מסמך קורס

ענף יסודות

יסודות

כותב: אלמוג וגמן ציפרוט

נכתב: 27.10.2017
עודכן: 05.11.2022

קורס יסודות

הקדמה

מסמך זה מכיל את כל המידע הנחוץ על מנת להגיע לתפקיד בענף.
המסמך מקיף ומכיל חומרים אשר ידאגו שהחניך יהיה מוכן לעבודה בצוות הן בפן הטכנולוגי והן בפן המקצועי.
הקורס נבנה מנקודת הנחה שלחניך אין רקע בפיתוח אתרים אך היא מתאימה גם לחניכים עם ידע קודם.
במידה והחניך מכיר נושא מסויים לעומק, יוכל אותו חניך לגשת לחונך ולקבל הדרכה לגבי איך לגשת לאותו הנושא או במידה והבנת החניך רחבה, יאשר לו החונך לדלג עליו.

הקורס יתמקד בנושאים הבאים:

- הבנה בסיסית של מה זה אתר ואיך הוא עובד.
- בניית אתר בסיסי בעזרת HTML ו־CSS.
- לימוד Javascript באופן יסודי ומקיף.
- בניית צד שרת ב־NodeJS.
- עבודה נכונה בצוות.
- ניהול קוד בצורה נכונה עם Git.
- לימוד דרך העבודה של הענף.
- תכנון מבנה פרויקט.
- הבנת הצורך בבדיקות.
- תכנון מערכת שלמה.
- מבצוע מערכת שלמה.
- הבנה בסיסית של תחום ה־DevOps.
- פריסת הקוד על הפלטפורמה בעזרת CI/CD.

בכל נושא יינתן לחניך מקור מידע עיקרי ממנו מומלץ ללמוד את החומר. בנוסף למקור המידע העיקרי, יינתנו מקורות מידע נוספים מהם ניתן ללמוד חלקים מסוימים. תוך כדי המעבר על החומר יינתנו משימות תרגול בסיסיות שעל החניך לעשות. משימות אלו יבדקו על ידי החונכים ויהוו שלב הכרחי על מנת לעבור לחלק הבא.
במהלך הקורס החניך יפתח פרויקט מרכזי בשלבים. הפרויקט יכלול בתוכו את כל נושאי הקורס ויהווה פרויקט גמר.

מבוא

הענף שלנו נוקט בגישה של קוד פתוח ושימוש בטכנולוגיות חדשות ומתקדמות, לכן, אנו מתעסקים במגוון רחב מאוד של טכנולוגיות, דבר אשר לא מאפשר כתיבת קורס אחד מקיף.
מסיבה זו, חילקנו את הקורס למספר חלקים:

1. **חלק ראשון - צד לקוח.**
   חלק זה מתמקד בבסיס של בניית אתרים, הוא יסביר מה זה אתר ואיך בונים אותו.
   את חלק זה כל החניכים יעשו.

2. **חלק שני - צד שרת וניהול פרויקט.**
חלק זה מתמקד בצד השרת בכללי וב־NodeJS בפרט, בנוסף, חלק זה יציג את המבנה הרצוי לפרויקטים בענף ואיך יש לעבוד בצוות.
לכן, לא כל החניכים יצטרכו לגשת לחלק זה, דבר אשר תלוי בצוות אליו הם יגיעו.
3. **חלק שלישי - Frameworks.**
חלק זה עוסק ב־Frameworks בהם נעבוד בצד הלקוח, ומתמקד ב־React.
4. **חלק רביעי - מתודולוגיות פיתוח וארכיטקטורה**
חלק זה מלמד על ארכיטקטורת פרויקטים והטכנולוגיות \ הדרכים המאפשרים לממש אותן.
5. **חלק חמישי - העשרת חובה - DevOps**
חלק זה מלמד על תחום ה־DevOps תוך כדי מיקוד בענף שלנו.

מסמך זה לא מיועד להיות מקור הידע שלכם, הוא רק נועד להדריך אתכם ולתת לכם קו פעולה כללי לדרך הלמידה.
על מנת שתדעו שאתם לומדים כראוי, ניתן לכם משימות אותם תצטרכו לבצע.
משימות אלו יעזור לכם להבין שאתם לומדים במסלול הנכון ולא הולכים לאיבוד בחומר.

בהצלחה!

נהלי קורס

קיימת לקורס קבוצה ב־GitLab, יש לעלות אליה תרגילים כשמתבקש.
על מנת לקבל הרשאות לקבוצה יש לפנות לחונך.

בנוסף, קיימת תיקייה ב־Google Drive. אל תיקייה זו יש להעלות את כל התשובות לתרגילים **בסיום** כל תרגיל (בקש מהחונך שיצור תיקייה על שמך).

יש לעבוד על כל תרגיל בקורס **לבד** אלא אם צוין אחרת.

בכל בוקר יתקיים סטטוס קורס אשר יבדוק את התקדמותכם בה:
- אין לאחר לסטטוס, יש להודיע מראש על אי-יכולת הגעה בזמן.
- יש לעלות בעיות כלליות בסטטוס, ופרטניות מול החונך.
- יש להציג את ההתקדמות שלכם בקורס.

לפני שאתם מתחילים

הקורס מתבצע בשיטת הלימוד העצמי. בכל חלק מוצעים מספר מקורות למידה. המגוון הרחב במקורות למידה נועד לאפשר לאנשים עם שיטות לימוד עצמי שונות, להבין את החומר הנלמד בצורה הקלה ביותר עבורו.
לכן, אתם יכולים לבחור את המקור המתאים ביותר עבורכם.
אך שימו לב, ישנם מקורות אשר רשמנו שהם פחות מומלצים, כמו הקורסים ב־Udemy, זאת עקב אורכם הארוך והקושי של רוב האנשים להתרכז בהם וללמוד מהם בצורה יעילה. לכן, אנו ממליצים ללמוד מן המקורות הטקסטואליים.

חלק ראשון - צד לקוח

חלק זה בקורס יעזור לכם להבין איך אתרי אינטרנט עובדים, איך בונים אותם, איך מעצבים אותם ואיך גורמים להם לעשות בדיוק מה שתרצו שהם יעשו.

מקור הידע הראשי בחלק זה יהיה מסמך לימוד בניית האתרים של MDN.
אינכם חייבים לעבור על כל מקור זה, אם יש דברים אותם אתם כבר בטוח יודעים, אתם יכולים לדלג עליהם.

מה זה אתר אינטרנט?

אתר הוא בעצם איגוד הקבצים שהלקוחות (אתם) מקבלים בגישה אל כתובת תקנית של אתר אינטרנט.
בגישה אל הכתובת, השרת של האתר אליו אתם מנסים לגשת, ישלח לכם את הקבצים אשר מרכיבים את האתר (במידה והשרת "יסכים" לשלוח לכם את הקבצים). אותם קבצים יכולים להכיל קבצי HTML, CSS, javascript, תמונות, סרטונים וכדומה.
כאשר הקבצים יגיעו אל הדפדפן (הלקוח), הדפדפן "יתרגם" את הקבצים ויציג אותם, כך תוכלו לראות את האתר.

מבוא

לפני שתתחילו ללמוד איך לבנות אתר, עליכם להבין מה זה אתר ואיך הוא פועל.

במודול Getting started with the Web קראו את:

- Installing basic software
- What will your website look like
- Dealing with files
- How the web works

שאלות מבוא

ענו על השאלות הבאות:
(ניתן גם לקרוא ממקורות אחרים על מנת לענות על השאלות).

1. הסבר במילים שלך מהם המושגים הבאים:
   a. HTML
   b. CSS
   c. JavaScript
2. ענה על השאלות הבאות בפירוט:
   a. מה קורה כאשר אני פונה אל כתובת מסוימת בדפדפן, למשל אל www.google.com?
   b. מה ההבדל בין צד לקוח לצד שרת?
   c. מה זה localhost?

HTML

HTML הינה שפת תגיות אשר מייצגת את מבנה ותוכן דפי האינטרנט. זו היא **איננה** שפת תכנות ותפקידה הוא לייצר שלד של אלמנטים אותם הדפדפן יכול "לקרוא" ולהמיר לתצוגה ויזואלית ללקוח.
קראו את:

- HTML basics
- What's in the head? Metadata in HTML
- HTML text fundamentals
- Creating hyperlinks
- Document and website structure

מומלץ להיעזר ב־W3schools על מנת ללמוד אילו אלמנטים קיימים ואיך להשתמש בהם.
אין צורך ללמוד לעומק כל אלמנט, אך מומלץ להכיר את כולם ולדעת על קיומם.

CSS

CSS הינה שפה שנועדה לתאר איך דף אינטרנט (הכתוב ב־HTML) צריך להראות.

קראו את:

- CSS basics
- CSS syntax
- w3schools CSS Tutorial
- CSS tricks FlexBox
- CSS tricks CSS grid

לאחר הקריאה, בצעו את המשימות הבאות:

משימות HTML ו־CSS

בענף שלנו, בדומה לחברות אזרחיות, ישנו לרוב מעצב אשר אחראי על מראה האתר. על המפתחים בענף לבנות אתר אינטראקטיבי על בסיס ה"תמונות" שהמעצב הכין.
לפניכם דוגמאות לעיצובים. בעזרת HTML ו־CSS כתבו את האתרים הבאים:
בהמשך, מצורפים קישורים למקורות שיכולים לעזור לכם בתרגילים.

הדגשים:

1. אין להשתמש ב־javascript בתרגיל.
2. האתר צריך להיות רספונסיבי - יש לדאוג שהאתר יתפקד בצורה תקינה במסכים שונים ובשינוי גודל הדפדפן (אין צורך להתאים ל־Mobile).
3. אין צורך להוסיף את הפרסומות שיש באתר.
4. יש להשתמש ב־CSS ולא ב־Inline Style. (מדוע?)

משימה 1 - דף התחברות

עצבו דף התחברות (Login) לאתר על פי אחת התמונות אשר נמצאות בדף המשימה הנקרא "משימה 1 - עיצוב דף התחברות" בתיקיית התרגילים.
יש לבחור תמונה אחת ולעצב את הדף בעזרת HTML ו־CSS כך שיהיה דומה ככל הניתן לתמונה שבחרתם.

משימה 2 - עיצוב דף ראשי

עצבו דף ראשי לאתר אינטרנטי לחנות יין בעזרת HTML ו־CSS.
יש לעצב את הדף על פי התמונה "משימה 2 - עיצוב דף ראשי" שבתיקיית התרגילים. הדף שתעצבו צריך להיות דומה ככל הניתן לתמונה.
תוכלו לעשות חנות לאו דווקא ליין, אלא לכל מוצר שתרצו, תוכלו להשתמש בכל תמונה שתמצאו באינטרנט.

מקורות נוספים לעזרה:

- Your first HTML form
- The native form widgets
- Debugging HTML
- CSS Debugging

לאחר סיום המשימות קראו ברפרוף את המקורות הבאים:

- Sending form data
- Form data validation

JavaScript

JavaScript הינה שפת תכנות עילית אשר מאפשרת הפיכה של דף אינטרנט לאינטראקטיבי.
בחלק זה ישנם 3 מקורות מידע עיקריים:

1. The Modern Javascript Tutorial - הינו המקור המומלץ לקורס, זהו מקור טקסטואלי.
2. בנוסף, ישנו גם קורס אינטרנטי ב־udemy בשם JavaScript: Understanding the Weird Parts , זהו קורס ארוך שלא מתאים לכל אחד, לכן מומלץ להשתמש רק במקור הראשון.

   במידה ומסיבה מסוימת אתם מעדיפים להשתמש בקורס, עדכנו את החונך והשתמשו בפרטים הבאים על מנת להתחבר למשתמש של הענף:

   שם המשתמש:
   yesodot.anaf@gmail.com
   סיסמא:
   Crzhk2014!

3. בנוסף, ישנם 2 ספרים בהם ניתן להיעזר:
   a. You Don't Know JS
   b. Eloquent Javascript

למידת בסיס השפה

קראו את An introduction ב־The Modern Javascript Tutorial.
ואת JavaScript Fundamentals.

לאחר מכן, **התחילו את משימת JavaScript** תוך כדי קריאה של המקורות הבאים:

- Document בחלק 2 של The Modern Javascript Tutorial
- Introduction to Events באותו חלק.
- Same Origin Policy
- Mozilla Canvas Tutorial

מקור קריאה נוסף (בונוס):

- JavaScript the right way

משימת JavaScript

כתבו אתר רספונסיבי (מבנו ישתנה בהתאם לגודל המסך בצורה שתשמור על תפקודו התקין ונראותו) בעזרת HTML, CSS ו־javascript אשר יכלול את הדפים הבאים:

דף ראשון - דף ראשי

יכיל קישורים לכל שאר הדפים ובנוסף תופיע בו השעה והתאריך העדכניים (על השעה להתעדכן באופן שוטף).

דף שני - כרטיסיות

- הדף יכיל Form עם 3 שדות Input וכפתור כאשר שדה ה־Input הראשון ייצג שם, השני ייצג מקצוע והשלישי Email.
- כאשר נלחץ על הכפתור הוא יוסיף "כרטיס" בו יוצג שם הבן אדם, המקצוע שלו והדואר האלקטרוני שלו.
- ניתן יהיה להוסיף כמות כרטיסים בלתי מוגבלת.
- הכרטיסים יהיו צמודים אחד לשני באותה שורה, כאשר לא יהיה מקום בשורה הם ירדו למטה, לשורה חדשה, ויהיו מסודרים מימין לשמאל.
- ניתן יהיה למחוק כרטיסים על ידי לחיצה על כפתור מחיקה שיופיע עליהם רק כאשר עוברים על כרטיס עם העכבר.
- יש **למחוק את המילה** "פקיד" אם היא מופיעה באחד המקצועות (ולהשאיר את שאר האותיות שהוכנסו).
- תוודאו שבשדה ה־Email באמת מופיע Email תקין, אם לא, תחליפו את ה־Email ב־"valid@email.com".
- שם בן האדם חייב להכיל לפחות 2 אותיות (לא מספרים, לדוגמא). במקרה והוא לא מכיל 2 אותיות, אל תאפשרו ליצור (או לשמור) את הכרטיס.
- כאשר נסגור את הדפדפן ונפתח אותו מחדש, כל הכרטיסים שיצרנו ישארו. (רמז: יש לשמור אך ורק את התוכן של הכרטיסיות ולא HTML)

דף שלישי - פעולות על מערכים

הדף יכיל 10 כרטיסיות. הכרטיסיות יציגו את המידע מהמערך בקובץ הבא (כל כרטיסיה תציג אובייקט אחד מהמערך, תעתיקו את המערך לפרוייקט שלכם). מעל הכרטיסיות יהיה תפריט אשר יכיל את האלמנטים הבאים:

1. Select Box - יאפשר לבחור שדה (מהאובייקטים) עליו הפעולות יתבצעו.
   השדות אשר יופיעו בבחירה:
   a. Age
   b. Name
   c. Admin
   d. Grades
   e. Address
2. Input (טקסט) - בו יוזן הערך עליו יבוצע החיפוש\פעולות.
3. Button בשם Find - יציג רק את הכרטיסיות שהערך שהוכנס ב־Input מספק את התנאים הבאים בהן:
   a. עבור שדה age - הציגו את כל הכרטיסיות שגילם גדול מהגיל שהוזן.
   b. עבור שדה name - הציגו את כל הכרטיסיות ששמם מכיל את הערך שהוזן.
   c. עבור שדה admin:
      i. במידה והוזן "true" - הציגו את כל הכרטיסיות שהן אדמינים.
      ii. במידה והוכנס "false" עליכם להחזיר את כל היוזרים שאינם אדמינים.

