #

1      # **התקוה**  
2  
3      פָּלֶל עַזְזֵד בְּלִבְבָּשׁ פְּנִימָה  
4      , בְּפֶשֶׁת יְהוָה נִזְמָנָה  
5      , וְלֹפְאָתָה מִזְרָחָה קָדְיָמָה  
6      , עַל צְוֹפִיה לְצִיוֹן  
7      ^  
8      , עַזְזֵד לֹא אָבְדָה תְּקִוָתָנו  
9      , סְתִמְנוּה בַת שְׁבָוֹת אֲלִפָּים  
10     , לְהִיוֹת עִם צְפָלִי בָּאָרֶץ נָנו  
11     . אָרֶץ צִיוֹן וַיְרֻשָּׁלַיִם  
12

```
git add readme.md
git commit -m"New verses"
git checkout main
git merge alternate-verses
```

```
C:\local\hatikva>git checkout main
Switched to branch 'main'

C:\local\hatikva>git checkout -b alternate-verses
Switched to a new branch 'alternate-verses'

C:\local\hatikva>git add readme.md

C:\local\hatikva>git status
On branch alternate-verses
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   readme.md

C:\local\hatikva>git commit -m"New verses"
[alternate-verses 0eaddf9] New verses
 1 file changed, 11 insertions(+), 6 deletions(-)
 rewrite readme.md (89%)

C:\local\hatikva>git checkout main
Switched to branch 'main'

C:\local\hatikva>git merge alternate-verses
Updating 1290e4b..0eaddf9
Fast-forward
  readme.md | 13 ++++++++----
  1 file changed, 9 insertions(+), 4 deletions(-)
```

## פתרון קונפליקטים

מתוך `main`, בצעו מיזוג לבראנו '`.add-punctuation`'

מה קרה?

אין פותרים את מה שקרה?

## פתרון

המיזוג נעשה באמצעות ידוע שאנו ב-`main` והפקודה:

`git merge add-punctuation`

```
C:\local\hatikva>git branch
  add-punctuation
  alternate-verses
* main

C:\local\hatikva>git merge add-punctuation
Auto-merging readme.md
CONFLICT (content): Merge conflict in readme.md
Automatic merge failed; fix conflicts and then commit the result.
```

מה שקרה הוא קונפליקט. התרחש כי ניסינו למזג בראנץ' לקטע קוד שהשתנה.

```
hatikva > ⓘ readme.md > abc # עד לא אבדה תקנות #
1  # התקנות
2
3  Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
4  <<<<<< HEAD (Current Change)
5  כל עוד בלבב פנימה
6  נפש יהודיה הומיה
7  ולבאות מזרחה, קדימה
8  ענו לאין צופיה
9  ,
10 , ועוד לא אבדה תקנות
11 , התקנות בת פנויות אלפים,
12 , להיות עם חפשי בארכון
13 . ארץ ציון וירשלים
14 =====
15 עוד לא אבדה תקנות
16 התקנות הפה
17 מושב לארכז אבותינו
18 >>>> add-punctuation (Incoming Change)
19 |
```

עם GitLens אפשר לנוקוט פעולות באמצעות תפירת קטן או לעשות את הכל טקסטואלי. למשל, אפשר לקבל את שני השינויים:

התקהה > ⓘ readme.md > abc # התקהה

1      # התקהה  
2  
3      פֶל עֲזָד בְּלַבְבָ פִנְיִמָה  
4      , נְפַלֵ שִׁיחַ הַזְמִינָה  
5      , וְלֹפָאַתִי מִזְרָחָה, קָדְיִמָה  
6      , עִירְצִוְנוֹן צְוֹפִיה  
7  
8      , עֲזָד לֹא אָבְדָה תְקֻנוֹתָנוּ  
9      , סְתֻקְנוּה בַת שְׁגָנָות אַלְפִים  
10     , לְהִיוֹת עִם זְפַלְלִי בָאָרֶץ נָנוּ  
11     . אָרָק צִוְנוֹן גִּירְוָשְׁלִים  
12     עֲזָד לֹא אָבְדָה תְקֻנוֹתָנוּ  
13     סְתֻקְנוּה חַבּוֹשְׁנָה  
14     מְשֻׁבָב לְאָרָק אָבּוֹתֵינוּ  
15     . לְעִירְבָה קָדוֹד חַנָה  
16

לקבל את השינוי incoming, כלומר זה שרצינו למזג:

התקוה > ⓘ readme.md > abc # התקוה

```

1  # התקוה
2
3  עוד לא אבדה תקונתנו
4  התקונה הפולשנה
5  מושוב לארץ אבוטניינו
6  •לעיר בה דוד חגה
7

```

או להעדיף את הקויום:

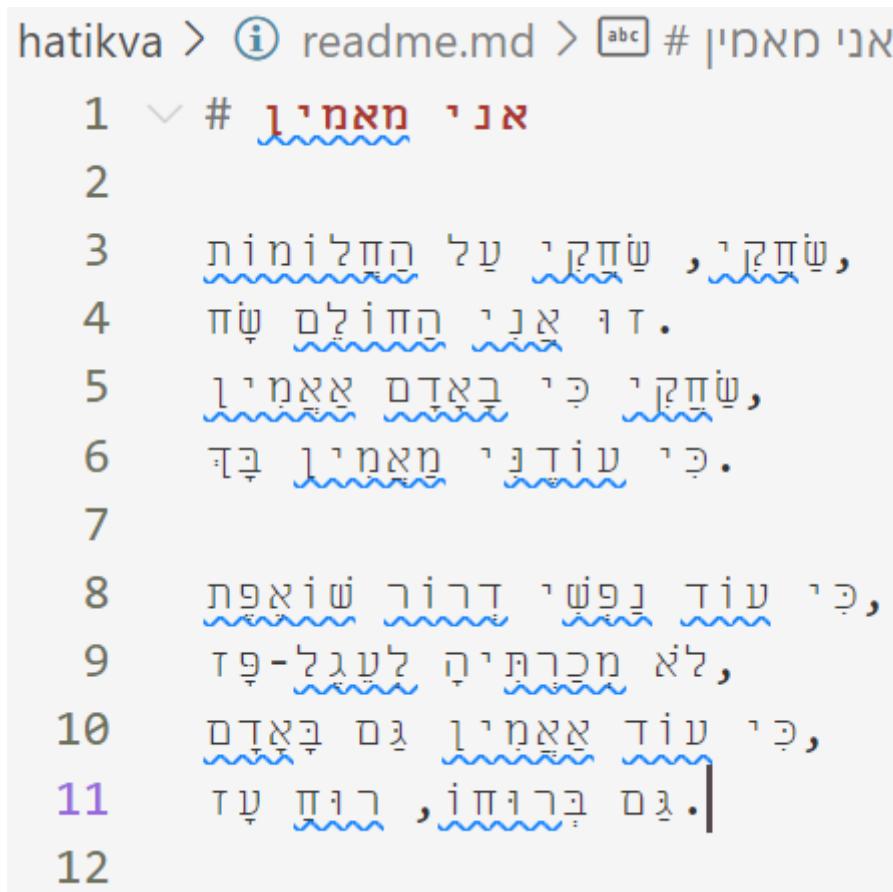
התקוה # התקוה

```

1  # התקוה
2
3  כל עוד בלבב פגימה
4  , נפש בהודו הומיה
5  , כל פתמי מזרע, קדיימה
6  , עית לנצח צופיה
7
8  ,
9  ,
10 ,
11 ,
12

```

אפשר גם לעבוד באופן ידני וליצור קוד מאוחד שמחבר את שני השינויים או מבטל אחד מהם. למשל, להקליד ידנית טקסט חדש לחלוטין:



```

ANI CAIMI # ani maamit
1
2
3 , שָׁמַךְנִי עַל הַפְּלִזּוֹמוֹת
4 . זֶה אֲנִי כְּחֻזֵּל מִשָּׂה
5 , שָׁמַךְנִי בְּאַדְם אֲמִינִי
6 . כִּי עוֹד בְּיַמְּמָאָמִינִי בְּדָ
7
8 , כִּי עוֹד בְּנִפְשֵׁר דָּרוֹר שׂוֹאָפָת
9 , לֹא מְכַרְתִּיכְ לְעִגְלָ-פָז
10 , כִּי עוֹד אֲמִינִי גַּם בְּאַדְם
11 . גַּם בְּרַוחַן, הוּא עַז
12

```

לא משנה איפה דרך נוקטים, אחורי שפוטרים את הקונפליקטים ומוחקים את:

&gt;&gt;&gt;&gt;&gt;HEAD

=====

&gt;&gt;&gt;&gt;&gt; add-punctuation

מבצעים קומיט נוסף. מקובל לכתב בהערה .Resolve conflicts

```
C:\local\hatikva>git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:  README.md

no changes added to commit (use "git add" and/or "git commit -a")

C:\local\hatikva>git add README.md

C:\local\hatikva>git commit -m"Resolve conflicts"
[main 13c09b5] Resolve conflicts
```

וכך פותרים את הקונפליקט.

## סיכום

אם עברתם על הפרקים לפי הסדר והקפדתם לעשות את התרגול האחרון, אתם יודיעים גיט. בתחילת כל דבר יהיה קשה, יהיו הודעות שגיאה מזורמת, יהיה קשה להבין למה יש קונפליקטים, מפת הבראנצ'ים לא תהיה שלמה בראשם, לא תבינומתי לעשות מרג' או מה צריך לעשות rebase. אבל ככל שייעבור הזמן, כך זה יבוא לכם יותר בקלות, בדיקות כמו רכיבת על אופניים וניהגה. בהתחלה יש קושי טכני ממש בתפעול ההילוקים או הcidon, אבל אחר כך זה נהפך לטבע שני.

יש עוד המון מידע על גיט שלא כיסינו כאן – למשל git alias או git hooks. הם מיועדים למשתמשים מתקדמים של המערכת – מתכנתים בכיריהם (סנוירים) או ארכיטקטנים המגדירים את המערכת ואת היישוֹלְגִיט. המידע הזה אינו נדרש על מנת לעבוד עם גיט ומתקנתים מן השורה כמעט אינטנסיבית במסגרת תפקידים. הידע בಗיט הוא הסטנדרט בתעשייה. אי-אפשר לעבוד בתחום ולא להכיר את גיט ולבוד אותו. זו הסיבה שצריך מיד להתרגל לעבוד אליו באופן מסודר לפי היישוֹלְגִיט המקבול, גם בפרויקטים האישיים שלכם וМОבן שבפרויקטים של קוד פתוח. השלב הבא הוא להתחיל לעבוד על הפרויקט שלכם עם גיט ולהתחיל לחפש בಗיטהאב הזדמנויות לתרומות קוד. אין דרך אחרת לצבור ניסיון בוגוט.

פרק 2

# פרוייקטים לדוגמה

# פרויקטים לדוגמה

## למה זה חשוב?

פרויקטים לדוגמה הם קרייטיים בכל מה שקשור להבנת החומר. בספרים הקודמים – "לلمוד ג'אווהSCRIPt בעברית", "للمוד Node.js בעברית", "للمוד ריאקט בעברית", "للمוד MySQL בעברית" ו"للمוד jQuery בעברית" – יש לא מעט תרגילים והפניות לאתרי תרגול. אבל יש הבדל בין תרגול טכני, שמלמד שליטה בשפה, לבין בניית פרויקט מكيف המשלב כמה פרדיגמות של תוכנות ומכליל לא מעט קוד. לא מעט בוטקאמפים ובתי ספר לתכנות מקפידים על כך שהסטודנטים שלהם יבנו פרויקטי גמר כדי שייהי להם מה להראות בקורס החיים. זו הסיבה שגם בספר זה יש פרויקטים לדוגמה – כדי שמי שלומדים מהם יוכלו להתנסות בהם וייהי להם בסיס לפרויקטים עצמאיים שאפשר לשלב בקורס החיים. פרויקט מרשימים לדוגמה יספק לכם יתרון על פני מי שיש לו אותם קורות חיים והשכלה, אך אין לו פרויקט זהה.

## איך לומדים מהפרויקטים לדוגמה שיש בספר?

בחילק הזה של הספר ישנו שלושה פתרונות לדוגמה. הראשון הוא וידג'ט (פייסט קוד הניתנת להשתלה באתרים ומציגה מידע) קטן שמציג תמונות מ-[imgflip.com](#). השני הוא אפליקציה סטטית של ריאקט והשלישי הוא מגנון לסריקה של אתרים ושמירת התוצאות במסד נתונים. המלצתה שלי היא לקרוא קודם את תיאור הפרויקט, ורק אז לחשב איך הייתם מתכוונים אותו. נסו לשרטט אותו, לתקן את מבנה הקוד, אפילו על דף, ולפרט איך רכיבים שאתה ייראו ויתקשרו זה עם זה. מנהל פיתוח חכם אמר לי פעם ש"שבועיים של כתיבת קוד חוסכים שעתיים של תכנון". מתכוונים, בוודאי בתחילת דרכם, ממהריהם לצלול לתוך פרויקט ולהתחיל לכתוב, אך חשוב דווקא לפני הכתיבה לתקן ולהבין את זרימת המידע, לתקן בקווים כלליים איך ייראו הקוד והקומפוננטות, ומה יהיו הפלט והקלט. נכון, זה נראה מעט תלוש ובטח לא מעניין מספיק, אבל זה חשוב מאד, במיוחד כמשמעותו לבניית מערכות בעולם האמיתי.

אחרי שתכננתם את הפרויקט, נסו לראות אם התכנון שלכם זהה לתכנון שאני הצעתי. נסו למשתמש את הפרויקט בדרך שלכם ולאו דווקא בדרך שלי. אם נתקעתם – נסו לראות מה הפתרון שאני הצעתי ואם הוא מתאים לכם. הדרך הטובה ביותר היא דווקא הארוכה – לנסות לבנות את הפרויקט בדרך שלכם, עם הטעויות השונות. הטעויות הן חלק ממשמעותי ביותר מהתלמיד.

בסוף כל פרויקט יש רעיונות להמשך הפיתוח. אפשר ורצוי לנסות לפתח את הפרויקט להלאה – לבנות פיצ'רים חדשים ולעשות בדיקות או כל דבר נוסף שתחשבו עליו. תוכנה לומדים דרך הידים ודרך בניית הפרויקטים. כדאי מאוד לא לוותר וגמ אם לא הצליחם לבנות בעצמכם או למש בעצמכם, לפחות להבין עד הסוף את קוד הפרויקט ולבנות את ההצעות המפורטות בסיכום כל פרויקט.

אסכם ואומר שהדרך הטובה ביותר להפיק תועלת מהפרויקטים לדוגמה היא:

1. לקרוא את תיאור הפרויקט ואז לכתוב איך הייתם ממשים אותו.
2. לקרוא את התכנון, להבין אם יש שוני בין הדרך שאתם חשבתם אליה לבין הדרך שאני חשבתי אליה ולתקן את התכנון שלכם אם מצאתם הבדלים מהותיים.
3. להתחיל למש בעצמכם. אם נתקעתם, אפשר לבדוק איך אני עשית את זה – בהתאם לשלב שנתקעתם בו.
4. להגיע לשלב שבו הפרויקט עובד למגורי, אפילו אם זה באמצעות העתקה של הקוד שאני הצעתי.
5. להמשיך את הפרויקט בהתאם להצעות בסיכום.
6. לבנות פרויקט דומה אחר עם קוד ותכנון שלכם ולהעלות לgitהאב.

## הכנסת פרויקט הדמו אל קורות החיים

פרויקט משלכם הוא לגמרי חלק מהר诏מה המקצועית שלכם ויכול להיכנס לקורות החיים. הוא לא נחשב כמו תרומה ממשמעותית לקוד פתוח, אבל הוא נחשב. הוא גם פותח פתח לדיוון מקצועי על הקוד שלכם בזמן ראיון עבודה.

אם אתם משלבים בקורות החיים פרויקט משלכם שמוצג כפרויקט גמר או כפרויקט עצמי, חשוב שהוא יהיה מורכב, שייהי בניו לפי סטנדרטים (למשל הקפדה על `eslint`, כדי שמוסבר גם בפרויקטים עצם וגם בנספחים בספר "למוד ג'אווה ספרייפט בעברית"), שייהי בגיטהאב ושתיהיה גרסה חיה שלו. להכנס פרויקט לגיטהאב אתם כבר אמרוים לדעת. יצירת גרסה חיה לפרויקט מסוימת במלואה בפרקeos על דיפלומנט.

אם אתם יכולים ליצור פרויקט שהוא לא דמה, אלא משתמשים בו ממש, זה cocci טוב שיכול להיות. אני מכיר מתכנתים שדריכם המקצועית החלה בבניית פרויקט גמר שהוא גם פרויקט שימושי, למשל מערכת ניהול קטנה שטוענת כרטיסי תן ביס, מערכת ניהול למשרד עורכי דין או יידגיטים משוכלים לחניות. קוד שאתם מפתחים ואנשימים משתמשים בו שווה הרבה יותר מאשר פרויקט גמר או פרויקט לדוגמה, גם אם הם מושקעים מאוד.

כן או כן, חשוב להכנס לקורות החיים כל דבר הקשור לקוד שעשיתם, בין שמדובר בפרויקט לדוגמה ובין שבפרויקט שעשיתם ללקוח או בפרויקט שלכם שימושים בו. את הפרויקט מננים יחד עם פירוט קצר עליו ועל מטרתו, כולל פירוט של הטכנולוגיות שהשתמשתם בהן כדי ליצור אותו וקייםו לגיטהאב ולדמו חי.

פרק 3

# ויזגורט ג'אוושה סקייפט המציג תמונה מם פופולרי



# וידגט ג'אווהסקרייפט המציג תחומותם פופולרי

## הגדרת הצורך

לקוח בעל אתר חדש ביחס קוד ג'אווהסקרייפט שיציג לו את המם (meme) הפופולרי ביותר שיש באתר imgflip על מנת להציג את הנתונים. הלקוח הסביר שיש API המציג את הממים הפופולריים ביותר עם שמותיהם ב:

[https://api.imgflip.com/get\\_memes](https://api.imgflip.com/get_memes)

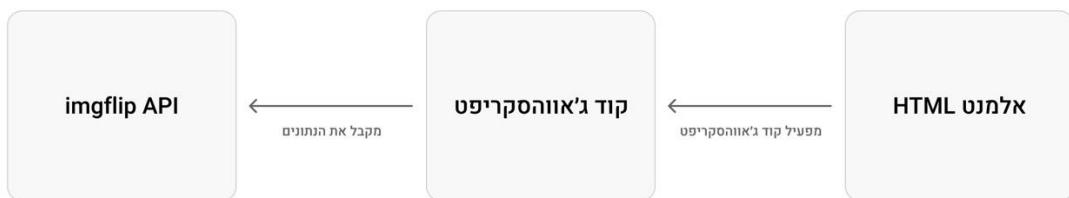
רוחב הוידגט לא יכול לעלות על 300 פיקסלים.

## הבנת הוידגט

הקוד הוא קוד ג'אווהסקרייפט טהור ללא פרימורוק שצמוד לקוד HTML. ככלומר, צריך ליצור קוד HTML ולהציג אליו קוד ג'אווהסקרייפט, שמתחבר אליו ומשנה את האלמנט שיש בו – מכניס אליו את המידע שהוא מקבלים מה-API.

האתגר המרכזי הוא להתחבר ל-API, להבין את זרימת המידע ולהכניס את המידע אל ה-DOM.

זרימת המידע של הוידגט באופן נאיyi נראה כך:



## תיכנון

ניצור קובץ HTML שיש בו אלמנט `<div>` עם `id`. הקוד שלנו יפעיל את הפעולות וייצוק את התוכן אל ה-`DOM` באמצעות האלמנט שמחוזר על ידי `getElementById`. התוכן יתקבל ב-`AJAX`.

## יצירת התשתייה

התשתייה פה היא דף HTML בודד שיש בו את ה-`div` שלנו וכן תגית `script`.

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8" />
    <title>Embedded Widget</title>
  </head>
  <div id="imgflip-widget"></div>
  <body>
    <script>
      // Here will be our script
    </script>
  </body>
</html>
```

את הדף זהה נפתח עם דפסון ונוכל לעבוד.

הבעיה היא שבכל שינוי נctrar לגשת אוטומטית אל הדפסון ולרפרש. באופן אישי, אני מעדיף לעבוד מול [codepen.io](http://codepen.io) בקטיעים כאלה, או להתקין מודול של `jsNode` בשם `http-serve`. אני אסביר על המודול הזה – אבל מי שלא מכיר כלל את `jsNode` עלול להתבלבל, ולפיכך מומלץ לו לדלג אל הסעיף הבא (או לקרוא את הספר "למידה `jsNode` בעברית".)

## הפעלת `serve` המיעוד לפיתוח סטטי

אם יש לנו קובץ `index.html`, אנחנו יכולים לעבוד עם `serve`. כל מה שצריך לעשות זה ליצור פרויקט `Node` בתיקייה שבה הקובץ נמצא באמצעות הקלדת:

```
npm init
```

בטרמינל ואז התקנת `serve` באמצעות:

```
npm i serve
```

ההפעלה היא באמצעות הקלדת הפקודה:

```
npx serve
```

וזהו. מהריגע שהפקודה הוקלדה, יש לנו שרת בשם `localhost:5000` שמתחדש בכל שינוי, זהה מכך מאוד את הפיתוח של קוד סטטי.

→ `static-widget npx serve`

**Serving!**

- Local: <http://localhost:5000>
- On Your Network: <http://192.168.2.57:5000>

Copied local address to clipboard!

## צירמת קריאה לשרת

החלק הראשון הוא ליצור פונקציה אסינכורונית. כיוון שיש לנו קוד אסינכורוני שմבצע קריאה לשרת, אנחנו חייבים לעבוד אסינכורונית והקוד שלנו יהיה בתוך פונקציה כזו. נתחיל בכתיבת פונקציה פשוטה שתכיל כמעט את כל הקוד שלנו:

```
async function init() {
```

```

    console.log('hello world');
}

init();

```

נבדוק ונראה שבאמת קיבלנו בקונסולה שלנו hello world

השלב הבא הוא ליצור את הפונקציה שקוראת ל-`API`. נשתמש במנגנון `fetch` שקורא ל-`JSON`. נשתמש גם במודול `async await` המודרני. זהו הקוד המלא, וכבר ניתן אותו:

```

async function getMemes() {
  const url = "https://api.imgflip.com/get_memes";
  const response = await fetch(url);
  const json = await response.json();
  return json.data.memes;
}

async function init() {
  const memes = await getMemes();
  console.log(memes);
}

init();

```

הפונקציה `getMemes`, שהיא לב הויידג'ט שלנו, היא פונקציה אסינכורונית המשתמשת ב-`fetch` – הדרך הנכונה לעשות AJAX. משתמשים ב-`async/await` כדי לבצע את הקריאה, וכן נמנעים משימוש יתר ב-`then`. הקריאה מתבצעת עם שלוש השורות האלו:

```

const response = await fetch(url);
const json = await response.json();
return json.data.memes;

```

כיוון שאנו מעוניינים אך ורק במקרים מסוימים, אנחנו לוקחים את התגובה ומוציאים רק את:

```
json.data.memes;
```

אם תבדקו, תראו שיש עוד מידע שחווץ מהשרות. כיוון שאנו מעוניינים רק בـ`memes`, אנו מתיחסים רק אליהם.

פונקציית `getMemes` לא תעבור אם לא נקרא לה. אנו קוראים לה בפונקציית `init` ומכניסים את התשובה לקונסולה על מנת שנוכל לראות שהכל עובד:

```
async function init() {
  const memes = await getMemes();
  console.log(memes);
}
```

ב-`API` של ה-`response` מקבלים אובייקט גדול שאפשר לבדוק אותו עם הקונסולה או בלשונית הנטוורק:

Name	Headers	Preview	Response	Initiator	Timing
get_memes			<pre>{   "success": true,   "data": {     "memes": [       {         "id": "181913649",         "name": "Drake Hotline Bling",         "url": "https://i.imgur.com/lc...",         ...       },       ...     ]   } }</pre>		

אפשר לראות את המבנה של התגובה ולהבין למה צריך רק את `.data.memes`.

אם הכל עובד, השלב הבא הוא לבצע מניפולציה ב-`DOM` ולהכניס לשם את המידע הרצוי.

## הכנסת המידע ל-DOM

השלב הבא הוא לחת את האובייקט הראשון וליצור לו ייצוג ב-DOM. צריך להכניס תמונה וtekst ב-HTML. למשל, עלינו ליצור באמצעות `document.createElement` את אלמנט התמונה ואלמנט הטקסט ולהכניס אותו אל `div` שה-`id` שלו הוא `imgflip-widget`. אנחנו גם צריכים להגדיר את התמונה במקסימום רוחב של 300 פיקסלים.

הנה הקוד המלא:

```
async function init() {
    const containerElement = document.getElementById("imgflip-
widget");
    const memes = await getMemes();
    const img = document.createElement("img");
    img.width = "300";
    img.src = memes[0].url;
    containerElement.appendChild(img);
    const coverText = document.createElement("p");
    coverText.innerText = memes[0].name;
    containerElement.appendChild(coverText);
}
```

נכתב אומנו שורה אחר שורה.

```
const containerElement = document.getElementById("imgflip-
widget");
```

השורה זו בעצם מקבלת את ה-`reference` לאלמנט הראשי שלו שאיתו עובדים ובתוכו יוצרים את שאר האלמנטים.

```
const memes = await getMemes();
```

את השורה זו בעצם כבר כתבנו. אנו קוראים בה לשירות ומקבלים את המידע שצרי – מערך של כל הממימ. אנחנו צריכים את הראשון.

```
const img = document.createElement("img");
```

זהו ייצור אלמנט DOM מסווג תמונה. האלמנט זה הוא אלמנט `img` וברגע שיוצרים אותו, מכניסים אותו למשתנה בשם `img`. הוא עדין לא קיים חוץ מהזיכרון.

```
img.width = "300";
img.src = memes[0].url;
```

אנו מגדירים את רוחב התמונה ל-300 ומגדירים את ה-`src` של התמונה, ה קישור שלה, מתוך מערך `memes`.

```
containerElement.appendChild(img);
```

זו השורה החשובה – אנו מוסיפים את אלמנט `img` לקונטינר.

```
const coverText = document.createElement("p");
coverText.innerText = memes[0].name;
containerElement.appendChild(coverText);
```

שלוש השורות האלו עושים את אותו הדבר, רק שבמקום אלמנט תמונה משתמש באלמנט `p` – אלמנט פסקה ב-HTML - עברו שם המם.

והנה הקוד המלא:

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8" />
    <title>Embedded Widget</title>
  </head>
  <div id="imgflip-widget"></div>
  <body>
    <script>
      async function getMemes() {
        const url = "https://api.imgflip.com/get_memes";
        const response = await fetch(url);
        const json = await response.json();
        return json.data.memes;
      }

      async function init() {
```

```
const containerElement =
document.getElementById("imgflip-widget");
const memes = await getMemes();
const img = document.createElement("img");
img.width = "300";
img.src = memes[0].url;
containerElement.appendChild(img);
const coverText = document.createElement("p");
coverText.innerText = memes[0].name;
containerElement.appendChild(coverText);

}

init();
</script>
</body>
</html>
```

אם נסתכל על הקוד באמצעות הדפסן (בין שבפתחה ישירה ובין שב-server), באמת נראה את הממ הפולרי ביותר, עם הכיתוב שלו.



## Drake Hotline Bling

הלקוח בעצמך יקבל את הקוד הזה. אם הוא ידיביך אותו באתר (כלומר את קוד ה-HTML), יהיה לו וידגיט שיעבוד.

## סיכום

יש מי שצולל ושר אל הפרויקטים שיש בפרויקט או לפרויקט המורכב באמצעות עם MySQL ו-Node.js, אבל יש לא מעט פרויקטים בג'אווהסקרייפט בלבד. בדרך כלל אלה פרויקטים לפරילנסרים. לא חסרים בוני אתרים שרצוים וידגיטים קטנים של חדשות, של ממינים או של מג האוויר באתריהם שלהם. בוני אתרים כאלה יכולים לשכור אתכם או

לחלופין, אם אתם בוני אתרים בעצמכם, קוד זהה באתר הורדפרס שלכם יכול להעניק לכם כלים לחת שירות מוצלח ואיכותי ללקוחות שלכם.

אף על פי שמדובר בקוד פשוט יחסית, כמתכנתים בתחילת דרככם תיתקלו בקשישים. אין צורך להיבהל או לש��ע ברחמים עצמים. התקЛОת, חיפוש התקЛОת והבנה שמדובר בתקלות פשוטות שהייתם צריכים לראות קודם – זהו שכר הלימוד שצריך לשלם.

