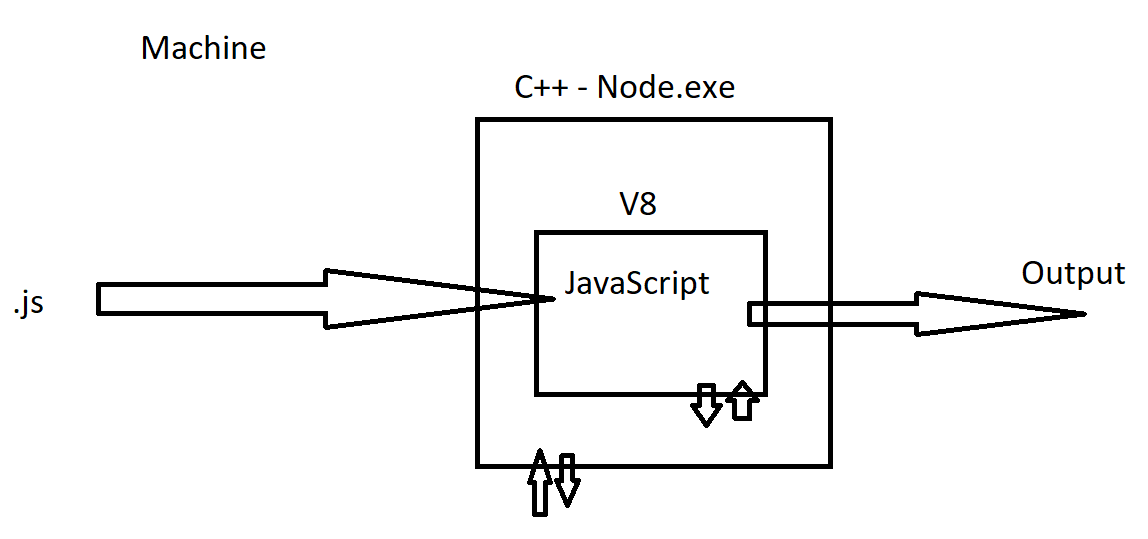
# **Node.js**

**Node.js**

סביבת הרצה של JavaScript מחוץ לדפדפן.



אין ב-Node.js:

* window
* DOM
* document
* תצוגה
* alert

יש בה בנוסף ל-JS במסגרת דפדפן:

* גישה ל-File System
* יצירה / קריאה של קבצים ותיקיות
* גישה למסדי נתונים
* משאבי מערכת ההפעלה

אתר רשמי: <https://nodejs.org/en>

**package.json**

זהו קובץ הגדרות ברמת פרויקט Node.js. מכיל שם, גרסה, dependencies ועוד.

**Package Manager**

מערכת הקיימת על שרת מרוחק והספקת ספריות עבור טכנולוגיית פיתוח כלשהי.

לדוגמה עבור C# קיים Package Manager בשם NuGet

לדוגמה עבור PHP קיים Package Manager בשם Composer

עבור Node.js קיים Package Manager בשם NPM – Node Package Manager.

**Package**

פרויקט Node.js שלם.

**global**

זהו האובייקט הגלובלי ב-Node.js (מקביל ל-window).

מכיל תשתית מסוימת עבור Node.js

**Module**

זהו קובץ js. אחד ויחיד.

כל קובץ כזה מכיל אוטומטית IIFE בתוכו בכדי לנתק את הקוד שלנו מהאובייקט הגלובלי – global

קיימים שלושה סוגי מודולים:

1. Custom Module – מודול שאנו בונים.
2. Built-In Module – מודול שקיים ב-Node.js כברירת מחדל. חלק מתשתית Node.js.
3. NPM Module – מודול הקיים ב-Package Manager שאנו עובדים איתו.

התקנה: npm install module-name

או בקיצור: npm i module-name

הסרה: npm uninstall module-name

קיים גם סוג של Module שאנו יכולים להתקין ברמת כל המחשב.

npm install --global module-name

או בקיצור: npm i -g module-name

זאת אך ורק עבור ספריות המכילות פקודות CLI – Command-Line Interface

**nodemon**

זו ספרייה המאפשרת להריץ פרויקט Node ע"י מנגנון Hot-Reloading

קיצור של Node Monitor

זו ספרייה המכילה פקודת CLI להרצת הפרויקט ולכן היא מותקנת ע"י npm i -g nodemon

**Promise**

זהו אובייקט Native המאפשר להריץ קוד אסינכרוני ויודע לדווח לנו ברגע שהסתיים – הצלחה או כשלון.