   שימו לב: על הקוד שלכם לא להיות case sensitive. כלומר, במידה והוזן "TrUe", עליכם להציג את כל הכרטיסיות שהן אדמין.

   d. עבור שדה grade - עליכם להציג את כל הכרטיסיות שממוצע הציונים שלהן גדול מהערך שהוזן.
   e. עבור שדה address ו־houseNumber - הערך יוזן על פי הפורמט fieldname.value ויוצגו הכרטיסים שהערך שלהם שווה לערך שהוזן (ה־value).
      דוגמא:
      עבור city.ashdod עליכם להציג את כל הכרטיסיות של אנשים שגרים באשדוד.
4. Button בשם "All Grades Greater Than" - הציגו רק את הכרטיסיות ש**כל** אחד מהציונים שלהן גדול מהציון שהכנסתם ב־Input.
5. Button בשם "Some Grades Greater Than" - הציגו רק את הכרטיסיות ש**לפחות** ציון אחד שלהן גדול מהציון שהכנסתם ב־Input.
6. Button בשם "Array Filter and Manipulation" - הציגו את הכרטיסיות שממוצע הציונים שלהן קטן מהמספר שהוזן ב־Input **ו**מספר הבית שלהם גדול מהמספר שהוזן.
   יש להוסיף לגיל (age) של הכרטיסיות שמוצגות את המספר שהוזן.
   דוגמא:
   אם המספר שהוזן הוא 30, הגיל של הבן אדם הוא 12, ממוצע הציונים שלו הוא 29 ומספר הבית הוא 50 שנו את גיל הבן אדם מ־**12** ל־**12+30** (42) שכן התנאים מתקיימים. במידה והתנאים לא מתקיימים אין להציג את הכרטיס.

דף רביעי - אלגוריתמיקה

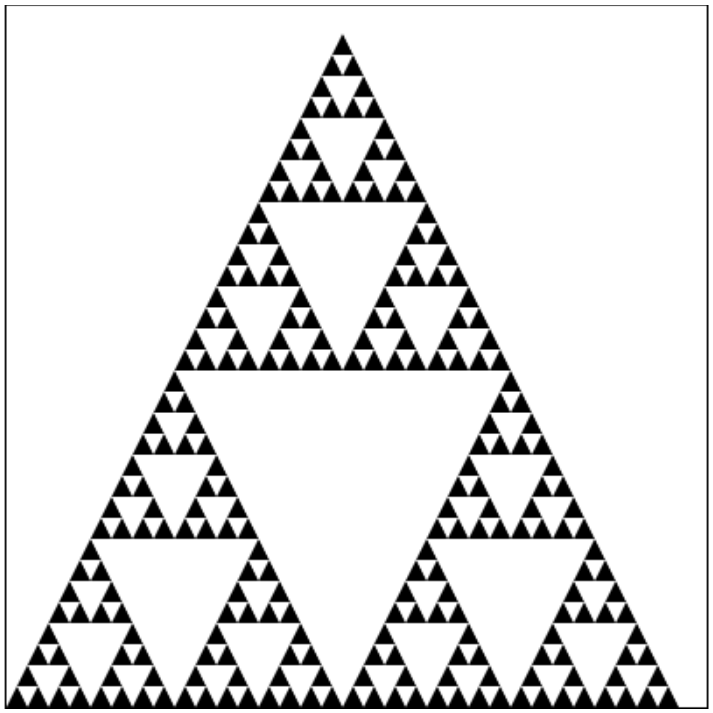
1. החלק הראשון של הדף יכיל שדה Input ו־Button. בהכנסת String ל־Input ולחיצה על ה־Button, שדה ה־input ישנה את רקעו לירוק במקרה וה־string הינו Palindrome ולאדום במקרה ולא.
2. החלק השני של הדף יכיל:
   2 שדות Input ו־2 כפתורים:
   a. שדה Input אחד יתן להכניס מחרוזת באנגלית.
   b. שדה Input שני יתן להכניס מספר בין 0 ל־25.
   c. כפתור בשם Encrypt אשר "יקדם" כל אות בטקסט שהוכנס במספר שהוכנס בשדה ה־Input השני.

   דוגמא:
   במקרה והמחרוזת שהוכנסה היא: "ac" והמספר שהוכנס הינו 3 המחרוזת תשתנה למחרוזת הבאה:
   "df" כל אות התקדמה ב־3 אותיות (a ל־d **ו**־c ל־f).
   d. כפתור בשם Decrypt אשר "יחזיר" כל אות בטקסט כמספר שהוכנס בשדה ה־Input השני (הפוך ממה שמתואר בסעיף הקודם).

דף חמישי - רקורסיה

יש לבצע את 2 המשימות הבאות בעזרת רקורסיה:

- יכיל שדה שיקבל מספר שלם. המספר השלם ייצג את n האיברים הראשונים בסדרת פיבונאצ'י אשר יוצג באותו הדף.
- יכיל 2 Sliders אשר ישנו את גודל כל משולש וכמות המשולשים במבנה הבא אשר יוצג ב־Canvas:



שימו לב לדגשים הבאים:

- סדרו את הקוד בפונקציות במהלך התרגיל.
- אין חובה ליצור עמודי HTML נפרדים, אך חובה לספק קישורים לכל "דף" בנפרד.
- חלקו את הקוד לקבצים שונים.
- יש לתת לפונקציות ולמשתנים שמות רלוונטיים.
- יש להעלות את התרגיל לתיקייה שלכם ב־Google Drive בסיום העבודה ולעדכן את החונך.
- יש לדאוג לעמוד בסטנדרטים, על פי ה־Style Guide של Airbnb.

סטנדרטים

כאשר מספר מתכנתים עובדים על אותו הפרויקט, מומלץ לשמור על אחידות.
כאשר עובדים באופן אחיד בצוות ובענף, קל יותר למתכנתים לקרוא קוד של מישהו אחר ולהבין איך הוא פועל.

בנוסף, כאשר מישהו יקרא את הקוד שלכם לאחר שכבר עזבתם, הוא לא יצטרך את העזרה שלכם על מנת להבין את פעולתו הבסיסית של הקוד, ולא רק זה, גם כאשר אתם תחזרו לקוד לאחר מספר חודשים יהיה לכם קל יותר להבין אותו.

על מנת שנשמור על אחידות, אתם נדרשים לעבוד על פי ה־Style Guide של Airbnb (הסטנדרטים של airbnb ל־Javascript).

בהמשך, כאשר (ואם) תגיעו ל־NodeJS, תוכלו להשתמש ב־eslint אשר ידאג לשמור על הסטנדרטים של הקוד שלכם על ידי בדיקתו וזריקת אזהרות כאשר חלק בו אינו תקין (linter).

שמות משתנים ופונקציות

פונקציה עם שם מובן יכולה להבדיל בין פונקציה שנצטרך לקרוא שורה שורה על מנת להבין את מה שהיא עושה לבין פונקציה שנוכל להבין מה היא עושה על ידי קריאת שמה בלבד.

קראו את המאמר הבא: The art of naming variables.
מעכשיו, עליכם לתת שמות מובנים והגיונים למשתנים ולפונקציות שלכם.

סטנדרטים כללים

חוץ מהסטנדרטים המצויים ב־Style Guide של Airbnb יש לשמור על הסטנדרטים הבאים:

1. שמות משתנים יכתבו ב־camelCase כאשר האות הראשונה תמיד תהיה קטנה, לדוגמא:

```
let personName;
```

2. יש לתת שמות הגיוניים למשתנים, כך שנוכל להבין למה הם משמשים על ידי שמם בלבד.
   a. אין לכתוב הערה על שם משתמש, במקרה ואתם מרגישים שיש צורך בכך, זה אומר שהשם שלו לא טוב מספיק.
3. קבועים (const) אשר מהווים שמירת ערך Hard Coded כמו מספר ספציפי (למשל ערך pi) יש לכתוב באותיות גדולות כאשר הרווח בין המילים יכתב בעזרת '_' (קו תחתון), לדוגמא:

```
const ERATH_GRAVITY = 9.807
```

4. יש לחלק פונקציות לפיסות קוד קטנות, כאשר כל אחת עושה משהו ספציפי (מבלי להגזים).
5. יש לתת לפונקציות שמות שמייצגים פעולה (שמות פעולה), לדוגמא:

```
function getNumberOfPeople()...
```

הערות בקוד

ישנם מקומות אשר מחייבים את המפתחים שלהם לרשום הערות בקוד, לפעמים על כל פונקציה. דבר זה יוצר קוד עמוס ולא קריא, אשר מקשה על תחזוקתו והבנתו.
בנוסף, גישה זו נותנת לגיטימציה למפתחים לרשום פונקציות לא ברורות, אשר הבנתן תלויה בהערות שלהן בלבד.

אנו נשתמש בחוקים הבאים על מנת להבין מתי לרשום הערה על הקוד שלנו ומה לרשום בה:

1. פונקציות בפרט וקוד בכללי אמורים להיות "Self Documenting" עד כמה שניתן. כלומר, הקוד צריך להיות פשוט מספיק על מנת שניתן יהיה להבין מה הוא עושה מבלי להוסיף לו הערות. את זה ניתן לעשות בעזרת:
   a. שמות משתנים אשר מסבירים באופן ברור מה תפקידו של המשתנה (במקום לרשום i, תרשמו personIndex, או במקום לרשום score, תרשמו "currentGameScore"
   b. שמות פונקציות יסבירו את תפקידה של הפונקציה, ותמיד יציינו שם פעולה (מפני שפונקציות הן פעולות). לדוגמא: "person" אינו שם תקין לפונקציה והוא אינו מסביר מה היא עושה, לעומת זאת "getPerson" מסביר בדיוק מה המטרה של הפונקציה ואפילו מה היא מחזירה לי.
   c. פונקציות יהיו קצרות ופשוטות כמה שניתן על מנת להקל על הבנתן (בנוסף לסיבות אחרות שמעודדות פונקציות קצרות ופשוטות, כמו Reusability).

כאשר נגיע לפונקציה שלא ניתן להבין בקלות את משמעותה ללא הערות, נעשה בה Refactor וננסה לפשט אותה. אבל, מה קורה כאשר לא ניתן לפשט את הקוד?
יש מקרים בהם הקוד הוא פשוט ככל הניתן אך עדיין לא ברור. במקרים כאלו מותר, ואף מומלץ לכתוב הערה שתסביר אותו.

2. "קוד אומר לכם איך, הערות אומרות לכם למה", על פי אמרה זו, הקוד שלכם צריך להראות איך אתם עושים משהו ובמקרה ומה שהם עושים לא ברור מסיבה מסוימת (נגרם מגורם חיצוני למשל) אז תסבירו בהערה למה עשיתם את זה.

בנוסף לחוקים שצוינו, קראו את המאמרים הבאים ותפעלו על פיהם בעתיד:

1. When Good Comments Go Bad
2. Code Tells You How, Comments Tell You Why
3. Putting comments in code: the good, the bad, and the ugly

ניהול קוד

כאשר כותבים פרויקט, מומלץ לעבוד עם כלים שיעזרו לנו לעקוב אחר התקדמות הקוד שלנו, לשמור אותו במקום בטוח, לנהל אותו ולשתף אותו באופן חכם עם חברי הצוות.
לכן, מומלץ להשתמש בכלים ל־version controls, כמו Git, ובשרת repo של git כמו־Github או Gitlab.

Git גם נועד לשיפור העבודה בצוותים וזה הכלי בו אנו משתמשים בצוותים בענף על מנת לשמור על סדר ותקינות הקוד.

מרגע זה, העלו כל פרויקט ל־Repository בקבוצה האישית שלכם ב־Gitlab.
יש להשאיר את ה־master נקי ולא לעבוד עליו לעולם אלא אם צויין אחרת.
עליכם לפתוח branch בשם develop ולעבוד עליו (אפשרי ואף מומלץ לפתוח branches נוספים אשר יצאו ויעשו Merge אליו בהמשך).
לאחר שסיימתם את העבודה על הפרויקט יש לפתוח merge request מ־develop אל ה־master (לבדוק שהוא לא WIP / draft) ולשייך את ה־merge request לחונך.

למדו על git מהמקורות המצוינים בהמשך תוך כדי פתירת התרגילים ב־משימות Git.

קראו את git handbook מ־Try Github.
אתם יכולים לנסות את המקורות מ־Learn by doing באותו אתר:
- Visualizing Git
- Learn Git Branching

מקורות נוספים לעיון:
- oh shit git
- מידע מעמיק על git rebase
- ספר - Pro Git

Commit Messages

קראו את How to Write a Git Commit Message ועקבו אחר הסטנדרטים האלו כאשר אתם כותבים תיאורים ל־commits.

משימות Git

תרגיל 1

העלו את התרגיל שעשיתם במשימת Javascript אל התיקייה שהחונך יפתח לכם ב־gitlab.

תרגיל 2

1. צרו פרויקט חדש בשם "git_exercise" בתיקייה שלכם ב־gitlab.
2. דרך הסביבה הלוקאלית שלכם (המחשב שלכם) צרו קובץ בשם README.md, הוסיפו אותו ל־master בפרויקט שלכם (הלוקאלי). תנו ל־commit תיאור (message) רלוונטי.
3. צרו branch חדש מה־master בשם "readme-update" (בלוקאלי).
4. רשמו "git_exercise_1" בקובץ והעלו אותו ל־remote של ה־branch החדש שיצרתם (צרו בפרויקט שלכם ב־gitlab ענף בעל אותו השם כמו הענף הלוקאלי שיצרתם).
5. צרו branch חדש **מה־master** בשם "readme-update-2" (בלוקאלי)
6. ב־branch החדש שיצרתם, גשו לקובץ README.md (בשלב זה הקובץ אמור להיות ריק, אם הוא לא, מחקו את ה־branch ונסו שוב ליצור אותו מה־master, אם הוא עדיין לא ריק, התחילו את התרגיל מההתחלה) ורשמו בו "git_exercise_2".
7. לאחר ששמרתם את השינויים המקומיים, העלו את ה־branch החדש שיצרתם אל ה־remote שלו.
8. בסביבה המקומית, בצעו merge מ־"readme-update" ל־master.
9. בסביבה המקומית, בצעו merge מ־"readme-update-2" ל־master.
10. פתרו את ה־merge conflict בסביבה המקומית באופן שבו השינויים של "readme-update-2" יכנסו לתוקף ב־master ולא אלו של "readme-update".
11. העלו את השינויים ל־remote.

פעולות אסינכרוניות

JavaScript הינה שפה בעלת תהליכון (thread) אחד, דבר שאומר שהיא יכולה להריץ רק שורה אחת של קוד בכל רגע נתון.

באתרים יש לנו פעולות שלוקחות זמן רב (ביחס לפקודות רגילות), כמו למשל תקשורת עם השרת או גישה ל־dom.
על מנת למנוע Blocking (מצב בו הקוד "תקוע" מפני שהוא מחכה לפעולה ארוכה שתסתיים) נוכל להשתמש בפעולות אסינכרוניות בעזרת callbacks וב־promises.

קראו על Promises ו־Callbacks, ואת המאמר הבא אשר מסביר לעומק איך פעולות אסינכרוניות עובדות ב־JavaScript ובצעו את המשימה הבאה.
בנוסף ניתן לראות את הסרטון הבא ולהשתמש בכלי loupe שמציג איך ה־Call Stack, ה־Event Loop וה־Callback Queue עובדים.

מקורות נוספים:

1. סרטון: JavaScript promises In 10 Minutes
2. סרטון: JavaScript Async Await
3. מאמר: Callbacks, Promises, and Async

משימת אסינכרוניות

1. צרו קובץ JS וממשו בו פונקציית Sleep אשר מקבלת כפרמטר מספר שניות ומחזירה Promise שעושה resolve לאחר מספר השניות שצוין.

2. עבור כלל הפונקציות הבאות, יש להשתמש בפונקציית ה־Sleep על מנת לוודא שה־Promise יחזיר תשובה רק לאחר זמן רנדומלי של שניות, בין 1 ל־5.

   a. פונקציית prefix המקבלת טקסט ומחזירה את הטקסט כ־Promise מוצלח בפורמט הבא:

```
message info: <text>
```

   b. פונקציית isOddArray המקבלת מערך ומוודאת שכל המספרים במערך אי-זוגיים.
   במידה והתשובה היא חיובית, הפונקציה תחזיר Promise מוצלח שמכיל את המחרוזת 'odd'.
   במידה והתשובה היא שלילית, ה־Promise ייכשל.
   במידה ונקלט קלט לא תקין, כגון מחרוזת במקום מערך, ה-Promise ייכשל עם ההודעה "Not an array".

   c. פונקציית sum המקבלת שני מספרים כקלט.
   במידה והמספר הראשון גדול מהמספר השני, הפונקציה תחזיר Promise מוצלח שמכיל "First number is bigger".
   במידה והמספר השני גדול מהמספר הראשון, הפונקציה תחזיר Promise מוצלח שמכיל "Second number is bigger".
   במידה והמספרים שווים או שמוכנס קלט לא תקין, ה-Promise ייכשל עם ההודעה "Invalid input".