#### רעיונות להמשך הפיתוח:

1. להציג יותר מתמונה אחת.
2. להוסיף אнимציה של טעינה בזמן שמחכים לתשובות מהשרת.
3. לבנות קROLSלה של תМОנות (אפשר לחפש ברשת ולהבין איך בונים קROLSלה נזאת).

פרק 4

# אפקט אוליה-ჰציית ריאקט לבדיקה משתמש בGITLab



# אפליקציה ציירית רקט לבדיקת משתמש בטיחות

## הגדרת הצורך

לפחות בикש אפליקציית ריאקט עבור המגייסים של החברה, שבה הוא מזין שם משתמש ומתקבל מידע על מספר הריפוזיטורי הפתוחים של המשתמש וגם רשימה מסודרת שלהם ומתי הם נוצרו. המגייסים של החברה, שאינם מכירים היטב את גיטהאב ומתכונים לגלוש בה, יכולים להשתמש במכשיר שהוא לא של גיטהאב על מנת לקבל מידע מסודר על התרומה של המשתמש גם בלי לנoot במכשירים של גיטהאב.

## הבנת האפליקציה

על מנת לקבל מידע מגיטהאב אפשר להשתמש ב- API (Application Programming Interface) שלו. מדובר בדרך להתמכשות עם מאגרי מידע על מנת להציג מהם נתונים.

לGITAEVB יש API פתוח ו- API שצורך להזדהות כדי לקבל ממנו את המידע. למרבה השמחה, במקרה שלנו ה- API אינו דורש רישום. למשל, כניסה אל <https://api.github.com/users/barzik/repos> הריפוזיטורי הפתוחים של המשתמש. אתם מוזמנים ויכולים להיכנס לכתובת כדי לראות את המידע.

כניסה אל <https://api.github.com/users/barzik/gists>

תחזר לנו גם מידע מסודר על הגיסטים של המשתמש. גיסט הוא בעצםקובץ טקסט קצר שנכל להשתמש בו ליצור ולשמור, ללא צורך ברישום, מה שאומר שאפשר להסתפק באפליקציה סטטית בלבד.

شرطוט נאיבי של זרימת המידע של האפליקציה ייראה כך:



כלומר, יהיה לנו אפליקציה סטטית, שפרויקט מאד מתאימה עבורה. היא תכיל services שיקראו לגיטהאב ויציגו את התוצאה בפני הלambda.

## תכנון

אחרי שהבנו את הדרישות, נחשב מעט על התכנון.  
בפני הלambda תוכג תיבת חיפוי שבה הוא יצרך להזין את שם המשתמש, למשל barzik.  
הזהנה והליך יעבירו את שם המשתמש לפונקציה, שאנו קוראים לה service. היא קוראת לגיטהאב בכתובת:

```
https://api.github.com/users/\${user}/repos
```

user מייצג את שם המשתמש בגיטהאב. התוצאות יוחזרו אל האפליקציה ואז יצרנו להוות מוגנות בפני הלambda בפורמט טבלי.

הקומפוננטות הריאקטיות יהיו:

1. טופס חיפוי.
2. קומפוננטת רשימה.
3. קומפוננטה של שורה בראשימה.
4. הקונטינר שיחזיק את הכלול, יכיל, עם סטייט, את התוצאות משנה המוקמות ויעדכן את קומפוננטת הרשימה.

## יצירת התשתית – create react app

על create react app למדנו בספר "לימוד ריאקט בעברית". מדובר באפליקציה תשתית (bootstrapper) נוחה ביותר להתחלה של פרויקט ריאקט, שכבר מכילה את כל מה שאנו צריכים. נגיע לתיקיות האב שבה נרצה להקליד את האפליקציה שלנו ונקליד:

```
npx create-react-app github-user-history-viewer
```

inicrs לתיקייה ונלחץ על npm start. האפליקציה תעלה מיד. ענשוו הזמן לשנות. נפתח את ה-IDE שלנו ונתחיל לעבוד.

## יצירת הקונטינר

אפליקציה ריאקט נתענת ב-.js.App, אבל עדיף לגעת בקובץ זהה כמו שפותות. ניצור תיקייה שנקראת components ובה ניצור קובץ בשם Container.jsx. נכניס בו תוכן דמה:

```
const Container = () => {
  return (
    <div>
      <h1>האפליקציה שלך</h1>
    </div>
  )
}

export default Container;
```

אחר לכך ניגש ל-.js.App, המלא בתוכן דמה של ריאקט שנוצר על ידי Create React App ונהליף את התוכן 7:

```

import './App.css';
import Container from './components/Container';

function App() {
  return (
    <div className="App">
      <Container />
    </div>
  );
}

export default App;

```

זהו השינוי האחרון שנצרך לעשות ב-`App`. מכשיו כל השינויים והתוספות יהיו בקונטינר.

אם נפתח את הדף ונכון אותו ל-`localhost`, נראה שהוא מציג את "האפליקציה" שלי". אם זה לא מוצג או שיש תקלה, עשיתם משהו לא נכון וכדאי לחזור בחזרה ולתken את הדברים.

## יצירת טופס החיפוש

ניצור קומפוננטה נוספת שתכיל את שדה החיפוש. היא בעצם תקבל את הקלט מהמשתמש ותשמש בו כדי לתרשא את ה-`API` של אתר גיטהאב. אנו נשתמש בטופס סטנדרטי של HTML, קלומר `form` וכפתור `submit`. בקומפוננטה אנו יכולים ליצור פונקציה שתשתלט על אירוע `submit`, תיקח את המידע ותעביר אותו להלה בקולבך. הפלט של הקומפוננטה יהיה פשוט קולבך שיכיל בתוכו את המילה שהלקוח חיפש. מי שיازין לקולבך יקבל את המידע:

```

const Input = (props) => {
  const handleSubmit = (event) => {
    event.preventDefault();
    const text = event.target.text.value;
    props.handleSubmit(text);
  };
}

```

```

return (
  <form onSubmit={handleSubmit}>
    <input
      style={{ width: '300px' }}
      name="text"
      type="text"
      placeholder="Please type user name in GitHub -
i.e. barzik"
    />
    <button type="submit">Submit</button>
  </form>);
}

export default Input;

```

השורה המעניינת פה היא:

```
props.handleSubmit(text);
```

היא קוראת לـ``handleSubmit`` שמעבירים לה בـ``props`

בואו נבדוק את הקומפוננטה. נכנס אותה בـ``asx.Container`` ונעביר לה `handleSubmit` עם `console.log`, רק כדי לראות מהה עובד:

```

import Input from "./Input";

const Container = () => {

  const handleSubmit = (user) => {
    console.log(user)
  }
  return (
    <div data-testid="general-repo-container">
      <h2>טופס חיפוש שם משתמש בגיטהאב</h2>
      <Input handleSubmit={handleSubmit} />

    </div>
  )
}

```

```

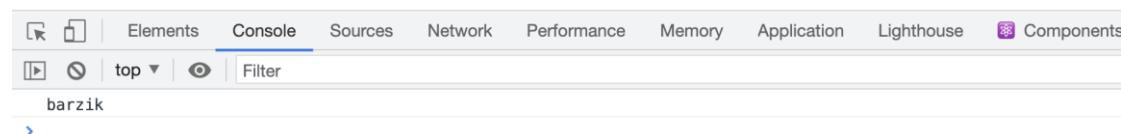
    )
}

export default Container;

```

התוצאה שאנו רוצים לקבל היא צזו – נקליד מחרוזת טקסט בקומפוננטה, נלחץ על הכפתור (או על אנטר) ונקלט בקונסולה את מה שהקלדנו. ככה נודע שההלווק אכן עובד כצפוי.

### טופס חיפוש שם משתמש בגיטהאב



## יצירת ה-`service`

אנו יכולים להשתמש בספריות שונות על מנת ליצור קשר עם API, אך במקרה הזה `fetch` בסיסי יספק. מדובר בעוליה אסינכרונית, אז נשתמש ב-`async/await`.

בספר "למוד ריאקט בעברית" יש הסבר מكيف על `service`. מדובר בפונקציה ג'אוوهסקריפט שאינה קומפוננטה. היא מקבלת קלט, יוצרת קשר עם API באמצעות `fetch` ומחזירה את המידע. כיוון שמדובר בעוליה שלוקחת זמן (לשנות בקשה לשרת ולקבל מידע) אנו משתמשים בפונקציה אסינכרונית.

יצור תיוקה בשם `services` וקובץ בשם `repoService.js` שיכיל את הקוד זהה:

```

const repoService = async (user) => {
  const url = `https://api.github.com/users/${user}/repos`;

```

```

const response = await fetch(url);
const repos = await response.json();
return repos;
};

export default repoService;

```

מדובר בקוד ממש פשוט. אנו בונים את ה-`url` לפי הקלט של המשתמשים, שולחים אותו באמצעות `fetch` לגיטהאב ומחזירים את ה-JSON.

נבדוק אותו באמצעות חיבור ל-`Container`. במקום להציג את מה שנשלח מהמשתמש בקונסולה, נשלח את זה ל-`service` ואז נציג את התוצאה בקונסולה. כך נוכל לראות שהכל עובד:

```

import Input from './Input';
import repoService from './services/repoService';

const Container = () => {
  const handleSubmit = async (user) => {
    const reposListFromServer = await repoService(user);
    console.log(reposListFromServer);
  }
  return (
    <div data-testid="general-repo-container">
      <h2>טופס חיפוש שם משתמש בגיטהאב</h2>
      <Input handleSubmit={handleSubmit} />
    </div>
  )
}

export default Container;

```

בגدول, מה שהוספנו זה `import` ל-`repoService` ואת הקוד הזה:

```
const reposListFromServer = await repoService(user);
```

```
console.log(reposListFromServer);
```

אם הכל עובד, ברגע שנלחץ על הכפתור של השילוח נראה בקונסולה את התוצאות –  
מערך של אובייקטים. כל אובייקט מייצג ריפוזיטורי עם המון מידע עליון:

### טופס חיפוש שם משתמש בגיטראב

The screenshot shows the GitHub search interface with the query 'barzik' entered in the search bar. Below the search bar, there are tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Lighthouse, Components, and Profiler. The 'Console' tab is selected. The results show 30 repositories, each with a small preview icon, the repository name, and its full name. The repositories listed include 'blockersaas', 'branch-name-lint', 'careful', 'ci-example', 'cywp', 'digital-fingerprint', 'drupal-jasmine-integration', 'eslint-plugin-angular-component-api', 'example-nodejs-clock-heb', 'expanding\_box', 'git-last-commit', and 'github-user-history-viewer', among others.

```
▼ (30) [{}]
  ▶ 0: {id: 92423060, node_id: 'MDExO1JlcG9zaXRvcnk5MjQyMzA2MA==', name: 'blockersaas', full_name: 'barzik/blockersaas', private: false}
  ▶ 1: {id: 106296638, node_id: 'MDExO1JlcG9zaXRvcnkxMDYyOTY2Mzg=', name: 'branch-name-lint', full_name: 'barzik/branch-name-lint', private: false}
  ▶ 2: {id: 95448395, node_id: 'MDExO1JlcG9zaXRvcnk5NTQ0ODM5NQ==', name: 'careful', full_name: 'barzik/careful', private: false, ...}
  ▶ 3: {id: 163529817, node_id: 'MDExO1JlcG9zaXRvcnkxNjM1Mjk4MTc=', name: 'ci-example', full_name: 'barzik/ci-example', private: false}
  ▶ 4: {id: 359821907, node_id: 'MDExO1JlcG9zaXRvcnkzNTk4MjE5MDc=', name: 'cywp', full_name: 'barzik/cywp', private: false, ...}
  ▶ 5: {id: 93934770, node_id: 'MDExO1JlcG9zaXRvcnk5MzknDc3MA==', name: 'digital-fingerprint', full_name: 'barzik/digital-fingerprint'}
  ▶ 6: {id: 13367791, node_id: 'MDExO1JlcG9zaXRvcnkxMzM2Nzc5MQ==', name: 'drupal-jasmine-integration', full_name: 'barzik/drupal-jasmine-integration'}
  ▶ 7: {id: 107850114, node_id: 'MDExO1JlcG9zaXRvcnkxMDc4NTAxMTQ=', name: 'eslint-plugin-angular-component-api', full_name: 'barzik/eslint-plugin-angular-component-api'}
  ▶ 8: {id: 24790393, node_id: 'MDExO1JlcG9zaXRvcnkxNDc5MDM5Mw==', name: 'example-nodejs-clock-heb', full_name: 'barzik/example-nodejs-clock-heb'}
  ▶ 9: {id: 10913073, node_id: 'MDExO1JlcG9zaXRvcnkxMDkxMzA3Mw==', name: 'expanding_box', full_name: 'barzik/expanding_box', private: false}
  ▶ 10: {id: 65070653, node_id: 'MDExO1JlcG9zaXRvcnk2NTA3MDY1Mw==', name: 'git-last-commit', full_name: 'barzik/git-last-commit', private: false}
  ▶ 11: {id: 407477281, node_id: 'MDExO1JlcG9zaXRvcnk0MDc0NzcyODE=', name: 'github-user-history-viewer', full_name: 'barzik/github-user-history-viewer'}
  ▶ 12: {id: 58868830, node_id: 'MDExO1JlcG9zaXRvcnk10Da20DazMA==', name: 'orunt-css-url-embed', full_name: 'barzik/orunt-css-url-embed'}
```

עכשוינו אנו צריכים לגשת לבניית התצוגה.

## קומפוננטת List

אנחנו צריכים להציג את הרשימה שלנו, ואת זה עושים באמצעות קומפוננטת `List`. זהה קומפוננטה פשוטה שהקלט שלה הוא בעצם מערך האובייקטים שאנו מקבלים מגיטהאב. אם נסתכל על איבר במערך, נראה שהוא מכיל המון מידע, שאות רובו אנחנו לא צריכים, אבל זה הקולט של הקומפוננטה. היא מקבלת מערך ומוציאה רשימה.

נוהג לפצל קומפוננטת `List` ל-`List` כללי ול-`Item` קטן יותר. ב-`List` יש לו לפחות שםعبירה כל איבר במערך ל-`Item`, וה-`Item` מציג אותו לפי הקריטריונים שאנו צריכים.

קומפוננטת `Item` תיראה כך:

```
const Item = (props) => {
  return (
    <div>
      <h2>{props.repo.name}</h2>
      <small>{new
Date(Date.parse(props.repo.created_at)).toLocaleDateString()}<
/small>
      <p>{props.repo.description}</p>
      <a rel="noreferrer" href={props.repo.html_url}
target="_blank">{props.repo.html_url}</a>
    </div>
  )
}

export default Item;
```

הקלט של הקומפוננטה הוא `repo` והפלט שלה הוא אסז שמנדר את ה-`repo` – זה אובייקט גדול מאוד שאנו צריכים רק כמה חלקים ממנו. מבצעים פְּרָסּוֹר (parsing) של התאריך מ-`timestamp` אל תאריך בפורט קרייא.

מי שמכיל את ה-Item זה ה-List:

```
import Item from './Item';
```

```
const List = import Item from './Item';

const List = (props) => {
  let rows = [];
  props.repos.map((repo, index) => rows.push(<Item
key={index} repo={repo} />))
  return (
    <div>
      <div>
        ריפורטורייז פתוחים {rows.length} ללקוח יש.
      </div>
      <div className="list-group">
        {rows}
      </div>
    </div>
  )
}

export default List;
```

הקלט של האפליקציה הוא repos, מערך. נעשה map על ה-repos, שהוא מערך של אובייקטים וכל אובייקט בו מרונדר כ-Item.

זה החלק התציגתי. עכשו אפשר לבדוק את הכל ולהעביר לחלק התציגתי את המידע האמתי. חוזרים ל-Container.

## לחבר את הכל יחד

از יש לנו קומפוננטה שמקבלת קלט מהמשתמש, כלומר קוראת ל-service שמקבל מידע, ויש לנו קומפוננטה שמציגה את המידע הזה. אין קישורים בין ה-items לבין קומפוננטת List שמציגה את המידע? במקרה זהה נצורך סטייט. הסטייט יאותחל

כמערך ריק, אך מי שייכלס אותו יהיה ה-`service`. מי שייחובר אליו יהיה ה-`List`. אם הסטייט ישתנה, ה-`List` ירונדר מחדש אוטומטית.

כרגע, החיבור הסופי בין הקומפוננטות הוא המורכב ביותר. הנה הקוד המלא, ואני אסביר אותו שורה אחר שורה:

```
import { useState } from 'react';
import Input from "./Input";
import List from "./List";
import repoService from '../../services/repoService';

const Container = () => {
  const [reposList, setRepos] = useState([]);
  const handleSubmit = async (user) => {
    const reposListFromServer = await repoService(user);
    setRepos(reposListFromServer);
  }
  return (
    <div data-testid="general-repo-container">
      <טופס הייפוש שם משתמש בGITLAB>
      <Input handleSubmit={handleSubmit} />
      <List repos={reposList} />
    </div>
  )
}

export default Container;
```

נתחיל ב-`import`. אנו קוראים פה לכל הקומפוננטות ול-`service` וגם להוק של `:useState`.

```
import { useState } from 'react';
import Input from "./Input";
import List from "./List";
import repoService from '../../services/repoService';
```

הגדרת הסטייט נעשית עם השורה זו וגם אתחולו כמערך ריק. יש לנו בעצם את הסטייט עצמו, reposlist, ואת הפונקציה שמעדכנת אותו, setRepos:

```
const [reposList, setRepos] = useState([]);
```

את handleSubmit כתבנו כבר; זהה הฟונקציה שמחוברת לkomponennet Input. מה חדש פה הוא השורה:

```
setRepos(reposListFromServer);
```

שמעדכנת את הסטייט בכל פעם שמתבצע קלט עם תוצאות ה-service.

```
const handleSubmit = async (user) => {
  const reposListFromServer = await repoService(user);
  setRepos(reposListFromServer);
}
```

השורה החדשה היחידה שיש היא זו:

```
<List repos={reposList} />
```

כאן פשוט מעבירים ל-List את הסטייט, שמתעדכן בפונקציית השליחה.

אם הכל תקין, כאשר נזין שם משתמש בגייטהאב, נקבל תוצאות:

**טופס חיפוש שם משתמש בGITהאב**

<input type="text" value="barzik"/>	<input type="button" value="Submit"/>
-------------------------------------	---------------------------------------

ללקוח יש 30 ריפורטוריים פתוחים.

**blockersaaS**

5/25/2017

SAAS Blocker

<https://github.com/barzik/blockersaaS>**branch-name-lint**

10/9/2017

Lint your branch names

<https://github.com/barzik/branch-name-lint>**careful**

6/26/2017

**סיכום**

הראינו כאן אפליקציית ריאקט סטטיית פשוטה. אף שהיא סטטית, היא בעלת ערך ולא חסרות אפליקציות כאלה, גם בטלפונים ניידים. ריאקט היא פלטפורמה נחרת כיוון שי create react app חוסף מאייתנו הרבה כאבי ראש בתקנות, אך למורות הפשטות של האפליקציה ושל הסביבה – למכננים בתחילת דרכם המשימה זו יכולה להיראות עצומה וקשה. אם הייתם שגיאות ותקלות – הן חלק מהתיכון הלימוד. כתיבת קוד ראשוני אמורה להיות קשה ולכלול טעויות וביעות וחיפוש בגוגל של בעיות שמתרור בשן פשוטות ממש. זה שכר הלימוד שלכם על הידע. מתכנת מנוסה יכול לבנות את האפליקציה זו בחצי שעה. למכננת בתחילת דרכו זה יכול לקחת הרבה הרבה זמן – אבל זו ממשות הניסיון. אם כתבתם, הצעתם והגעתם לשלב הזה גם לאחר שעות רבות וטעויות אינספור – בפעם הבאה זה ייקח קצת פחות זמן.

**رجوعات للمشكل הפיתוח:**

1. הודיעת שגיאה מסודרת אם לא הוקלד שם משתמש.
2. חיפוש בGESUT והציג המידע.
3. עיצוב האפליקציה עם UI material.
4. כתיבת טסטים אוטומטיים לקומפוננטות.

פרק 5

# אפליהציה מודרנית לניהול מעקבים על אתרי אינטרנט



# אפליקציה מורכבת לניהול מעקבים על אתרי אינטרנט

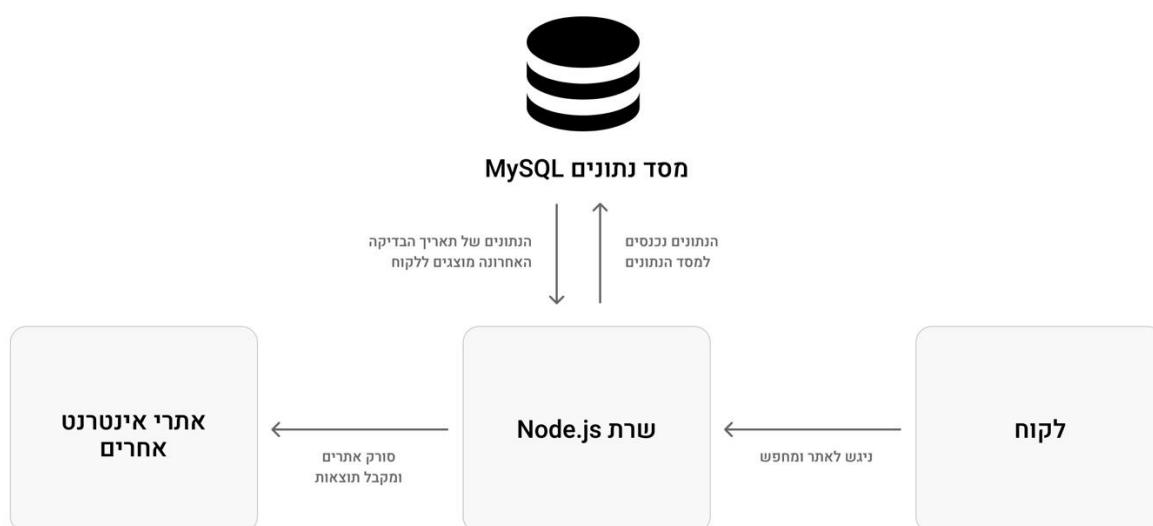
## הגדרת הערך

לקוח שמנהל מסחר בורסאי בิกש מכם מערכת הטראה המיועדת לאתרי אינטרנט. הוא נכנס מדי בוקר לעשרה אתרים (ואפלו יותר) וסורק ידנית שם של חברה שהוא עובד אליה. הוא רוצה מערכת שכאשר יוכל לאליה, היא תיכנס לאתרי אינטרנט (מתוך רשימה מוגדרת מראש כרגע), תחפש את הטקסט שהוא יגדר לה בזמן אמת ותכניס את התוצאות למAGER נתונים שהוא יוכל לקרוא.

## הבנת האפליקציה

אפשר לראות שלא מדובר באפליקציה סטטית או בפורונט בלבד. יש לנו כאן עבודה עם שרת, שיבצע את הסריקה לפי בקשת הלוקוח, יכניס את התוצאות למסד נתונים וגם יוציא אותם משם כדי להציגם ללוקוח.

שרטוט נאיבי של האפליקציה נראה כך:



אם נחשוב מעט על מסכי האפליקציה, נבין שיש לנו כרגע דף אחד, שבו הלקוח מכניס מילה וЛОוחץ על "חיפוש", מתבצע חיפוש ואז הלקוח רואה את תוצאות החיפושים האחרונים. זה הכל. אולי בעתיד יהיו עוד דפים – למשל מסך שבו הלקוח יוכל לשנות את רשימת האתרים שנסרקיהם (כרגע הרשימה היא סטטית), או מסך נוסף עם תוצאות עבר, אבל זה מה שאנו צריכים כרגע.

## תיכנון

אחריו שהבנו את הצרכים יותר לעומק, הגיע הזמן לתכנון. נחלק את התכנון לשלוות החלקים של היישויות השונות: מסד הנתונים, דף ה-HTML שהלקוח רואה, שהוא הפונט אנדר, והשרת עצמו, שהוא חלק המורכב יחסית.

מבחןת מסד הנתונים, אנחנו צריכים כרגע רק טבלה אחת בשם scans שמכילה:

1. IP מהה בשם pi\_scan.
2. את הזמן שבו האתר נסרק (timestamp).
3. את כתובת האתר (מחוROT טקסט VARCHAR).
4. את המילה שנסקרה (מחוROT טקסט VARCHAR).
5. את השאלה אם המילה נמצאה (בוליאני שנממש באמצעות enum של 1/0).

מבחןת ה-HTML אנחנו צריכים:

1. Form פשוט עם מילת חיפוש וכפתור "חפש". הטופס ישלח את הנתונים ב-POST.
2. טבלה שמציגה את המידע בנוגע לחיפוש.

את ה-HTML ניצור עם אקספרס, שעליה למדנו בספר "לימוד JavaScript בעברית".

מבחןת שרת ה-`js.Node` אנחנו צריכים:

1. מנגנון שיעודע להגיש את ה-HTML ב-GET. ה-HTML יהיה ריק ועם טופס פשוט בלבד.
2. מנגנון שיעודע לקבל POST מהטופס ואז:
  - א. לבצע את הסריקה על רשימת אתרים מוגדרת מראש.

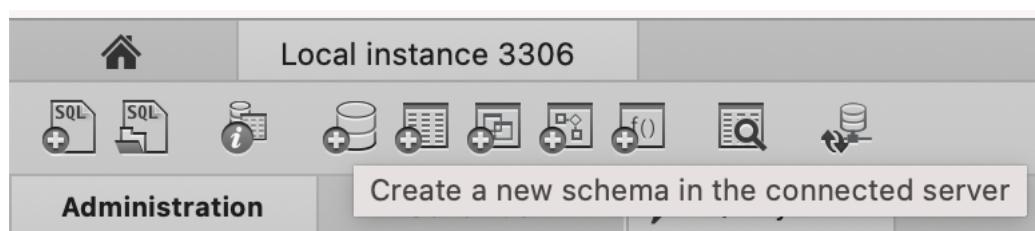
- ב. לכתוב את הנתונים שהגיעו מהסריקה למסד הנתונים.
- ג. לקרוא את כל הנתונים ממסד הנתונים ולהחזיר אותם עם HTML בטבלה מלאה.
- כלומר, יש לנו כאן שתי נקודות כניסה – GET להצגה של טופס "נקי" ו-POST לקבלת מידע מהטופס.

## סביבה העבודה והפיתוח

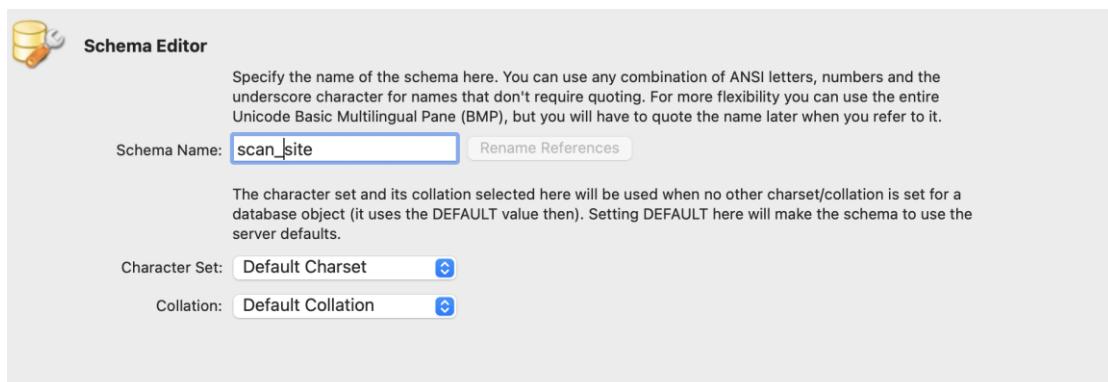
נתחיל בהקמת שתי הסביבות העיקריות שלנו במחשב המקומי. הראשונה היא שרת MySQL וממסד נתונים עם טבלה אחת בפורמט שקבענו. השנייה היא שרת Node.js עם אקספרס וקובץ `gitignore` ו-`eslint` בסיסי.

