Node.js

Node - צומת. כשיש רשת של מחשבים, מחשב בודד נקרא Node.

מנוע להרצת קוד JavaScript מחוץ לדפדפן.

בשנת 2009 היו שלוש אפשרויות עיקריות לפיתוח שרתים:

1. מייקרוסופט - עלתה הרבה כסף, והצריכה ידע בכלי מייקרוספוט, וגם הוכחה כלא יעילה ל-scale של אינטרנט (ראה מקרה MySpace למשל)
2. ג'אווה - מצד אחד ג'אווה הייתה שפה קשה ומסובכת, ורצה בעיקר על שרתי יוניקס יקרים.
3. לאמפ - LAMP - Linux, Apache, MySQL, PHP/Perl/Python

שרת ה-Web של ה-LAMP היה Apache

אפאצ'י, היה שרת CGI - Common Gateway Interface



נוד פותח על ידי בחור בשם ריאן דאל



מרכיבי Node.js:

1. מנוע V8 של Chromium
2. ספריית לינוקס שנקראת libuv כדי לממש את הפעולות האסינכרוניות
3. איבנט לופ - event loop

תוצר הלוואי המבורך:

1. ג'אווה סקריפט בפרונטאנד
2. בבקאנד PHP
3. דטהבייס: MySQL
4. פרונטאנד: ג'אווהסקריפט
5. בקאנד: ג'אווהסקריפט
6. דטהבייס (מונגודיבי): ג'אווהסקריפט

זה בעצם ה-FullStack המקורי. המושג נטבע ברגע שהייתה ג'אווהסקריפט בכל אחד מרכיבי ה-stack.

Nodemon

מאפשר להריץ תוכניות נוד רגישות לשינויי קוד. כלומר ההרצה תתרחש מחדש בכל פעם שנשמור שינויים לקוד.

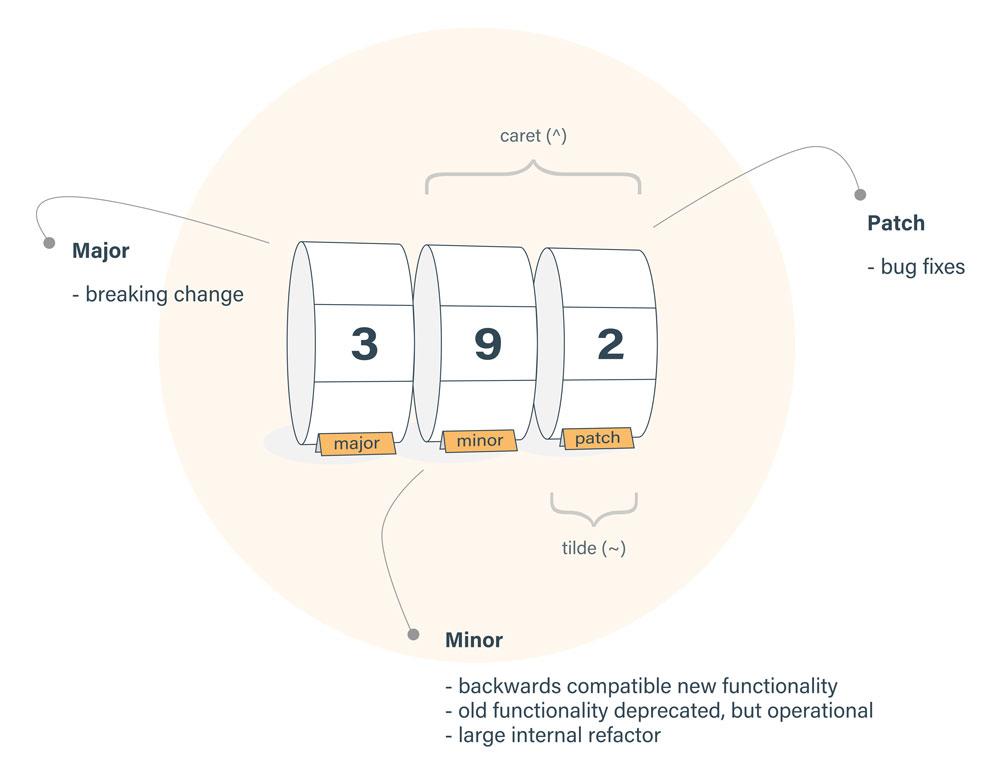
התקנה: npm i -g nodemon (התקינו כאדמין)

מודולים.