3. קראו לפונקציות sum, isOddArray ו-prefix כך שהפלט שלהן תמיד יהיה באופן הבא:

```
First number is bigger/Second number is bigger
odd
message info: <text>
message info: <text>
odd
Invalid input
Not an array
message info: <text>
Invalid input
```

4. חיקרו איך ניתן לפשט את השורות שלפניכם באמצעות שימוש ב־Promise והחילו את מה שלמדתם בפונקציות שלכם, במידת הצורך:

```
new Promise((resolve) => resolve(5));
new Promise((resolve, reject) => reject(new Error('Error')));
```

5. קראו על async/await. צרו קובץ JS חדש, העתיקו לשם את הקוד הנוכחי ושנו את הקוד כך שישתמש ב־async/await בפונקציות sum, prefix, isOddArray ובפונקציות שקוראות להן.
6. חיקרו איך ניתן להפעיל את sum, prefix ו־isOddArray במקביל ולשמור את התוצאות שלהן, כך שלא תצטרכו לחכות שכל פעולה תסתיים כדי שיהיה ניתן להריץ את הפונקציה הבאה בתור.

הרצאה

עליכם להכין הרצאה (כולל מצגת) על נושא הקשור בפיתוח אתרים או נושא טכנולוגי אחר.
בחרו נושא וגשו אל המפקד המקצועי להתייעצות וקבלת אישור על הנושא הנבחר.
את ההרצאה תציגו מול שאר החניכים ואנשים מהענף.

משימת קורס - צד לקוח

כתבו משחק בעזרת HTML, CSS ו־Javascript ללא שימוש בחבילות נוספות.

- יש להשתמש ב־Canvas על מנת להציג את המשחק.
- על המשחק להכיל פונקציונלית מתקדמת
  - לדוגמא: יצירת מפות דינמיות, אויבים חכמים, גורמים שישנו את פעולתו הבסיסית של המשחק בצורה דינמית.
- יש לאפשר עצירה של המשחק (מבלי לסגור אותו) והמשכה מאותה נקודה לאחר העצירה.
- מומלץ לקרוא על מחלקות לפני תחילת כתיבת המשחק. במידה ופרדיגמת הפיתוח שלכם מתאימה לכך, ניתן להשתמש במחלקות בכתיבת הקוד.
- יש לעקוב אחר הסטנדרטים שלמדתם.
- על הקוד להיות קריא ומובן
  - יש לתת שמות הגיוניים לפונקציות ולמשתנים.
  - יש למחוק קוד שלא בשימוש.
  - יש לתת שמות הגיוניים לקבצים.
  - חובה להפריד בין קבצי HTML, CSS ו־Javascript.
  - יש לחלק את הקוד למספר קבצים (לבצע חלוקה הגיונית).
- יש להעלות את המשחק לתיקייה שלך ב־Gitlab
  - יש לבצע Commits בתדירות סבירה ולתת להם תיאורים הגיוניים ומשמעותיים.
- בסיום המשימה יש להעלות את המשחק גם לתיקייה האישית ב־Drive.

מקורות קריאה מומלצים:

- Mozilla Game Development
- Efficient animation for web games
- JavaScript game tutorial

בסיום כתיבת המשחק יש להציגו מול החונכים ושאר החניכים, כולל הצגת הקוד שלו.

העלאה על VM

מה זה VM? למה נצטרך אותו? איזה סוג מערכת הפעלה תהיה עליו? איך נתחבר אליו?

- צפו בסרטון **עד דקה 11:26.**
- אז, כפי שהבנתם, **VM = Virtual Machine = מכונה = שרת -** הינה סביבה מבודדת שמנצלת את המשאבים (CPU, RAM ו-Memory) שיושבים על המחשב שלכם או ישירות על שרתים פיזיים.
- ה־VM יכול להיות מסוג מערכת הפעלה כרצונכם.
- ברוב הענף קיימת אינטראקציה קבועה עם מערכת ההפעלה (Ubuntu (Linux distro, לכן חשוב שתכירו ותשתמשו בה. עם זאת, המחשבים שלכם כרגע הם בעלי מערכת הפעלה Windows **ואסורים לשינוי**, לכן, אנחנו מקצים לכל חניך VM שבו יוכל להתנסות עם Ubuntu ועם הרעיון של remote development.
- שלבי עבודה עם ה־VM:
  - קראו את הדוקומנטציה **מההתחלה עד החלק (לא כולל)** של Generate SSH Keys Using PuTTY, ועקבו אחרי ההוראות. לאחר מכן בצעו את הפעולות הבאות:
    - העתיקו את תוכן הקובץ id_rsa.pub (אם מסתכלים דרך ה־File Manager זהו הקובץ שתוכנו מתחיל ב ...ssh-rsa), העלו אותו לטבלה. הודיעו לחונכים של אותו החלק.
    - לאחר שתקבלו אישור מהחונכים הריצו את הפקודה הבאה:
      ssh <vm_username>@<vm_ip or dns>
      תמצאו את פרטי ההתחברות במסמך האקסל המצורף: מסמך VM
    - אם תראו את הטרמינל של המכונה הוא `(vm_username@vm_name:~$)` אם כן, הצלחתם להתחבר בהצלחה למכונה.
    - עריכת הקוד שלכם דרך הטרמינל של המכונה היא מאוד לא נוחה ולכן קיים תוסף ל־VSCode שמאפשר התחברות למכונה הוירטואלית ועריכת קבצים במכונה דרך ה־VSCode הלוקאלי. קראו על התוסף, קראו רק את החלקים הבאים:
      1. Install the extension
      2. Connect using SSH
      3. Port forwarding

      שימו לב לא להוריד NodeJS מהמקור הבא, הסתמכו על מקורות אחרים
    - שימו לב שהפורטים של המכונה סגורים כברירת מחדל אז על מנת לשלוח בקשות למכונה מבחוץ צריך לפתוח פורטים ספציפיים אליהם רוצים לגשת. קראו על החלק של הרצת האפליקציה ו־Port Forwarding במאמר.
  - כמה דגשים לשימוש:
    - ה־VM פעילים בין השעות 07:00-00:00.
    - אין גישה ל־VM מחוץ לכותלי הבסיס, הנושא נאכף ולכן לא תוכלו להתחבר.

שימו לב! ה־VM שלכם זמניים לתקופת הקורס - משמע שבסוף הקורס ימחק המידע משם.

משימה

היכנסו לאתר, קראו ותרגלו את הפרקים הבאים (אפשר ואף מומלץ להתנסות ב־VM):

- Getting Started
- Command Line
- Advanced Text - Fu

משימת VM

העלו את המשחק שכתבתם על ה־VM שברשותכם (רמז: השתמשו בתוסף **Live Server** של VSCode).
ענו על השאלה:
מדוע ניתן לראות את המשחק רץ על הכתובת "localhost" בדפדפן הלוקאלי כאשר הוא רץ על ה־VM?

חלק שני - צד שרת

מבוא - צד שרת

קראו את Introduction to Server Side של MDN.
ניתן גם לקרוא על NodeJS ב־w3schools.

NodeJS

לחלק זה ישנם 2 מקורות מידע עיקריים:

1. הסרטון Node.js Tutorial for Beginners ב־youtube והמדריך ב־Tutorials Point שהוא המקור המומלץ ללמידה בחלק זה.
2. קורס אינטרנטי ב־Udemy בשם Learn and Understand NodeJS.
במקרה והנכם מעדיפים ללמוד מהקורס, עדכנו את החונך והתחברו ל־Udemy עם הפרטים המצוינים בחלק javascript.

תראו את הסרטון Node.js Tutorial for Beginners והמשיכו לשאלות הבאות:

שאלות

ענו על השאלות הבאות תוך כדי למידת NodeJS:

1. מה ההבדל בין קוד אסינכרוני (Asynchronous) וקוד סינכרוני (Synchronous)?
    a. תנו דוגמא לקוד אסינכרוני ב־NodeJS.
    b. תנו דוגמא לקוד סינכרוני ב־NodeJS.
2. מה ההבדל בין Callback ל־Promise?
    a. מהם החסרונות של Callbacks?
3. מהו ה־event loop ב־NodeJS?
4. האם ניתן להריץ קוד של NodeJS על הדפדפן?
5. מהו NPM?
6. מנו שלושה הבדלים בין JS בדפדפן ו־NodeJS, וחמש פונקציות שיש באחת ואין בשניה.

סקריפטים ב־Node.js

למרות שהשימוש העיקרי של Node.js הוא בניית web servers היא גם יכולה לשמש כשפה לכתיבת סקריפטים.

פתרו את התרגילים הבאים:

Scripts - משימה 1

- כתבו סקריפט ב־NodeJS בשם "file_generator" שיקבל 2 ארגומנטים (arguments):
  - הארגומנט הראשון יהיה מספר אשר ייצג את כמות הקבצים שהסקריפט יצור בתיקיה בשם "created_files".
  - הארגומנט השני יהיה מספר אשר ייצג את כמות המילים הרנדומליות שהקובץ הראשון יכיל. כל קובץ (חוץ מהקובץ הראשון) יכיל כמות כפולה של מילים מהקובץ שהגיע מלפניו.
- הסקריפט ירשום ב־console את שם כל קובץ שהוא יצר ואת כמות המילים שנרשמו בו.
- דוגמא:
  הפקודה הבאה
  Node file_generator 3 4
  תיצור 3 קבצים כאשר הקובץ הראשון יכיל 4 מילים, הקובץ השני יכיל 8 מילים והקובץ השלישי יכיל 16 מילים.
- הדגש: הסקריפט צריך להיות אסינכרוני.

Scripts - משימה 2

כתבו סקריפט ב־NodeJS בשם "jokes" שיקרא מקובץ בשם ".env" את משתני הסביבה הבאים:

1. כמות בדיחות: JOKE_AMOUNT
2. נושא הבדיחות: JOKE_SUBJECT

הסקריפט יבצע את הדברים הבאים:

- יכתוב לקובץ בדיחות בנושא הנבחר תוך בדיקה שבדיחה לא תופיע פעמיים.
- במידה ואין משתנה סביבה בשם JOKE_AMOUNT, כמות הבדיחות תהיה 50.
- ידפיס שגיאה מתאימה במידה והנושא הנבחר אינו קיים
- ידפיס שגיאה מתאימה במידה בכמות הבדיחות קטנה מ־50.

יש להשתמש בחבילה one-liner-jokes.

Scripts - משימה 3

כתבו סקריפט ב־NodeJS בשם "file_mover" אשר יעביר את כל הקבצים והתיקיות מתיקייה בשם "files_to_move" לתיקייה בשם "moved_files".

- הסקריפט יעביר כל קובץ וכל תיקייה אשר נמצאים בתיקייה המצוינת.
- כאשר נוצר קובץ\תיקייה חדשים בתיקייה, הסקריפט יעביר גם אותם לתיקייה היעד (ללא צורך בהרצה מחודשת של הסקריפט).
- הסקריפט ירשום ב־console את שמות הקבצים והתיקיות שהוא העביר (אין צורך להדפיס את שמות הקבצים והתיקיות המקוננות).
- בנוסף, הסקריפט ירשום לקובץ בשם "moved_files.txt" את שמות כל הקבצים והתיקיות שהוא העביר.

Scripts - משימת סיכום

כתבו סקריפט אשר יהווה ממשק טקסטואלי לטיפול בקבצים.
הממשק יוצג ב־cmd ויכיל את הפונקציות הבאות:

1. מחיקה של קובץ בהינתן שמו
2. יצירה של קובץ בהינתן שם
3. כתיבת טקסט לקובץ (יקבל את הטקסט ואת שם הקובץ)
4. יצירת תיקייה בהינתן שם.
5. מחיקת תיקייה בהינתן שמה.
6. יצירת קובץ בתיקיה.
7. מחיקת קובץ מתיקיה.
8. איחוד קבצים - יקבל שמות של שני קבצים ויוסיף את תוכן הקובץ השני בסוף התוכן של הקובץ הראשון. הקובץ השני ימחק.
9. יציאה מהמערכת.

דגשים:

1. אפשר להעזר בחבילות NPM.
2. חלקו את הפרויקט לקבצים בצורה מסודרת והגיונית.
3. **בונוס**: גרמו לממש להיות כמה שיותר ייחודי ויפה.

ניהול פרויקט ב־Git

Merge Requests \ Pull Requests

כאשר אנו רוצים לעבוד על פיצ'ר חדש בפרויקט שלנו, לשנות בו פונקציונליות, לתקן בו באג או לשנות בו כל דבר אחר, אנו נצטרך לפתוח branch חדש ולעבוד עליו.
כאשר נסיים לעבוד נצטרך לבצע Merge Request (ב־GitLab) או Pull Request (ב־GitHub) מה־Branch עליו עבדנו.

Merge Request \ Pull Request או בקצרה MR \ PR היא בעצם בקשה לשלב את הקוד עליו עבדנו, ב־Branch אחר. Branch זה יכול להיות כל Branch בפרויקט, לדוגמא master branch.

כאשר נבצע MR, מפתח אחר שעובד על הפרויקט (ולפעמים, אם הפרויקט כתוב על ידכם בלבד, אתם) יצטרך לאשר אותו. לפני שאותו מפתח יאשר את ה־MR, הוא יצטרך לבדוק אותו, ולראות שהוא לא פוגע בפונקציונליות, ב־Business Logic או בסטנדרטים של הפרויקט.
אותה בדיקה של המפתח נקראת Code Review, בסופה, אם המפתח יאשר את ה־MR, אותו Branch שעבדתם עליו, ישתלב עם ה־Branch המבוקש.

סקרי קוד - Code Review

Code Review היא הפעולה של בדיקת MR \ PR על ידי מפתח בפרויקט (לרוב, אך לפעמים גם יכול להיות מפתח מצוות אחר).

בכל פרויקט יש לבצע Code Review לכל Merge Request שמשתלב עם אחד מהענפים הראשיים (master develop וכדומה).

הגשת Merge Request לבדיקה

לפני שאתם מגישים MR עליכם לוודא את הדברים הבאים:

1. ה־MR מכיל אך ורק קוד אשר קשור ישירות למהות ה־MR.
2. ה־MR קטן ככל האפשר (מומלץ לשנות דבר אחד בכל MR, ולא גם לתקן באג וגם להוסיף feature לדוגמא).
3. כל הבדיקות חייבות לעבור.
4. ה־MR שלם, כל קוד בו חיוני ואין בו קטעי קוד שלא עובדים.

ביצוע Code Review

כאשר אתם מבצעים Code Review שימו לב לדברים הבאים:

1. אין שגיאות לוגיקה בקוד.
2. אין באגים נוספים שנוצרו בעקבות הקוד.
3. הקוד לא שובר שום חלק בפרויקט (אלא אם זאת מטרתו)
4. הקוד ממש את משמעות\דרישות ה־MR (לפי שמו או תיאורו).
5. ה־Merge Request מכיל אך ורק קוד שרלוונטי אליו.
6. הקוד קריא ומובן.
7. הקוד יעיל מספיק.

8. ה־commit messages עומדים בסטנדרטים
9. הקוד ממומש בצורה הגיונית, תחשבו על איך אתם הייתם ממשים אותו.
10. הקוד תואם לסטנדרטים של הצוות.
11. הבדיקות האוטומטיות "מכסות" את כל הקוד ומקרי הקצה האפשריים.
12. שונו בדיקות קיימות במקרה והקוד החדש דורש זאת (breaking changes לדוגמא).