### הקמת מסד נתונים

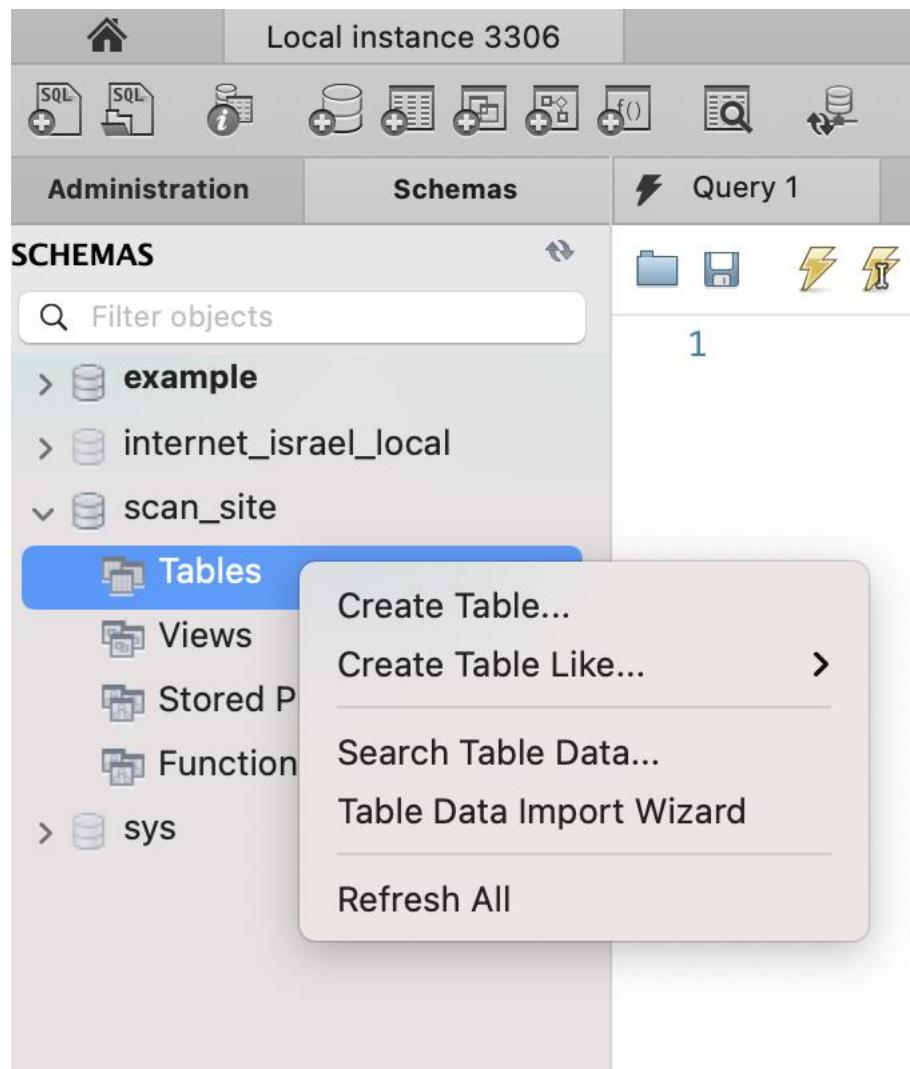
נפתח את MySQL Workbench וניצור מסד נתונים חדש:



נקרא לו `:scan_site`



אחריו שיצרנו את מסד הנתונים, נבחר בו, נלחץ על Table וنبחר ב>Create table



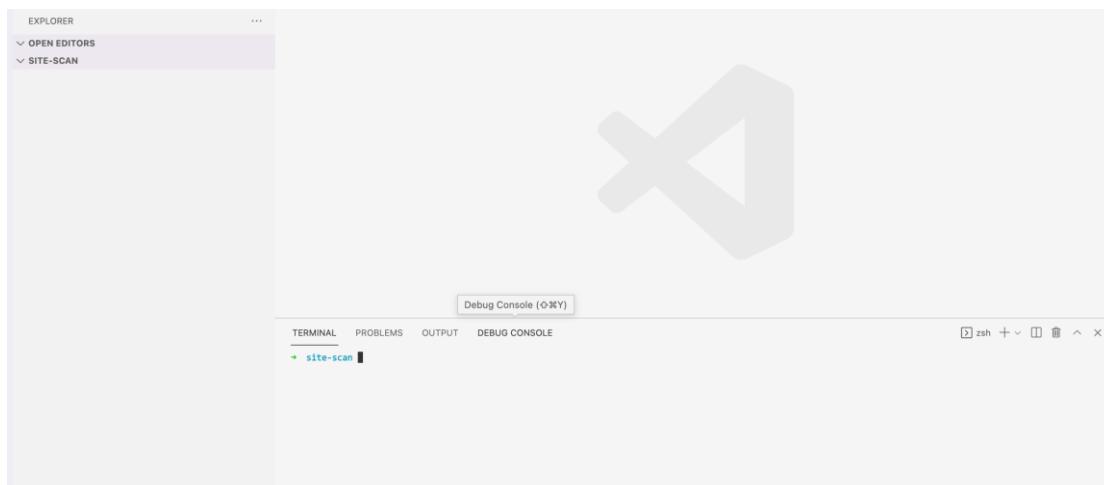
נכיס בצדקה מרכזת את הנתונים שכבר הגדרנו בשלב התכנון:

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
scan_id	BIGINT	✓	✓	✓				✓		
timestamp	BIGINT			✓						
url	VARCHAR(45)									NULL
keyword	VARCHAR(45)									NULL
found	ENUM('1', '0')									NULL
<click to edit>										

אחריו שסיימנו, נבחן את הטבלה ונראה שהכל תקין. אפשר להתקדם לשלב הבא.

## יצירת התשתיות של Node.js

ניצור תקייה במחשב שלנו ונפתח את ה-`Visual Studio Code` שמכיל אותה. מהנקודה הזו נעבד אך ורק עם ה-`IDE`, כולל עבודה בטרמינל:



ניצור פרויקט גיט חדש עם `git init` ופרויקט `Node.js` קיים עם `npm init`:

```

TERMINAL      PROBLEMS      OUTPUT      DEBUG CONSOLE
_____
→ site-scan git init
Initialized empty Git repository in /Users/barzik/local/hebdevbook-examples/site-scan/.git/
→ site-scan git:(main) npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (site-scan) █

```

נשתמש ב-.gitignore מיוחד ל-.js שנמצא באתר הרשמי של גיטהאב, פה:

<https://github.com/github/gitignore/blob/master/Node.gitignore>

ניצור קובץ בשם .gitignore ונכנס אליו את הטקסט הזה – מדובר בסוגי קבצים רבים שמתעלמים מהם ולא ניכנס להסביר על כל אחד מהם. קחו את הקובץ כפשותו:

```

# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
lerna-debug.log*
.pnpm-debug.log*

# Diagnostic reports (https://nodejs.org/api/report.html)
report.[0-9]*.[0-9]*.[0-9]*.[0-9]*.json

# Runtime data
pids
*.pid
*.seed
*.pid.lock

# Directory for instrumented libs generated by
jscoverage/JSCover

```

```
lib-cov

# Coverage directory used by tools like istanbul
coverage
*.lcov

# nyc test coverage
.nyc_output

# Grunt intermediate storage (https://gruntjs.com/creating-plugins#storing-task-files)
.grunt

# Bower dependency directory (https://bower.io/)
bower_components

# node-waf configuration
.lock-wscript

# Compiled binary addons (https://nodejs.org/api/addons.html)
build/Release

# Dependency directories
node_modules/
jspm_packages/

# Snowpack dependency directory (https://snowpack.dev/)
web_modules/

# TypeScript cache
*.tsbuildinfo

# Optional npm cache directory
.npm

# Optional eslint cache
.eslintcache
```

```
# Microbundle cache
.rpt2_cache/
.rts2_cache_cjs/
.rts2_cache_es/
.rts2_cache_umd/

# Optional REPL history
.node_repl_history

# Output of 'npm pack'
*.tgz

# Yarn Integrity file
.yarn-integrity

# dotenv environment variables file
.env
.env.test
.env.production

# parcel-bundler cache (https://parceljs.org/)
.cache
.parcel-cache

# Next.js build output
.next
out

# Nuxt.js build / generate output
.nuxt
dist

# Gatsby files
.cache/
# Comment in the public line in if your project uses Gatsby and not Next.js
# https://nextjs.org/blog/next-9-1#public-directory-support
# public
```

```
# vuepress build output
.vuepress/dist

# Serverless directories
.serverless/

# FuseBox cache
.fusebox/

# DynamoDB Local files
.dynamodb/

# TernJS port file
.tern-port

# Stores VSCode versions used for testing VSCode extensions
.vscode-test

# yarn v2
.yarn/cache
.yarn/unplugged
.yarn/build-state.yml
.yarn/install-state.gz
.pnp.*
```

השלב הבא הוא להתקין express. נעשה את זה עם פקודה אחור בשם express-generator --view=ejs. שטיוצר שרת אקספרס מוכן. נרץ אותו מהתיקייה שלנו. ייווצרו לנו תיקיות וקוד וכן dependencies ב-package.json.

```
destination is not empty, continue? [y/N] y

create : public/
create : public/javascripts/
create : public/images/
create : public/stylesheets/
create : public/stylesheets/style.css
create : routes/
create : routes/index.js
create : routes/users.js
create : views/
create : views/error.jade
create : views/index.jade
create : views/layout.jade
create : app.js
create : package.json
create : bin/
create : bin/www

install dependencies:
$ npm install

run the app:
$ DEBUG=site-scan:* npm start

→ site-scan git:(main) ✘
```

בסוף ההתקנה, כפי שאפשר לראות, נדרש לבצע `npm install`, וזה מה שנעשה.

אחר מכן, כדי להפעיל את האתר, נקליד:

```
DEBUG=site-scan:* npm start
```

ונקבל חיוי שהאתר פועל בפורט 3000.

השלב הבא הוא להכנס אל `localhost:3000` בדפדפן ולראות אם הכל עובד. זה אמרור לעובוד:



# Express

Welcome to Express

## התקנת eslint

חלק שימושי מaicות הפיתוח שלנו הוא התקנת לינט שיעבוד לוקלית. נעשה את זה באמצעות הפקודה `eslint --init` ומענה על השאלות בהתאם לפרויקט שלנו, בדיק [לפי התמונה הבאה](#):

```
→ site-scan git:(main) ✘ npx eslint --init
✓ How would you like to use ESLint? · style
✓ What type of modules does your project use? · commonjs
✓ Which framework does your project use? · none
✓ Does your project use TypeScript? · No / Yes
✓ Where does your code run? · node
✓ How would you like to define a style for your project? · guide
✓ Which style guide do you want to follow? · airbnb
✓ What format do you want your config file to be in? · JavaScript
Checking peerDependencies of eslint-config-airbnb-base@latest
Local ESLint installation not found.
The config that you've selected requires the following dependencies:

eslint-config-airbnb-base@latest eslint@^5.16.0 || ^6.8.0 || ^7.2.0 eslint-plugin-import@^2.22.1
✓ Would you like to install them now with npm? · No / Yes
Installing eslint-config-airbnb-base@latest, eslint@^5.16.0 || ^6.8.0 || ^7.2.0, eslint-plugin-import@^2.22.1
```

נכיס אל ה-`package.json` שלנו את הסкриיפטים הבאים:

```
"lint": "npx eslint '**/*.{js}' --ignore-pattern
node_modules/",
"lint:fix": "npx eslint '**/*.{js}' --ignore-pattern
node_modules/ --fix"
```

הרצה שלהם תבצע בדיקת איקות קוד אוטומטית.

## התקנת nodemon

כיוון שאנו עוסקים באקספרס, בכל שינוי שאינו ב-`js` אנו נדרשים לבצע הפליה של השרת ואז העלה שלו. עם `nodemon` לא חיבבים לעשות את זה – זה קורה אוטומטית, ואני ממליץ להשתמש בו. איך עושים את זה? מתכוונים אותו באמצעות:

```
npm install --save-dev nodemon
```

מוסיפים ב-`json` פקודה חדשה מתוך פקודת `start`. במקום `node` כתובים `:nodemon`

```
"start:dev": "nodemon ./bin/www",
```

זה הכל. משתמשים בפקודת `start:dev` במקום `run start` במהלך הפיתוח. מעכשיו, בכל פעם שתראו כאן "להפעיל את השרת ולפתחו אותו שוב", לא תצטרכו לעשות את זה, כיון שבכל שינוי של קובץ, `nodemon` יעשה את זה עבורכם ותוכלו לראות את זה בשורת הפקודה.

## בנית התצוגה

### עיצוב הדף הראשוני – של טופס החיפוש

inicnes לתקינות `views` ונחפש את `index.ejs`. זה הטמפלט שלנו. אם נשנה ונרפרש את השרת, נראה שהוא השתנה.

**שימוש לב:** אם אתם לא מוצאים את קובצי `js` ובמקום זה יש לכם קובצי `ejs`, התחילה מההתחלת והקפידו לכתוב את הפקודה שיצרת את שרת האקספרס: `-npx express-generator --view=ejs`

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
```

```

</head>
<body>
  <h1><%= title %></h1>
  <form method="POST">
    <input id="scantext" required name="scantext"
type="text" placeholder="Insert text for scan" />
    <button type="submit">Scan</button>
  </form>
</body>
</html>

```

אם נרפרש את העמוד, נראה את הטופס:

← → ⌂ ⓘ localhost:3000

# Express

אם נכניס טקסט ולחץ על submit, נקבל הודעה שגיאה. מדוע? כי אנו שלוחים מידע ב- POST אך עדין לא הגדרנו באקספרס נתיב שמקבל את המידע זהה.

## ציירת דף הצגת התוצאות

כאמור, דף הצגת התוצאות הוא עבודה עם המידע שהוא מקבלים ב-POST. ניכנס ל- routes/index.js וניקח את הפרמטר scantext שנשלח אליו מהטופס (הגדרנו אותו ב- name וב-`id` של אלמנט `input`):

```

var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function (req, res, next) {
  res.render('index', { title: 'Welcome to the scan site' });
});

/* POST home page. */
router.post('/', function (req, res, next) {
  const scantext = req.body.scantext;
  // This is the mock data
  // TODO: get it from DB
  const scanresults = [
    {
      scan_id: '1',
      timestamp: 'mockTimeStamp',
      url: 'mockUrl',
      keyword: 'mockKeyword',
      found: '1',
    },
  ];
  res.render('index', {
    scanresults: scanresults,
    title: `Search result for: ${scantext}`,
  });
});

module.exports = router;

```

אפשר לראות שמקבלים את הparameter ומעבירים אותו כ-`title` מנת לבדוק את הנתיב החדש. נסגור את התהיליך של אקספרס, נריץ אותו שוב, נכנס טקסט מסוים לשדה החיפוש ונשגר אותו. הפעם, כיוון שיש מי שיקבל את ה-POST, לא נראה הודעה שגיאה אלא את הטקסט שנשלח:

A screenshot of a web browser window. At the top, there are navigation icons (back, forward, refresh) and a address bar containing the text "localhost:3000". Below the address bar, the main content area displays the heading "Search result for: hello" in large bold letters. Underneath the heading are two buttons: "Insert text for scan" (highlighted with a blue border) and "Scan".

← → ⌂ ⓘ localhost:3000

# Search result for: hello

[Insert text for scan](#) [Scan](#)

אבל אנחנו צריכים גם את התוצאות. כיוון שעדין אין לנו אותן, כי לא בנוינו את המנגנון שמבצע את הבדיקות לאתרים ולא את המנגנון שמבצע את הכתיבה והקריאה למסד הנתונים, נכניס מידע דמה. בשפה המקצועית זה נקרא `mock`. המידע הזה, שאינו אמיתי, פשוט יסייע לנו בעיצוב הפירונט אנד, וכך כבר נגדיר אותו ואת הפורמט:

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function (req, res, next) {
  res.render('index', { title: 'Welcome to the scan site' });
});

/* POST home page. */
router.post('/', function (req, res, next) {
  const scantext = req.body.scantext;
  // This is the mock data
  // TODO: get it from DB
  const scanresults = [
    {
      scan_id: '1',
      timestamp: 'mockTimeStamp',
      url: 'mockUrl',
      keyword: 'mockKeyword',
    }
  ];
  res.json(scanresults);
});
```

```

        found: '1',
    },
];
res.render('index', {
    scanresults: scanresults,
    title: `Search result for: ${scantext}`,
});
});

module.exports = router;

```

אם נPILE את האפליקציה ונרים אותה שוב ונרפרש, ולאחר כך נכנס שוב טקסט ונלחץ על שיגור, נראה שדבר לא השתנה – כי אנו צריכים להכניס את ה-`locals` `scanresults` לתוך טבלה מסודרת. אנחנו צריכים לנתח לולה שתיקח את המערך שיש ב-`scanresults` ותמייר אותו לטבלה. ה-`else` שלנו יראה כך:

```

<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel="stylesheet" href="/stylesheets/style.css" />
  </head>
  <body>
    <h1><%= title %></h1>
    <form method="POST">
      <input
        id="scantext"
        name="scantext"
        required
        type="text"
        placeholder="Insert text for scan"
      />
      <button type="submit">Scan</button>
    </form>
    <% if(!locals.scanresults){ %>
      <table border="1">

```

```

<thead>
  <tr>
    <th>ID</th>
    <th>TimeStamp</th>
    <th>URL</th>
    <th>Keyword</th>
    <th>Found (1 if yes, 0 if not)</th>
  </tr>
</thead>
<tbody id="lecturesTableData">
  <% if(scanresults.length){ %> <% for(let i = 0; i <
scanresults.length; i++) { %>
    <tr>
      <td><%=scanresults[i].scan_id%></td>
      <td><%=scanresults[i].timestamp%></td>
      <td><%=scanresults[i].url%></td>
      <td><%=scanresults[i].keyword%></td>
      <td><%=scanresults[i].found%></td>
    </tr>
  <% } %> <% }else{ %>
    <tr>
      <td>No Results found</td>
    </tr>
  <% } %>
</tbody>
</table>
<% } %>
</body>
</html>

```

אפשר לראות שמדובר בג'אוوهסקריפט פשוט שלוקח מערך שבו כל איבר הוא אובייקט ופותח את האובייקט זהה לשורה. כל שורה בטבלה היא איבר במערך. כמו כן, המידע הלא אמיתי שלנו הוא מערך בעל אובייקט אחד, אבל הקוד הזה יעבד גם על מאות אובייקטים סמסודרים במערך:



## Search result for: hello

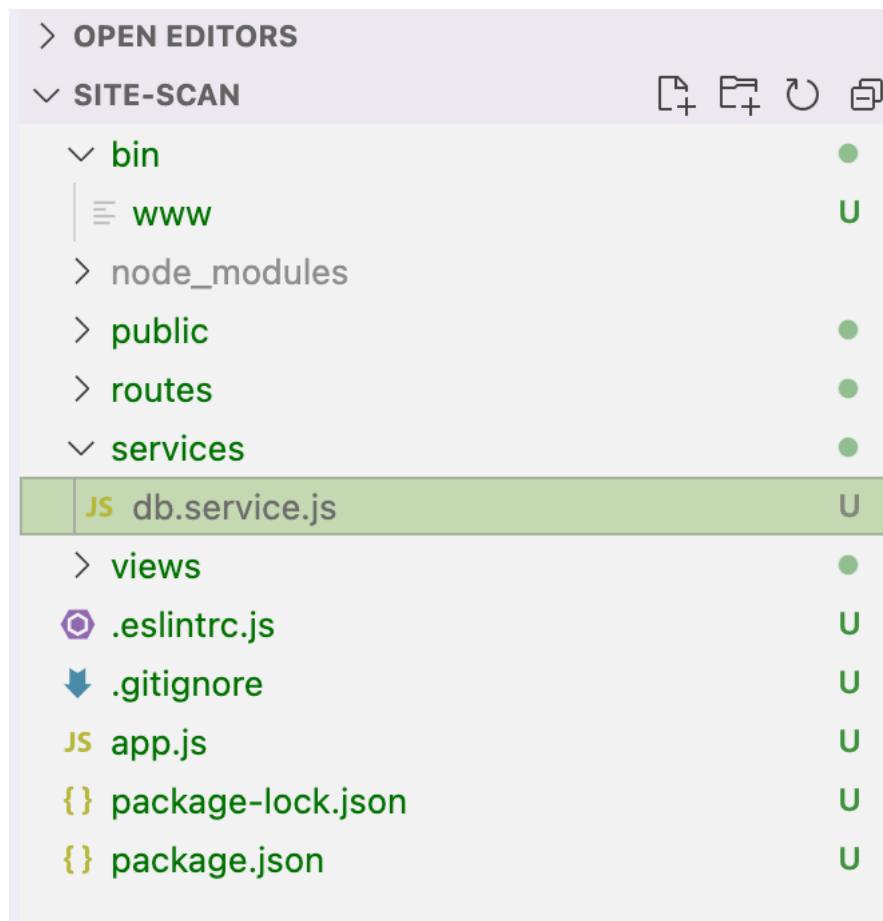
Insert text for scan		Scan		
ID	TimeStamp	URL	Keyword	Found (1 if yes, 0 if not)
1	mockTimeStamp	mockUrl	mockKeyword	1

כל התצוגה מכוערת למדי, אבל כרגע זה לא מעניין אותנו כי אנחנו עדיין בתהליך הבניה. אפשר לומר שסימנו עם הצד הפונקציוני של התצוגה! זה לא רע, אבל אנחנו עדיין צריכים להשלים שני חלקים מרכזיים. החלק הבא שנעבד עליו הוא קרייה וכתיבה אל מסד הנתונים.

## בנית התשתיות של תקשורת עם מסד הנתונים

כדי לבצע את הקרייה והכתיבה אל מסד הנתונים אנחנו צריכים מודול שנקרא `mysq`. למדנו עליו בספר "לימוד Node.js בעברית", וקל מאד להשתמש בו. ממש נטעיק את הקוד שמאחדר לאותם.

ראשית נתקן אותו באמצעות `npm i mysql`. ניצור תיקיה בשם `services` וביה ניצור קובץ בשם `db.service.js`. שם ניצור את הקלאס שיבצע את החיבור ונייצא אותו. כיוון שהמודול עובד עם קולבקים, נגרום לו להשתמש בפרומיס:



החלוקת של הקוד ל-services היא משהו שמקובל מאוד לעשות כאשר יוצרים פעלת של חיבור למסד הנתונים.

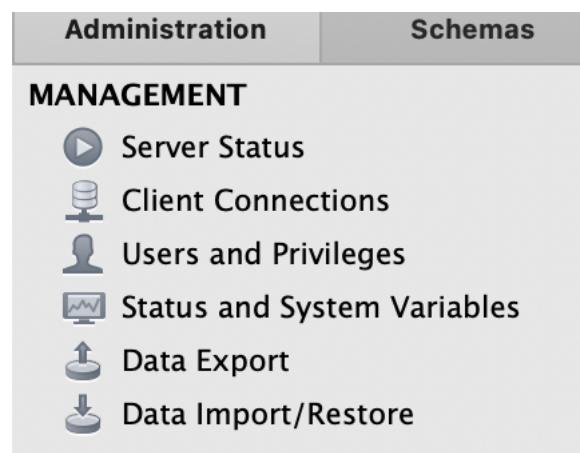
ניצור את הקלאס הבא:

```
const mysql = require('mysql');

class DBConnection {
  constructor() {
    this.pool = mysql.createPool({
      host: localhost,
      user: db_user,
      password: 123456,
      database: scan_site,
    });
  }
}
```

```
module.exports = new DBConnection();
```

מדובר בחיבור פשוט מאוד למסד הנתונים, שכאמור למדנו לעילו בעבר. זהו חיבור ראשוןינו הכלול את שם המשתמש והסיסמה של מסד הנתונים, במקרה זה scan\_site. שם משמש וסיסמה לשימוש יוצרים באמצעות MySQL workbench – לשונית administration והגדרת שם משתמש וסיסמה עם פריבילגיות של קרייה וכתיבה למסד הנתונים :scan\_site



## אבטחת מידע

רגע אחד – האם זה רעיון טוב להכניס את שם המשתמש והסיסמה של MySQL לקוד? נכון, כמובן, שמדובר בקוד שרצה על המחשב שלנו בלבד, אבל בקרוב נרצה לעשות דיפלוי לקוד או להעביר אותו להלאה, אז מסד הנתונים לא יהיה על המחשב שלנו אלא מסד נתונים אמיתי – כלומר זה רעיון ממש- ממש רע.

איך פותרים את העניין הזה? איך משתמשים בסיסמאות בלי להכניס אותן לקוד? התשובה היא - בעזרת משתני סביבה. למשל, במקום לכתוב `npm start`, נקליד:

```
DB_HOST=localhost DB_USER=scanneruser DB_PASS=123456
DB_DATABASE=scan_site npm start
```

ונעשה את זה בכל הרצה. בהתחלה זה מרגינ', אבל מעבר לאבטחה – אחר כך, כשנעשה דיפלוי, זה ממש יקל לנו.

על מנת לעשות את זה נשתמש במודול `dotenv`, וכך נשמר את הסיסמה ושם המשתמש של מודול זהה מאפשר לנו להכניס משתנים בשורת הפקודה ולקבל אותם בקובד שלנו.

נתקין את מודול `dotenv` באמצעות:

```
npm i dotenv
```

ונכניס את שתי השורות הבאות ל-`app.js`, ממש בהתחלה:

```
const dotenv = require('dotenv');
dotenv.config();
```

הקוד של הקלאס שלנו ייראה כך:

```
const mysql = require('mysql');

class DBConnection {
  constructor() {
    this.pool = mysql.createPool({
      host: process.env.DB_HOST,
      user: process.env.DB_USER,
      password: process.env.DB_PASS,
      database: process.env.DB_DATABASE,
    });
  }
}
```

עכשו אפשר להמשיך.

## קריאה ממסד הנתונים

אחרי שיצרנו את הקלאס של החיבור, ניצור מתודה בשם `readscans` שתבצע את הקריאה. בדוקו מנטזיה של מודול `mysql` יש כמה וסברים בהיריים, גם אם לא קראתם את פרק החיבור למסד הנתונים בספר "לימוד Node.js בעברית", וקריאה היא דבר מאד-מאוד פשוט:

```
const mysql = require('mysql');

class DBConnection {
  constructor() {
    this.pool = mysql.createPool({
      host: process.env.DB_HOST,
      user: process.env.DB_USER,
      password: process.env.DB_PASS,
      database: process.env.DB_DATABASE,
    });
  }

  readScans(word) {
    return new Promise((resolve, reject) => {
      const columns = ['scan_id', 'timestamp', 'url',
        'keyword', 'found'];
      this.pool.query(
        'SELECT ?? FROM scans WHERE keyword = ?',
        [columns, word],
        (error, results) => {
          if (error) return reject(error);
          resolve(results);
        },
      );
    });
  }
}

module.exports = new DBConnection();
```

כדי לשים לב שאני עובד כאן עם פרומיסים כדי שאוכל להשתמש ב-`async await` בקוד שלי ולא ב콜בקים. אם הקונספט נראה לכם מוזר ומשונה, כדאי שתשகיעו בلمידה של העניין זהה, כי זו הדרך לפתח כיוון.

על מנת לראות שהקריאה עובדת כמו שצריך נכניס למסד הנתונים שלנו מעט מידע דמה. ניכנס ל-MySQL Workbench וניצור שתי שורות בטבלה באמצעות לחיצה על הכפתור הימני בעכבר על הטבלה ובחירה ב"הוספה", או פשוט באמצעות הקלדת `INSERT` ישירות לקונסולה:

scan_id	timestamp	url	keyword	found
1	33444	https://example.com	example	1
2	334455	https://site.com	example	0

השלב הבא הוא לבדוק את הקלאס שלנו, לראות שהכל מתחבר. נעשה זאת ב-`route` של `index`, היכן שכבר הכנסנו מידע דמה שדומה אחד לאחד למה שהוא אמור לקבל מהקריאה ל-MySQL – אוטם שמות של עמודות ואוטם נתונים.