ישנם 3 סוגי מודולים:

1. מודולים שאנחנו מפתחים בעצמנו (custom modules)
2. מודולים שנגישים לנו מתוך Node.js עצמו
3. מודולים הזמינים ציבורית (או ארגונית) דרך NPM. אלו בעצם custom models שלא אנחנו פיתחנו.באמצעותם ניתן לנצל את כח הקהילה/הארגון כדי לייעל את הקוד שלנו ולא לפתח את הגלגל מחדש בכל פעם.

גרסאות מודולים ב-NPM



לכל מודול NPM יש גרסה בת 3 מספרים בפורמט:

Major.minor.patch

למשל:

1.4.5

גרסת ה-patch מקודמת בתיקוני באגים

גרסת ה-minor מקודמת בתוספות לקוד

גרסת ה-major מקודמת בתוספות לקוד ששוברות גרסאות קודמות.

בקבתי ה-package.json שלנו בחלק ה-dependencies נוכל לקבוע אילו גרסאות נרצה לקבל באמצעות סימנים מיוחדים:

1.4.5 - יביא לנו אך ורק את גרסה זו ספציפית

^1.4.5 - יביא לנו את גרסת ה-minor הגבוהה ביותר שימצא מעל הגרסה המצויינת

~1.4.5 - יביא לנו את גרסת ה-patch הגבוהה ביותר שימצא מעל הגרסה המצויינת

\* או latest יביא לנו את הגרסה האחרונה שימצא

תפקיד ה-package-lock.json

תפקידו לוודא שכל חברי הצוות שעובדים על פרוייקט אחד מסויים, יעבדו על אותן גרסאות של dependencies (בפועל אי אפשר לאכוף את זה, אבל בכל פעם שמישהו מריץ npm install ה-dependencies שלו יתעדכנו עד לרמת ה-package-lock ולא מעבר, ורק כשמישהו יריץ יזום npm update ויעלה גרסה מעל למה שמצויין ב-package-lock רק אז, אחרי שהוא יעשה קומיט לקובץ ה-lock כל שאר חברי הצוות יצטרכו להסתנכרן איתו

יצירה יזומה של קובץ package.json: npm init -y

פיבלוש מודולים ל-npm:

1. npm login
2. Npm publish

עידכון מודול ב-npm:

1. לקדם את הגרסה בקובץ package.json לפי השינוי שעשינו
2. npm publish

Debuuging

הפעולה שאנו המפתחים עושים כדי למצוא שגיאות בקוד. בעיקר שגיאות לוגיות.

כיום אנו יודעים לדבג באמצעות console.log

אבל ל-IDEs בדרך כלל יש debugger-ים יותר משוכללים, ועם המעבר לתוכנת צד שרת, הגיע הזמן להציג אותם.

Ts-node

npm i ts-node -D

זהו מודול המאפשר לקמפל טייפסקריפט בזכרון ולהריץ אותו מבלי שיישמר לקובץ.

משמעות @types

כל npm module הכתוב ב-JavaScript ורוצה לאפשר למפתחים להשתמש בו גם ב-TypeScript צריך לפרסם Types. למה:

1. כדי שהקומפיילר של TypeScript יוכל לאכוף את הטיפוסים שלו
2. וכדי לאפשר InteliSense ב-IDE

פירסום types נעשה באחת משתי דרכים:

1. הכללת קבצי \*.d.ts במודול המפובלש
2. או פרסום הטייפים כ-npm module נפרד בספרייה הגלובלית של @types. מי שבוחר בשיטה זו, ומפבלש ל-npm מודל בשם למשל myLovelyModule יצטרך לפבלש מודול בשם @types/myLovelyModule