כתיבת הערות ל־Merge Request

כבודקים, על כל דבר לא תקין שאתם מוצאים ב־Merge Request עליכם לרשום הערה מתאימה על ההערה להיות:

1. בונה - יש להיות מאחורי ההערה משמעות, יש להעיר רק על דברים משמעותיים.
2. מכבדת - יש לכתוב את ההערה בטון מכבד ורציני.
3. מפורטת - יש לפרט בדיוק מה הבעיה ולא לכתוב הערות קצרות ולא מובנות.
4. יחודית - אין צורך לציין את אותה הבעיה מספר פעמים, אפילו אם היא מופיעה מספר פעמים בקוד.

היתרונות של Code Reviews

1. שמירה על איכות הקוד בפרויקט.
2. העברת ידע בין המפתחים (גם מצד הבודק וגם מצד הנבדק).
3. הורדת כמות הבאגים בפרויקט (כאשר יש יותר עיניים על פיסת קוד, הסיכוי לבאג נמוך יותר).

נקודות נוספות

1. כל אדם בצוות צריך לעבור Code Review, גם המפתח הכי בכיר.
2. כמפתח שמקבל ביקורת ב־Code Review, קבלו את ההצעות, ככה תוכלו ללמוד יותר (אלא אם הן גורעות מאיכות הקוד).
3. תהיו חיוביים כמבצעי Code Review, אל תזלזלו או תרשמו דברים לא מכבדים כביקורת.
4. אל תבצעו את ה־Code Review מהר מדי, אתם תפספסו דברים חשובים, אך אל תחרגו מ־60 דקות לכל Code Review.

חומרים נוספים לקריאה

- Five reasons why the code review process is critical for developers
- Importance of code reviews
- Code reviews
- Best practices for peer code review
- Code review best practices

משימת Code Review

בצעו Code Review למשימה הראשונה והשנייה של Scripts ב־NodeJS ל־2 חניכים שונים (תבדקו משימה אחת לכל חניך). במקרה ואין חניכים שסיימו את המשימה, דלגו עליה כעת אך חזרו אליה במועד מאוחר יותר.

כתיבת Web Server ב־NodeJS

במשימות הבאות עליכם להשתמש ב־Postman על מנת לבדוק שהשרתים עובדים.

Web Server - משימה 1

כתבו שרת ב־NodeJS **ללא שימוש ב־Express או באף חבילה חיצונית אחרת**.

- השרת יאזין לפורט המצוין במשתנה סביבה (environment variable) בשם SERVER_PORT, או, במקרה והמשתנה לא מוגדר, הוא יאזין לפורט המוגדר בקובץ קונפיגורציה (config file) אשר יוגדר על ידכם.
- השרת יחזיר תשובת HTTP לנתיבים הבאים:
  - בקשת **POST** אל הנתיב "**api/numbers/prime/validate**" תחזיר true במקרה וערך כל ה־properties שבגוף הבקשה (body) הם מספרים ראשוניים, אחרת היא תחזיר false. השרת יתייחס לכל property בגוף ההודעה ללא תלות בשמו.
  - בקשת **GET** אל הנתיב "**api/numbers/prime?amount=n**" תחזיר את **n** המספרים הראשוניים הראשונים, כאשר **n** הינו מספר בין 1 ל־32 שמתקבל ב־query parameter בשם amount.

Express

Express הינה חבילת NPM אשר משמש כפריימוורק ל־NodeJS ומקל על כתיבת צד השרת בעזרתו.
העיקרון הראשי מאחורי Express הוא ה־Middlewares שלו המתווים את דרך פעולת של השרת והדרך בה יטפל בכל בקשה.

ה־Middlewares בעצם מהווים רשימה של פעולות אשר יקרו בכל בקשה שהשרת יקבל.

Web Server - משימה 2 - Express

בצעו את משימת ה־Web Server הראשונה, רק כעת השתמשו ב־Express.
הוסיפו לשרת את הנתיב "API/numbers/prime/display" אשר בבקשת GET ישלח דף HTML ללקוח ויציג לו את עשרת המספרים הראשוניים הראשונים בכרטיסים (div לכל מספר) המסודרים האחד ליד השני (יש להשתמש ב־SSR במשימה, אין להריץ Script בצד לקוח).

על מנת לפתור את התרגיל ניתן לעיין במקורות הבאים:

- Installing Express
- Hello World Example
- Basic Routing
- Serving Static Files
- Routing
- Writing Middlewares
- Using Middlewares

Best Practices

קראו את Best Practice Performance.

Error Handling

- Express Error Handling
- Node Production Practices
- Building Robust Node Applications: Error Handling

בסיסי נתונים

בחלק גדול מהפרויקטים, עליכם יהיה לשמור מידע עבור מטרות מסוימות.
על מנת לשמור את המידע, אנו נשתמש בבסיס נתונים (Database).
קראו את המאמרים הבאים על בסיסי נתונים:

1. Database Introduction
2. NoSQL vs SQL

בקורס זה, נתמקד בשימוש ב־MongoDB.

MongoDB

MongoDB הינו בסיס נתונים לא רלציוני, המבוסס על Documents, לכן, אין בו SQL (נקרא גם NoSQL).
קראו על MongoDB באתר הרשמי.
התמקדו בחלקים הבאים:

- CRUD Operations
- Data Models
- Aggregation

Indexes

קראו על אינדקסים ב־MongoDB במאמר הבא: MongoDB Optimization - Indexes and when to use them.

משימות MongoDB

משימה 1

צרו בסיס נתונים עם מונגו אשר יחזיק מידע על סופרים והספרים שהם כתבו.
בסיס הנתונים צריך להכיל את המידע הבא על כל ספר (לא בהכרח ב־Collection של הספרים):

- שם הספר
- תיאור הספר
- תאריך פרסום הספר
- כותב הספר
- מספר עמודים

ואת המידע הבא על הסופרים:

- שם פרטי
- שם משפחה
- שנת לידה

ה־DB צריך להיות אופטימלי לשאילתות הבאות:

- הצגת כל הספרים של סופר מסוים (כולל כל הפרטים על הספרים)
- חיפוש ספר לפי השם או התיאור שלו (חיפוש אשר לא ידרוש הכנסת שם מדויק של הספר)
- הצגת כל הספרים מעל 250 עמודים כאשר הם מסודרים מהכמות הנמוכה לגבוהה.

את המשימה יש לבצע בעזרת סקריפט ב־NodeJS (אין צורך להרים שרת):

- יש להשתמש ב־Driver של Mongo ולא ב־mongoose.
- יש להגדיר את הפונקציות הבאות:
  - פונקציה שתיצור Document ותשים אותו ב־Collection המתאים (לכל סוג Document)
  - פונקציה לכל שאילתא שצוינה למעלה אשר תדפיס את ה־Documents שהתקבלו.
- יש ליצור אינדקסים בתוך הסקריפט עצמו.
- יש לתכנן את ה־Collections בצורה כזו שכל השאילתות, בממוצע, יפעלו בצורה הכי טובה בהתחשב בזה שצריך לדאוג שכולן יעבדו כמה שיותר טוב (יהיו מהירות).

משימה 2

קראו על Aggregation ובצעו את המשימה הבאה:
כתבו שאילתה שתחזיר את כל הספרים **בעלי יותר מ־200 עמודים**, **שנכתבו בין שנת 2015 ל־2020**, של ה**סופרים ששמם הפרטי מתחיל ב־"P"**.
השאילתה **תחזיר רק את שמות הספרים ואת שם הסופר שכתב אותם**.
השאילתה תמיין את הספרים על פי שם הסופר (מהנמוך לגבוה) ואז על פי כמות העמודים (מהנמוך לגבוה).
השאילתה תתבסס על בסיס הנתונים שתכננתם במשימה 1 בחלק זה.
יש לכתוב את השאילתה בעזרת אגרגציה.
יש לפתור את המשימה בעזרת Mongoose וליצור Models ו־Schemas עבור כל Collection.

נק' למחשבה...

1. מה יקרה אם נפנה ל־DB וננסה לשלוף מידע בפעולות שאינן אסינכרוניות?
2. למה חשוב לשלוף מידע לפי שדה יחודי (Unique)?

TypeScript

TypeScript הינה שפת תכנות אשר "עוטפת" את JavaScript ומתורגמת אליה באמצעות Transpiler.
TypeScript מוסיפה הגדרות סטטיות לטיפוסים של משתנים שונים ומוסיפה תמיכה ב־features מודרניים (רק במהלך כתיבת הקוד) אשר לא בהכרח נתמכים בדפדפנים וסביבות ההרצה השונות של JavaScript.

מעתה - השתמשו ב־TypeScript עבור כל המשימות הבאות בקורס.

קראו בקצרה על TypeScript בדוקומנטציה הבאה.

Transpiler

Transpiler הוא סוג של מתרגם אשר לוקח קוד מקור הכתוב בשפה אחת וממיר אותו לקוד מקור שווה ערך בשפה אחרת.

קראו על מה זה Transpile ומדוע אנו צריכים להשתמש ב־Javascript Transpilers.

היתרונות

- מאפשר להגדיר את מבנה המידע בו אנו משתמשים בקוד (ממשתנים פשוטים ועד פרמטרים בפונקציות וערך ההחזר שלהם).
- מספק תיעוד בסיסי לקוד - הוא מאפשר לכתוב קוד קריא ומובן יותר מאשר ב־JavaScript ומקל על כתיבת הקוד בעקבות כך.
- מאפשר לנו להשתמש ב־features החדשים ביותר של JavaScript שעדיין לא נתמכים בכל הסביבות, מה שהופך את הקוד שלנו ליפה יותר, קריא יותר, ונוח יותר לכתיבה וגם מאפשר לנו לא להישאר מאחור בעקבות מגבלות שונות של סביבות שונות.
- משתלב עם סביבות פיתוח (IDE) רבות ויוצר חווית פיתוח נוחה יותר עבור המתכנת (בדיקה סטטית של שגיאות, רמזים לגבי הגדרות של פונקציות, השלמה אוטומטית ועוד)

חשוב לשים לב:
- **TypeScript לא מהווה תחליף לבדיקות** - למרות שסביבות פיתוח רבות יתריעו לנו על שימוש לא נכון בקוד, אלו בדיקות לזמן כתיבת הקוד בלבד, ולא ל־Runtime (זמן ריצת הקוד).
- TypeScript מכילה את JavaScript, לכן חשוב מאוד להכיר טוב את בסיס JavaScript ולא להתעלם ממנו.
- TypeScript יכול להתאים לרוב הפרויקטים, אך כאשר כותבים קוד קצר, יש מצב שכתיבה בו רק תעכב את המפתח, לכן חשוב לדעת מתי להשתמש בו ומתי לא.

מקורות קריאה נוספים:
השוואה בין TypeScript ל־JavaScript:
- what is typescript and why use it
- why use typescript good and bad reasons
- TypeScript vs JavaScript: Should you migrate your project to TypeScript?

ספר על TypeScript:
- TypeScript Deep Dive - Why TypeScript

Tsconfig.json

בגלל ש־TypeScript לרוב לא מגיע מובנה בפרויקט שלנו , אנחנו צריכים להגדיר מראש שאנחנו מתכננים להשתמש בו, וגם איך בדיוק אנחנו נשתמש בו.
על מנת שנוכל לכתוב ב־TypeScript ולהריץ את הקוד שלנו, עלינו להגדיר קובץ שנקרא tsconfig.json. קראו עליו ועל המשמעות שלו.
תוכלו למצוא פירוט של ההגדרות והשדות האפשריים שיכולים להופיע בקובץ בתיעוד הבא.

משימת TypeScript

צרו פרויקט חדש אותו תכתבו ב־TypeScript. היעזרו במדריך הבא ליצירת פרויקט ראשוני.

בפרויקט זה יהיה עליכם לכתוב שרת express בדומה למה שעשיתם במשימת ה־Web Server.

השרת שתכתבו ייצג חנות ספרים.
על כל ספר בחנות יש לשמור את המידע הבא:
- מחרוזת של שם הספר (שם הספר חייב יהיה להכיל לפחות תו אחד שאינו מספר)
- מחרוזת של שם הסופר
- מספר סידורי ייחודי
- ז'אנר:
  - אחד מתוך חמישה ז'אנרים מוגדרים המורשים בספריה.

***שימו לב:** אין להשתמש בבסיס נתונים לשמירת הנתונים. יש לשמור אותם בזיכרון התכנית - כלומר, בסוף התכנית המידע יעלם.
על השרת לאפשר את הפעולות הבאות:
1. יצירה של ספר חדש.
2. קבלת ספר לפי:
   a. שם הספר

   או

   b. מספר סידורי

שימו לב: יש ליצור route יחיד עבור החיפוש הזה, כלומר - ערך החיפוש יגיע לאותו משתנה כל פעם.
עליכם יהיה לבדוק בקוד עצמו האם החיפוש הוא לפי מספר סידורי או שם בהתאם לערך שהתקבל.
3. קבלת כל הספרים בספריה ללא המספר הסידורי שלהם.

על מנת שתוכלו לבצע את כל הפעולות הנדרשות כראוי, עליכם תחילה להבין מה הם טיפוסים וכיצד מגדירים אותם.
קראו את הפירוט ב־TypeScript Handbook על מה הם טיפוסים, וגם את הפירוט בלינקים הבאים:
- Advanced Types
- Utility Types

כלי עזר לפיתוח

Linter

Linter הינו כלי שסורק את הקוד שלנו ומסמן בעיות שהוא מצא בו על פי ההגדרות שהגדרנו אותו. ה־linter יכול לסמן באגים, בעיות תכנות, אי עקיבה אחר סטנדרטים וכדומה.

אנו משתמשים ב־Linter על מנת לוודא שאנחנו עוקבים אחרי הסטנדרטים שהגדרנו בחלק הראשון של הקורס.
תוכלו להשתמש ב־NPM packages כמו eslint ו־extensions ל־vs code על מנת להשתמש ב־linter.

כדי לעקוב אחר הסטנדרטים של הענף עם eslint, השתמשו ב־config של airbnb ל־eslint.

מעכשיו, כתבו את כל הקוד שלכם בעזרת Linter על מנת שתוכלו לעמוד בסטנדרטים בקלות.

בהמשך, כאשר תשתמשו ב־TypeScript אתם מוזמנים לקרוא את המדריך הבא שיעזור לכם להגדיר את eslint ו־prettier לעבוד עם typescript (כמובן שתגדירו גם את הסטנדרטים של airbnb)

התקנת Linter

1. הורידו את התוסף eslint ל vscode.
2. התקינו את ספריות הNPM הבאות:
   a. eslint
   b. eslint-config-airbnb-base
   c. eslint-plugin-import

3. הוסיפו קובץ eslintrc.json. והדביקו שם את הקוד הבא:

```
{
  "parser": "@typescript-eslint/parser",
  "extends": [
    "plugin:@typescript-eslint/recommended",
    "airbnb-typescript/base"
  ],
  "parserOptions": {
    "ecmaVersion": 2020,
    "sourceType": "module",
    "project": "./tsconfig.json"
  },
}
```

ביטול חוקי Lint

במידה ויש חוקים שלא מתאימים לפרוייקט שלכם ותרצו לכבות אותם (למשל חוק שלא מאפשר לכתוב console.log), יש לכך 3 שיטות:

1. לכתב דרך ה־Rules שב־eslintrc.json.
2. לכבות על שורה ספציפית.
3. לכבות על קובץ שלם.

ארכיטקטורת קוד

לכל מפתח יכולה להיות דרך אחרת לעיצוב הקוד שלו.
הדרך הטובה ביותר ללמוד דברים חדשים היא להתנסות ולהבין איזה סידור קוד מתאים לאיזה פרויקט ואיזה סידור קוד מתאים לך ספציפית.

בקורס אנו נתמקד בארכיטקטורת קוד מבוססת שכבות אשר מבוססת על ארכיטקטורה רב שכבתית.

ארכיטקטורה זו מחלקת את הקוד למספר שכבות, כאשר כל שכבה מתקשרת עם השכבה שמתחתיה ובמקרים מסוימים, השכבה מתקשרת בינה לבין עצמה.