כדי שנוכל להשתמש ב-`async/await`, נוסיף `async` לפני הfonקציה, פה:

```
router.post('/', async (req, res, next) => {
```

כך הקוד ייראה עם התוספת שלנו. כבר אין מידע דמה, אלא מידע שמניע מסודן הנתונים:

```
var express = require("express");
var router = express.Router();
const dbConnection = require('../services/db.service');

/* GET home page. */
router.get("/", function (req, res, next) {
  res.render("index", { title: "Welcome to the scan site" });
});
```

```

/* POST home page. */
router.post('/', async (req, res, next) => {
  const { scantext } = req.body;
  const scanresults = await dbConnection.readScans(scantext);
  res.render('index', {
    scanresults,
    title: `Search result for: ${scantext}`,
  });
});

module.exports = router;

```

אם הכל תקין, כנסנו את Node.js ונפתח אותו שוב ונחפש את example, נראה שהוא מקבלים את מידע הדמה שהכניסנו למסד הנתוניים:

← → ⌂ ⓘ localhost:3000

## Search result for: example

Insert text for scan		Scan		
ID	TimeStamp	URL	Keyword	Found (1 if yes, 0 if not)
1	33444	https://example.com	example	1
2	334455	https://site.com	example	0

## כתיבה למסד הנתוניים

את הקריאה עשינו, אבל עכשיו יש צורך גם לכתוב. הכתיבה גם היא תישנה באותו קלאס, במתודה שנקראת writeScans. היא משתמשת ב-SET על מנת להכניס שורה, כמתואר בדוקומנצייה של מודול mysql, וגם פה נסתמך על אובייקט שמניע מבחוץ וככתבו:

```
const mysql = require('mysql');
```

```
class DBConnection {
  constructor() {
    this.pool = mysql.createPool({
      host: process.env.DB_HOST,
      user: process.env.DB_USER,
      password: process.env.DB_PASS,
      database: process.env.DB_DATABASE,
    });
  }

  readScans(word) {
    return new Promise((resolve, reject) => {
      const columns = ['scan_id', 'timestamp', 'url',
        'keyword', 'found'];
      this.pool.query(
        'SELECT ?? FROM scans WHERE keyword = ?',
        [columns, word],
        (error, results) => {
          if (error) return reject(error);
          resolve(results);
        },
      );
    });
  }

  writeScans(scan) {
    return new Promise((resolve, reject) => {
      this.pool.query(
        'INSERT INTO scans SET ?',
        [scan],
        (error, results) => {
          if (error) return reject(error);
          resolve(results);
        },
      );
    });
  }
}
```

```

}

module.exports = new DBConnection();

```

על מנת לבדוק שהכל עובד נשתמש באובייקט דמה שנכנים ל-`routes` ב-`POST`. הרי מה שיקרא אחרי שנסרים את הכל הוא סריקה -> הכנסה למסד הנתונים -> קריאה ממסד הנתונים. אז פשוט נדמה את הסריקה ואת התוצאה המתקבלת – רק כדי לראות שהכתביה עובדת:

```

// Demo object - it will come from the scans.
scan = {
  timestamp: 3333,
  url: 'http://mozilla.com',
  keyword: 'example',
  found: '1',
};
await dbConnection.writeScans(scan);

```

הקוד המלא ייראה כך:

```

var express = require("express");
var router = express.Router();
const dbConnection = require('../services/db.service');

/* GET home page. */
router.get("/", function (req, res, next) {
  res.render("index", { title: "Welcome to the scan site" });
});

/* POST home page. */
router.post("/", function (req, res, next) {

  // Demo object - it will come from the scans.
  scan = {
    timestamp: 3333,
    url: 'http://mozilla.com',
    keyword: 'example',
    found: '1',
  }
}

```

```

};

await dbConnection.writeScans(scan);

const { scantext } = req.body;
const scanresults = await dbConnection.readScans(scantext);
res.render('index', {
  scanresults,
  title: `Search result for: ${scantext}`,
});
});

module.exports = router;

```

עכשו נל שעלינו לעשות הוא להפעיל ולהעלות מחדש את האפליקציה שלנו, לープרש ולשלוח שוב `example` בשדה החיפוש. השילחה, שנעשית כאמור ב-POST, תגרום קודם כולם לכתיבה למסד הנתונים של השורה שלנו ואז לקרואיה, כך שגם הכל תקין ולא נקבל שגיאות – נראה שלוש תוצאות באפליקציה, וגם בבדיקה במסד הנתונים נראה שלוש שורות. השורה השלישית היא בדוק כמו האובייקט שלנו:

← → ⌂ ⓘ localhost:3000

## Search result for: example

Insert text for scan		Scan		
ID	TimeStamp	URL	Keyword	Found (1 if yes, 0 if not)
1	33444	https://example.com	example	1
2	334455	https://site.com	example	0
3	3333	http://mozilla.com	example	1

השלמנו את הפעולות מול מסד הנתונים. יש לנו כתיבה ווש לנו קריאה. עכשו נותר לנו רק לעשות את הסריקה עצמה – החלק המורכב יחסית.

## כתיבת הקוד שמבצע סריקה

הסריקה היא החלק האחרון של האפליקציה שלנו. אנחנו צריכים לכתוב את הקוד שסורק אתרים ומאתר בהם את המילה שהמשתמש מזין. התוצאות מוגנות למסד הנתונים וכן הלאה.

ניצור כמה services לשם כך. הראשון הוא service שמבצע קריאה, השני מבצע חיפוש במידע המגיעה מהאתר, והשלישי הוא זה שמאחד את כולם. כל השירותים יהיו אסינכרוניים.

### סריקה

מבצע סריקה של תוכן האתר באמצעות axios, ספרייה פופולרית מאוד שעובדת ב-`Node.js` וקוראת מידע ממש כמו דף-דף. כל מה שצריך לספק לה זה את כתובות הדף, והוא מחזירה את המידע הזה.

נתקין את `axios` באמצעות:

```
npm i axios
```

ניצור בתיקיית `services` את הקובץ `scraper.service.js` ונכנסים אליו את הקוד הבא:

```
const axios = require('axios');

const scraper = async (target) => {
  const result = await axios.get(target, {});
  return result.data;
};

module.exports = scraper;
```

זהו קוד פשוט ביותר שנלקח מהדוגמה שיש בדוקומנטציה של `axios`. אני חושב שאין צורך להסביר עליו יותר מדי. בגודל, לב `service` הוא:

```
axios.get(target, {});
```

### חישוף במידע המגיע מהסריקה

המידע המתקבל מ-axios הוא HTML. על מנת לחפש ב-HTML אנו צריכים ספרייה ש יודעת לפרש את ה-HTML זהה ולמצוא בו דברים, כמו ספריית cheerio הדוקומנטציה והמודול נמצאים פה: <https://www.npmjs.com/package/cheerio>

נתקין את cheerio עם:

```
npm i cheerio
```

הדוגמה הראשונה היא בדיק מה שאנו צריכים – קבלת HTML וחישוף אם יש מילה כלשהי שאנו חיפשימים. ניצור service בקובץ js.service.js

```
const cheerio = require("cheerio");

const parser = (input, word) => {
  const $ = cheerio.load(input);
  const result = $($`*:contains("${word}")`);
  if (result.text()) {
    return '1';
  } else {
    return '0';
  }
};

module.exports = parser;
```

גם פה, הקוד פשוט ביותר: פונקציה מקבלת את ה-HTML ואת המילה שאנו מחיפשים. טוענת את הטקסט ל-`cheerio` בדיק כדי שנכתב בדוקומנטציה ומבצעת את החישוף.

לב ה-service הוא שתי שירותות:

```
const $ = cheerio.load(input);
const result = $('*:contains("${word}")');
```

## קריאה וסירה והכנסה לאובייקט

עכשו נאחד בין שני השירותים עם service נוסף בשם finder, שיימצא בתיקיית .finder.service.js. הוא זה שיקרא לשירות קריית האתר, יבצע את הפרסור ויחזר לנו אובייקט מוכן לכתיבת הקוד הנתונים:

```
const scraper = require('./scraper.service');
const parser = require('./parser.service');

const finder = async (url, keyword) => {
  const html = await scraper(url);
  const found = parser(html, keyword);
  return {
    timestamp: Date.now(),
    keyword,
    url,
    found,
  };
};

module.exports = finder;
```

## לחבר את הכל

כעת נותר רק לחבר את הכל. אנחנו צריכים בעצם קוד שלוקח את מילת החיפוש שהמשתמש הזין, מעביר אותה לـjs.finder.service ומקבל ממשם את התוצאות. את התוצאות צריך להכניס לתוך מסד הנתונים ולהוכיח שההכנסה תסתיים. אחרי שההכנסה הסתיימה ומסד הנתונים עודכן, אפשר לקרוא ממנו את התוצאה המעודכנת ולהציג אותה. עשינו את הכל, אבל עכשו נעשה את זה עם מידע אמוני. צריך לזכור שהכל אסינכרוני, מה שאומר המון שימוש בـawait/async. מצד אחד זה קוד הרבה יותר אלגנטי, בלי hell callback, טירוף של then או בלגן אחר. מצד שני, זה קוד שנראה קצת יותר מורכב כי יש שימוש בـall.Promise וגם בـmap.

ראשית אציג את הקוד ואז אסביר, שורה אחר שורה. זה החלק המורכב ביותר של המערכת, כי אנו מחברים את כל היחידות שבנו:

```
const express = require('express');

const router = express.Router();
const dbConnection = require('../services/db.service');
const finder = require('../services/finder.service');

/* GET home page. */
router.get('/', (req, res, next) => {
  res.render('index', { title: 'Welcome to the scan site' });
});

/* POST home page. */
router.post('/', async (req, res, next) => {
  const { scantext } = req.body;
  const sites = ['https://example.com'];
  const scansPromises = sites.map((site) => finder(site, scantext));
  const scans = await Promise.all(scansPromises);
  const dbConnectionPromises = scans.map((scan) =>
    dbConnection.writeScans(scan));
  await Promise.all(dbConnectionPromises);
  const scanresults = await dbConnection.readScans(scantext);
  res.render('index', {
    scanresults,
    title: `Search result for: ${scantext}`,
  });
});

module.exports = router;
```

זו השורה הפושאה ביותר שכבר כתבנו. אנחנו מקבלים את הטקסט שהמשתמש מחפש ומשתמשים בו:

```
const sites = ['https://example.com'];
```

זהו מערך של אתרים. כרגע הוא מוגדר באופן סטטי בלבד. בעתיד אולי נבצע קריאה של אתרים שונים מסד הנתונים, אבל עכשווי הוא סטטי. מה שחשוב זה לראות שמדובר במקרה ולא במחוזת טקסט, כי אף על פי שהבדיקה נעשית מול אתר אחד, אנחנו תומכים בשורה של אתרים:

```
const scansPromises = sites.map((site) => finder(site, scантext));
```

השורה הבאה, שהיא בעצם הגדלת מערך האתרים ל-finder, היא זו שעלולה ממש לבלבול. Sites זה מערך, הגדרנו אותו בשורה שלמטה. המап הוא הדרך הקונבנציונלית להכניס ל-finder שלנו, שמחזיר פורמייס, את כתובות האתרים ולקבל את כל הפורמייסים לערך:

```
const scans = await Promise.all(scansPromises);
```

משתמשים ב-Promise.all כדי לקבל את התשובות מה-finder. ברגע שמערך הפורמייסים מה-finder מושלם, כלומר כל האתרים נסרקו – כל תוצאות הסריקה נכנסות לקבוע scans. כלומר scans מכיל עכשווי את כל תוצאות הבדיקה:

```
const dbConnectionPromises = scans.map((scan) => dbConnection.writeScans(scan));
```

בדוק באותו אופן מכניסים את כל תוצאות הסריקות אל dbConnection.writeScans. זו אולי השורה המורכבת ביותר שיש בסקריפט. עוברים על ה-scans עם map ושולחים אל writeScans כל scan בנפרד. הכל אסינכרוני ובצמם מקבלים מערך של הבתוות שצריכות להתמסח:

```
await Promise.all(dbConnectionPromises);
```

כאן ממתינים שישתיימו כל ההכנסות למסד הנתונים:

```
const scanresults = await dbConnection.readScans(scантext);
res.render('index', {
```

```
scanresults,
  title: `Search result for: ${scantext}`,
});
```

זהו השלב האחרון. אחרי שכל הכתובות הסתיימו, אפשר לבצע קריאה של מסד הנתונים עבור המילה שלנו, להכניס את הכל לתוכה ולהציג למשתמש את הסריקה.

אם הכל עבד כמורה, כשהנחפש את המילה `example` במסד הנתונים, נקבל תוצאות :

A screenshot of a web browser window. The address bar shows 'localhost:3000'. The main content area displays the heading 'Search result for: example' followed by a table with one row of data. The table has columns: ID, TimeStamp, URL, Keyword, and Found (1 if yes, 0 if not). The data row is: 1, 1632040971698, https://example.com, example, 1.

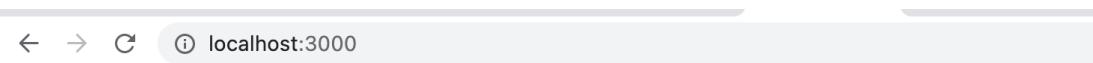
ID	TimeStamp	URL	Keyword	Found (1 if yes, 0 if not)
1	1632040971698	https://example.com	example	1

## Search result for: example

Insert text for scan		Scan		
ID	TimeStamp	URL	Keyword	Found (1 if yes, 0 if not)
1	1632040971698	https://example.com	example	1

זהו יעבוד גם אם נוסיף עוד אתרים, כמו למשל:

```
const sites = [ 'https://example.com' , 'https://internet-
israel.com' ];
```



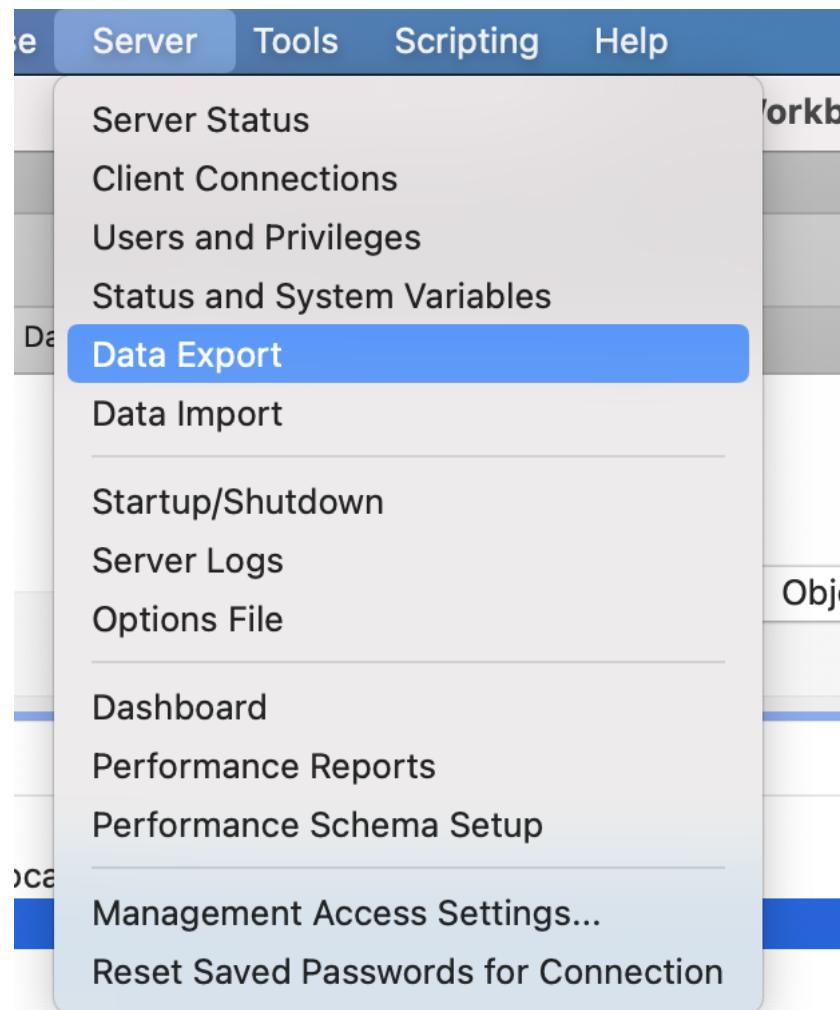
## Search result for: example

Insert text for scan		Scan		
ID	TimeStamp	URL	Keyword	Found (1 if yes, 0 if not)
1	1632040971698	<a href="https://example.com">https://example.com</a>	example	1
2	1632041040985	<a href="https://example.com">https://example.com</a>	example	1
3	1632041040846	<a href="https://internet-israel.com">https://internet-israel.com</a>	example	0

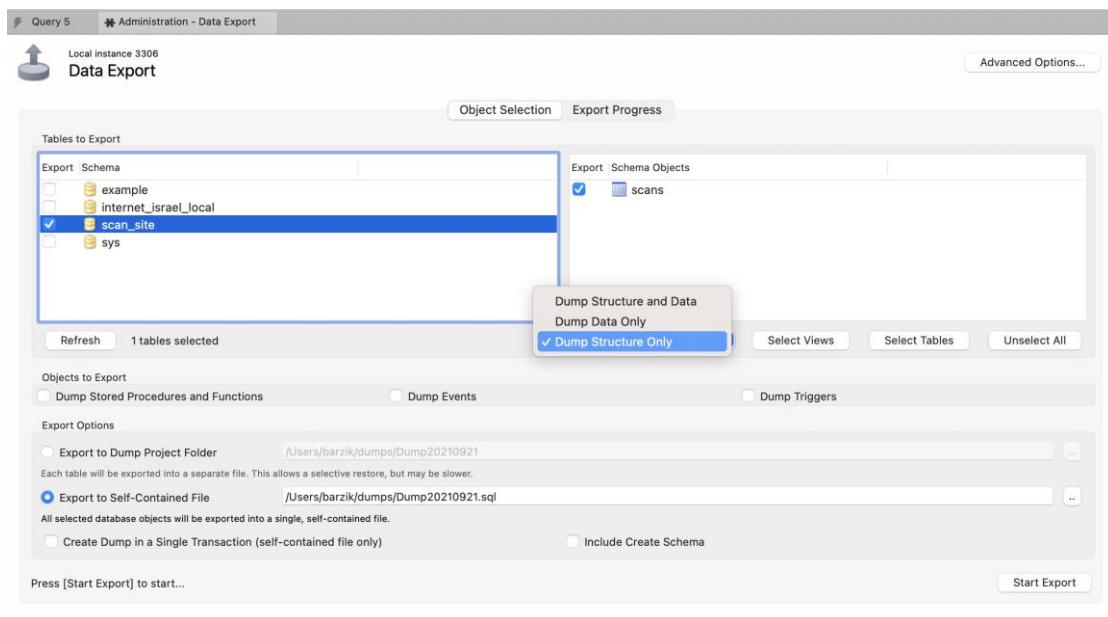
## יצוא מסד נתונים לגיט

כאמור, כל קוד של כל פרויקט לדוגמה אמר או להיות בגיט, עם קומיטים, פול ריקוסטים ואפיוño סביבת דמו. נשאלת השאלה – איך בדיק מיצאים מסד נתונים מסווג MySQL לגיט?

אפשר בהחלט להכניס קוד SQL MySQL, שהרצה שלו יוצרת את מבנה הטבלאות. איך עושים את זה? באמצעות כלי יצוא המידע MySQL Workbench. נכנסים לשונית Server ובוחרים יצוא מידע:



השלב הבא הוא לבחור את מסד הנתונים שרצים לגבות ולבחרו במבנה או במידע:



התוצאה תהיה קובץ SQL שאפשר להכניס לקובץ משלו. אפשר גם לכתוב סקורייפט ב-`Node.js` ליבוא מסד הנתונים זהה. או, כאשר עושים דיפולוי, לזכור ליבוא את מסד הנתונים באמצעות פקודה פשוטה:

```
mysql -u [user] -p [database_name] < [filename].sql
```

הסבר נוסף על כך מובא בפרק על דיפולוי בהמשך.

## סיכום

המערכת שתיארנו עכשו היא המערכת המורכבת ביותר שמתוארת בספר והוא מכילה לא מעט לוגיקה ומורכבות. מפתח מנוסה יפתח את המערכת זו בתוך שעה-שעתיים ובאיכות גבוהה. למפתח לא מנוסה זה ייקח הרבה יותר זמן, אבל זו המשמעות של ניסיון. נראה לא יהיה לכם קל. תטעו לא פעם, גם אם תעתקו את הקוד הוא לא יעבד, תקבלו שגיאות שונות ותצטרכו לשrox שעות על חיפוש בגוגל ולשבור את הראש רק כדי להבין בסוף שהכל קרה בגלל שיטת פסיק או שגיאת כתיב כללה. אבל זה בדוק תהליך הלמידה. אם עשיתם את הכל והכל עבד מיד מההופה – סימן שהספר לא עשה את העבודה עבורכם. תהליך הלימוד הואTeVוות. לטעות ושוב לטעות, לבזבז את הזמן בחיפושים של שגיאות ובעיות ותקלות. לנסות שוב ושוב ושוב עד שמצלחים – זה לא זמן מבוזבז. זה הלימוד.

אפשר לפתוח פיצ'רים נוספים במערכת – נסו את הפיצ'רים הבאים:

3. במקומות `timestamp`, נסו לכתוב תאריך מסודר.
4. במקומות 0 או 1, נסו להציג " נמצא" או " לא נמצא".
5. נסו לעצב את הטבלה ולהוסיף דינמיות.
6. נסו להכניס דף נוסף של ניהול רשימת האתרים, במקום רשימה סטטית.
7. למתקדים: נסו להחליף את `sys` בריект.

פרק 9

# דילוי



## דיפולוי

דיפולוי (Deployment) או בקצרה Deploy (או פריסה, זהה המונח הרשמי בעברית שאיש לא משתמש בו, והוא שם כולל למגוון תהליכי שעובר הקוד עד שהוא מגיע למצב שאנשים יכולים להשתמש בו. בדרך כלל מדובר בשורה של תהליכי שימושים מוביילים להעתקה ולהרצה של קבצים בשרת אינטרנט – מחשב שמחובר לשרת חיצונית בעלת כתובת IP חיצונית הכוולה שם מתחם (דומיין, כמו למשל [hebdevbook.com](http://hebdevbook.com)) או ללא שם מתחם.

דוגמה טובה היא למשל אתר בריאקט שניי בונה ומריץ עם create react app, כפי שŁemduo בספר "למוד ריאקט בעברית". הסיבה רצה במחשב שלי בסביבת localhost, אבל אחרי שסיימתי לפתח את הקוד והcoil מוכן – מה אני עושה עכשיו? איך אני מעלה את זה לשרת אינטרנט כדי שאנשים אחרים יראו את זה?

דוגמה נוספת נוספת היא יידג'ט מבוסס ג'אווסקריפט או אפיילו jQuery שעבוד בסביבת אתר וורדפרס. אחרי שסיימתי את הכתיבה שלו – איך אני מפיז אותו ללקוח כדי שייעבוד באתר הוורדפרס שלו?

דוגמה נוספת היא קוד של שרת Node.js שMRIIZ את אקספרס. האקספרס MRIIZ אותו מורכב עם ריאקט. איך אני מעלה את האתר הזה לשרת כדי שאנשים יוכלו לגשת אליו? הפעולה הזאת של העלה, בין שמדובר בהעלאה ראשונית ובין שבעדכון, נקראת דיפולוי ויש מגוון דרכים לעשות אותה. בחברות הייטק גדולות תהליך הדיפולוי הוא באחריות אנשי ה-DevOps (Development Operations), שהאחראים על המסע של הקוד מהסביבה (environment) שלנו אל הסביבה שבה ל��וחות יכולים לראות אותו. סביבה היא שם כללי לשרת או לקובץ של שירותי, למסדי נתונים או לכל מרכיב אחר של מה שמציג את הקוד ללקוח שלנו. יש כמה סביבות כאלה:

סביבה development – סביבת פיתוח/בדיקות, שהיא בדרך כלל הסביבה שבה התוכנה או האתר רצים על המחשב שלנו כאשר אנו מפתחים.

סביבה staging – סביבת בדיקות/תצוגה שכבר משתמשת בתנאים אמיתיים של הלוקחות. הסביבה הזאת אינה חשופה לאינטרנט באופן חופשי ומאפשרת לבדוק את תקינות האתר שלנו בסביבה קרובה לסביבה האמיתית.

סביבה production – הסביבה האמיתית שלקוחות משתמשים בה.

בחברות מרכבות בעלות מוצר מורכב, אנשי ה-DevOps מתחזקים מגוון שלם של תהליכי שנוudo להעביר את הקוד שהמתקנים כתובים אל הסביבה המתאימה. אבל לא כולם עובדים בחברת הייטק, במיוחד לא בפרויקט ראשון או בפרויקטים פרילנס, וחשוב להכיר את התהליכים שהקוד שלנו עבר ולדעת איך בעצם מציבים את האתר, האפליקציה או הווידג'ט שלנו בשירות כדי שייה זמין לאנשים אחרים בראשת. זה נראה כמו CAB ראש או חומר מיותר ללמידה, אבל בפועל זה חלק מהיכולות המקצועיות של מתכנת, ובראיונות עבודה שואלים לא פעם על הליך הדיפלומנט שקוד הפרויקטים שיצרתם עבר.

נדגים פה כמה שיטות לדיפלומנט, אבל חשוב לציין שיש המונח חברות שספקות שירותי, שירותים ענן ודריכים לעשوت דיפלומנט. אין דרך אחת טובה או ספק אחד מושלם, ו מבחינתי אין כאן בהכרח המלצה להשתמש בשירותים שלהן.

## אפליקציה סטטית או לא סטטית

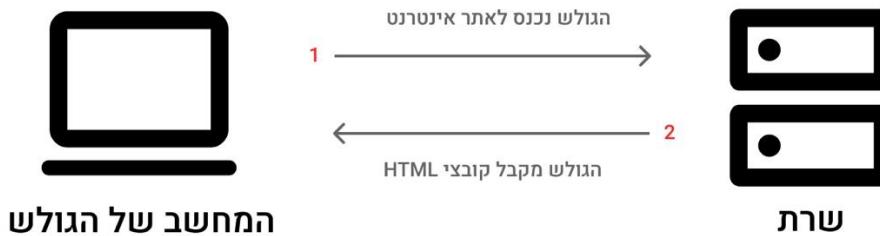
כשמדובר על דיפלו לשרת, הדבר הראשון שצריך להבין הוא אם הקוד שלנו הוא סטטי או דינמי. מה זאת אומרת? נזכיר לידע בסיסי על איך שרת באינטרנט עובד. שרת הוא בעצם מחשב רגיל שמחובר לרשת ומאזין לתקשורת. כאשר מגיעה בקשה בפורט 80, הוא מזכיר מידע, בדרך כלל HTML, ג'אווהסקריפט ו-CSS. החזרת המידע הזה נקראת גם הῆגשה או Serve, מלשון Server.

התהליך של גישה לאתר הוא פשוט. לquo מקליד שם מתחם בדף שלו, הדף דין מתרגם את שם המתחם לכתובת IP באמצעות שרת DNS ויוצר קשר עם כתובות ה-IP זו. השרת שি�ובן בכתובות ה-IP מקבל את הבקשה ופועל בהתאם. אם מדובר בבקשת גישה בלבד לティー הראשית, השרת יחזיר קבצים. כך שרת אינטרנט עובד. הדף דין יכול להיות דף סלולרי, אפליקציה על מחשב או על טלפון נייד או סטרימר. הפורט יכול להיות 80 או 3000 (סבירנו את זה בספר "למוד Node.js בעברית").

הקבצים האלה יכולים להיות סטטיים ל gamble. כאמור, השרת מגיש אותם בכל פעם ללא שום שינוי. ניתן שהג'אווהסקריפט שעלייהם יקרה ל- API חיוני כלשהו, אבל הקבצים שהשרת שלנו יגיש יהיו אחידים.

לחילופין, הקבצים יכולים להשתנות בהתאם למשתמש. למשל, אתר חדשות ששואב את המידע שלו ממAGER הכתבות, או פייסבוק, שמנגן לכל משתמש פיד אישי:

## אתר סטטי



## אתר דינמי



אם מדובר ב-HTML ובעג'אוהסקרייפט בלבד? זו אפליקציה דינמית. אם האפליקציה שלנו נוצרה עם `create react app`, אז היא אפליקציה סטטית. לעומת זאת, כל מה שצריך זה להציג אותה בשרת שיעודו להגיש קובצי HTML. אתר דינמי הוא אתר המבוסס על `Node.js express` ובכל פעם שמשתמש ניגש אליו, הוא מייצר את הקבצים, או API מבוסס `Node.js` שמבצע פעולות שונות מול מסד נתונים.