Express

פריימוורק לכתיבת קוד שרת בנוד\טייפסקריפט. נועדה להקל עלינו את כתיבת השרת, להכליל פעולות שנרצה לעשות באופן די קבוע ושהשרת הבסיסי של נוד לא מאפשר לנו, למשל:

* לפענח יו אר אלים
* לפענח פרמטרים
* לפענח post bodys
* להחזיר לקליינט תשובות בפורמטים שונים
* לסדר את הקוד שלנו

בסה"כ אקספרס לנוד, היא כמו ריאקט לואנילה ג'אווהסקריפט. תמיד נעדיף להשתמש באקספרס על פני נוד בסיסי.

מודל העבודה של אקספרס, מודל ה-middleware



Backend Layers



ORM - Object Relational Mapping

כלים שממפים אובייקטים בשפת תכנות לטבלאות בדטהבייס. כיום לא נהוג לכתוב קוד SQL אלא להשתמש בכלי ORM שמייצרים את ה-SQL בעתצמם. אנחנו לא נשתמש ב-ORM

Express Router

ראוטר הוא סוג של מיני אפליקציית אקספרס שמאגד בתוכו ראוטים בעלי מאפיינים דומים. למשל כל הראוטים שקשורים ל-products

DTO - Data Transfer Object

זהו אובייקט המייצג יישות מידע, אך מכיל בתוכו אך ורק את הנתונים ולא שום פעולה עליהם. בגדול, אובייקט זה יכול להיות משותף בין ה-FE וה-BE.

Node-config

מודל של npm שעובד בצורה כזו:

* אנו מגדירים קונפיגורציה כללית ככל שניתן בקובץ config/default.json
* עבור כל סביבה שאליה נרצה לדפלה את האפליקציה ניצור קובץ json חדש בשם הסביבה, בה נדרוס את הקונפיגורציה הדיפולטית, איפה שנצטרך.

Middlewares

* מידלוור סטנדרטי הוא פונקציה שחתימתה היא (request, response, next). כדי לצאת ממידלוור, צריך לבצע response.end או next()
* אררור מידלוור (error middleware) הוא מידלוור מיוחד שהוא פונקציה שחתימתה היא (error, request, response, next). כדי להפעיל אררור מידלוור, צריך לבצע next(err)
* מידלוור נטענת על ידי פונקציית use באקספרס או ב-router
* אקספרס נותן לנו utility functions עבור כל פעלי ה-http כדי להקל עלינו. כלומר, גם פונקציות get, post, put, patch, delete הן פונקציות הטוענות מידלוורים.
* עלינו לזכור שבסופו של דבר, במבט על, נוכל להסתכל על אפליקציית אקספרס ולהבין את מפת המידלוורים שיצרנו:



Input validation

חובה עלינו לוודא שהתכנים שאנו מקבלים מהמשתמשים (קרי מאפליקציות ה-FE למיניהן) תואמים למבנה שבו אנו רוצים לקבל אותם. למשל, יש שדות חובה, יש טיפוסים מסויימים לשדות, יש שדות שאנחנו לא מוכנים לקבל וכיו"ב.

לצורך כך נשתמש ב-npm שנקרא joi

**Opinionated vs unopinonated**

* אקספרס היא פריימוורק מסוג שנקרא unopinionated - אין לאקספרס מושג מה אנחנו רוצים לפתח. הדבר היחיד שהיא יודעת זה שאנחנו רוצים לפתח שרת ווב. מעבר לזה כלום. ולכן נצטרך לכתוב יותר קוד שיכיל את ה-opinion שלנו.
* פריימוורק אחר מאד פופולרי היום, נקרא nest.js והוא מסוג opinionated, כלומר, מבחינתו אנחנו רוצים לייצר שרת rest לסביבת microservices. והוא יכין הרבה דברים מראש, כך שנצטרך לכתוב פחות קוד. אבל - כל דבר שנסט לא הצליח לחשוב עליו מראש ובכל זאת נרצה ליישם באמצעותו, יהיה לנו הרבה יותר קשה להשיג זאת.