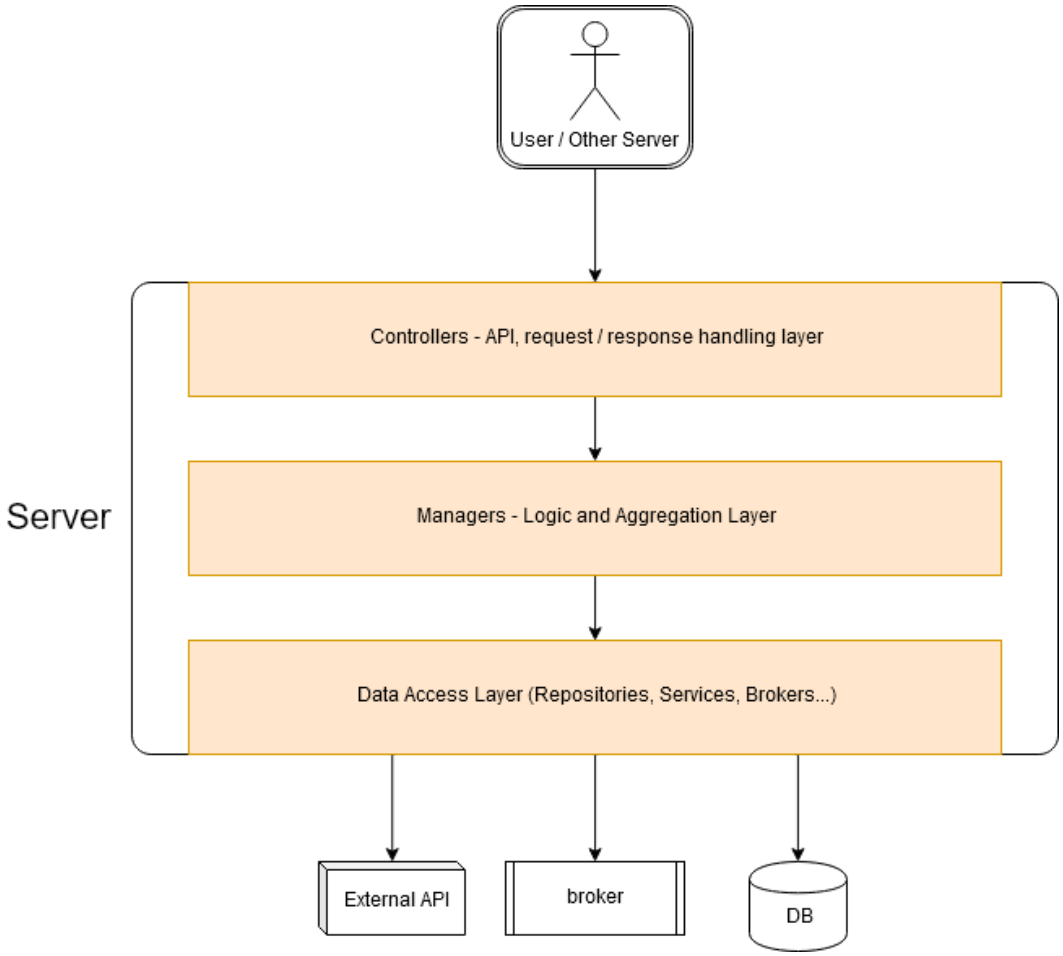
נתנו שמות משלנו לכל שכבה מפני שאין סטנדרט ידוע לשמות אלו:

השכבות הן כדלהלן:

1. **Controller** - זוהי שכבת הטיפול בבקשות שמגיעות אל ה־API של השרת. לדוגמא, במקרה ולשרת יש API מבוסס HTTP ואנו משתמשים ב־express, נשים בשכבה זו את ה־middlewares אשר מטפלים בבקשה ומחזירים תשובה ללקוח.
   חברים בשכבה זו יפנו אל שכבת ה־Manager.
2. **Manager** - שכבה זו אחראית על ה־business logic של השרת ועל איחוד מידע בין features שונים. שכבה זו תבצע פעולות לוגיות, בדיקות קלט מבחינת business logic (לדוגמא: בדיקה האם קבוצה מלאה לפני הוספת אדם אליה) ואיחוד מידע המגיע ממספר מקורות שונים.
   חברים בשכבה זו יפנו לשכבת ה־Data Access Layer ובנוסף, יוכלו לפנות לחברים אחרים בשכבה על מנת לבצע איחודי מידע ובדיקות הקשורות ל־business logic.
3. **Data Access Layer** - זוהי השכבה אשר ניגשת אל מקורות מידע שונים אשר יכולים להיות או חיצוניים או חלק מהשרת עצמו או מתקשרת עם מקורות חיצוניים על מנת לבצע בהם פעולות \ להתריע על פעולה שקרתה בשרת.
   בשכבה זו אנו משתמשים במספר שמות על מנת לתאר גישה למקורות מידע שונים:
   a. **Repository** - כאשר אנו ניגשים אל בסיסי נתונים (Data Base).
   b. **Service** - כאשר אנו ניגשים אל API של שירות חיצוני (שרת אחר או third party)
   c. **Broker** - כאשר אנו מתקשרים עם message broker. (**אין צורך להתעמק בנושא זה כרגע**)

שימו לב: סוג זה יכול להיות שייך גם ל־controllers וגם אל ה־Data Access Layer כתלות בכיוון ההודעה (הגיעה אלינו או נשלחה מאיתנו):

i. במקרה וההודעה נשלחה מאיתנו, נשרשר לסוף השם את המילה publish, אשר מציינת יצירה ושליחה של הודעה (broker.publish)
ii. במקרה וההודעה הגיעה אלינו, נשרשר לסוף השם את המילה subscribe, אשר מציינת "הרשמה" לקבלת הודעות מסוג מסוים (broker.subscribe).



היתרונות

1. מזעור הצורך ב־refactoring בעת שינוי ה־API או ה־Data bases בהם אנו משתמשים:
   a. במקרה ונחליף DB נצטרך לשנות רק ה־Repositories (שכבת ה־Data Access Layer).
   b. במקרה ונחליף Framework או נשנה את ה־API שלנו, נצטרך לשנות רק את ה־Controllers.
2. קריאות הקוד:
   a. מפני שכל ה־business logic נמצא רק ב־Managers, קל להבין אותו (את ה־business logic) ואין צורך להסתכל במספר קבצים או לתהות האם אנו מפספסים משהו בהבנת הלוגיקה.

b. הפונקציות יותר קטנות והקבצים פחות עמוסים בקוד, דבר המקל על הבנתו.
3. Code review - כאשר נעשה MR/PR ויעשו לנו Code Review, לבודק יהיה מושג ראשוני מה השתנה בקוד לפי הקובץ ששונה, דבר אשר יקל על הבדיקה.
4. מניעת שכפול קוד:
a. מפני שה־Managers מכילים רק את הלוגיקה, ניתן לתקשר בין Managers ולשמור על ה־Business Logic מבלי לשכפל אותו בכמה Managers.
לדוגמא: אם יש צורך להוסיף בן אדם רנדומלי לקבוצה כאשר יוצרים אותה, נממש את הפעולה ב־Manager של הקבוצות. במקרה ו־Manager אחר ירצה ליצור קבוצה על מנת לממש business logic אחר הוא יוכל להשתמש במימוש שב־Group Manager ולא יצטרך לממש אחד בעצמו.
b. במקרה וישנם מספר Features שה־Repository שלהם זהה או דומה, ניתן ליצור Class בסיסי שיממש פעולות אלו וה־Repositories השונים ירשו ממנו.

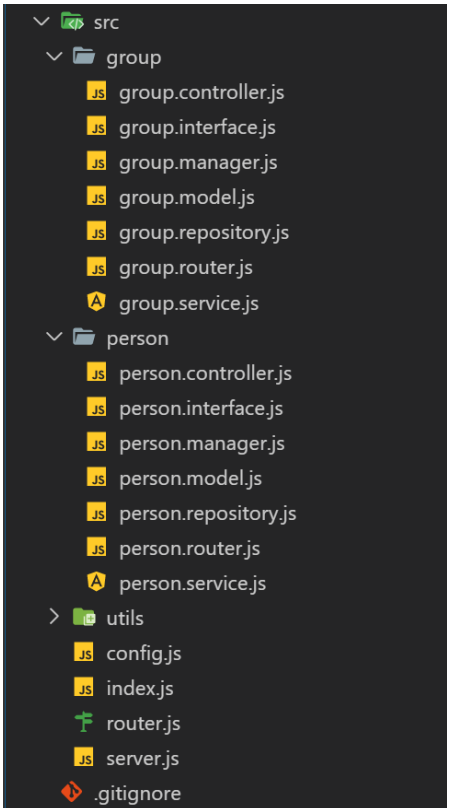
מבנה תיקיות

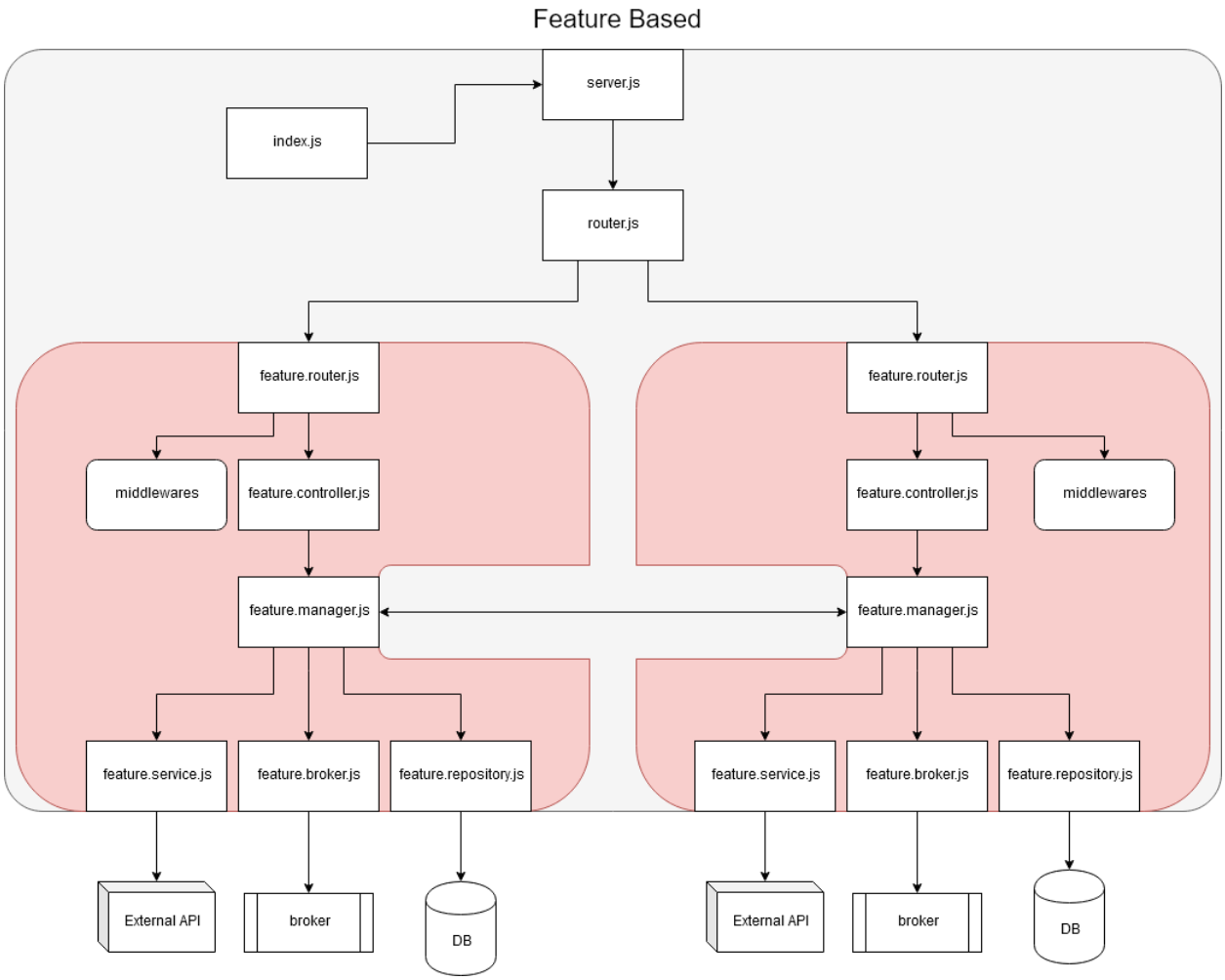
ישנם מספר דרכים בהם ניתן לסדר את הפרויקט שלכם בצד שרת.
שתי הדרכים המוכרות ביותר הן סידור על בסיס שכבות וסידור על בסיס תכונות (feature).
רוב הפרויקטים אצלנו בנוים על בסיס תכונות, אך אתם יכולים לבנות את שלכם כיצד שמתאים לכם, כל עוד יש לכם סיבה הגיונית לכך. הדגשה: סידור תיקיות הינו משנה את מהות הקוד או הארכיטקטורה שלו.

סידור על פי תכונות

סידור על פי תכונות אומר שלכל feature שקיים בפרויקט תהיה תיקייה משלו, כאשר כל הקבצים והשכבות הקשורות אליו יהיו באותה התיקייה.

בסידור זה כל תיקייה מכילה את כל השכבות של התכונה (משכבת הטיפול בבקשה ועד שכבת הגישה לנתונים).
חוץ משכבת ה־managers, שום אלמנט ב־feature לא יוכל לפנות ל־feature אחר.





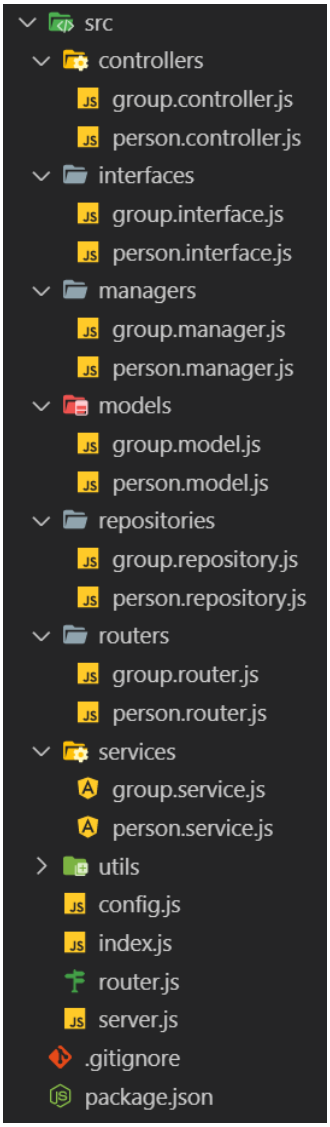
הדיאגרמה מציגה סידור תיקיות מבוסס פיצ'רים כאשר כל קובץ מיוצג בעזרת מלבן עם זוויות חדות וכל תיקייה מיוצגת על ידי מלבן עם זוויות מעוגלות.
תיקיית ה־feature מיוצגת על ידי מלבן אדום והתיקייה הראשית על ידי מלבן אפור.
אלמנטים הנוגעים בקצה תיקייה מייצגים כניסה או יציאה ממנה.
לדוגמא:

- ה־server.js מאפשר גישה חיצונית **ל**אפליקציה מפני שהוא אחראי על העלאת השרת.
- ה־Repository מגדיר גישה חיצונית **מ**האפליקציה אל DB.