## אפליקציה סטטית

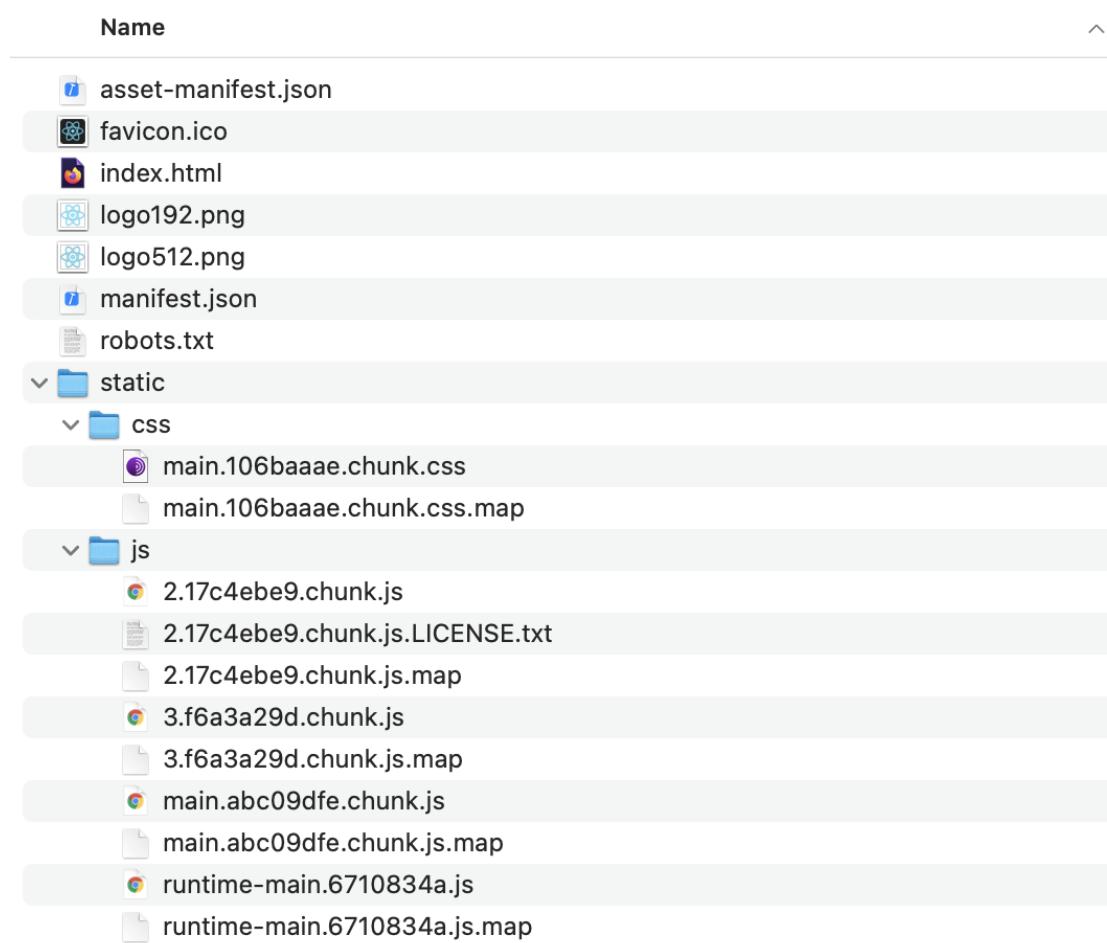
באמצעות `create react app` למשל אנו יכולים לבצע תהליך `build` שבסיומו יהיו לנו קובץ HTML וכן קובצי CSS/GAHOHESKRIFUT. אנחנו יכולים ליצור אפליקציה סטטית גם עם ג'אוהסקרייפט פשוט לגמרי או עם `jQuery` וקובצי CSS/GAHOHESKRIFUT. את הקבצים האלה אנו צריכים להציג בשרת שיעודו להאזין לתנועה בפורט 80 ולהחזיר אותם למי שմבוקש.

הפתרון הזול והפשוט ביותר הוא לשכור שרת או חלק משרת (זה נקרא גם אחסון שיטופי). שכירות שרת משתנה בהתאם לגודל השרת ולאיכותו – שרת טוב יותר, עם יותר

ליבורן, זיכרון וسطح בדיסק הקשיח, יעליה יותר משרת צנווע יותר, בדיקן כמו כל מחשב. שרת שיעמוד לשימושכם הבלתי ייה יהיה יקר יותר משרת שיתופי שתחלקו עם לקוחות אחרים.

נדגים זאת בעזרת אפליקציה סטטית פשוטה המציגה, באמצעות API, רשימה של כל הפרויקטים של משתמש גיטהאב (כבר הוסבר לעילו כאן בפרק המוקדש לדוגמאות לאפליקציות). מה שחשוב לדעת הוא שמדובר באתר פשוט שמציג מידע על משתמשי גיטהאב. למשל, בהקלחת שם המשתמש שלי, barzik, אני מקבל את כל הפרויקטים על שמי. אמנם יש חלקיים דינמיים באפליקציה (משגרים בקשה אל ה-`API` של גיטהאב), אבל האפליקציה עצמה סטטית. כל מה שהליך צריך הוא לטעון את ה-`HTML`, `HTML` והג'אווהסקריפט וקובצי `CSS`. לא נדרש חישוב מהשרת, הכנסה או הוצאה למסד נתונים וכו'. כל מה שהשרת צריך לעשות הוא להגיש לכל מי שմבקש את הקבצים האלה.

איך יודעים שמדובר באפליקציה סטטית? אחרי שהרכננו את `run build mock`, בתיקית ה-`build` יהיו לנו קובצי `CSS`, `HTML` וג'אווהסקריפט בלבד:



מה שהשתת צריך לעשות הוא לגייס את הקבצים האלה ללקוח. כמובן, לדעת להחזיר אותם ללקוח שמנכינס את כתובת האתר בדףן. זהו שרת שיודע להחזירקובץ סטטי.

## דיפולו של אפליקציה סטטית באמצעות שרת אחסון

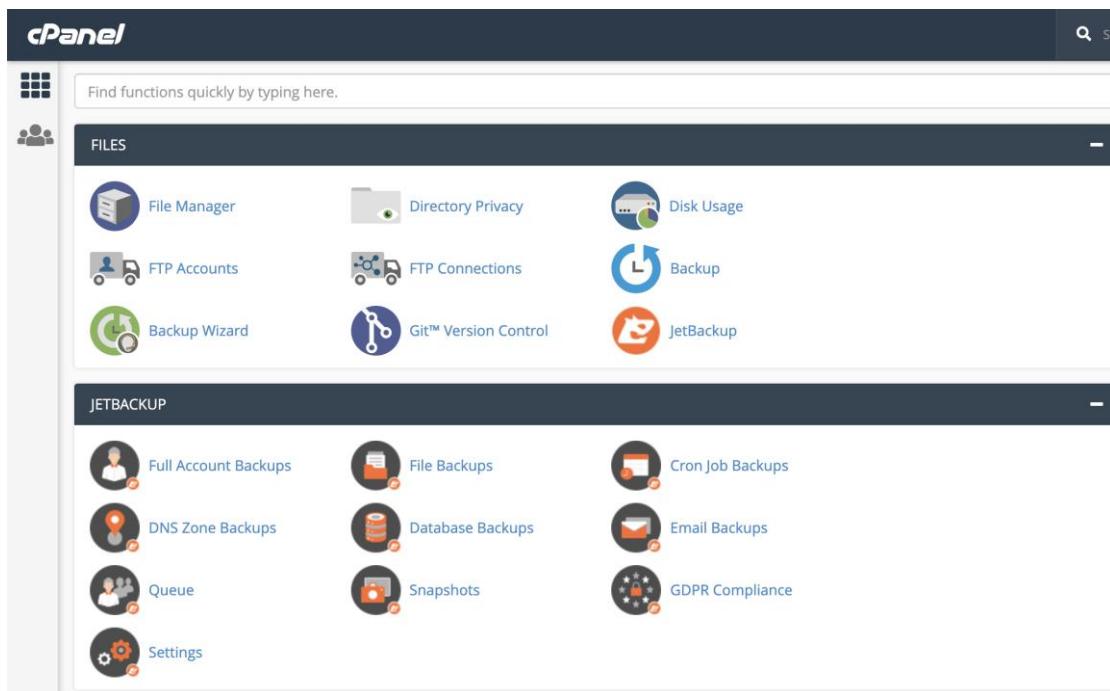
### שיתופי

הדרך הפופולרית ביותר לאחסון אפליקציה סטטית היא שרת שיתופי. כאמור, תוכנות מותקנות על מחשבים חזקים (יותר ופחות), ואנחנו חולקים את המחשב עם לקוחות אחרים. הם כמובן לא יכולים לגשת לקבצים שלנו ויש הפרדה, אבל אנחנו חולקים את אותו מחשב, בדיקות כמו מחשב במעבדת מחשבים בתיכון או באוניברסיטה. בשיעור שלנו אנחנו עובדים על מחשב מסוים, אבל בשיעור אחר סטודנט אחר לחוטין ישמש בו. על המחשב הזה צריכה להיות מותקנת תוכנה שיודעת לגייס, בפורט 80, את קובצי ה-HTML, CSS והג'אוויסקריפט ללקוח. התוכנה זו נקראת שרת, והשירותים הפופולריים ביותר הם Apache (אפאצ'י) או Nginx (אנג'ן אייקס), שיודעים לקבל בקשה ממשתמש שנכנס לדומין ולהחזיר את index.html וקבצים נוספים.

אפשר לרכוש חשבון בשירותים כאלה במחיר של דולרים בודדים בחודש. מובן שהשירותים האלה יהיו חלשים ולא יעדכו בעומס מבקרים רב, אבל אפליקציה סטטית בדרך כלל לא תקשה עליהם. כשרוכשים את החשבון על השרת, רוכשים בדרך כלל גם שם מתחם (דומיין) – אך לאណון בכך במספר הזה.

את רוב השירותים השיתופיים בעולם מנהלים עם מערכת שנקראת cPanel. כאשר שוכרים שרת שיתופי, מומלץ לבדוק שהאתר אכן מספק מערכת נזו. זהה בעצם מערכת גרפית שמחילה את הטרמינל ואייתה אפשר לעשות פעולות שונות. כשרוכשים שרת שיתופי, מקבלים קישור למשק הניהול שהוא ה-[cPanel](#), שם משתמש וסיסמה והוראות ל קישור שם המתחם שרכשנו אל השירות השיתופי.

панל ניהול של cPanel נראה כך:

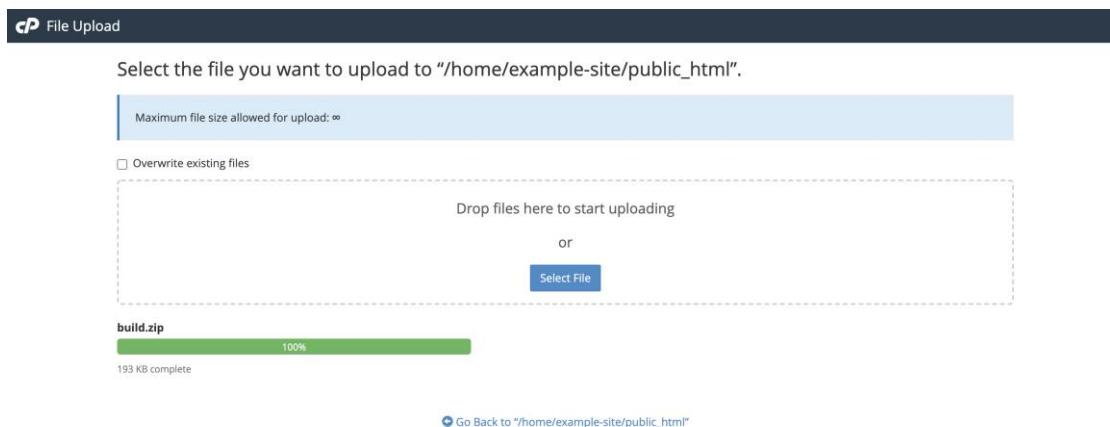


מה שמשמעותו שם זה ה-File Manager?

Name	Size	Last Modified	Type	Permissions
cagefs	4 KB	Dec 27, 2020, 11:08 AM	httpd/unix-directory	0771
.c.selector	4 KB	Dec 27, 2020, 5:18 PM	httpd/unix-directory	0755
cpaddons	4 KB	Oct 10, 2019, 6:37 AM	httpd/unix-directory	0755
.cppanel	4 KB	Today, 12:57 PM	httpd/unix-directory	0700
.phorde	4 KB	Aug 27, 2021, 6:01 PM	httpd/unix-directory	0700
.htpasswdws	4 KB	Feb 8, 2020, 11:10 AM	httpd/unix-directory	0750
jbm	4 KB	Dec 16, 2019, 3:02 PM	httpd/unix-directory	0755
.spamassassin	4 KB	Oct 5, 2019, 9:47 PM	httpd/unix-directory	0700
subaccounts	4 KB	Apr 23, 2021, 11:54 AM	httpd/unix-directory	0700
.trash	4 KB	Jul 1, 2021, 2:00 AM	httpd/unix-directory	0700
etc	4 KB	Today, 12:57 PM	httpd/unix-directory	0750
logs	4 KB	Yesterday, 3:37 PM	httpd/unix-directory	0700
mail				
public_ftp				
public_html				
ssl				

יש לא מעט תיקיות על השרת – חלקן פנימיות, והתקייה שזמיןنا למשתמשים חיצוניים נקראת `html`, או `www` בחלק מספקי האחסון. כל מה ששמות שם יהיה זמין לגולשים מכל העולם, אם מדובר בקובץ סטטי. נעה לשם את כל הקבצים והתיקיות. אפשר להשתמש בממשק הנוח – להיכנס לתיקיית `public_html`, ללחוץ על `upload` ולהעלות את כל הקבצים שיש בתיקיית `public_html`. במקום להעלות ידנית את כל הקבצים, אפשר לנaze אוטם בקובץ `zip` (קיז), להעלות אותו כקובץ אחד ולפתח אותו גם כן באמצעות הממשק הגרפי.

כך זה נראה – העלאה:



מיד אחרי הפעלה, כל הקבצים יהיו בתיקייה, וחשוב לוודא שקובץ ה-`index.html` יהיה בתיקייה הראשית, כיוון שהשרת, בין שמדובר ב-`Apache` ובין שב-`Apache`, מגיש למשתמש אוטומטית את הקובץ זהה כאשר הוא נכנס לתיקייה הראשית.

זהו, כך מבצעים דיפלוי. אפשר לראות שמדובר בתהליך פשוט מאוד וידני. וזה גם הבעיה בשרתים מסוימים – אין בהם כלים מותחנים שיביצעו עדכון של הקוד ברגע שהוא משתנה בגייט. מצד שני, זה פשוט וдол ויכול להתאים מאוד בתנאים מסוימים. למשל, לאחר התרגול של הספר "למוד ג'אווהסקריפט בעברית" הוא אפליקציה סטטית הנמצאת בשרת.

## דיפלוי של אפליקציה סטטית באמצעות GitHub pages

שירות GitHub pages החינמי מאפשר להציג אפליקציות סטטיות באמצעות גיטהאב. השירות הוא בבעלות חברת גיטהאב, נמצא בכתובת `so.githhub.io` וכל משתמש בגיטהאב יכול להשתמש בו ללא רישום נוספת. השירות הזה מיועד בעצם להציג את הפרויקט, דמו שלו או יישום שלו. הוא לא מיועד ליישומים בתשלום, אבל נادر כסבירות תצוגה לפרויקטים שרצו להציג. מקובל להעלות פרויקט גמר או פרויקטים אחרים ל-

GitHub pages ולהציב את הקישור ב"על אודות הפרויקט" וכן בדף קורות החיים.

הDİFOLIוֹ פה הוא פשוט ונעשה באמצעות מודול `mkch` בשם `gh-pages`. כדי להשתמש במודול דרך שורת הפקודה, תצטרכו חשבון גיטהאב וכן אפשרות לתקשר אליו באמצעות SSH, בדיק כפי שלמדנו בתחילת הספר בפרק על גיט.

אפשר לעשות דיפולו בהתאם לטכנולוגיה, אבל אני אסביר על אפליקציה סטטית שנבנית באמצעות `create react app`, כפי שלמדנו בספר "לימוד ריאקט בעברית".

## התקנת `gh-pages`

המודולים שמשמשים אותנו לאפליקציה מוגדרים בקובץ `package.json`. כדי להשתמש ב-`gh-pages` לדיפולו, צריך להתקן אותו. נתקין את `gh-pages` כ-`dependency` של `dev` של הפרויקט באמצעות כניסה לתיקיית הפרויקט ו从此 הקלה בטרמינל של הפקודה:

```
npm install gh-pages --save-dev
```

אחר לכך נגדר ב-`package.json` של הפרויקט את המקום אליו נועשה דיפולו. למשל:

```
"homepage": "http://barzik.github.io/github-user-history-viewer",
```

הכתובת תיראה כך:

<https://barzik.github.io/github-user-history-viewer>

במקום `barzik` יהיה שם המשתמש שלכם בגיטהאב ובמקום `-viewer` – שם הפרויקט שלכם.

אחרי כן נוסיף את הפקודות הבאות תחת `:scripts`:

```
"predeploy": "npm run build",
"deploy": "gh-pages -d build"
```

זה הכל. כאשר נרצה לעשות דיפלוי, פשוט נקליד `run deploy` ומקה. הפקודה מרים את ה-`build` שיש ב-`app` ב-`create react app` וזו שולחת את תיקיית `build` ל-`GitHub` `pages`. כיוון שאנחנו עושים את זה דרך הטרמינל שמודר לעבודה עם SSH ועם המפתחות שלנו, אין צורך בזיהוי או באימומות נוספת:

```
File sizes after gzip:
43.93 KB  build/static/js/2.17c4ebe9.chunk.js
1.63 KB   build/static/js/3.f6a3a29d.chunk.js
1.19 KB   build/static/js/runtime-main.cae19196.js
1.09 KB   build/static/js/main.abc09dfe.chunk.js
292 B     build/static/css/main.106baaae.chunk.css

The project was built assuming it is hosted at /github-user-history-viewer/.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.

Find out more about deployment here:
  https://cra.link/deployment

> static-app@0.1.0 deploy
> gh-pages -d build

Published
```

מיד אחרי שיופיע `published`, נוכל לראות את האפליקציה שלנו בעמוד שהגדכנו בדף, וכמונו גם כל אדם אחר בעולם. כך אפשר ליצור דמו מרשימים. אפשר, באמצעות GitHub actions, להגדיר אוטומטיות דופליי בכל `release` וכן לייצר `Continuous Deployment`.

## דיפלוי של אפליקציה סטטית באמזון

אמזון היא חברת סחר ענקית שמוכרת בזכות החנות העצומה שלה, אבל היא גם חברת תשתיות אינטרנט עצומות שירותים הענן שלה, ה-`i-Services`, (`AWS`) (Amazon Web Services). עולם ה-`AWS` הוא רחב ואנשי ה-`DevOps` מכירים אותו לעומק, אך נפוצים מאוד בעולם. עולם ה-`AWS` הוא רחב ואנשי ה-`DevOps` מכירים אותו לעומק, אך אפשר לעשות דיפלוי לאפליקציות סטטיות באמזון גם בלי להכיר לעומק את מגוון השירותים העצום של החברה. ישנם שירותים שונים, כמו `Lightsail` או `Amplify`, המאפשרים דיפלוי פשוט מאוד. נדגים באמצעות `Amplify`.

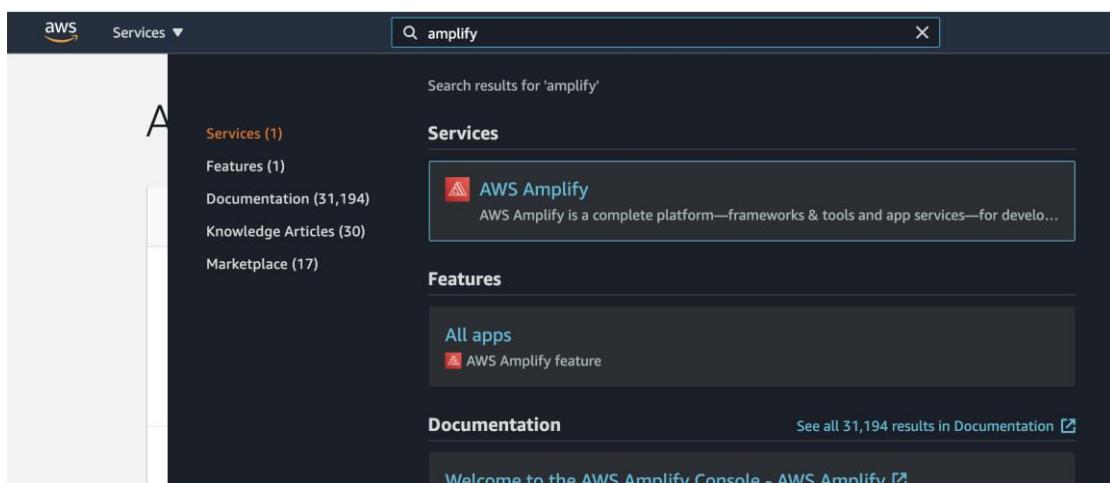
## יצירת חשבון באמזון

יש ליצור חשבון באמזון בכתובת `aws.amazon.com`. שימו לב – זה לא אותו חשבון כמו בחנות אמזון אלא חשבון אחר לגמרי. על מנת ליצור אותו צריך להזין כרטיס אשראי בינלאומי בתוקף, שלא יהיה בחיבור ראשוני. לקוחות חדשים באמזון מקבלים לא מעט שימושים בחינם, ואם היקף השימוש באפליקציה לא מגיע להגדרות שיש ב-`Free Tier`, לא מחויבים כלל. על שימוש ממשמעותי (למשל יותר מהמשה גיגabit תעבורת לחודש) מחויבים, אבל הוצאות של השירותים המוצגים פה לא גבוהה בכלל.

מה שכן – חובה לבחור שם משתמש וסיסמה חזקים, כיון שגם פה את כרטיס האשראי. שירות AWS הם בהחלט מטרה לפריזות ולהונאות, ותוקפים יכולים לחדוג על חשבון פרוץ בחיבורים אסטרטגיים.

## כניסה ל-amplify

אחרי שייצרנו את החשבון, נחפש את amplify בשירותיהם:



:Get Started - נגיע לדף ניהול שלו ונלחץ על הכפתור הכתום -



# AWS Amplify

Fastest, easiest way to develop mobile and web apps that scale.

[GET STARTED](#)

שימוש ללב: בכל זאת מדובר באתר דינמי, והcube'ים של העיצוב أولי ישתנו. בעת כתיבת השורות האלה (אפריל 2022) כך נראה העיצוב, וסביר להניח שהוא כך בעתיד, אך גם אם הממשק ישתנה, התהילה יישאר בkowskiים כלליים דומה למtooar כאנ. גם אם תגינוו למשק ועל הcapetor יהיה כתוב Start והוא יהיה במצב אדום – נראה זה יהיה הcapetor הנכון.

נגייע למסך שבו נבחר באפשרות של **:Host your web app**

## Get started

### Develop

### Create an app backend

Setup a backend to enable data, authentication, or storage capabilities. Then integrate them in your app with just a few steps.

[Get started](#)

### Deliver

### Host your web app

Connect your Git repository to continuously deploy your frontend and backend. Host it on a globally available CDN.

[Get started](#)

נבחר בGITLAB:

**Host your web app**

AWS Amplify offers a fully managed hosting service for web apps. Connect your repository in the Amplify Console to build, deploy, and host your web app.

**From your existing code**

Connect your source code from a Git repository or upload files to host a web app in minutes.

- GitHub
- Bitbucket
- GitLab
- AWS CodeCommit
- Deploy without Git provider

Amplify Console requires read-only access to your repository.

**Continue**

נבחר מיד למסך שבו נוצרה הדרישה לאישור החיבור לגיטהאב (ויתכן שנידרש להתחבר לגיטהאב קודם לכך):

Authorize AWS Amplify (eu-west-1)

AWS Amplify (eu-west-1) by [aws-amplify-console](#) wants to access your [barzik](#) account

**Repositories**  
Public and [private](#)

Organization access  
[codeworth-gh](#) X [Request](#)

**Cancel** **Authorize aws-amplify-console**

Authorizing will redirect to  
<https://eu-west-1.console.aws.amazon.com>

אחרי הצלחת החיבור נחזיר למסך שבו קיבל חיובי על ההצלחה ונידרש לבחור את הריפורזיטורי ואת הבראנץ' שנרצה לעשות לו דיפלוי:

**Add repository branch**

**GitHub**

**GitHub authorization was successful.**

**Repository service provider**

**Recently updated repositories**  
If you don't see your repository below, please push a commit and then click the refresh button.

**barzik/github-user-history-viewer**

**Branch**  
Select a branch from your repository.

**main**

Connecting a monorepo? Pick a folder.

**Cancel** **Previous** **Next**

במסך הבא יוצג לנו קובץ טקסטואלי קל לקרוא (הפורמט נקרא YAML). יש שם את התרגום הטקסטואלי של ההגדרות. בדרך כלל לא תהיה בעיה עם ההגדרות, אבל חשוב לשים לב שפקודת `build` נמצאת שם:

**Configure build settings**

**App name**  
Pick a name for your app.  
**github-user-history-viewer**

Name cannot contain periods

**Build and test settings**  
We've auto-detected your app's build settings. Please ensure your build command and output folder (baseDirectory) are correctly detected.

```

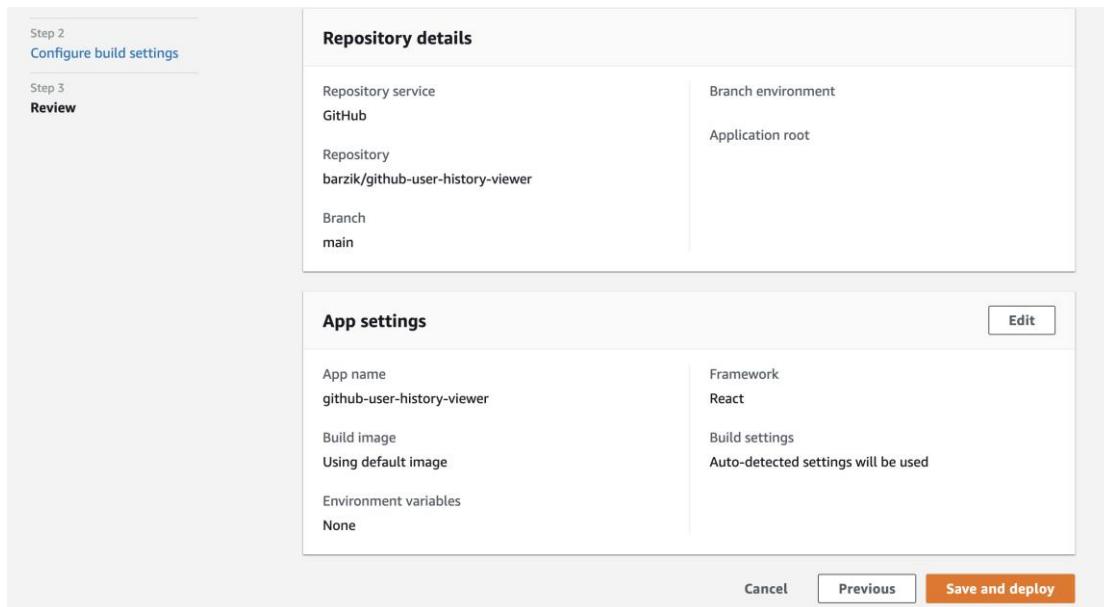
1 | version: 1
2 | frontend:
3 |   phases:
4 |     preBuild:
5 |       commands:
6 |         - yarn install
7 |     build:
8 |       commands:
9 |         - yarn run build
10 |     artifacts:
11 |       baseDirectory: build
12 |       files:
13 |         - '**/*'
14 |     cache:
15 |       paths:
16 |         - node_modules/**/*
17 |

```

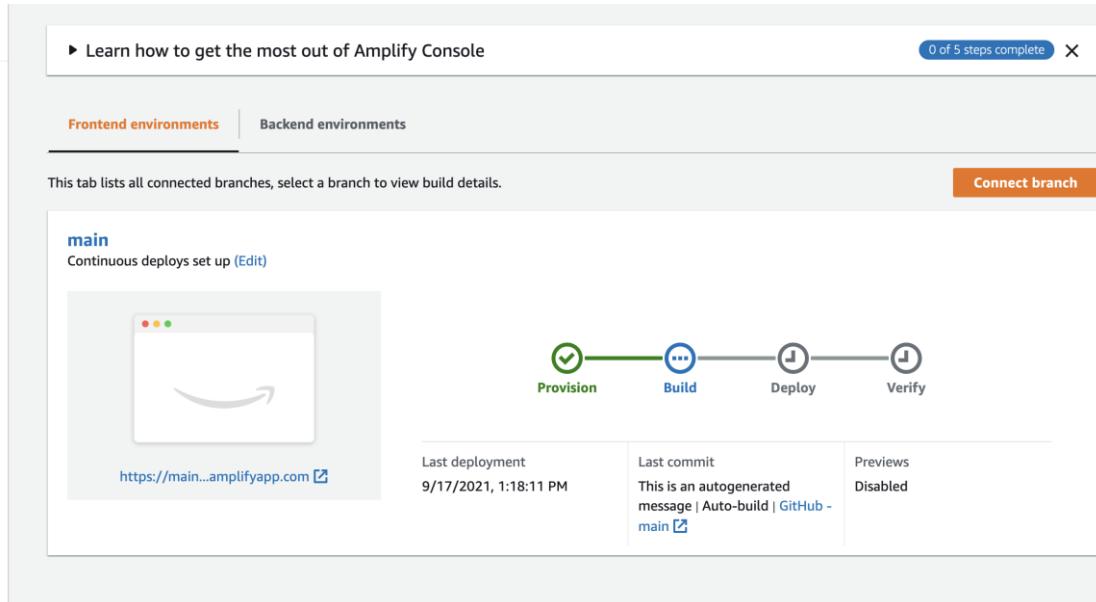
Build and test settings

**Download** **Edit**

אחרי שנודע, נעשה save and deploy



עכשו נשאר רק ללחכות ולראות שהכל נבנה כמו שצריך:



אפשר ללחוץ על הקישור כדי להיכנס לאפליקציה שלנו ולראות אותהעובדת.