כמה שנים לאחר המצאת האובייקט – התווספו שתי מילים שמורות ל-JavaScript: async ו-await.

מילים אלו פועלות אך ורק על Promise.

async מוגדרת לפני הפונקציה שאנו בונים בכדי לציין למערכת שאנו כותבים פונקציה המכילה קוד אסינכרוני.

await נכתבת לפני אובייקט ה-Promise. היא מבצעת מספר דברים:

1. היא ממתינה לסיום פעילות ה-Promise.
2. ההמתנה לא תוקעת את כל המערכת אלא רק את הפונקציה הנוכחית שלנו ומשחררת את כל JavaScript להמשך פעילות.
3. מחזירה את התוצאה שאובייקט ה-Promise מדווח בעת הצלחה.

**Observable**

אובייקט המיועד להריץ קוד אסינכרוני שמדווח רצף של נתונים לאורך זמן.

ניתן להירשם אליו ולקבל את הדיווחים ממספר מקומות בקוד.

יכול לדווח שגיאה אחת (ואז הוא נסגר).

צריך לדווח אירוע נוסף של complete ברגע שרצף המידע הסתיים.

**To Promisify**

הפיכת קוד ישן, שעובד ע"י successCallback/errorCallback לכזה שעובד ע"י Promise

בכדי שנוכל לקרוא לקוד זה ע"י async-await.

**TypeScript**

שפת תכנות שנכתבה ע"י מייקרוסופט ב-2013.

מכילה קומפיילר שהופך את הקוד ל-JavaScript.

יתרונות:

1. TypeScript הינה Strongly Typed כאשר JavaScript אינה.
2. יש לה קומפיילר המבצע שני דברים: 1. בודק שגיאות ומתריע. 2 הופך את הקוד ל-Native JavaScript (נקרא Transpiler).
3. בגלל שהיא Strongly Typed קיים IntelliSense כמו שצריך.
4. יש לה תחביר OOP מלא, בדומה לשפות מתקדמות כמו #C או Java כאשר ה-OOP של JavaScript לוקה בחסר!
5. ועוד...

התקנת הקומפיילר של TypeScript:

npm i -g typescript

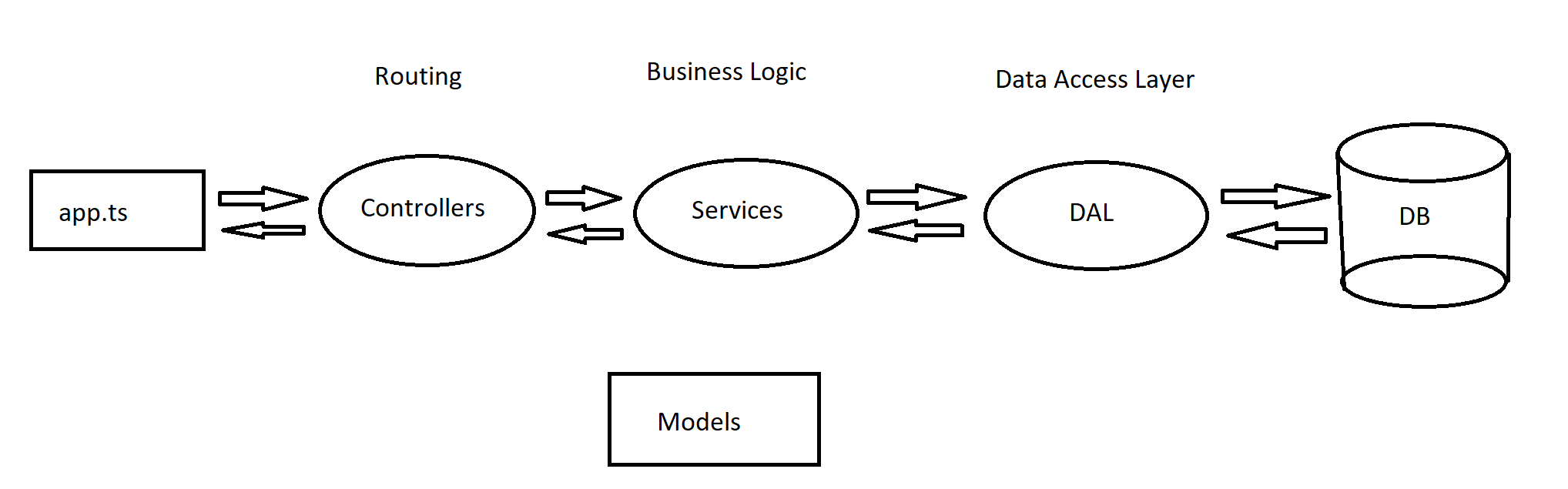
או לחילופין – התקנת ts-node המאפשר להריץ קוד Node.js שנכתב ב-TypeScript:

npm i -g ts-node

**Express**

ספריית Node.js לבניית REST API.

**Layered REST API**



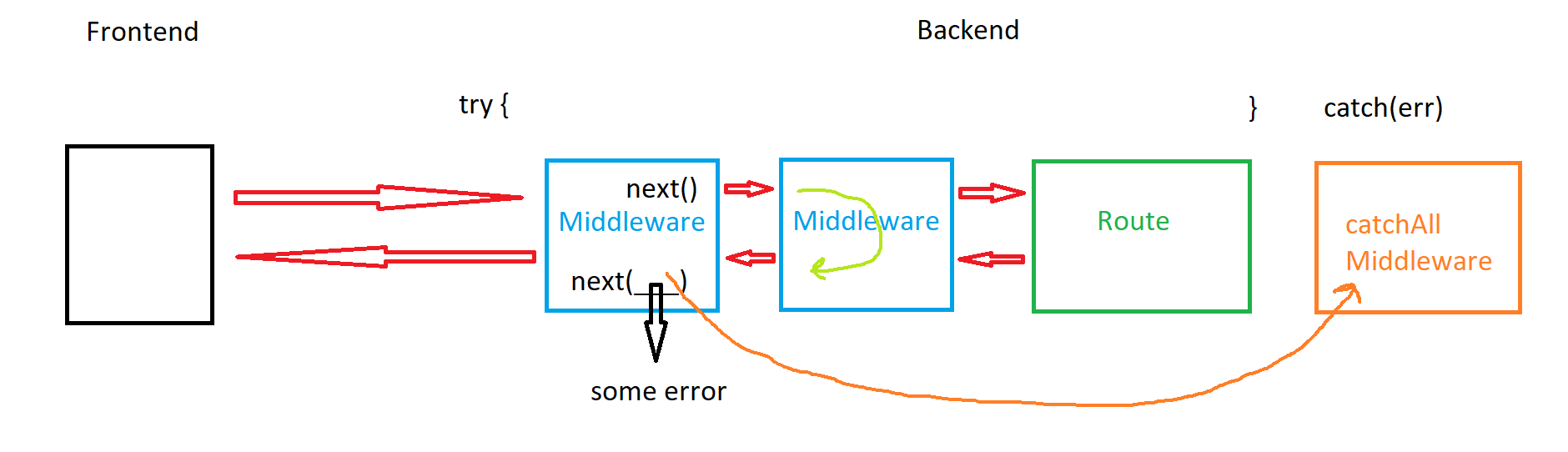
**Middleware**

זו פונקציה המתבצעת בצורה אוטומטית בין ה-Request לבין ה-Response.

Middleware יכול להתבצע אוטומטית עבור כל ה-Requests, עבור חלק מהם או עבור Request ספציפי.

Middleware יכול לבצע פעולה ולהמשיך את ה-Request הלאה ליעד שלו.

Middleware יכול להפסיק את ה-Request ומידית להחזיר Response כאשר במקרה כזה ה-Request לא יגיע ליעד שלו.



**DB**

ישנם שני סוגים של מסדי נתונים:

1. מסד נתונים רלציוני – Relational Database.

זהו מסד נתונים המכיל טבלאות וקישורים בין הטבלאות.

שפת SQL פועלת על מסדי נתונים אלו. לכן יותר קל לבצע שאילתות מורכבות.

דוגמאות: MySQL, Postgress, MSSQL – Sql Server, Oracle Server...