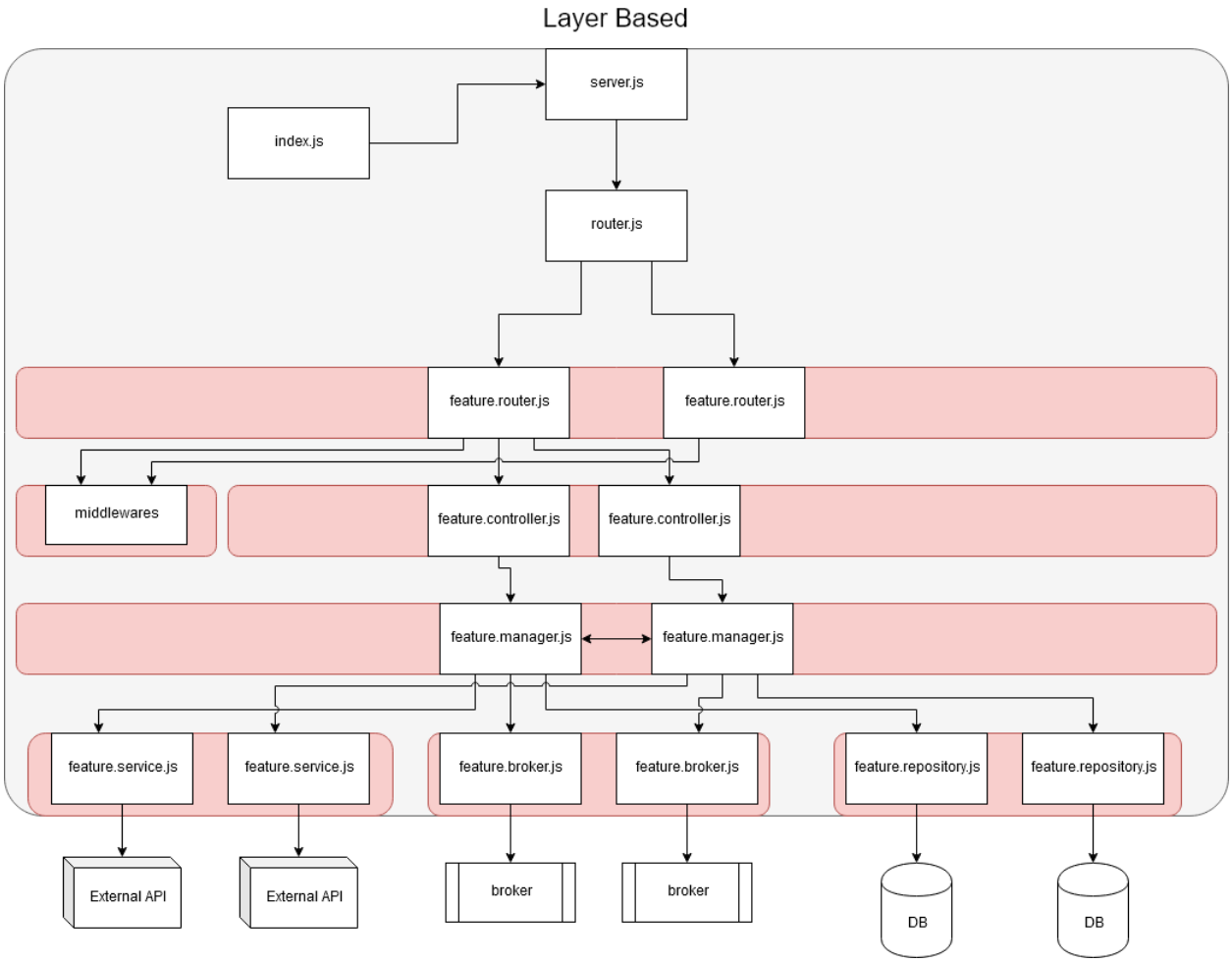
(הגדרת החיבור ל־DB נמצאת לרוב ב־Index.js, אך הפונקציות שניגשות אליו נמצאות בשכבת ה־Repository)

סידור על פי שכבות

סידור על פי שכבות אומר שלכל שכבה יש תיקייה משל עצמה, כאשר בכל תיקייה יופיעו כל התכונות שיש להן את השכבה הזו.



בסידור זה כל תיקייה המייצגת שכבה מכילה את כל ה־features בשרת אשר מכילים את שכבה זו.



דיאגרמה זו מתארת סידור על פי שכבות. כל שכבה בתיקייה שונה כאשר מספר features שונים מאותה שכבה נמצאים בה.

ניהול גרסאות

גרסאות סמנטיות

גרסאות סמנטיות הן סטנדרט למספור גרסאות תוכנה אשר משתמש במבנה הבא להגדרת כל גרסה:
Major.Minor.Patch

אחת הסיבות להשתמש בגרסאות סמנטיות היא למנוע שבירה של מערכות אחרות אשר משתמשות ב־API שלנו על ידי "סימון" גרסאות אשר עלולות לשבור\לשנות פונקציונליות ב־API.

קראו את ההסבר באתר הרשמי של Semantic Versioning.

Conventional Commits

זהו מפרט להוספת commits הקריאים גם לבני אדם וגם למכונה (למחשב).
כאשר נכתוב בעזרת מפרט זה, נוכל להוסיף אוטומציות על ה־commits שלנו, דבר שיכול לעזור בשחרור גרסאות וביצוע אוטומציות אחרות.

קראו את ההסבר באתר הרשמי והתחילו לרשום הודעות commits על פיו מהיום.

הסוגים השונים של ה־commits הן כדלהלן (לקוח מהסטנדרטים של Angular):

Type	Description
build	Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, NPM)
ci	Changes to our CI configuration files and Scripts (example scopes: Travis, Circle, BrowserStack, SauceLabs)
docs	Documentation only changes
feat	A new feature
fix	A bug fix
perf	A code change that improves performance
refactor	A code change that neither fixes a bug nor adds a feature
style	Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)
test	Adding missing tests or correcting existing tests

כאשר אתם רושמים type ל־commit יש לזכור להצמיד ":" (נקודותיים) אל ה־type ולשים רווח (space) אחרי הנקודותיים ורק אחרי הרווח לרשום את תיאור ה־commit.
דוגמא:

feat: add groups module

בנוסף, ניתן להוסיף Scope ל־commits בעזרת סוגריים "()" אשר יהיו צמודות ל־type ולנקודותיים

דוגמא:

fix(group module): fix problem in group creation API

יש להשתמש מעתה במפרט של Conventional Commits בפרויקטים שלכם.

משימת קורס - חלק 1 - צד שרת

כתבו שרת אשר ינהל אנשים וקבוצות.

- יש לכתוב את השרת ב־TypeScript.
- השרת יאפשר יצירה, קבלה, עדכון ומחיקה של קבוצות (CRUD).
- השרת יאפשר יצירה, קבלה, עדכון ומחיקה של אנשים (CRUD).
- חוקים:
  - בן אדם יכול להיות שייך למספר קבוצות שונות.
  - אותו בן אדם לא יוכל להופיע יותר מפעם אחת בקבוצה ספציפית.
  - קבוצה תוכל להכיל מספר בלתי מוגבל של אנשים.
  - קבוצה תוכל להכיל מספר בלתי מוגבל של קבוצות.
  - קבוצה לא תוכל להכיל קבוצה ספציפית יותר מפעם אחת.
  - קבוצה לא תוכל להכיל את עצמה.
  - קבוצה לא תוכל להיות אב קדום של עצמה.
  - מחיקת קבוצה תמחק את כל הקבוצות שהיא מכילה.
  - מחיקת קבוצה לא תמחק את האנשים שבה.
  - קבוצה לא יכולה להיות מוכלת בכמה קבוצות.
- על השרת ליחצן את הפעולות הבאות (חוץ מהפעולות CRUD הרגילות):
  - קבלת כל הקבוצות והאנשים בהיררכיה של קבוצה מסוימת.
  - חיפוש בן אדם בקבוצה לפי השם שלו.
  - קבלת כל הקבוצות שבן אדם מסוים נמצא בהן.
- עליכם להשתמש ב־MongoDB כבסיס הנתונים שלכם (ניתן להשתמש ב־mongoose).
- עליכם להוסיף indexes מתאימים ב־MongoDB על מנת שהבקשות יעבדו כמה שיותר מהר.
- עליכם לכתוב Unit Tests ו־Integration Tests. יש לכתוב בדיקות אשר יבדקו את כל התנאים המצוינים בחוקים למעלה, בנוסף לבדיקות בסיסיות של הקוד.
- עליכם לטפל בשגיאות של המשתמש ובשגיאות שלכם (Error Handler).
- עליכם לעקוב אחרי הסטנדרטים לשיום נתיבים.
- יש לכתוב את הקוד על פי הסטנדרטים שנלמדו ולהשתמש ב־Linter.
- עליכם לעבוד עם Git ולבצע Commits באופן תדיר; עליכם לשמור על פורמט ה־Conventional Commits בתיאור ה־Commit.
- ניתן להשתמש בכל חבילת Npm או Framework שתרצו.

משימה זו מכיל שילוב של נושאים רבים, רובם נמצאים בהמשך החלק הנוכחי של הקורס.
התחילו את המשימה ועבדו על פי סדר הגיוני; קודם התחילו את בסיס השרת, עם Express, לאחר מכן הוסיפו התממשקות ל־MongoDB, לאחר מכן הוסיפו בדיקות ולבסוף תעלו את האפליקציה על Docker.
יש להתחיל עם Typescript על תחילת הפרויקט ולא לעבור באמצע.

בפרויקט אמיתי סדר כתיבת הפרויקט לא יראה כך בהכרח, הסדר נקבע על מנת שדרך הלמידה שלכם תהיה פשוטה יותר.

בדיקות

כל פרויקט, בסופו של דבר, צריך לעבור שינויים במהלך דרכו, אם זה ל־refactoring או להוספת features.
כאשר אנו מוסיפים או משנים קוד בפרויקט, קיים סיכוי שנשבור את פעולתו התקינה או שנפגע בפעולתו של קוד אחר.
אנו עושים בדיקות כדי שלאחר כל שינוי, נוכל לבדוק שהכל עובד כמצופה.

בכל פרויקט שתכתבו, כולל במשימת קורס, תצטרכו לעשות טסטים.
אותן הבדיקות שתעשו, לא רק יעזרו לכם, אלא גם למתכנתים האחרים שיגיעו לפרויקט אחריכם.

ישנם מספר סוגי בדיקות אשר רלוונטיות אליכם כרגע:

- Unit Testing
- Integration Testing

תקראו על נושאים אלו ותבינו את המשמעות שלהם.
לאחר מכן, עליכם ללמוד להשתמש בחבילות המתאימות ולהוסיף בדיקות לקוד שלכם.

אלו החבילות המקובלות לשימוש כיום ב־NodeJS:

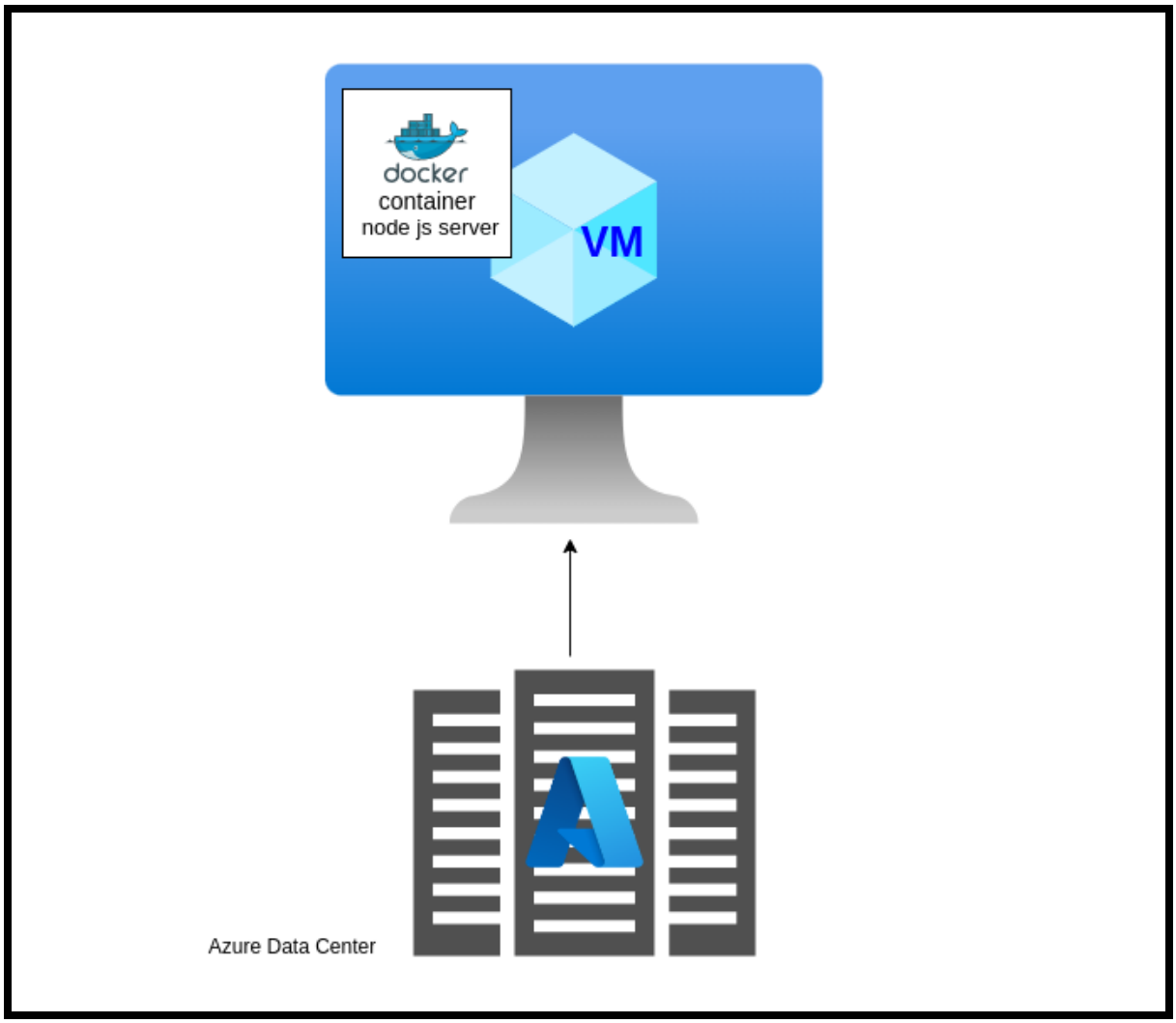
- Jest
- Mocha
- Chai
- SuperTest

באפשרותכם לבחור חבילות אחרות ולהשתמש בהן, אך מומלץ להתייעץ תחילה עם החונכים או עם מקורות ידע אחרים בענף.

הרמה על Docker

דוקר היא תוכנה בקוד פתוח המאפשרת התקנה והרצה של אפליקציות בתוך סביבה וירטואלית מבודדת הנקראת **Container**. כל Container כולל תוכנות, ספריות וקבצי קונפיגורציה משלו. באופן זה, ניתן להריץ אפליקציות בצורה מבודדת וכך למנוע מהן "להפריע" לריצה של אפליקציות אחרות, וליהנות מסביבה מותאמת ואופטימלית לכל אפליקציה.
צפו בסרטון ההסבר הבא והתייחסו בדוגמא של Ruby On Rails ל־MongoDB.

כשסיימתם את משימת הקורס - צד שרת, בניתם שרת של NodeJS והרצתם אותו לוקאלית במחשב שלכם.
במשימה הבאה, תרימו את השרת שיצרתם על סביבה מבודדת נוספת בתוך ה־VM (על Container).
אז בעצם, תצרו סביבה נפרדת בתוך הסביבה הנפרדת.
להסבר יותר מפורט והבנה של שתי הסביבות, צפו בסרטון עד דקה 6:46.



כתיבת Dockerfile

לאחר שלמדנו את הקונספט של Docker, נלמד איך בפועל עובדים הדברים ונרים את האפליקציה שלנו על Container בעצמנו.
בשביל ללמוד את הבסיס, צפו בסרטון הבא שילמד אתכם על הקובץ Dockerfile, שממנו נבנה Image, שממנה לבסוף נרים Container.

סטנדרטים כללים לכתיבת Dockerfile:

- יש להשתמש ב־Image בסיס קל וקטן ככל האפשר (שיקח כמה שפחות זיכרון)
- על כל Image להכיל רק את הקבצים הרלוונטיים לפעולת התכנית

קראו את הדוקומנטציה, וכתבו לפיה את ה־Dockerfile המתואר, בנו Image והריצו אותו ב־VM שלכם.
הרמתם את האפליקציה על Container אך היא נופלת - בדקו ב Logs מדוע.

Docker Compose

Docker Compose הוא כלי שפותח כדי לעזור להגדיר ולשתף אפליקציות מרובות Containers.
עם Compose, נוכל ליצור קובץ YAML כדי להגדיר את השירותים ובאמצעות פקודה אחת להרים את הכל.
אם בדקתם את הלוגים במשימה הקודמת - הבנתם שאתם צריכים להרים גם Container של MongoDB שיתחבר ל־container של האפליקציה שלכם. היעזרו בסרטון הבא, כתבו קובץ Compose והרימו את האפליקציה ואת ה־MongoDB, היעזרו בסרטון.