אם מההו לא עובד, אפשר ללחוץ על ה-`build` כדי לבנות שוב או להשתמש בקונסולה כדי לראות אם יש הודעות שגיאה. לא אמורות להיות כאלה, אבל אם יש – זה חלק מהלימוד.

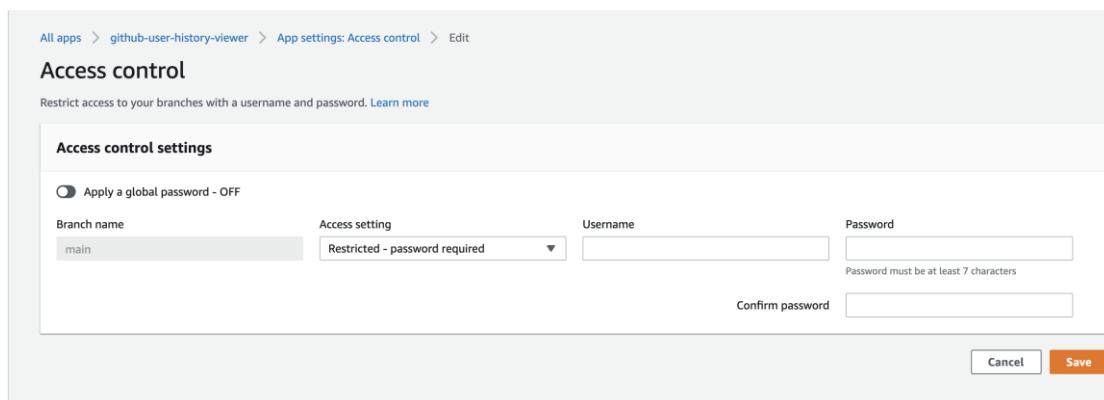
אפשר לבדוק אם הדומיין נראה בערך כה:

`/https://main.d2cpchovvn7bly.amplifyapp.com`

זה דומיין לא נראה במיוחד. אפשר לרכוש דומיין ולחבר אותו ל-amplify כך שייהיה לנו שם מתאם משלנו. ישנן הוראות מדויקות, בהתאם למקום שבו רוכשים את הדומיין, פה:

<https://docs.aws.amazon.com/amplify/latest/userguide/custom-domains.html>

אפשר להגביל כניסה אל האפליקציה באמצעות שם משתמש וסיסמה שמנדרירים בקונסולה. זה לא יחזק מים עבור משתמשים רבים, אבל עשוי להגביל את הכניסה אם מדובר במקרה שלא רוצים לחסוף או בשירות שמယעד למטעים. בתפריט הצדדי יש סעיף `access control`, שבו אפשר להגיע לתפריט הגבלת הכניסה:



## אפליקציה דינמית

כאשר יש פעולה של צד שרת, בין שמדובר בשפת PHP ובין שבבידוטנט או ב-`.js`,  
`Node.js`, יש לנו אפליקציה דינמית. רוב האתרים המוכרים כיום הם אפליקציות דינמיות. יש להם מסד נתונים מסווג מונגו או MySQL והם מציגים דפים שונים למשתמשים שונים או דפים שונים בנקודות שונות בזמן.

בדרך כלל לאפליקציה דינמית יש כמה חלקים:

1. שפת תכנות בצד השרת – למשל Node.js. השפה זו צריכה לזרוץ בשרת והיא אחראית ברוב המקרים ליצירת שרת האינטרנט שיעודו לעבוד עם פורט 80 (בספר "לימוד Node.js בעברית" יש הסבר מקיף על פורטים) או עם כלים אחרים. ישנן שפות שמצריכות שרת אינטרנט.
2. מסד נתונים – למשל MySQL, שמכיל את נתונים המשמשים, סיסמות, כתבות, טוקבקים, מידע על רכישות – כל מידע שהוא.
3. קבצים סטטיים – קובצי ג'אווהסקרייפט שהליך מורד וهم רצים עצמם (בדיקות כמו אפליקציה סטטית), קובצי CSS, תמונות, וידאו וכו'.

כל החלקים האלה, שבรวมם הם קבצים, צריכים לעלות למכונה כלשהי שתדע להריץ את השרת ואת מסד הנתונים ולטפל בכך. מדובר בדיפלוי מעת יותר מורכב כי דברים עלולים להשתבש. למשל, הגודל הפיזי בגיגabytes שהקצינו למסד הנתונים עלול להיגמר. אם למכונה אין מספיק זיכרון, תהליך ה-Node.js אולי לא ירוז כמו שצריך. אבל עם הכלים והפתרונות של היום, דיפלוי אינו דבר מורכב במיוחד.

נדגים את הדיפלוי באמצעות אפליקציה Node.js וMySQL מורכבת, שבנינו בפרק על הפרויקטים לדוגמה.

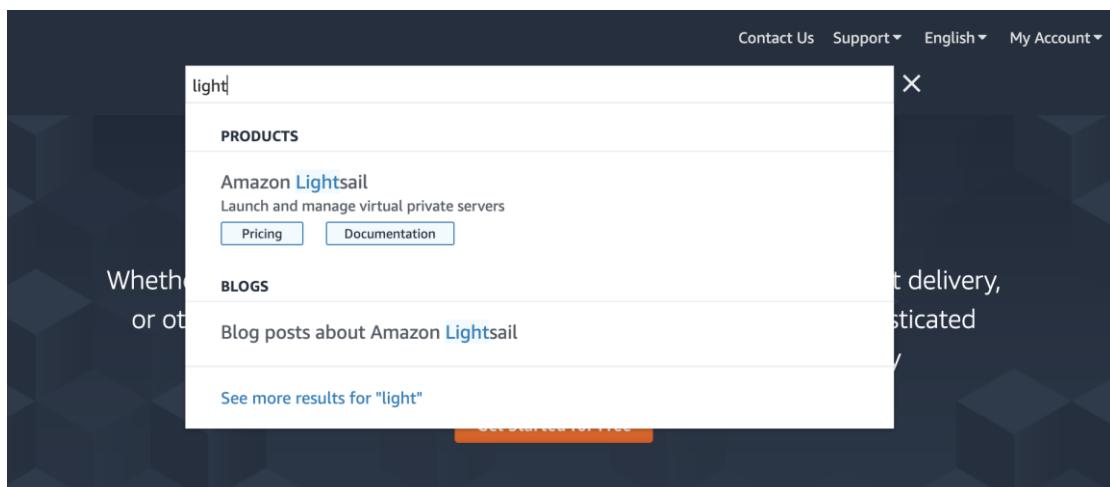
## דיפלוי באמזון

ישנן דרכים רבות לעשות דיפלוי מנט לאפליקציה עם קוד מורכב באמזון, והמערכות תלויות בסקליל שלנו – כמה אנשים אנו רוצים שיישתמשו באפליקציה בו בזמן. אפשר לכתוב ספר שלם רק על דיפלוי של אפליקציה עם Node.js ומסד נתונים באמזון כיוון שנitin לבנות רמת מרכיבות לאפליקציה שתומכת במילוני משתמשים בשנייה. מובן שרתם מרכיבות כזו גם דורשת לא מעט כסף, שכן בהיקפים כאלה השירותים של אמזון לא ניתנים בחינם. אך לא זו המטרה שלנו כאן, אלא להראות איך עושים דיפלוי באמזון לפרויקט קטן, שאחר כך אפשר להציג דמו שלו ואמור לשמש אלפי אנשים ביום במקרה הטוב, ולעשות את זה בחינם או בעלות נמוכה מאוד.

אחת הדרכים הטובות ביותר לעשות דיפלו באמזון לאפליקציות בסקליל נמוך היא להשתמש ב-Lightsail, סביבה זולה 3.5 דולר בחודש ושלושת החודשים הראשונים בחינם בזמן כתיבת سورות אלו) המספקת שירות לינוקס שאפשר לעבוד בו. העבודה היא דרך שורת פקודה בטרמינל ולא דורשת התקנות מיוחדות.

אם כבר ייצרתם חשבון AWS, בהמשך לפרקי הקודם, התחברו אליו ותגינו לדף המוצרים.

אם לא פתחתם חשבון, נפתח חשבון ב-AWS בכתובת <https://aws.amazon.com> החשבון דורש כרטיס אשראי בינלאומי פעיל ובתוקף (בהתחלת אין חיוב). נחפש בתוך מגוון המוצרים את המילה Lightsail וניכנס לדף המוצר:



## יצירת המכונה

בדף Lightsail נלחץ על כפתור Create instance. זהו כפתור כתום גדול:

Create instance

כאמור, עיצוב האתר עשוי להשתנות, אבל גם אם הכפתור אדום וכ כתוב בו **Create a new instance**, לא צריך להיבהל. מדובר במסך גרפי ומה שחשוב זה רוח הדברים – יצירת מכונה וירטואלית באמזון.

במשך הבא נקבע את תכונות המcona ומה תהיה מותקן עליה. נבחר במקום שאנו רוצחים ובפלטפורמת לינוקס. המיקום חשוב, כיון שהשאיפה היא למקם את האפליקציה שלנו קרוב ללקוחות:

## Create an instance

### Instance location

 You are creating this instance in **Ireland, Zone A** (eu-west-1a)  
 Change AWS Region and Availability Zone

### Pick your instance image

Select a platform



אחרי כן נלחץ על OS Only (OS – Operating System) וنبחר את האוביונטו החדש ביותר. מדובר במערכת הפעלה יידידותית מבוססת לינוקס, שMRIIZA היבט שפויות תכונות רבות. היא בקוד פתוח וזולה מאד להפעלה גם באמזון:

### Select a blueprint

Apps + OS  OS Only



#### Ubuntu 20.04 LTS

Ubuntu 20.04 LTS - Focal. Lean, fast and powerful, Ubuntu Server delivers services reliably, predictably and economically. It is the perfect base on which to build your instances. Ubuntu is free and will always be, and you have the option to get support and Landscape from Canonical.

Learn more about Ubuntu on the [AWS Marketplace](#).

By using this image, you agree to the provider's End User License Agreement. ©2008-2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#)

השלב הבא הוא לבחור חבילה – אם מדובר בדמו, מומלץ לבחור חבילה זולה, אבל כזו שיש לה מינימום של גיבוב אחד של זיכרון. זה קרייטי עבור MySQL. תמיד אפשר לשדרג אותה מאוחר יותר:

You can add a shell script that will run on your instance the first time it launches.

You are using the **default** SSH key pair for connecting to your instance.  
 Change SSH key pair

Automatic snapshots create a backup image of your instance and attached disks on a daily schedule.

Enable Automatic Snapshots

### Choose your instance plan (?)

**New!** Check out our new 16 GB and 32 GB RAM bundles!

Sort by:



:Create instance **下一步**

### Identify your instance

Your Lightsail resources must have unique names.

#### TAGGING OPTIONS

Use tags to filter and organize your resources in the Lightsail console. Key-value tags can also be used to organize your billing, and to control access to your resources.

[Learn more about tagging.](#)

#### Key-only tags (?)

#### Key-value tags (?)

**Create instance**

©2008-2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) 1

מיד לאחר היצירה נחזיר למסך המוצר, ועכשו נוכל לראות את המכונה שלנו נוצרת.  
נctrיך להמתין כמה דקות:

# Good morning!

**Instances**      Containers      Databases      Networking      St

Sort by **Date** ▾



**Ubuntu-1**

512 MB RAM, 1 vCPU, 20 GB SSD

Pending

52.19.120.206

2a05:d018:7a9:1d00:a15f:f274:1afd:720f

Ireland, Zone A

אחרי המתנה של כמה דקות, המכונה תהיה מוכנה **לפעולה**:

# Good morning!

Instances    Containers    Databases    Networking    St

Sort by Date ▾

 Ubuntu-1 512 MB RAM, 1 vCPU, 20 GB SSD	<span>Running</span>	52.19.120.206 2a05:d018:7a9:1d00:a15f:f274:1afd:720f Ireland, Zone A
-----------------------------------------------------------------------------------------------------------------------------	----------------------	----------------------------------------------------------------------------

אפשר ללחוץ על האיקון הכתום באורת הטרמינל, ויפתח לנו, על גבי הדף, טרמינל לינוקס. גם אם אנחנו חלונות, הטרמינל הזה אינו שונה מה-cmd של חלונות.

כל מה שצრיך לעשות עכשו הוא:

1. להתקין `Node.js`.
2. להתקין MySQL במכונה הlokalityt.

## התקנת Node.js

יש פירוט מלא של התקנת לינוקס בספר "למוד Node.js" בעברית. בקצרה, יש להריץ את הפקודות הבאות על מנת להתקין Node.js חדש:

```
curl -sL https://deb.nodesource.com/setup_16.x -o
nodesource_setup.sh

sudo bash nodesource_setup.sh

sudo apt-get install -y nodejs
```

```
Ubuntu-1 – Terminal | Lightsail
lightsail.aws.amazon.com/ls/remote/eu-west-1/instances/Ubuntu-1/terminal?protocol=ssh

## Run `sudo apt-get install -y nodejs` to install Node.js 16.x and npm
## You may also need development tools to build native addons:
##   sudo apt-get install gcc g++ make
## To install the Yarn package manager, run:
##   curl -sL https://dl.yarnpkg.com/debian/pubkey.gpg | gpg --dearmor | sudo tee /usr/share/keyrings/yarnkey.gpg >/dev/null
##   echo "deb [signed-by=/usr/share/keyrings/yarnkey.gpg] https://dl.yarnpkg.com/debian stable main" | sudo tee /etc/apt/sources.list.d/yarn.list
##   sudo apt-get update && sudo apt-get install yarn

ubuntu@ip-172-26-14-169:~$ sudo apt-get install -y nodejs
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  nodejs
0 upgraded, 1 newly installed, 0 to remove and 211 not upgraded.
Need to get 26.2 MB of archives.
After this operation, 123 MB of additional disk space will be used.
Get:1 https://deb.nodesource.com/node_16.x focal/main amd64 nodejs amd64 16.9.1-deb-1nodesource1 [26.2 MB]
Fetched 26.2 MB in 1s (19.6 MB/s)
Selecting previously unselected package nodejs.
(Reading database ... 59624 files and directories currently installed.)
Preparing to unpack .../nodejs_16.9.1-deb-1nodesource1_amd64.deb ...
Unpacking nodejs (16.9.1-deb-1nodesource1) ...
Setting up nodejs (16.9.1-deb-1nodesource1) ...
Processing triggers for man-db (2.9.1-1) ...
ubuntu@ip-172-26-14-169:~$ node -v
v16.9.1
ubuntu@ip-172-26-14-169:~$
```

Ubuntu-1  
52.19.120.206

זה הכל. עכשיו יש Node.js על המחשב.

## התקנת MySQL

יש פירוט מלא של התקנת MySQL על לינוקס בספר "למוד MySQL בעברית". בקצרה, עושם זאת כך:

הרכבת הפקודה אישור של כל מה ששאליהם אותנו. זה עשוי לדרוש זמן:

```
sudo apt upgrade
```

אחר מכן:

```
sudo apt-update
```

התקנת MySQL עם:

```
sudo apt install mysql-server
```

אחרי כן ניתן לכנס אל הקונסולה של MySQL עם:

```
sudo mysql
```

נראה שכעת יש `mysql` לפני הטקסט שלנו. זו אינדיקציה לכך שאנו בקונסולה של MySQL. נגדיר את הסיסמה למשתמש `root` באמצעות הפקודה:

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH
mysql_native_password BY 'Your password';
```

אנא זכרו שמדובר בשורת שפתוח לעולם והשתמשו בסיסמה שאינה 123456 או q1w2e3. אחרי הקלדת הפקודה `alter table` ולחיצה על אנטר נצא באמצעות הפקודה `.exit`.

ניתן שוב לטרמינל של MySQL באמצעות הפקודה:

```
mysql -u root -p
```

והקלדת הסיסמה שלנו. זו סיסמת `root`.

```
ubuntu@ip-172-26-1-224:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 8.0.26-0ubuntu0.20.04.2 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> exit
```

## אבטחת מידע של MySQL

כיוון שמדובר בשרת MySQL בסביבת פודקשן (כן, גם אם זה אתר קטן שמרימים לדוגמה), כדאי להקשיח אותו, ככלומר להפוך אותו לקשה יותר לפריצה. זה נעשה בклות באמצעות הרצת סкриיפט פשוט שבא עם MySQL באמצעות הפקודה:

```
sudo mysql_secure_installation
```

הפקודה שואלת כמה שאלות פשוטות – לגבי אכיפות סיסמאות (מומלץ לבחור בשתיים), איסור על כניסה למסד הנתונים מהרשת (זאת אומرت גישה רק מהמכונה. זה מונע שימוש ב-MySQL Workbench אך מדובר בפעולה קרייטית), מחיקת מסד נתונים test ועוד שאלות. מומלץ לענות על כלן ב-yes.

עכשו יש לנו מכונה עם MySQL ועם Node.js ואפשר להתחיל לעבוד. אם הפרויקט שלנו נמצא בריפוזיטורי של גיטהאב (ציבורי או פרטי), אין פשוט יותר להשתמש בGIT על מנת למשוך אותו.

אם מבנה הנתונים נמצא בקובץ sql בGIT, צריך ליבא אותו לתוך מסד הנתונים באמצעות הקלדת השורה זו בטרמינל של השרת:

```
mysql -u [user] -p [database_name] < [filename].sql
```

שרת MySQL רץ כל הזמן. על מנת להריץ את התהילך של Node.js יש להשתמש ב-node, כפי שהסביר בספר "למוד Node בעברית".

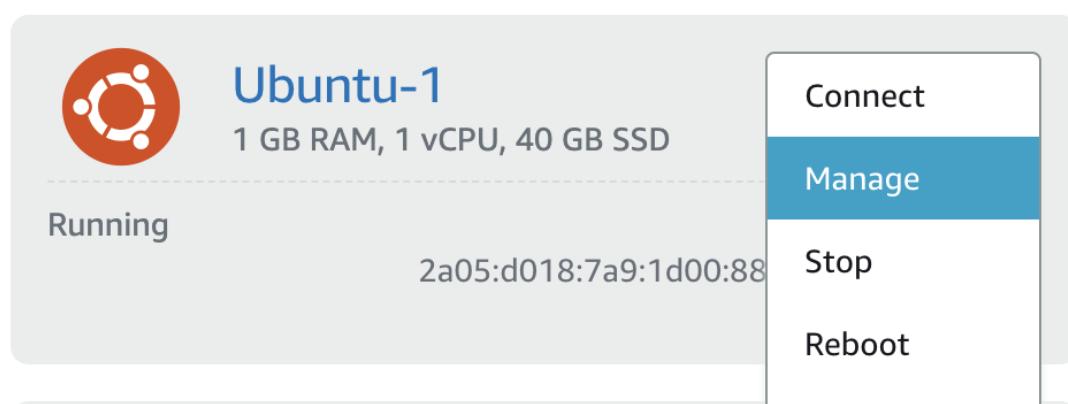
## הציגת האפליקציה

אפשר לגשת לשרת מכל דפדפן שהוא, ואם יש לנו אפליקציה שרצה על פорт 80, נראה אותה בקלות. על מנת לברר מהו ה-IP החיצוני של השרת נלחץ על שלוש הנקודות ואז: Lightsail Manage בחלק של השירות שלנו בדף המוצר של

# Good afternoon!

**Instances**      [Containers](#)      [Databases](#)      [Networking](#)      [S](#)

Sort by **Date** ▾



ונכל לראות את כתובת ה-IP החיצונית. אמזון משנה לעיתים את הכתובת הזאת, ולכן אפשר ואפיו רצוי להפוך אותה לסתטית באמצעות בחירה ב-IP **Create static IP**:

ברגע שיש לנו כתובת IP סטטית, אנו יכולים לחבר אותה לדומין שלנו ולהציג את האפליקציה או פשוט למסור אותה ללקוח.

## דיפולו ב-Heroku

Heroku היא ספקית ענן גדולה וידידותית מאוד למפתחים. ביגוד למשק הגרפי של אמזון, ניהול של Heroku יכול להנהל רובו ככולו בטרמינל, דבר שמאוד מקל את התפעול ואת הדיפולו שלו לתוכנים מנוסים שמתרוגלים בעבודה מול CLI, ומצד שני עלול להיראות מסובך לתוכנים בתחילת דרכם. הגישה של Heroku היא דיפולו באמצעות גיט. אנו מגדירים remote שנקרא Heroku (בדוק כמה שמגדירים upstream) וdockers אליו. יש דחיפה של קוד חדש? יש דיפולו.

## פתיחה חשבון והגדרות ראשוניות

inicnes לאתר של Heroku בכתובת <https://www.heroku.com> ונפתח חשבון מפתחים חינמי. זהו הצעד הראשון. הצעד השני הוא להוסיף את המפתח הציבורי שלנו

לחשבו על מנת שנוכל להזדהות. כאמור, Heroku עובדת מול גיט ובשיטות של גיט. בפרק על ההזדהות מול גיטהאב הסבירתי על מפתח ציבורי ומפתח פרטי ואין ליצור אותם. אם לא עשיתם את זה – זה הזמן. את המפתח הציבורי יש להציב גם באתר של Heroku. נכנסים ל-[Settings](#) וגולמים להגדרות מפתח ה-SSH:



רק תזכורת – על מנת לקבל את המפתח הציבורי, אם אתם על חלונות יש לכתוב בטרמינל את השורה:

```
type %HOMEDRIVE%<%HOMEPATH%\.ssh\id_ed25519.pub
```

ולהעתיק אותה.

אם אתם במק או לינוקס או משתמשים ב-[Git Bash](#), יש לכתוב בטרמינל את השורה:

```
clip < ~/.ssh/id_ed25519.pub
```

והיא כבר מועתקת אל ה-[clipboard](#).

העבודה ב-Heroku נעשית עם הכלי שלהם שנקרא [Heroku CLI](#). אפשר להתקין אותו בקלות על חלונות, על לינוקס או על מק בלחיצה על הקישור הזה:

<https://devcenter.heroku.com/articles/heroku-cli#download-and-install>

ההתקנה תפעיל את הכלי של Heroku גם בטרמינל שלכם וגם (אם אתם בחלונות) [Git Bash](#).

נבדוק אם ההתקנה עבדה באמצעות כניסה לטרמינל והקלדה של הפקודה:

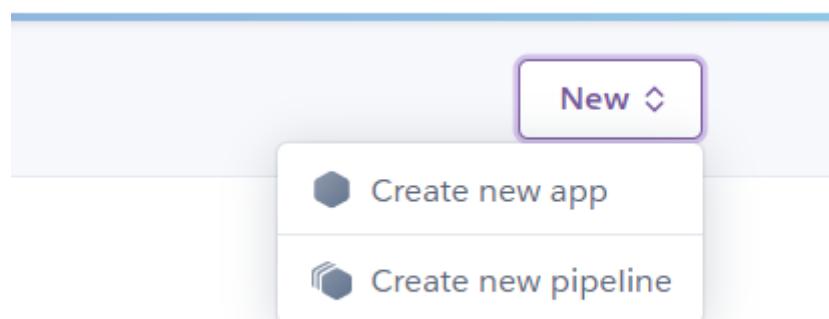
```
heroku login
```

מכניסים את המיל, הסיסמה וAIMOTOT צד שלישי אם צריך, ואם הכל תקין, הכל מוחובר ואפשר להתחיל לעבוד.

```
C:\Users\barzik>heroku login
Enter your Heroku credentials:
Email: ran@bar-zik.com
Password: *****
Logged in as ran@bar-zik.com
```

### יצירת האפליקציה

אחרי שהתקנו את CLI Heroku, הגדרנו את המפתח הציבורי והכנו בעצם את התשתית, הגיע הזמן ליצור את האפליקציה. נחזור לעמוד הראשי באתר Heroku, נבחר בלחצן **New** וナルחץ על **Create new app**:



כדי לשים לב שמדובר באתר וشعיצובים של אתרים משתנים. אם תגעו לכפתור במצב אחר, עם כתוב שונה, לא צריך להיבהל. צילומי המסך כאן הם באמת להמחשה בלבד.

השלב הבא הוא לבחור שם לאפליקציה שלנו או להשאיר את המקום ריק לבחירה אוטומטית. שתי אפשרויות נוספת הן בחירת אזור (בוחרים בדרך כלל באזוריים קרובים

לאלה שרוב הלקוחות של האפליקציה שלנו נמצאים בהם) ו-pipeline, שעליו לא נלמד בספר זהה. עדיף, בשלב הלימוד, לא לגעת באפשרויות ברירת המחדל:

Create New App

---

App name

Choose a region

United States ▼

---

Add to pipeline...

---

Create app

אחרי הלחיצה, האפליקציה עצמה נוצרת ואפשר לראות את שמה. במקרה זה -  
:`infinite-harbor-10475`

HEROKU

---

Personal > infinite-harbor-10475

---

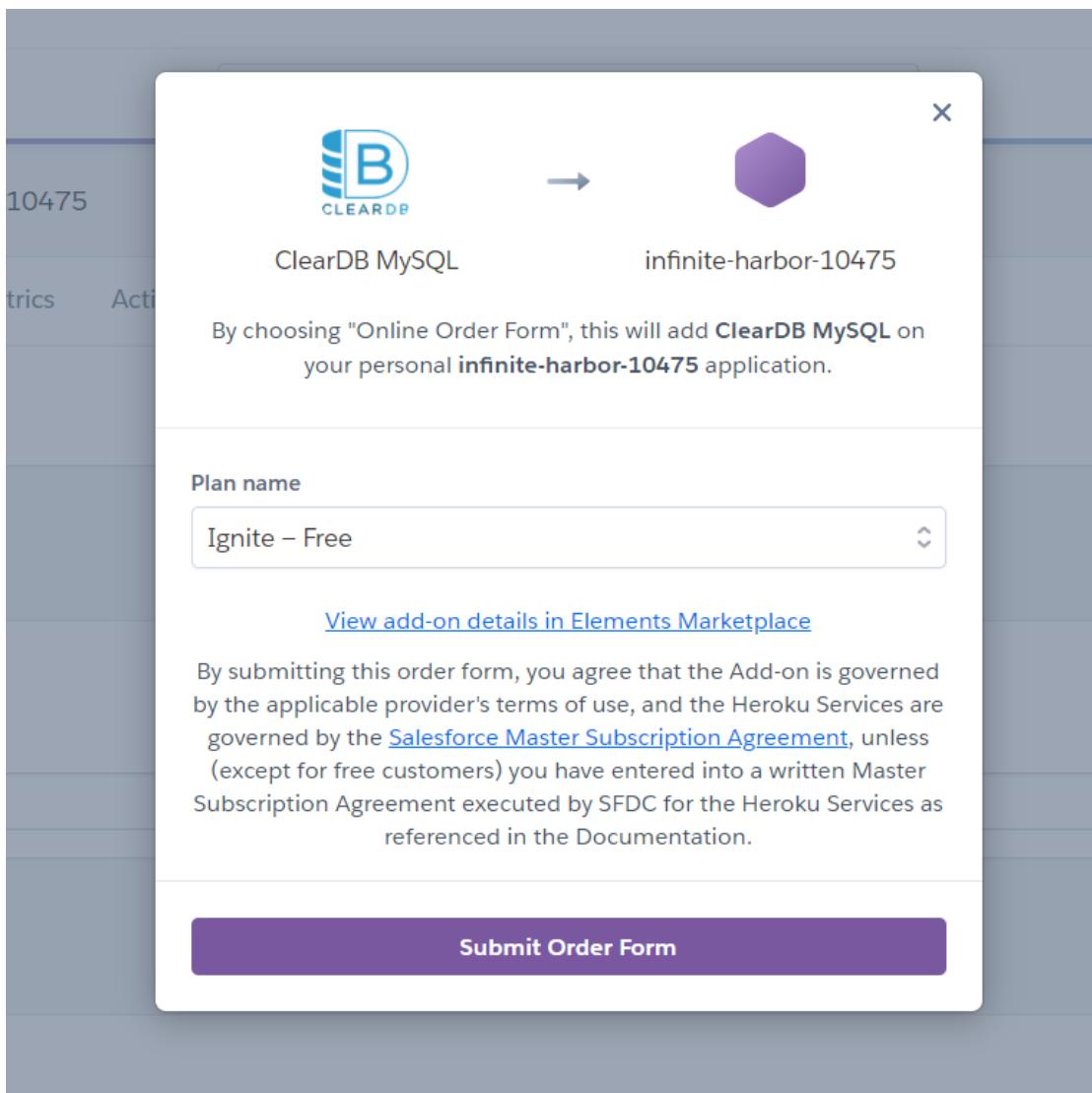
[Overview](#)   [Resources](#)   [Deploy](#)   [Metrics](#)   [Activity](#)

השלב הבא הוא הגדרת מסד הנתונים. ניכנס לשונית Resources וنחפש ons Add ,

ושם MySQL

The screenshot shows the Heroku dashboard for the application 'infinite-harbor-10475'. At the top, there's a navigation bar with tabs for Overview, Resources, Deploy, Metrics, Activity, Access, and Settings. The 'Resources' tab is currently selected. In the main content area, there's a section titled 'Dynos' which displays a message: 'This app has no process types yet' and 'Add a Procfile to your app in order to define its process types. [Learn more](#)'. Below this, there's a section titled 'Add-ons' with a search bar containing 'mysq'. A button labeled 'Find more add-ons' is located to the right of the search bar. Under the search bar, a list of add-ons is displayed, with 'ClearDB MySQL' being the first item.

נבחר ב-**MySQL** ClearDB MySQL המאפשרת תוכנית חינמית. אפשר לשלם יותר על מנת לקבל שרת בעל נפח גדול יותר:



אחרי הלחיצה נראה שיש לנו מסד נתונים. עכשיו אפשר לעבוד:

The screenshot shows the Heroku dashboard's "Add-ons" section. It lists the installed "cleardb" add-on with the message: "The add-on cleardb has been installed. Check out the documentation in its Dev Center article to get started." A search bar is at the top. Below it, the "ClearDB MySQL" add-on is listed with its plan ("Ignite"), price ("Free"), and status ("Up"). The estimated monthly cost is shown as "\$0.00".

## הדיפלוי של הקוד

ראשית נחבר את מה שאנחנו רוצים לעשות לו דיפלוי בgiatan Heroku באמצעות הפקודה  
הזה:

```
heroku git:remote -a infinite-harbor-10475
```

```
C:\local\scan-site>heroku git:remote -a infinite-harbor-10475
set git remote heroku to https://git.heroku.com/infinite-harbor-10475.git
```

זה בעצם יוצר remote חדש לגיט. בדוק כמו שיש לנו remote בשם `origin` לפרויקט שלנו בגיטהאב, ואם אנחנו יוצרים פורק יש לנו גם remote בשם `upstream`, יהיה לנו remote בשם נוסף, שנוכל לדחוף אליו קוד. פה הדחיפה לא תעדכן את גיטהאב – אלא תעדכן את Heroku שיש קוד חדש.

עכשו כל מה שאנחנו צריכים לעשות הוא לדחוף את הקוד באמצעות הפקודה:

```
git push heroku main
```

מיד נראה את הלוג של הפעולה, את הדחיפה של הקוד ואת התקנת המודולים (Heroku תזהה מיד שמדובר בפרויקט Node.js, ואם הקפדנו שהיא ב-`package.json` גם `npm start` תדע להריץ אותה מיד):

```
C:\local\scan-site>git push heroku main
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 8 threads
Compressing objects: 100% (21/21), done.
Writing objects: 100% (24/24), 81.43 KiB | 27.14 MiB/s, done.
Total 24 (delta 0), reused 24 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku-20 stack
remote: -----> Determining which buildpack to use for this app
remote: -----> Node.js app detected
remote:
remote: -----> Creating runtime environment
remote:
remote:       NPM_CONFIG_LOGLEVEL=error
remote:       NODE_VERBOSE=false
remote:       NODE_ENV=production
remote:       NODE_MODULES_CACHE=true
remote:
remote: -----> Installing binaries
remote:       engines.node (package.json): unspecified
remote:       engines.npm (package.json): unspecified (use default)
remote:
remote:       Resolving node version 14.x...
remote:       Downloading and installing node 14.17.6...
remote:       Using default npm version: 6.14.15
remote:
remote: -----> Installing dependencies
remote:       Installing node modules
remote:
remote:         > nodemon@2.0.12 postinstall /tmp/build_22bf5d99/node_modules/nodemon
remote:         > node bin/postinstall || exit 0
remote:
remote:       Love nodemon? You can now support the project via the open collective:
remote:         > https://opencollective.com/nodemon/donate
remote:
remote:       added 330 packages in 5.943s
remote:
remote: -----> Build
remote:
remote: -----> Caching build
remote:       - node_modules
remote:
remote: -----> Pruning devDependencies
remote:       removed 248 packages and audited 82 packages in 3.791s
remote:
remote:       12 packages are looking for funding
```

בסוף הפעולה נראה שהוא הצליח:

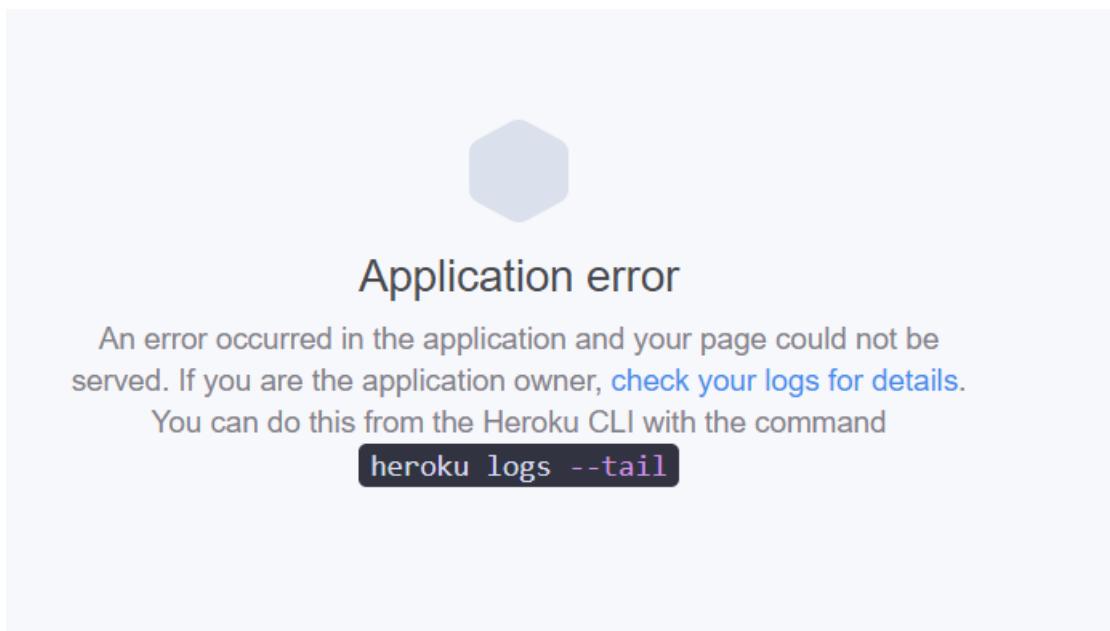
```

remote: -----> Build succeeded!
remote: -----> Discovering process types
remote:           Procfile declares types      -> (none)
remote:           Default types for buildpack -> web
remote:
remote: -----> Compressing...
remote:           Done: 33.8M
remote: -----> Launching...
remote:           Released v5
remote:           https://infinite-harbor-10475.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/infinite-harbor-10475.git
 * [new branch]      main -> main

C:\local\scan-site>

```

אם ניכנס אל הכתובת, שהיא בעצם שם האפליקציה, נוכל לראות שהנול עובד, בהנחה שמדובר באפליקציית Node.js בלבד. אבל אם משהו לא עובד – נראה מסך זהה:



אם נקליד בקונסולה שלנו את הפקודה זו:

```
heroku logs -tail
```

נראה מה השتبש. כאן למשל יש שגיאה שנוצרה כי ב-`start` וקח לא הורדתי את `:nodemon`

```

2021-09-22T09:12:30.604081+00:00 heroku[web.1]: Starting process with command `npm start`
2021-09-22T09:12:31.861752+00:00 app[web.1]:
2021-09-22T09:12:31.861768+00:00 app[web.1]: > site-scan@0.0.0 start /app
2021-09-22T09:12:31.861768+00:00 app[web.1]: > nodemon ./bin/www
2021-09-22T09:12:31.861769+00:00 app[web.1]:
2021-09-22T09:12:31.865401+00:00 app[web.1]: sh: 1: nodemon: not found
2021-09-22T09:12:31.868941+00:00 app[web.1]: npm ERR! code ELIFECYCLE
2021-09-22T09:12:31.869093+00:00 app[web.1]: npm ERR! syscall spawn
2021-09-22T09:12:31.869150+00:00 app[web.1]: npm ERR! file sh
2021-09-22T09:12:31.869223+00:00 app[web.1]: npm ERR! errno ENOENT
2021-09-22T09:12:31.871178+00:00 app[web.1]: npm ERR! site-scan@0.0.0 start: `nodemon ./bin/www` failed
2021-09-22T09:12:31.871248+00:00 app[web.1]: npm ERR! spawn ENOENT
2021-09-22T09:12:31.871303+00:00 app[web.1]: npm ERR!
2021-09-22T09:12:31.871350+00:00 app[web.1]: npm ERR! Failed at the site-scan@0.0.0 start script.
2021-09-22T09:12:31.871391+00:00 app[web.1]: npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

```

הפתרון הוא לתקן את ה-`start`, להסיר את `nodemon` ולדוחף שוב ל-Heroku. לא צריך להתבלבל או להיבחלה. השגיאות שיש שם הן בדרך כלל שגיאות רגילים של Node.js, שאפשר לשחזר גם בסביבה המקומית והן קלות להבין. חלק מהעבודה של המتنנטנים הוא לא להיבחלה מלוג של שגיאות אלא לקרוא אותו, להבין אותו ולטפל בו. בתחילת הדרך זה עלול להל치יך, אבל אחר כך מבינים שהוא נכון ושהקושי הוא דווקא במקום שאין שגיאות.

עדין, אם לא יהיה חיבור ל-MySQL, האפליקציה לא תעבור כמו שצריך. אנו צריכים לחבר את מסד הנתונים שלנו.

על מנת להתחבר למסד הנתונים, נחפש את פרטיו. כשהתחברנו למסד הנתונים, החיבור הכנס את הפרטים לאפליקציית Heroku שלנו. נמצא אותם באמצעות `heroku config`. נקליד את הפקודה זו ונחפש את השורה `CLEARDB_DATABASE_URL`. נראה שיש שם ערך צזה:

```
mysql://b686767e46de5a:638202010@us-cdbr-east-04.cleardb.com/heroku_9fc08dd54e7d83a?reconnect=true
```

במחזורת הטקסט הזה יש את כל הפרטים שאנו צריכים.

**שם המשתמש** הוא מה שמניע לפני הנקודות: `b686767e46de5a`,

**הסיסמה** היא מה שמניע אחריה `638202010`,

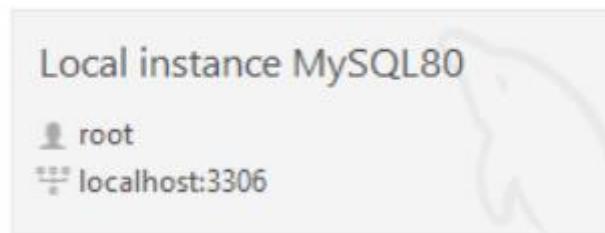
במקרה שלנו,

הHOST שלנו הוא us-cdbr-east-04.cleardb.com

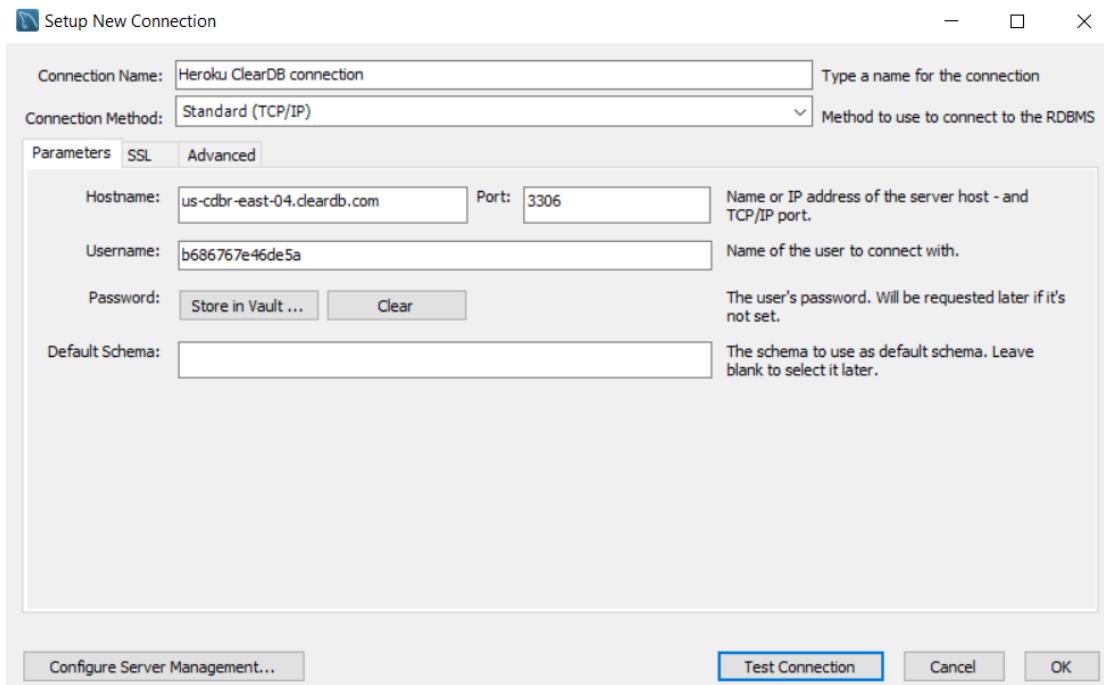
ושם מסד הנתונים הוא heroku\_9fc08dd54e7d83a

על מנת לעבוד עם מסד הנתונים אפשר ממש להתחבר אליו עם MySQL Workbench. עד כה בטח השתמשתם בו כדי להתחבר למסד נתונים מקומי משלכם, אבל אפשר בהחלט להתחבר אליו ולעבוד איתו על מסד נתונים מרוחק. נפתח אותו ונלחץ על סימן ההוספה ב宦boroirs:

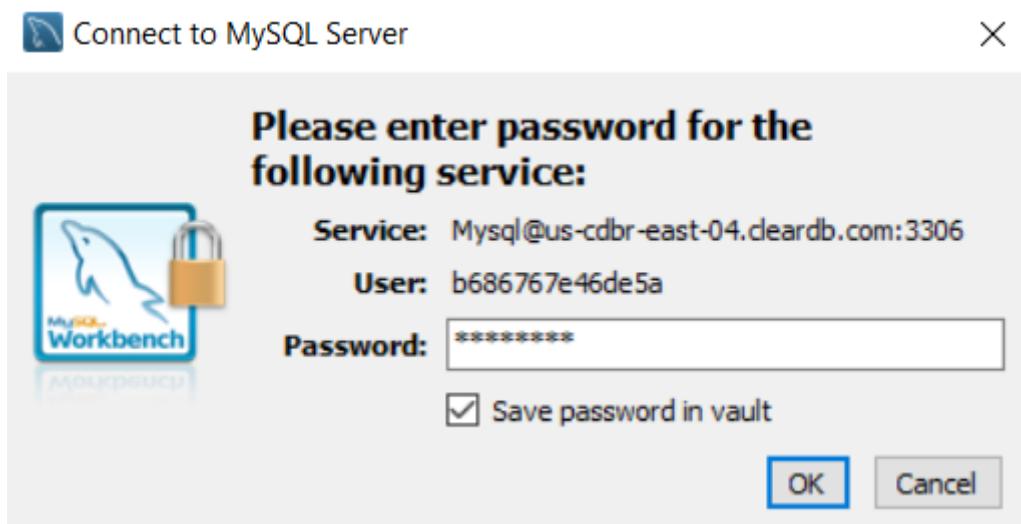
## MySQL Connections



אחריו.cn נגיעה למסך הוספת החיבור. נבחר לו שם, נכניס את hostname, את שם המשתמש ואת הסיסמה. נלחץ על **:Test Connection**

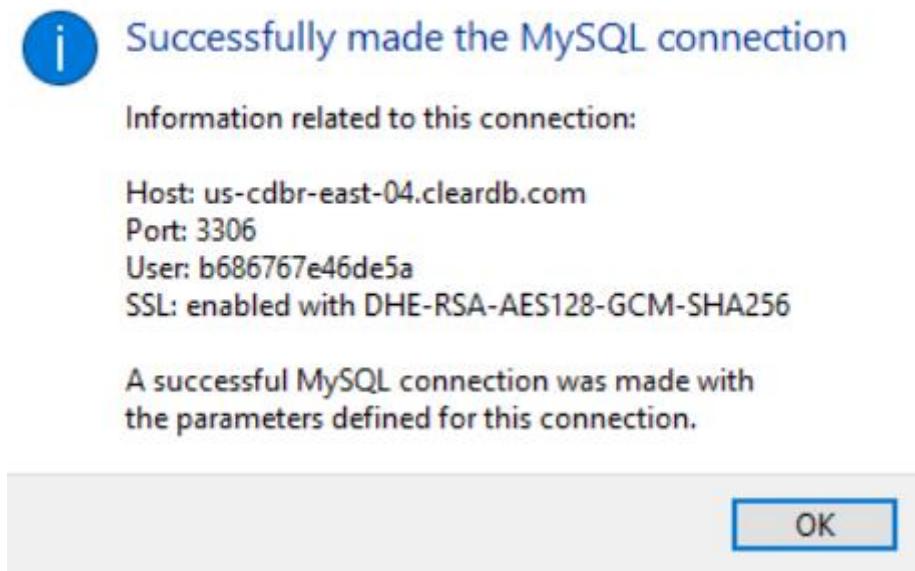


נידרש להקליד את הסיסמה שובה:



ואם הכל תקין, נקבל סימן להצלחה:

### MySQL Workbench



נראה שכעת יש לנו שני חיבורים, המקומי והמרוחק. אם נתחבר לחיבור המרוחק, נראה שיש לנו גישה:

### MySQL Connections ⊕ ⟳

<b>Local instance MySQL80</b> root localhost:3306	<b>Heroku ClearDB connection</b> b686767e46de5a us-cdbr-east-04.cleardb.com:3306
---------------------------------------------------------	----------------------------------------------------------------------------------------

הגישה תהיה מעט מוגבלת, בגיןו למסד הנתונים המקומי. לא נצליח ליצור עוד מסד נתונים או לעשות פעולות מסוימות על משתמשים, אבל כן נוכל ליצור טבלאות. למשל, ננסה לכתוב בלשונית הפקודה:

```
USE heroku_9fc08dd54e7d83a;
```

```
CREATE TABLE 'heroku_9fc08dd54e7d83a'. 'scans' (
  'scan_id' BIGINT NOT NULL AUTO_INCREMENT,
  'timestamp' BIGINT NOT NULL,
  'url' VARCHAR(45) NULL,
  'keyword' VARCHAR(45) NULL,
  'found' ENUM('1', '0') NULL,
  PRIMARY KEY ('scan_id'));
```

רק אל תשחחו להחליף את שם מסד הנתונים heroku\_9fc08dd54e7d83a בזה שלכם.  
תיווצר טבלה זהה לטבלה שצריך על מנת שהאפליקציה תספיק לסריקת אתרים, שעליה הסברתי בפרק על דיפלומנט, תעבור.

## חיבור מסד הנתונים לאפליקציה

עכשו צריך להזכיר את נתוני ה-HOST, שם מסד הנתונים, שם המשתמש והסיסמה אל האפליקציה. איך עושים את זה? מן הסתם לא בקוד, אלא דרך משתני סביבה. את משתני הסביבה מגדרים ב-**Heroku** באמצעות כניסה אל ההגדרות של האפליקציה **:Config Vars** וגלילה אל **Config Vars**

The screenshot shows the Heroku dashboard for the app 'infinite-harbor-10475'. The top navigation bar includes 'Personal' and the app name. Below it, a navigation menu has 'Settings' selected. The main section displays 'App Information' with details like App Name (infinite-harbor-10475), Region (United States), Stack (heroku-20), Framework (Node.js), Slug size (33.8 MiB of 500 MiB), and Heroku git URL (<https://git.heroku.com/infinite-harbor-10475.git>). At the bottom, there's a 'Config Vars' section with a note about how they change app behavior and a 'Reveal Config Vars' button.

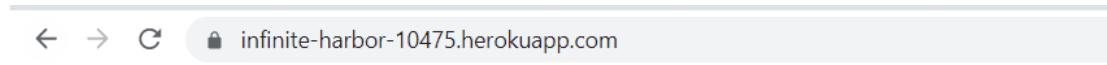
נלחץ על הכפתור Reveal Config Vars ונכנס לשם את משתני הסביבה שאנו צריכים:

The screenshot shows the 'Config Vars' section of the Heroku dashboard. It displays several environment variables with their values:

KEY	VALUE
CLEARDB_DATABASE_URL	
DB_HOST	us-cdbr-east-04.cleardb.com
DB_USER	b686767e46de5a
DB_PASS	
DB_DATABASE	heroku_9fc08dd54e7d83a
KEY	VALUE

A 'Hide Config Vars' button is visible at the top right.

אחרי זה, כל שנותר לנו לעשות הוא לבדוק את האפליקציה ולראות שהיא עובדת.  
במקרה שלנו, מדובר בסריקת האתרים:



## Search result for: example

Insert text for scan		Scan		
ID	TimeStamp	URL	Keyword	Found (1 if yes, 0 if not)
5	1632305769990	<a href="https://example.com">https://example.com</a>	example	1
15	1632305769968	<a href="https://internet-israel.com">https://internet-israel.com</a>	example	0

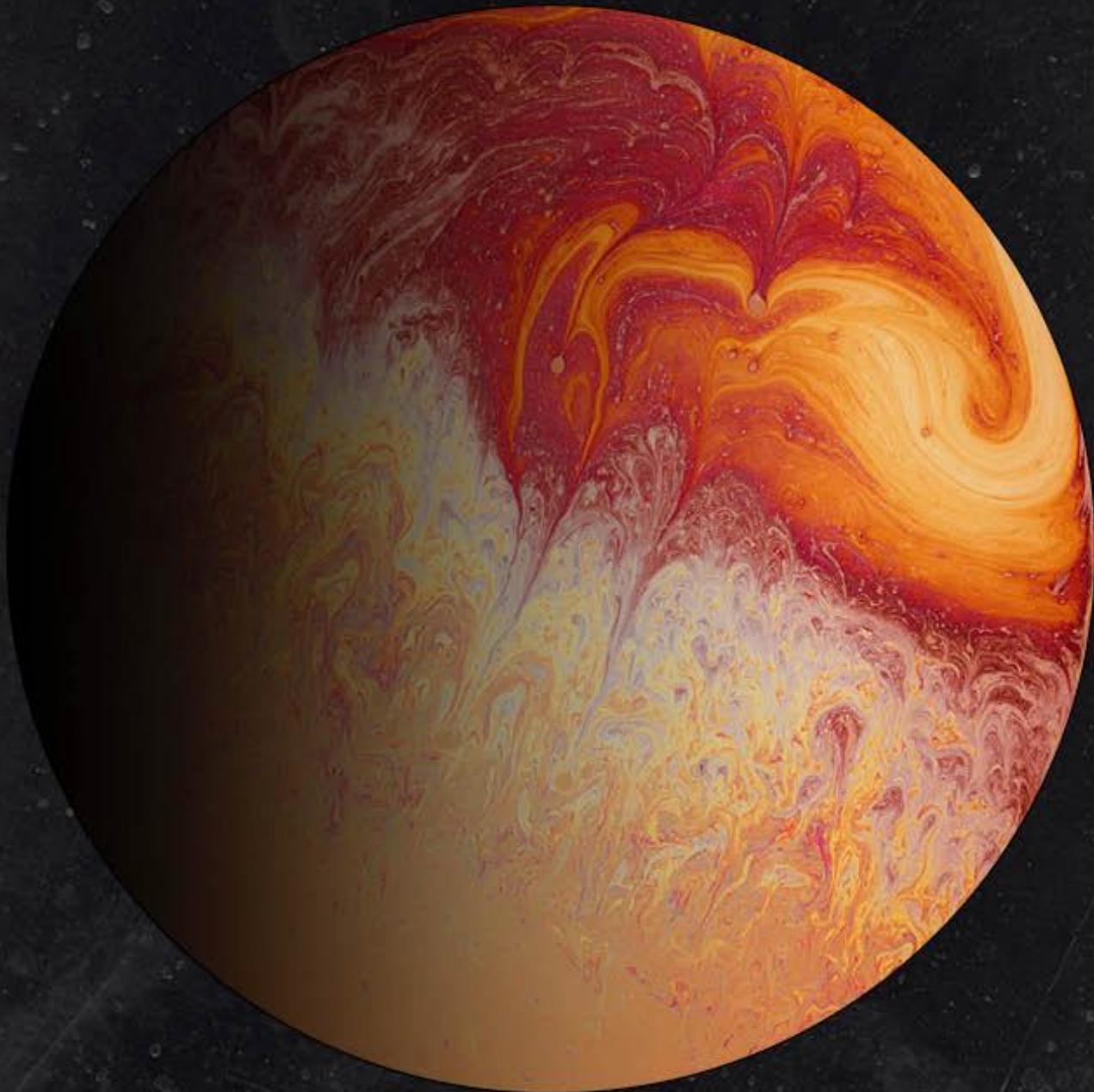
Heroku אינה מוגבלת רק לשורת הפקודה; אפשר לעשות דיפלומנט אוטומטי בכל גרסה (release) בגיטהאב או בשירותים אחרים, ובהחלט אפשר לחזור את Heroku ואת המדריך המצוין שלו למשתמש. השירות, אגב, נותן בחינם עד גבול מסוים, ומעבר לו צריך לשלם. כך Heroku וספקיו ענן עובדים – הםאפשרים למפתחים, כמונו, להתנסות בשירותים שלהם באפליקציות קטנות, מתוך הנחה שמתכנת שמכיר אותם גם יעדיף להשתמש בהם או שילחן על הנהלה להשתמש בהם. Heroku היא שירות נहדר, והאפשרות לעבוד איתה באופן חינמי לחולוינו יכולה להקל מאוד.

## לסיכום

חשוב לזכור שדברים יכולים להשתנות. ClearDB, למשל, יכולה להיסגר ואת מקומה יתפוך ספק אחר, העיצוב של האתר יכול להשתנות, וגם מהות הפקודות, והתהליך של דיפלומנט ראשוני, במיוחד למתקנים בתחילת דרכם, אינו קל. אבל כאן נכנס הניסיון לתמונה. השגיאות, ההטעמאות בהן, הקללות שתקללו אותי בהן מפני שלא כתבתי על טעות מסוימת או כי משאו יהיה לא ברור – הן הניסיון, ובלי הניסיון זהה לא תלמדו ולא תהיו אנשי מקצוע טובים.

פרק 7

# תרומה להוד פתוח



## תרומה לקוד פתוח

תרומה לקוד פתוח היא קריטית במיוחד למי שמחפש את העבודה הראשונה שלו בתחום, בין שמדובר בסטודנטים במדעי המחשב ובין שבוגרי קורסים בבתי ספר לתכניות, בוגרי בוטקאמפים ותוכניות הכשרה או מי שלמדו בלימוד עצמי. לא משנה מהו הרקע שלכם – אחד הדברים שמחפשים בעבודות, גם בעבודה ראשונה, הוא ניסיון, ולא מעט אנשים תהים – אם כולם מבקשים ניסיון – איך מושגים את הניסיון הזה? אז הינה אחת האופציות הקיימות.

- תרומת קוד פתוח היא ניסיון לכל דבר ועניין ובkoroot החיים היא אינדיקציה חיובית ליכולות של המועמד. מדוע? כיוון שתרומת קוד פתוח מראה על:
1. יכולת שימוש בגיט וב-flow git והבנה שלהן.
  2. יכולת הבנה של קוד שלא רק המועמד כתוב.
  3. הבנה של צורך.
  4. תקשורת עם צוות מפתחים שמנהלו את הריפוזיטורי ויכולת לתקשר את השינויים שבוצעו.

ככל שהריפוזיטורי שתרמתם לו גדול יותר ונחשב יותר, כך הערך של תרומתכם גבוה יותר. אם תרמתם קוד לריפוזיטורי גדול, כמו למשל React, Angular, ויו, אקספרס, nodemon או כל מודול אחר שמשתמשים בו מיליוןיי אנשים – אם בקוד שכתבתם משתמשים מיליוןיי מפתחים – האם לא מדובר באינדיקציה חד-משמעות ליכולות שלכם? התשובה היא – ודאי שכן, ובתהליך קבלה לעובדה למחרי משתמשים על תרומה לקוד פתוח ועל ניסיון.

בחילק הזה, החלק האחרון בספר, אסביר על תרומה לקוד פתוח, אדגנים וראאה לכם את הנטייב ואיך לתקשר את התרומה זו בkoroot החיים על מנת שהניסיון הזה יביא אתכם לעובדה הראשונה בתחום. זהו נטייב שתצרכו ללקת בו בעצמכם, אבל מי שילך בו יכול למןף את התרומה זו בראיונות עבודה ולהשתלב בתחום. את הידע הזה העברתי לבן הגדל שלו ולאנשים שישו עתי להם במצב העבודה הראשונה, וזה אולי החלק הכי חשוב בספר זהה – ואפילו בכל הספרים שכתבת!

## מהו קוד פתוח

המערכות, המודולים וחלקמשמעותי מהקוד שהראיתי גם בספר זהה וגם בספרים הקודמים הם בקוד פתוח. מה זאת אומרת קוד פתוח? זה אומר שהקוד שלהם גלוי וגם נמצא תחת רישיון קוד פתוח שמאפשר לכל אחד לעשות שינויים בהםתוכנה המקורית. כך, למשל, עורק הקוד Visual Studio Code הוא בקוד פתוח בGITהאב:

<https://github.com/microsoft/vscode>

כל אחד יוכל להוריד את הקוד, לעשות בו שינויים ולהפיץ אותו ללאה לפי תנאי הרישיון, בגיןוד לمؤلفי תוכנה אחרים שמוסרים הקוד סגור והוא מקור שלהם לא חשוב. יש לא מעט מוצרי תוכנה הוווב הוא עולם של קוד פתוח ברובו. המודולים שהשתמשנו בהם ב-node.js? ריאקט והקומפוננטות? כל ה-build, הדיפולימנט ובדיקות הקוד שהדגמנו איתן? קוד פתוח. גיט עצמה? קוד פתוח.

מה מרווחים חברות וארגוני קוד פתוח? התשובה היא: מפתחים. בטח יצא לכם לעבוד על מוצר קוד סגור מסוים (למשל וורד) ולהיתקל באגים מעצביים, לקרוא הוראות לא מדויקות או לחשב על תוספת (פייצ'ר) יפה שכדי להוסיף. מה אפשר לעשות עם הרצון הזה חוץ מהתלונן לתרמינה? שום דבר. במוצר קוד פתוח, מפתח יכול לתרום קוד ובכך לתקן את ההוראות, שנקרו גם דוקומנטציה, לתקן את הבאג, לתקן את בעיית האבטחה, להוסיף בדיקות אוטומטיות או אפילו פיצ'ר שלהם. ככלומר, החברה מרוויחה מוצר ממש טוב ולא עולה. למשל אוטומטי, החברה שמאחורי וורדפרס, פיתחה את וורדפרס בקוד פתוח. היא מרוויחה מהאחסון שלו וורדפרס, והעולם כולו מרוויח ממערכת מתוחזקת היטב. חברת RDDheat, שעומדת מאחורי מערכת הפעלה לינוקס בקוד פתוח, מרוויחה מהתקנות של מערכות שלא בחברות מסחריות, אבל הקוד הפתוח מאפשר למוצר שלה להיות ממש טוב. חברת מיקרוסופט, שב עבר יצאה נגד הקוד הפתוח, מאוד פעילה היום בעולם זהה ומחזיקה בעצם את גיטהאב ואת זוקה. הרוח שלה הוא מתשתיות הענן שלה, אז'ור, שימושה בתשתיות האלו ומידוד של מפתחים וחברות להשתמש בהן. עוד שהוא שהחברות מרוויחות זה מפתחים שמכירים את הקוד שלהן, וכך קל להן יותר לגייס עובדים. הן לא צריכים לבצע את ההכשרה במוצר הקוד הפתוח שהן משוקות.

מה מפתחים מרוויחים מכך שהם תורמים לקוד הפתוח? מלבד העובדה שהם מסיעים לתוך ליקויים במוצר שהם משתמשים בו או מוסיפים לו פיצ'רים – תרומה לקוד פתוח

היא דרך הנדרת להטבלת ולהרוויח ניסיון מקצועי רב. כך, למשל, יש מתכנתים שאחראים על פרויקטי קוד פתוח שהם יוזמו, שימושים בהם ב מיליון שרתיהם. לא הייתם רוצים לשכור אדם כזה? חברות וארגוני מחפשים לרוב אנשים שתורמים לקוד פתוח.

קוראי שורות אלו בהחלט יכולים לモצרי קוד פתוח ולהרוויח ניסיון משמעותי.

## **מי? אני?**

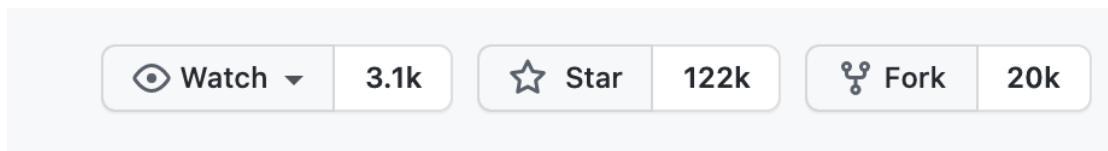
לא כל מתכנת מתחילה יכול לפתח פיצ'ר שלם לモצרים. אף אחד לא מצפה מכם שתתרומו שירות לקוחות של לינוקס, אבל אפשר לתהום קוד לרובה מודולים בלבד פיצ'רים משמעותיים. למשל בדוקומנטציה, בהוספת בדיקות אוטומטיות לモצרים שחרשות לו כלו, בתקוני לקליזיה או בתיקון באגים. יש לא מעט מוצרים נחשים בשוק שאתם יכולים לבחון אם לתרום להם.

כפי שכחתי קודם, התמורה לモצרים קוד פתוח קיים אינה פשוטה, גם אם זה כביכול "משהו קטן". סביר להניח שתצטרכו כמה וכמה ימי עבודה מאומצים לתמורה הראשונה שלכם, אבל מדובר בניסיון של ממש. כל תמורה שתבצעו לאחר התמורה הראשונה תהיה קלה יותר, ואם תתרמו מספיק לモצרים קוד פתוח, יש סיכוי גבוה שיימנו אתכם לـ*co-owners*, שותפים בפרויקט, זהה גם טיטל שמצויבים בקורס החינימ.

## **מה עדיף – פרויקט גמר או קוד פתוח?**

שניהם חשובים, אבל תמורה לモצרים קוד פתוח משמעותי שימושים בו נחשבת יותר מפרויקט גמר. מצד שני, בעבודה על פרויקט גמר מרוויחים היכרות עם מוצרי קוד פתוח שאפשר לתרום להם. אני הייתי אומר – גם וגם. פרויקט גמר (מטעם המכללה, בית הספר או אפילו למען יעד מסוים) הוא חשוב וכדי לציין אותו בקורס החינימ, אבל תמורה לקוד פתוח נחשבת הרבה יותר כיון שהיא אומרת המון על המתכנת שכתב את הקוד ועל ההבנה שלו, כפי שכבר ציינו.

אין פרויקט קוד פתוח זהה למשנהו. ככל שיש יותר הורדות חדשות לפרויקט יותר כוכבים (המקבילה של גיטהאב ל"לייקים"), כך הוא נחשב יותר. אפשר לראות את הכוכבים והפעולות פה:



במודולים של וקח אפשר לראות את מספר ההורדות בדף של המודול:



העדיפות היא לתרום לפרויקט שיש לו מספר מכובד של כוכבים. כמה זה מכובד? יותר מ-1,000, ואם לא כוכבים, אז הורדות. כמה? עשרות אלפיים. בפרויקטים קטנים יותר קל יותר לתרום (אם הם פעילים), אך התרומה נחשבת פחות. מצד שני, כמה שיותר – יותר טוב, ואפשר להתחיל בפרויקטים קטנים יחסית.

## איתור פרויקט שכדי לתרום לו

از מאיפה מתחילה? ראשית, מהה שלמדתם וברכיבים שמכירים טוב. אם למדתם ריאקט – השתמשתם בוודאי בקומפוננטות מסויימות שאתם מכירים היטב. ב-`Node.js` יש בוודאי מודולים מעוניינים שיוצא לכם לשימוש בהם. בגודל – כל רכיב ורכיב בפרויקטים המוצגים כאן ראוי לתרומה ואתם אמורים להכיר אותו ولو באופן שטхи. יש כאן הרבה מקום לטעם אישי ולהעדפות אישיות. יש מי שמתחרב יותר לקומפוננטות של ריאקט, יש מי שמתחרב יותר למודולים של אקספרס, ויש מי שדוקא נהנה יותר מ-`CLI`. כל אחד והפרויקטים שהוא מתחרב אליהם יותר. פשוט תיצרו בפרויקטים בגיטהאב ובקוד שלהם. האם אתם מבינים את הפרויקט? האם הוא מושך אתכם?



כשתחבנו את הריבזיטורי של הפרויקט, שימו לב אם יש שם באגים. הבאגים נמצאים בלשונית issues:

The screenshot shows the GitHub repository for cheeriojs/cheerio. The Issues tab is selected, showing 16 open issues. One specific issue is highlighted: #2096, titled "Unable to extract text from td". The issue was opened 13 days ago by bakman2. Below it, another issue is partially visible: "#2087 Selector types mismatch". Other issues listed include "README out of sync with v1.0.0-rc.10", "Provide method to read buffers with unknown encodings", and "RangeError: Maximum call stack size exceeded". The interface includes a search bar at the top and a sidebar on the left.

לעתים חלק מהבאגים מסומנים כ-good first issue וכשםם כן הם – באגים או נושאים שקל לטפל בהם:

The screenshot shows a specific GitHub issue for the cheerio repository. The issue is titled ".attr(name) should lowercase name property" and is categorized as a Bug. It is also marked as a "good first issue". The issue was opened on Oct 21, 2014, by psayre23. The issue description is partially visible.

בדרך כלל באגים הם דרך טובה לטפל בקוד. אם נראה לכם שאתם מוכנים, עשו פורק לריבזיטורי, בצעו את התיקון והצינו פול ריקווסט.

אפשר לבדוק גם בדף זהה את כל הבאגים שמסומנים **Causing good first issue** בכל הריפורזיטורים שיש בגייתהאכ:

<https://github.com/topics/good-first-issue>

דרך נוספת לתרומות היא לנתח בדיקות אוטומטיות, אם למוצר מסוים חסרות בנכאלו. אם יש coverage ב-README והוא נמוך מדי, אפשר להעלות אותו. אם אין בכלל coverage, אפשר להציע להוסיף צזה:

## cheerio

Fast, flexible & lean implementation of core jQuery designed specifically for the server.

[build](#) [passing](#) [coverage](#) 96% [backers](#) 31 [sponsors](#) 120

אם אתם לא יודעים איך להוסיף דבר צזה – זה הזמן לחקור. הספר הזה נותן בסיס לעובדה, אך כאמור לא יכול ללכט בנתיב זהה במקומכם. יש לכם את כל המידע כדי להבין איך מוסיפים בדיקת coverage לקוד ו איך מציעים את זה בפועל ריקווסט.

בדקו את הפול ריקווסטים ואת הפעולות בפרויקט טרם התרומה.  
לפניהם שתחללו תרומות, מומלץ שתקרוואו את הפול ריקווסטים שכבר יש לפרויקט ובמיוחד את אלה שעברו מרג'. האם הם מטופלים במתירות? מהו הנוסח שרוב הפול ריקווסטים עבדו לפיו ומהו נוסח הנכונות? בחלק מהפרויקטים, במיוחד גדולים ונחשים יותר, יש מדריך לתרומות קוד שמקשור מה-README ומומלץ לעבורי עליו. אם לא עברו עליו ותנסו לתרום, אבל לא לפני הסטנדרט שהפרויקט מצפה לו, הפול ריקווסט שלכם עלול להיזחות.

## דוגמאות לתרומה לקוד פתוח

אתן דוגמה מתרומות קוד של בני, עומר בר-זיק. כשהיה לקראת סוף שירותו הצבאי (ביחידה לא טכנולוגית), המלצתי לו לחפש פרויקטי קוד פתוח כדי לתרום להם ולהעシリ

את הרזומה שלו. את הידע שפירטתי בספר זהה העברי לו, ובמיוחד את החשיבות של תרומת קוד פתוח. ביקשתי את רשותו להשתמש בשתי דוגמאות של תרומת קוד שלו.

## תרומה ל-**Verdaccio**

זהרי דוגמה הנדרת לתרומת קוד כי היא תרומת לדוקומנטציה. קל מאד לזלزل בתרומה כזו כיון שככלול היא לא מחייבת ידע טכני, אבל מי שתורם קוד בדוקומנטציה מראה שיש לו ידע טוב באנגלית ושהוא יודע לעבוד עם גיט, לתקשר עם אנשים, לכתוב פול ריקווסט ולהבהיר את רצונו, וזה בפרויקט של 22 אלף כוכבים שיש לו שני מיליון הורדות בשנה. האם לפי דעתכם זה לא נחשב לניסיון? ברור שכן.

הערה שעומרי בחר לתקן היתה בעיה שהוא שם לב אליה כאשר השתמש במוצר במסגרת הפרויקט שלו. הדוגמה שהובאה שם לא עבדה. תיקון פשוט, אך שימושו לבטקסט של הפול ריקווסט, עד כמה הוא מפורט וסביר מה הבעיה. עומרி הסתכל על פול ריקווסטים אחרים שהיו שם וניסח את הפול ריקווסט שלו בדיק באותו אופן, דבר שהקל את אישור התרומה שלו.

<https://github.com/verdaccio/verdaccio/pull/2230>

זו דוגמה לתרומה קטנה ולא מרכיבת טכנית. לא פיתוח פיז'ר, לא בדיקה אוטומטית, אבל תרומה לפרויקט ממשועורי שאומרת המון על האדם שתרם, גם אם הוא לא תרם קוד ממשי. אולי התרומה זו לבדה לא תעsha את האימפקט ממשי, אבל כשהיא תעמוד לצד תרומות אחרות – ודאי יודאי שכן.

## תרומה ל-**dockly**

ה מוצר הזה כתוב ב-`Node.js` והוא CLI לניהול דוקר. זה נשמע מסובך, אבל עומרי השתמש בו בפרויקט אחר שלו, שעשה בשביל ההפוך, והכיר אותו. וזה הכלyi חשוב - ההיכרות שלכם עם המוצר:

<https://github.com/lirantal/dockly>

למוצר היה באג:

<https://github.com/lirantal/dockly/issues/124>

בכל GANG יש הוראות שחזור, והמעקב אחריהן הוא חלק ממשמעות מהפתרון. עומרី הצליח לשחזר את GANG וברגע ששחזר אותו, הצליח לפתור אותו.

הוא עשה פורק לקוד של המוצר, יצר בראנץ' ואז הכנס לבראנץ' את התיקון. הוא יצר פול ריקווסט מהבראנץ' שלו:

<https://github.com/lirantal/dockly/pull/153>

כדי לשים לב לטקסט של הפול ריקווסט. יש קישור למספר GANG והסביר בהיר על GANG עצמו ועל התיקונים. עומרី הצליח לתרוך מצוין GANG ואת התיקון:

OmriBarZik commented on May 9

**Summary**

Fixed issue #124 by destroying and creating the necessary widgets on each view load.  
this fixed the memory leak and enable us to use the same keyboard shortcut on different views.

**Proposed Changes**

- Added a method to `clearWidgets`
- Added a check to `list.widget.template.js` to check if `searchInput` is present

**Checklist**

- I added tests
- I updated the README if necessary
- This PR introduces a breaking change
- Fixed issue [MaxListenersExceededWarning: Possible EventEmitter memory leak detected.](#) #124
- I added a picture of a cute animal cause it's fun

ואכן, הקוד נכנס לגרסה הבאה של המוצר ואנשים יכולים להשתמש בו. היה GANG, נפתר.

## הכנסת התרומה לקוד הפתוח אל קורות החיים

### פרסום פרופיל הגיטהאב שלכם

העשירו את פרופיל הגיטהאב שלכם כמה שאפשר, עם תמונה פרופיל, תיאור ותרומות מועדפות. אפשר כמובן גם להציג README בדף האישי ולעצבו כרצונכם. אחרי כן פרסמו

את הקישור אליו בכל מקום אפשרי: בפרופיל שלכם בפייסבוק או בלינקדאין וכמוון בקורות החיים שלכם, לצד הקישור ללינקדאין. מקובל למקם את זה מיד לאחר התקציר עליהם.

## פרסום התרומה לקוד פתוח

הצלחתם לתרום קוד לפרויקט ממשמעותי? זה הזמן לפרסם זאת בלינקדאין, בפייסבוק, בטוויטר ובכל מקום אפשרי. זה לגמרי היישג שראוי לחגוג. הצלחתם להבין קוד של פרויקט שאנשים אחרים כתבו, להבין את הצורך, לעבוד עם גיט מספיק טוב כדי לעשות פורק ופול ריקווסט ולתקשר עם אנשים אחרים, ועכשו אנשיים רבים שאתם לא מכירים עומדים להשתמש בקוד הזה! זו בהחלט סיבה למסיבה וראוי לפרסם זאת איפה שאפשר, בצוירוף ההערה שאתם מ Chapman עבודה.

## הכנסת התרומה לקורות החיים

תרומה לקוד פתוח או (אם התמזל מזמן) שותפות בפרויקט קוד פתוח מצילה בהחלט כניסה לקורות החיים ניסיון. ראשית, בתקציר עליהם רשמו את השורה הבאה:

Open source contributer to dockly (3K stars), npq (500 stars) and Verdaccio (12K stars).

שנית, הוסיף חלק (אם אין לכם ניסיון רלונטי, הוא יהיה החלק הראשון) שהכוורת שלו תהיה Open source experience. התוכן יהיה סעיפים שמספרתים כל תרומה. בסעיף יהיו כתובים שם הפרויקט, מהי הטכנולוגיה שלו, מספר הכוכבים/הורדות שלו ומהוות התרומה שלכם, עם הקישור לפול ריקווסט. למשל:

1. Contribution to dockly, CLI for docker management written in Node.js (3K stars). I fixed a memory leakage issue:  
<https://github.com/lirantal/dockly/pull/153>.
2. Contribution to npq, Security CLI in Node.js (500 stars). I fixed a npm v7 issue: <https://github.com/lirantal/npq/pull/196>.

3. Contribution to Verdaccio, Private npm registry in Node.js (12K stars). | fixed the installation instructions in the documentation:  
<https://github.com/verdaccio/verdaccio/pull/2230>.

**שים לב:** כדאי לעשות הῆגה לטקסט זהה (ובכלל קורות החיים) בכל בודק איות, שיבדק גם את התקינות של הניסוח, כמו Grammarly.

החלק הזה יעשה את קורות החיים שלכם בניסיון רלוונטי ובוואדי יגרום להם להתבלט מעל קורות חיים אחרים, שיש בהם רק פרויקט גמר. מתכנת שיסתכל על קורות החיים האלה יבין למי שהצליח לתרום קוד לפרויקטים שימושיים כבר יודע לא מעט ובודאי יצליח להשתלב בתעשייה.

### חשיבות תרומת הקוד גם למפתחים מנוסים יותר

רבים מקוראי הספר זה יראו לנגד עיניהם את הכניסה להייטק כיעד. אך אם יורשה לי, ממרומי גלי – שיפור היכולת המקצועית לא מסתירים עם הקבלה המיווחלת לעובדה הראשונה. עולם ההייטק הוא עולם דינמי וצריך כל הזמן ללמידה, להתפתח ולהתעדכן. דרך טוביה לעשות זאת היא להשתתף בדיונים של קבוצות מקצועיות, להגיע למיטאים ולכנסים או לקרוא ספרים ומארקים מקצועיים. אבל דרך טוביה עוד יותר היא להמשיך לתרום לקוד הפתוח בקביעות גם כשעובדם. מעבר לסיפוק האישני, זה מסיע מאוד למידה – כך יהיה לכם מגע עם מפתחים מתרבותות אחרות, שידאגו לתקן או לעדכן את הקוד שנתרם, תלמדו סטנדרטים חדשים ושיטות חדשות, וכןפ על כך – תעשירו עוד יותר את קורות החיים. הם ייראו אטרקטיביים יותר, ואתם תקבלו הצעות עבודה טובות יותר.

המשמעות על מתכנתים מובטלים בגיל 40 היא רק אגדה, אבל מתכנתים שלא מתעדכנים ולא משתפרים ולא מצליחים למצוא עבודה אחרת היא תופעה קיימת. תרומה לקוד הפתוח היא הוכחה משמעותית ליכולות שלכם ובחילק מהמרקם מסויימת גם בראיון הטכני, שאינו פשוט גם בשלב מתקדם של הקריירה.

פרק 8

# סינום



## סיכום – מה עושים עכשו?

הספר הורכב מכמה חלקים עיקריים: בחלק הראשון למדנו על גיט – המערכת הסטנדרטית לניהול הגרסאות שנמצאת כמעט בכל חברת הייטק והוא גם הבסיס לאקויסיטם של הקוד פתוח. למדנו על גיט מהזוויות הטכנית וגם מהזוויות הקונספטואלית, והכי חשוב – למדנו איך להשתמש בו כדי לתרום קוד.

בחלק השני הובאו פרויקטים לדוגמה: הראשון השתמש בג'אווסהקריפט נקי, בשמו השני ג'אווסהקריפט וNIL, כדי ליצור וידג'ט קטן. השני היה אפליקציה סטטית של ריאקט, והשלישי היה אפליקציה מורכבת של Node.js וMySQL. ראיינו איך מבינים את הרכבים של פרויקט זה, מתכנים ובונים אותו. הדבר החשוב של מדרנו בחלק זה הוא אין בונים פרויקט גמר שאפשר להציג בקורס החיים.

בחלק השלישי דיברנו על דיפלומנט, שהוא חלק חשוב בכל פרויקט, שבו מעלים את התוצר לשרת חי ומציגים אותו לעולם, בין שלוקחות ובין שלמרαιינים טכנולוגיים, שירצטו לראות את המוצר. וכך תוכלו גם להציג את התוצר בקורס החיים.

בחלק הרביעי והאחרון דנו בקוד פתוח. הבאתי דוגמאות של תרומה לקוד פתוח וסבירתי למה ממש כדאי לתרום לו.

הספר הזה על חלקיו השונים יסייע לכל אחד ואחת לחת את הידע התיאורטי של לימוד התכנות ולמנף אותו לבניית פרויקטים ולצבירת ניסיון עבודה ממשית לטובת השתלבות בתחום. כאמור, המידע שהבאתי כאן הוא אותו מידע שלימדתי את בני, את חברי ואת עוקבי ברשותה שהשתלבו בתעשייה: איך לחת את הידע בתכנות ולהפוך אותו לידע בבניית מוצר ממשי, איך להציג את המוצרים האלה ואני להשתמש במידע שנלמד בבניית מוצרים כדי לתרום לפרויקט קוד פתוח, שמיילוני אנשים משתמשים בהם. עם תרומה לפרויקטים כאלה, אף אחד לא יוכל להתעלם מקורות החיים שלכם, גם אם אין לכם ניסיון בעבודה מעשית, גם אם הרקע שלכם אינו סטנדרטי, גם אם החזות שלכם שונה. לתעשייה התיאורטי יש מגרעות, אבל יש גם יתרון עצום אחד: תמיד יש מקום לאנשים שיודעים לכתוב קוד טוב ולבוד בצוות, ואת זה בדיק מוכחים כשתורמים קוד.

از מה עושים עכשו? אם יש לכם את הידע התיאורטי בתכנות ובגיט – בנו את הפרויקט שלכם לפי אחד המודלים שהוצגו פה, או במודל אחר למחר.עשו לו דיפלי והכניסו אותו לקורס החיים שלכם. השתמשו במידע שצברתם בפרויקט ובהיכרות שלכם עם

מודולים בקוד פתוח כדי לתרום להם או לפרויקטים דומים. שלוש עד חמיש תרומות לפרויקטים נחשים – זה כל מה שצריך. הכנסו גם אותן לקורות החיים שלכם, המודפסים ובלינקדאין.

קורות החיים שלכם צריכים להיות כתובים באנגלית אם אתם מגישים אותם לחברות שאין חברות ממשלתיות או סמי-משלתיות (בנקים, חברות ביתוח וכו'). בדקו את קורות החיים שלכם בעזרת Grammarly כדי לראות שאין שגיאות כתיב ושכל המונחים הטכניים מדויקים. הגישו את קורות החיים למשרות שנראות לכם, גם אם דורשים בהן ניסיון של שנתיים. עם תרומת קוד מספקת והודות לביקוש הגבוה למתכנתים בשוק, יש סיכוי גבוהה שיזמנו אתכם לריאיון. גם אם הניסיון שלכם הוא בתרומת קוד פתוח בלבד – זכרו, תרומה משמעותית לקוד פתוח היא ניסיון מקצועני לכל דבר ועניין. ממש כן.

ראיונות לתפקידים טכניים בהייטק הם שונים מראיונות לשרות אחרות. יבחנו בהם גם את האישיות שלכם, אבל ראיון טכני הוא לב התהיליך ובכל חברה הוא שונה. התכוונו לראיונות האלה בדיקות שמתכוננים ל מבחן. חזרו את החושים האלגוריתמיים שלכם בעזרת אתרים המספקים אתגרי תכנות, כמו <https://www.hackerrank.com>, ועשו ראיונות דמה עם חברים. מפה – זה עליהם.

תחום ההייטק הוא רחב, ויש אינספור מקצועות ואינספור חברות שמחפשות עובדים. הוא לא שמור רק למסימי מגמה ריאלית, לבוגרי אוניברסיטאות יוקרתיות או לבוגרי ייחידות צבאיות טכנולוגיות. אני יכול להגיד רק על עצמי: למדתי בתיכון בבית ים וקיבלתה 60 בשלוש יחידות מתמטיקה, בצבא הייתה עובדת رس"ר, כשהשתחררתי עבדתי כשומר בбанк. ובכל זאת, הצלחתי להשתלב גם לשגשג, ומأחורי קריירה ארוכה בתאגידים רבים. לאומיים כמתכנת בכיר וכארכיטקט תוכנה, כעיתונאי טכנולוגי העוסק באבטחת מידע וכמרצה בקורס האקדמית אונו. רשמתי 15 פטנטים על תוכנה וכתבתי חמישה ספרים על התחום. אם אני הצלחתי, עם נתוני פтиחה בעיתויים, גם אתם יכולים ויכולים.

**תודה רבה על הרכישה של הספר!**

עבדתי מאוד קשה על הספר זהה: שעותות רבות של כתיבה, הגאה, תיקונים ומעבר על תוכרי הערינה. יותר מ-1800 אנשים תמכו בספר זהה ואיפשרו לו לצאת לאור.

הספר אינו מוגן במערכת ניהול זכויות. כלומר ניתן לקרוא אותו בכל התקן ללא הגבלה. אם מתחשך לקרוא גם מהקינדל, גם מהמחשב וגם מהטלפון הנייד אין בעיה להעתיק את הקובץ שלוש פעמים ולשים אותו בתוך כל התקן. מתוך תקווה שהרוכש והתום לא ינצל את האמון שנתתי בו להעתקה סיטונאית של הספר לאנשים אחרים והפצה שלו. אני מאמין שרוב האנשים הוגנים.

העתק זהה נמכר ל:

yaniv.harpaz@gmail.com

בנוסף לדף זה - הקובץ מסומן בטבעת אצבע דיגיטלית - כלומר בתוך דפי הספר נჩבים אים פרטי הרוכש באופן שkopf למשתמש. כדאי מאוד להמנע מהעתקה של הספר לאלו שלא רכשו אותו באופן חוקי. אם ברצונכם להעביר את הספר למשהו אחר במתנה - העבירו לו את הפרטים שלכם באתר ומיחקו את העתק שנמצא ברשותכם.

תודה וקריאה נעימה!