**Regular expression (regex)**

זו שפה המתארת מבנה של מחרוזות באמצעות תבניות.

למשל:

איך נתאר את מבנה המחרוזת שהוא אימייל?

[john@mysite.com](mailto:john@mysite.com)

זה מתחיל במחרוזת בעלת מינימום תו אחד, המכיל תווים אלפא נומריים

אח"כ חייב להיות @

אח"כ חייב להיות שם דומיין שזה מחרוזת באורך 1 תווים לפחות.

ואז חייבת להיות . (נקודה)

ואז חייב להיות TLD שזו מחרוזת בת 2-8 תווים אלפאנומריים

אז מה זה רגקס?

אני אתן לך מחרוזת למשל : a#5@@mysite,com

התפקיד של הרגקס הוא להגיד, true or false האם המחרוזת הזאת נופלת בתבנית.

יש לרגקס עוד יישום, והוא לחפש תתי מחרוזות בתוך מחרוזות ארוכות. למשל, מצא לי את כל האימיילים במחרוזת הבאה:

[a@a.com](mailto:a@a.com);b@b.rugghdfj





טסטים

טסט היא תוכנה קטנה, שאנו כותבים כדי לבדוק את אמינות התוכנה המרכזית שאנו מפתחים.

טסט הוא חוזה בינינו לבין עצמנו. אתה מתחייב שהטסט תמיד יעבור. היה ובעתיד הטסט נכשל, משמע אחד משניים:

1. או שהטסט שגוי (סבירות נמוכה), ואז צריך לעדכן אותו
2. או שהוספנו באג

אם אתמול הרצנו את כל הטסטים, וכולם עברו. היום פיתחנו עוד קוד. ועכשיו אנו מריצים טסטים, ואחד לפחות נכשל, זה אומר שהוספנו באג.

וזו מטרת הטסטים - שאנו נמצא את הבאג, לפני שאחד המשתמשים ימצא את הבאג.

למעשה, בכל חברת תוכנה "נורמלית", PR (פול ריקווסט) שלא עובר טסטים (וכל פי אר מריץ טסטים) לא מתקבל.כלומר לא עושים לו merge לענף המרכזי של הקוד והקוד לעולם לא יגיע ל-production.

בפיתוח התוכנה המודרני, יש מושג שנקרא CI - continuous integration, משמעותו היא שכל קוד שרוצה להצטרף לכל אחד מענפי הגיט, עובר בדיקות אוטומטיות וגם נדחה אוטומטית אם הטסטים נכשלים.

יש גישה, שנקראת TDD - test driven development שלפיה, מפתחים את הטסטים עוד בכלל לפני שפיתחנו את הקוד.

סוגי טסטים שונים:

יש טסטים מכל מיני קטגוריות. Unit testing הוא הבסיסי ביותר, וכשמו כן הוא, הטסטים בודקים יחידות תוכנה קטנות ועצמאיות. כדי ליישם unit testing על קוד שאינו עצמאי, נאלץ (בכאב וסבל רב) לכתוב mock-ים לכל מה שאינו עצמאי בקוד. לדוגמא: יש לנו יחידת תוכנה שמתחברת לדטהבייס, נצטרך לכתוב מוק לדטהבייס.

עוד סוג של טסטים שנעבוד איתם, נקרא integration tests וכאן אנו כבר בודקים שילוב של כמה מערכות, למשל שרת ודטהבייס ללא מוקים. גם את זה נעשה על אפפ של אקספרס שנכתוב.\

לטסטים ב-javascript יש מספר פריימוורקים פופולריים. השניים הפופולריים ביותר הם mocha ו-jest. אנחנו נלמד jest.

npm i jest @types/jest ts-jest -D

npx ts-jest config:init

יש שתי גישות ל**איפה** כותבים טסטים.

1. גישת המוקה: מייצרים תיקיית tests וכותבים את כל הטסטים שם
2. גישת הג'סט: כותבים את הטסטים בצמוד למודולים שאותם הם בודקים