בדקו שהצלחתם - עשו בקשת API ל־Container של האפליקציה ושימו לב שאתם מקבלים את התשובה המתבקשת.

שימו לב להבדל בבקשות ה־HTTP בין הסרוויסים בהרצה על דוקר ובהרצה לוקאלית.

Docker Registry

A Docker registry is a service that hosts and distributes Docker images. In many cases, a registry will consist of multiple repositories which contain images related to a specific project. Within a given repository tags are used to differentiate between versions of an image.

מבלי לדעת, השתמשתם ב־Registry כזה שנקרא DockerHub; משם משכתם את ה־Image של Mongo.
תייגו את ה־Image של האפליקציה שלכם לפי גרסה והעלו אותו ל־Harbor Registry הענפי תחת פרוייקט על שמכם (בקשו את שם המשתמש והסיסמא מהחונכים).

כותב: אלמוג וגמן ציפרוט
נכתב: 27.10.2017
עודכן: 05.11.2022

חלק שלישי - Frameworks

מבוא

Framework הינה עטיפה לדרך הרגילה שבה אנחנו כותבים את הקוד באופן שיאפשר כתיבה פשוטה יותר שלו בטווח הרחוק.

במה אותן Frameworks עוזרות לנו?
הן מקצרות את זמן הפיתוח ומקלות עליו. כל Framework תקל בביצוע דברים שונים אבל רובן יעזרו בטיפול ב־HTML ובהזרקת מידע לתוכו. בנוסף, הן עוזרות בעיצוב, במבנה ובפונקציונלית של האתר.
ישנם מקרים בהם Frameworks יכולים לגרוע מזמן פיתוח האפליקציה, לרוב במקרים בהם האפליקציה פשוטה, ו־Framework יסבך אותה.
לכן מומלץ לשקול לכל פרויקט בנפרד, האם שימוש ב־Framework יקל על פיתוחו או יקשה עליו.

בענף אנחנו משתמשים ב־Frameworks מבוססי קומפוננטות (Components).
ה־Frameworks המשומשים ביותר כיום אשר משתמשים ב־Components הן Angular, React, Vue ו־Svelte, גם בענף שלנו אנו מפתחים בעזרת ארבעתן, כאשר כל צוות מתמקד באחת מהן.

מהן Components?

ישנם מספר סוגי Components, כאשר כל Framework יכול לממש אותן אחרת.
באופן כללי, קומפוננטה היא רכיב הניתן ל־"מיחזור", ניתן להשתמש בו במספר אזורים באפליקציה מבלי לכתוב את הקוד שלו מחדש.

קומפוננטות מכילות HTML, CSS ו־JS כך שהן משמשות רכיב פונקציונלי וויזואלי.

כפי שציינו מקודם, כל Framework יכול לממש אותן שונה, לכן שימוש בקומפוננטה מ־Framework אחד ב־Framework אחר לא בהכרח יעבוד.

אחד המאפיינים המאפשרים ל־component להופיע מספר פעמים באפליקציה באזורים שונים, הוא כמיסה (Encapsulation).
כמיסה אומרת שאפשר לשים את הקומפוננטה בכל אזור באפליקציה מבלי שאלמנטים אחרים ישפיעו עליו, או, מבלי שהוא ישפיע על אלמנטים אחרים סביבו (לדוגמא חוקי CSS).
דבר זה אפשרי בעזרת ה־Shadow DOM.

קראו את ה־section על Web Components ב־javascript.info.

קומפוננטות טיפשות וחכמות

ישנם 2 סוגי ראשיים של קומפוננטות:

1. קומפוננטות טיפשות
2. קומפוננטות חכמות.

קומפוננטות טיפשות

קומפוננטות טיפשות, או בשמן האחר, קומפוננטות ויזואליות, הינן קומפוננטות המתמקדות ב־UI, הן מקבלות מידע (מ־props או Input) אך לא מבקשות אותו (למשל בבקשת HTTP).

קומפוננטות חכמות

קומפוננטות חכמות יבקשו מידע ממקור חיצוני, לדוגמא בקשת HTTP אל השרת של האפליקציה.
בנוסף, הן יבצעו פעולות לוגיות, יגדירו את הפונקציונליות של האפליקציה וינהלו את ה־state שלה.

מומלץ להשתמש בקומפוננטות טיפשות בכל מקום בו זה אפשרי מכיוון שהן מאפשרות שימוש חוזר בהן, לעומת קומפוננטות חכמות.

לדוגמא:

רשימה של אנשי קשר - קיים מצב שבו נרצה להציג אנשי קשר ברשימה בכמה אזורים באפליקציה, כאשר כל אזור יציג רשימה אשר מכילה אנשי קשר שונים, למשל "מועדפים", אנשי קשר איתם יצרת קשר לאחרונה, ורשימה של כל אנשי הקשר.
במקרה ונשתמש בקומפוננטה טיפשה, נוכל להזריק אליה את אנשי הקשר שאנו מעוניינים בהם בכל חלק באפליקציה בנפרד, כך לא נצטרך להגדיר את הקומפוננטה מחדש בכל אזור (לא נצטרך לכתוב את הקוד שלה מחדש רק עם משהו קטן שונה).
מעל הקומפוננטה הטיפה, בכל אזור, נזריק לה מידע אשר יגיע מבקשת Http, אשר תוגדר בקומפוננטה חכמה.

במקרה ונחליט להשתמש בקומפוננטות חכמות (לדוגמא אחת אשר מבצעת קריאת Http על מנת לקבל את אנשי הקשר) נצטרך להגדיר קומפוננטה שונה לכל אזור באפליקציה, מפני שבקשת ה־HTTP תהיה שונה.

קראו את ה[מאמר הבא אשר מסביר על ההבדלים ביניהן](#).

React

React הינה ספריית javascript לבניית UI המבוססת על Components.
למדו את הבסיס של React על ידי קריאת המדריך learn by doing.

לאחר שהבנתם את הבסיס של React, קראו באופן מעמיק על הקונספטים העיקריים שלו.

שימו לב: עליכם לקרוא את כל הפרקים מ־Installation ועד Hooks כולל.
בפרק **Advanced Guides** עליכם לקרוא רק את תתי הפרקים הבאים:
context
fragments
higher order functions
Reconciliation
Refs and the DOM
Strict Mode
Type checking with PropTypes
בפרק **API reference** עליכם לקרוא רק את תתי הפרקים הבאים:
React
React.component
ReactDOM
Glossary

בצעו את כל המשימות שבכל הפרקים.
מדריך קונספטים עיקריים

שימו לב -
בעבר, העבודה עם React התבצעה עם LifeCycle Methods.
עליכם להכיר את הsyntax של גרסה זו שכן הרבה מערכות מבוססות עליה.
כיום, בעקבות הוספת Hooks ל־React סטנדרט מימוש ה־Components השתנה.

סטנדרטים - React

יש להוסיף Eslint, כפי שהוספתם לפרויקטים הקודמים. **שימו לב** - יש להוסיף פלאגין שיאפשר בדיקת קוד של JSX/TSX. בנוסף:

1. יש להשתמש ב־Arrow Functions **(ולא ב־Class Components!)**.
2. יש להשתמש ב־PascalCase בשיום פונקציות וקבצים (Button.tsx).
3. יש להשתמש בטייפסקריפט.

React - שאלות מבוא

1. מה ההבדל בין Functional Component ל־React.Component Class?
2. מה הבדל בין Props ל־State? מי שולט על שינוי של כל אחד מהם?
3. מתי רצים ה־Hooks הבאים:
   a. useState
   b. useEffect
4. מהי החשיבות של Key Attribute?
5. מה ההבדל בין ה־Attributes הבאים: className ל־class?

לאחר מכן התחילו את משימות Framework:

משימות Framework

משימה ראשונה:

צרו פרוייקט חדש, שיכיל דף אחד ראשי ו-2 קומפוננטות.
על הדף לכלול 4 Inputs שבהם ימלאו את הפרטים הבאים על אדם: שם פרטי, שם משפחה, אימייל וגיל.
יש להוסיף כפתור Submit, שבלחיצתו יתווסף האדם שיצרתם לרשימת אנשים.
הרשימה תהיה מתחת ל־Inputs ותציג את כל הפרטים עבור כל אדם.
עבור כל אדם ברשימה, יש להוסיף כפתור מחיקה שבלחיצה עליו האדם ימחק מהרשימה.
אם הגיל האדם גדול מ־20, השם שלו ברשימה יצבע באדום.
את האדם ה־5 ברשימה לא יהיה ניתן למחוק ממנה.

משימה שנייה:

כתבו אפליקציה לניהול משימות אשר תאפשר למשתמש להוסיף, למחוק, לשנות ולהציג את המשימות שלו.
לכל משימה יהיה שם ותיאור.

האפליקציה תכיל את הדפים הבאים:

- דף הצגת משימות
- דף יצירת\עריכת משימות

לאפליקציה לא יהיה צד שרת, לכן המידע לא ישמר ביציאה מהאפליקציה.
בדף יצירת ועריכת משימות יש להשתמש ב־API חיצוני על מנת להציג עובדה על חתולים :). את המידע תוכלו להשיג באמצעות פנייה ל־https://catfact.ninja/fact בבקשת GET.

משימת קורס - חלק 2 - צד לקוח

כעת, לאחר שכתבתם את צד השרת כתבו את צד הלקוח של המערכת אותה בחרתם בחלק הראשון בעזרת ה־Framework שלמדתם.

במשימה זו אתם יכולים להשתמש בכל חבילה (NPM) שתרצו, בתנאי שהוא עובד כראוי עם ה־Framework שלמדתם.

דרישות:

- לאפליקציה יהיו שני נתיבים:
  - /groups
  - /people
- כל נתיב שלא קיים באפליקציה יעשה Redirect לאחד מה־Routes הקיימים.
- בעמוד ה־Groups:
  תצוגה של כל הקבוצות הקיימות כך שניתן לראות בצורה ברורה מי האנשים שקיימים בכל קבוצה ומי הן הקבוצות הקיימות בכל קבוצה.
- בעמוד ה־People:
  תצוגה של כל האנשים שקיימים כך שליד כל בן אדם כתובות הקבוצות אליו הוא שייך.
- בכל עמוד ניתן יהיה לעשות את כל פעולות ה־CRUD (פעולות Create, Read, Update, Delete) - (גם בעמוד הקבוצות וגם בעמוד האנשים).
- מעבר נוח בין שני העמודים השונים שקיימים באפליקציה (ולא רק דרך שינוי ה־URL).
- שימו לב שאין דרישה מיוחדת על עיצוב האתר, עליכם לבחור לבד את הדרך שנראית בעיניכם נוחה לתפעול אפליקציה אינטרנטית.
- תוכלו להשתמש בחבילות עיצוב בפרויקט.

חבילות עיצוב

כפי שאמרנו בתחילת הקורס, בחברות אזרחיות קיימים מעצבים אשר מעצבים בשבילכם תבנית (template) לאתר ומספקים לכם תמונה על מנת שתעתיקו אותו. אבל, אצלנו בענף, אין מעצבים, לכן, אנו צריכים ללמוד לעצב בעצמנו ולנסות שהעיצובים יהיו כמה שיותר יפים, מקצועיים, קלים לשימוש ונוחים לעין.

מסיבה זו, מומלץ ללמוד עקרונות עיצוב אתרים, אחד מהעקרונות הידועים לעיצוב אתרים נקרא Material Design, מומלץ שתקראו עליו.

ישנם חבילות אשר מבוססות על Material Design:

- Angular Material - בשביל Angular
- Material-UI - בשביל React
- קיימות חבילות נוספות ו־Components סינגולריים.

גם במשימה זו עליכם לשמור על הסטנדרטים ולהשתמש ב־Git ולשתף אותנו עם ה־Repo שלכם.

העלאת צד לקוח על Docker Container

כתבו Dockerfile של צד הלקוח, צרו image ודחפו אותו אל ה־Harbor Registry הענפי תחת הפרויקט שלכם.
בשביל להצליח לעשות בקשות אל צד הלקוח, תצטרכו להשתמש ב־Nginx.
צפו בסרטון והיעזרו בדוקומנטציה (מיועדת למי שלמד React).
הוסיפו לצד הלקוח שבניתם Nginx, השתמשו ב־Docker Compose וקבצו את הכל ביחד.
העלו את הכל ובדקו שאתם מצליחים לבצע בקשות.

חלק רביעי - מתודולוגיות פיתוח וארכיטקטורה

מבוא

לפני שמתחילים לפתח פרויקט חדש עלינו לבחור מתודולוגיית פיתוח מסוימת שעל פיה נתכנן את הארכיטקטורה של המערכת.

ישנה כמות לא קטנה של מתודולוגיות פיתוח כאשר לכל אחת מהן יש עקרונות שונים אשר ישפיעו על הארכיטקטורה הסופית של המערכת ועל אופן הפיתוח שלה.

אנו נתמקד ב־2 מתודולוגיות 'הפוכות':

- Monolithic
- MicroServices

Monolithic vs Microservices

קראו על ההבדלים בין monolithic ל־Microservices במקור הבא או במקורות אחרים.

על מנת ללמוד יותר על microservice הנכם יכולים להשתמש במקור הבא: Microservices.io

משימת מבוא

ענו על השאלות הבאות:

1. מה ההבדל בין ארכיטקטורת Microservices ו־Monolithic? הרחיבו.
2. מה היתרונות של כל אחת מהן?
3. מה החסרונות של כל אחת מהן?
4. מתי מומלץ להשתמש בכל אחת מהן?
5. מה lose coupling אומר ולאיזו ארכיטקטורה עיקרון זה יותר מתאים?
6. מה הפלטפורמה הכי נוחה לפרוס אפליקציה שמורכבת מ־Microservices?

חישוב מבוזר

כאשר מפתחים אפליקציה בארכיטקטורת Microservices ניתן להשתמש בטכנולוגיות אשר מאפשרות חישוב מבוזר.

לימדו על RabbitMQ אשר עוזרת במימוש חישובים מבוזרים:

Message Queue

קראו אודות Message Queues ובמיוחד על RabbitMQ ובצעו את המשימה הבאה:

משימת Message Queues

- כתבו שני סרוויסים כאשר:
  - ה־Service הראשון יקבל בקשות HTTP POST.
    - ה־Service ישלח את תוכן ה־Body של כל בקשה ל־Service השני בעזרת messages של RabbitMQ.
    - ה־service יחזיר הודעת הצלחה ללקוח כל עוד ה־Message נשלח בהצלחה, ללא תלות באם הוא הגיע ל־service אחר.
  - ה־Service השני יקבל messages בעזרת RabbitMQ מה־Service הראשון
    - הוא ישמור תוכן כל הודעה בקובץ text בשם logs
    - כל הודעה תהיה בפורמט JSON.

Microservices - Components

ארכיטקטורת Microservices יכולים להיות מספר סוגים שונים של רכיבים, אנו נתמקד בבאים:

Services

אלו הם ה־Microservices עצמם.
כל service צריך לשמור על העקרונות הבאים:

1. Decoupled - אינו תלוי באף service אחר (לכן גם לא יתקשר עם service אחר)
2. Stateless - במידה ואפשרי, ה־service לא יכיל "state", זה אומר שמידע לא ישמר עליו, ואם הוא יפול, ויקום מופע אחר שלו שיחליף אותו, שום מידע לא יאבד בדרך.
3. Feature Based - מתאר "תכונה" אחת ויחידה במערכת.

דוגמאות ל־Microservices:

1. Comment Service - שירות אשר ישמור מידע על תגובות. המידע שישמר יכול להיות:
   a. טקסט התגובה
   b. ה־id של המשאב אליו התגובה קשורה
   c. סוג המשאב אליו התגובה קשורה
   d. ה־id של כותב התגובה
2. Folder Service - יכיל מידע על תיקייה, ישמור את השם שלה, את הנתיב שלה, ואת ההרשאות שלה.
3. File Service - ישמור פרטים על קבצים, לדוגמא השם שלהם, זמן העלאה שלהם, סוג הקובץ, id של התיקייה אליה הוא שייך.

Compositors

כל microservice אחראי על תכונה (feature) אחרת במערכת.
דבר זה יכול ליצור מספר בעיות כאשר רוצים לממש תכונה גדולה יותר.

לדוגמא, קיים Channel Service אשר מייצג ערוצים של סרטונים, בנוסף, קיים Video Service אשר מייצג סרטונים.
לשני השירותים אין תלות האחד בשני, אם אחד נופל מבעיה פנימית שלו, השירות השני ישאר למעלה (כל עוד לא היו לו בעיות משלו).

מה יקרה אם נרצה ליצור סרטון חדש ולשייך אותו לערוץ מסוים?
על מנת לא להכניס למערכת מידע שקרי\לא אמין, עלי לבדוק שהערוץ אליו אני משייך את הסרטון קיים.

איך אוכל לעשות זאת? הרי אסור לי לפנות ישירות מה־video service אל ה־Channel Service, דבר זה יצור Coupling (תלות) ביניהם ואלול לגרום למצב שאם ה־channel service נופל, גם ה־Video Service יפול.

הפתרון הוא שימוש ב־Compositor או בשמותיו האחרים, API gateway / Aggregator.

ה־Compositor ייצג תכונה אחת גדולה, על ידי שילוב של מספר תכונות קטנות (services) ובנוסף, ישמש כמנתב ל־Services שבאחריותו.

מה בעצם ה־Compositor יעשה?
במקרה שלנו, הוא יגש ל־channel service וישאל אם ה־channel אליו אני מנסה לשייך סרטון, קיים.
במקרה וכן, הפעולה תמשיך בביצועה, במקרה ולא, נזרוק שגיאה.

ובכללי, ה־Compositor ישמש כמעין מגדיר חוקים בתוך התכונה הגדולה.
הוא ידאג לגשר על הפערים אשר נוצרים בעקבות ביזור המערכת (פערים שלא היו נוצרים במקרה והיינו משתמשים ב־ארכיטקטורה monolithic).

בנוסף, ה־Compositor משמש גם כ־Populator
כאשר הלקוח יבקש מידע מ־service מסויים, ה־Compositor ידאג למלא את אותו מידע, במידע נוסף אם הוא רלוונטי.

דוגמא:
נניח ויש לנו Comment Service, שירות לתגובות.
אותו שירות שומר את טקסט התגובה, על מה הייתה התגובה, ומי כתב אותה.

בנוסף, יש לנו User Service אשר אחראי על אחסון פרטי המשתמשים במערכת.

אנו רוצים לקבל את כל התגובות למשאב מסוים (נניח סרטון), שירות התגובות יכול לתת לנו את המידע הזה בקלות.
אבל, אם נשתמש רק בו, הלקוח יראה את ה־id של כותבי ההודעות במקום את השמות שלהם, אשר מאוחסנים ב־User Service.

כמובן שאנחנו יכולים לשלוף את אותם שמות דרך צד הלקוח, אך זאת יכולה להיות פעולה די מסורבלת ואיטית (בקשת שם הכותב של כל הודעה). במקום לקבל בקשה אחת עם כל הפרטים, הלקוח יצטרך לחכות ל־2 בקשות, האחת אחרי השנייה (וזה במקרה ונצליח בבקשה אחת לקבל את כל המשתמשים הרלוונטיים), דבר האיטי בהרבה מבקשה אחת אשר מכילה את כל הפרטים.

הדרך היעילה והנוחה יותר תהיה לבצע את ה־Population ב־Compositor באופן הבא:

כאשר הלקוח מבקש תגובות על משאב מסוים, ה־Compositor יקבל את הבקשה, יפנה ל־comment service עם אותם פרטים (פרטי הבקשה).
ה־Compositor יקבל חזרה את כל התגובות, יוציא מהם את ה־userId ואז יבצע פניה ל־User Service ויקבל את כל המשתמשים.
לאחר מכן, הוא יחליף לכל תגובה את ה־userId בכל פרטי המשתמש (או רק השם שלו) ויחזיר את התגובות עם המשתמשים ללקוח.

מבנה ה־Compositor

ה־Compositor יפעל בדרך הבאה:

1. יתפוס את כל בקשות ה־Get שקשורות לתכונה אותה הוא מייצג אשר זקוקות ל־Population של מידע.
   a. ה־Compositor יבצע בקשה, בעצמו, לשירות הרלוונטי, ויקבל ממנו את המידע המבוקש.
   b. ה־Compositor יזקק את המידע הטהור אותו הוא צריך לקבל (למשל מערכת של user ids אותו הוא צריך להפוך ל־Users מלאים)
   c. ה־Compositor יבצע בקשה לשירות אחר, אשר יכול לספק את המידע הנחוץ ל־Population
   d. ה־Compositor ירכיב תשובה יחד עם המידע הנוסף שחזר ויחזיר אותה ללקוח.
2. יתפוס כל בקשת Create\Update\Delete אשר משפיעה על מספר תכונות (services) שונות, ויוודא שהן לא פוגעות בתקינות המערכת\המידע שבמערכת.
3. יבצע proxy, אל השירותים המתאימים, לכל הבקשות אותן אינו תפס בסעיפים הקודמים.

דוגמא ל־Compositors

Comment Compositor - יחבר בין כל השירותים אשר קשורים לתגובות (שירות התגובות עצמו, שירות המשתמשים, אשר מכיל את הפרטים על מגיבים פוטנציאליים, וכל השירותים אשר מייצגים תכונה עליה ניתן להגיב).

למשל, החיבורים יהיו:

1. Comment Service - פרטי התגובות
2. User Service - פרטי המשתמשים
3. Video Service - תכונה עליה אפשר להגיב
4. Channel Service - תכונה עליה אפשר להגיב.

ה־Compositor ישתמש בשירותי הסרטים והערוצים כדי לבדוק אם הערוץ\הסרטון עליו תגובה נוצרה באמת קיים.

ה־Comment Service יכול לפעול לבדו, ללא אף service אחר, כנל לגבי ה־User Service, ה־Video Service וה־Channel Service.

למרות זאת, תגובות צריכות להיות קשורות למשאב ספציפי, לכן, כדי "למלא" את ה־feature של התגובות, נשתמש ב־Comment Compositor כדי לתווך בין השירותים השונים ולאפשר תגובות עליהם.

נגדיש, ה־Comment Service לא חייב את ה־Comment Compositor על מנת לפעול. הוא יוכל לשמור מידע על תגובות על משאבים שונים ללא כל תלות בגורם חיצוני. אבל, על מנת לשמור על טוהר המידע במערכת, נעדיף שלא ליצור תגובות על משאבים שאינם קיימים, לכן נכניס את ה Comment Compositor כדי שישמש כעטיפת השלמה ל־Comment feature.

סיכום Compositors

לסיכום, Compositors אחראים על הדברים הבאים:

1. ניתוב בקשות לשירותים הרלוונטיים
2. אגרגציות - הוספת מידע לבקשות לשירות ספציפי על ידי קבלת מידע משירותים אחרים.
3. שמירה על נכונות המידע במערכת, מניעת שמירת מידע שקרי ושמירה על החיבור התקין בין שירותים במערכת.

משימת קורס - חלק 3 - Microservices

עליכם לקחת את המערכת שכתבתם בחלק הראשון של משימת הקורס ולהתאים אותו לארכיטקטורת Microservices, על ידי הפרדתו ליותר מ־Service אחד. בנוסף, יש לענות על השאלות הבאות:

1. מהם היתרונות של ההפרדה שעליכם לבצע כעת?
2. מהם החסרונות של ההפרדה?
3. מהם התכונות של המערכת שעליי להעביר ל־Compositor על מנת לשמור על עקרונות ה־Microservices? פרטו.
4. איזה בקשות יוכלו לעבור דרך פרוקסי אל הסרוויסים? מדוע?

עליכם לדאוג לשמור על העקרונות של Microservices בזמן ביצוע המשימה.

חלק חמישי - העשרת חובה בתחום ה־DevOps

DevOps היא פילוסופיה תרבותית שמאטמטת (מלשון אוטומטי) ומשלבת את התהליכים בין צוותי פיתוח תוכנה לבין צוותי האופרציה, על מנת לשפר את שיתוף הפעולה והפרודוקטיביות.
אפשר לחשוב על DevOps כעל אבולוציה של שיטת Agile, או כחלק חסר שלה מכיוון שעקרונות מסוימים של Agile מתממשים בצורתם השלמה ביותר רק כאשר משתמשים בפילוסופיית ה־DevOps.
פילוסופיית ה־DevOps מיישמת את החידושים של גישת ה־Agile בתהליכי פריסה של הקוד - צפו בסרטון הבא.
DevOps הוא תחום רחב מאוד - מהפילוסופיה שלו התפתחו תתי תחומים רבים (כמעט לכל phase שראיתם בסרטון) שמטרתם להתנהל טוב יותר בעולם של big data.

ההיסטוריה של DevOps היא פשוטה, אך מהפכנית.
הרעיון של הפילוסופיה צץ מתוך דיון בין אנדרו קליי ופטריק דבואס בשנת 2008. הם היו מודאגים מהחסרונות של גישת ה־Agile ורצו להמציא משהו טוב יותר. הרעיון החל להתפשט אט אט ולאחר האירוע - DevOpsDays שנערך בבלגיה ב־2009, הוא הפך למילת באזז בעולם.
DevOps הוא לא רק ניסיון להתייעלות, הוא צעד לקראת שינוי תרבותי.

- תהליך האינטגרציה של DevOps מגשר על אי ההבנות ומבטיח שיתוף פעולה בין- מפתחים, מהנדסי QA ומנהלי מערכת ((**Ops**) And Operations (**Dev**) Software Development). תקשורת והבנה טובים יותר יעזרו גם לצוותים להכיר בסדר העדיפויות שלהם.
- הטמעה של הפילוסופיה מבטיחה שמפתחים יכולים כעת לקחת חלק בפריסה, מנהלי מערכת יכולים לכתוב סקריפטים ומהנדסי QA יכולים לדבג תקלות ללא שימוש ידני בבדיקות. תהליכים יכולים להיות אוטומטיים ואף אחד לא צריך לחכות מכיוון שהם יכולים לבצע הכל בעצמם ב"שירות עצמי".

היתרונות של DevOps גורמים לפרודוקטיביות מרקיעת שחקים ומהירויות דלוור מהירות.

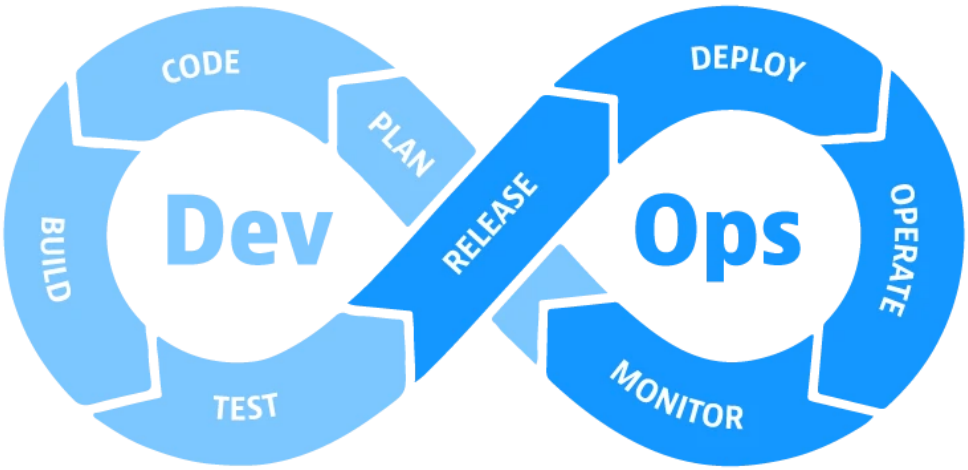
לסיכום,
DevOps הוא שינוי ארגוני שהתעשייה הייתה זקוקה לו מאוד.
הגשר שיצר DevOps בין פיתוח לאופרציה, התפתח כעת לנהר של פעילות שבו כולם תורמים באופן חיובי.

DevOps בענף

מכיוון ש- DevOps זה תחום בעל תתי נושאים מרובים כפי שהסברנו קודם, נמקד את החומר בעבודה של צוות DevOps בענף והשירותים שהוא מציע.
בשביל להעלות אפליקציה בצורה הטובה, הנוחה והזריזה ביותר נצטרך להכיר את הכלים הבאים שצוות Neptune (DevOps) מנגיש:

1. פלטפורמת אורקסטרציה חכמה (Kubernetes)
2. תהליך (pipeline) של DevOps טהור המורכב מ-CI/CD ופריסה על ה־Kubernetes
3. פלטפורמת Ansible שתפקידה לפרוס פלטפורמות/ תוכנות/ שירותים ולאפשר את הכל Xaas
4. ניהול והנגשת סביבת הפיתוח באינטרנט האזרחי (ה־Azure cloud)
5. Image registries
6. Vault לאבטחת משתני הסביבה

מיד נסביר מה הם הכלים הללו (הרלוונטים לקורס זה) וכיצד להשתמש בהם, אך קודם נסביר את אבני הדרך:



גישת ה־GitOps

GitOps uses Git repositories as a single source of truth to deliver infrastructure as code. Submitted code triggers the CI process, while the CD process applies requirements for things like security, infrastructure as code, or any other boundaries set for the application Framework. All changes to code are tracked, making updates easy while also providing version control should a rollback be needed.

גישה זו אומרת, שהקוד הקיים בגיט הוא הקוד המעודכן שרץ על הסביבה. הדרך למימוש גישה זו, היא בכך שכל push לגיט מהווה טריגר לתהליך CI/CD שתפקידו לפרוס על פלטפורמה (לדוגמא ה־Kubernetes) את הקוד החדש שעכשיו נדחף. בגישה הזו, נוכל לזהות באגים מהר יותר, לתקן אותם מהר יותר, לשחרר גרסאות מהר יותר ולבסוף גם להוציא מוצר יותר טוב!

תהליך ה־CI/CD

Continuous Integration and Continuous Delivery
Continuous Integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are running, and then an image is built and pushed to an image registry

Continuous Delivery automates the entire delivery process, including the deployment process on an infrastructure.
Those two create a self automatic service environment.

צפו בסרטון.

פלטפורמת ה־Kubernetes

Kubernetes or k8s is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. That way, you can track and control all of your Microservices with some very simple commands.

1. צפו בסרטון
2. צפו בסרטון עד לדקה 3:40

קוברנטיס הוא הפלטפורמה שעליה רוב מערכות הענף רצות, לפלטפורמה, כפי שראיתם יש יתרונות רבים והיא מיישמת את פילוסופיית ה־DevOps וגישת ה־Agile במהותה.
קוברנטיס מורכב מכמה שרתים (VM) שעליהם רצים Docker Containers, שבקוברנטיס נקראים - Pods.
(Pods הם אבסטרקציה מעל ה־Containers אבל זו הרחבה שאינה נדרשת לקורס זה).

משימה

העלו את האפליקציה שלכם לאחר ההפרדה ל־Microservices על ה־K8S והשתמשו ב־CI/CD Pipeline הענפי.
מדריך למשתמש

סיכום הקורס

כל הכבוד, הגעתם לסוף.
למדתם איך אתרים עובדים, איך בונים אותם, איך מעצבים אותם, איך גורמים להם לעשות בדיוק מה שתרצו שהם יעשו.

גם למדתם איך לבנות את צד השרת, ואיך לבנות את צד הלקוח בצורה מהירה ואיכותית.
בנוסף, למדתם איך כותבים קוד נכון, איך מנהלים את הקוד שלכם ואיך דואגים שהוא ישאר תקין לאורך כל תהליך הפיתוח.

בנוסף, נחשפתם לעולמות ה־DevOps, ולמדתם כיצד מוקמים ומועלים פרויקטים בענף שלנו.

כעת, אתם מוכנים להתחיל לעבוד על פרויקטים.

ברוכים הבאים לענף שלנו.

בהצלחה.