1. NoSQL

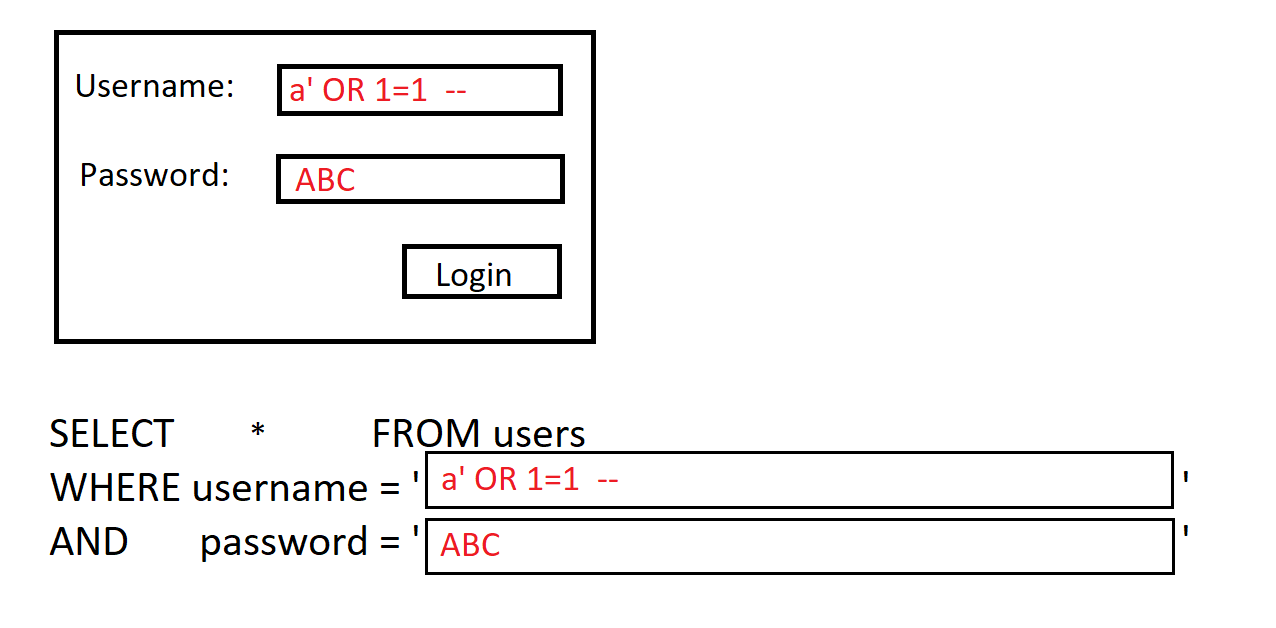
זהו מסד נתונים ללא טבלאות. המידע נמצא בו בפורמט אחר. לדוגמה JSON.

דוגמאות: MongoDB, Elastic Search, Cassandra, ...

אלו מסדי נתונים מהירים יותר המתאימים גם ל-BigData.

**SQL Injection**

התקפה נפוצה הנגרמת ע"י "הזרקת" חלק של שאילתת SQL לתוך תיבת טקסט או ערוץ קלט אחר, המשתלבת בשאילתה המקורית ומשנה את המשמעות שלה לצורך פריצה, גניבת מידע, מחיקת מידע, שיבוש טבלאות וכדומה.



**טיפול בקבצים**

ספרייה לטיפול בקבצים הנשלחים מצד הלקוח:

npm i express-fileupload

npm i @types/express-fileupload -D

ספרייה ליצירת UUID:

npm i uuid

npm i @types/uuid -D

**MongoDB**

מסד נתונים NoSQL בין הפופולריים ביותר היום. אין בו טבלאות, אין בו קישורים, המידע נשמר בפורמט BSON (Binary JSON).

**Collection**

זהו מערך של אובייקטים. מקביל לטבלה במסד נתונים רלציוני.

**Document**

זהו אובייקט אחד שקיים בתוך Collection. זה מקביל לרשומה.

**ORM**

Object Relational Mapper

זו ספרייה המאפשרת לייצג את המידע שבמסד הנתונים ע"י מחלקות ואובייקטים בקוד שלנו.

למחלקות הללו יש פונקציות Built-In המאפשרות לבצע פעולות על מסד הנתונים.

בדוטנט יש ORM בשם Entity Framework.

ל-MongoDB יש ORM בשם Mongoose.

התקנת Mongoose:

npm i mongoose

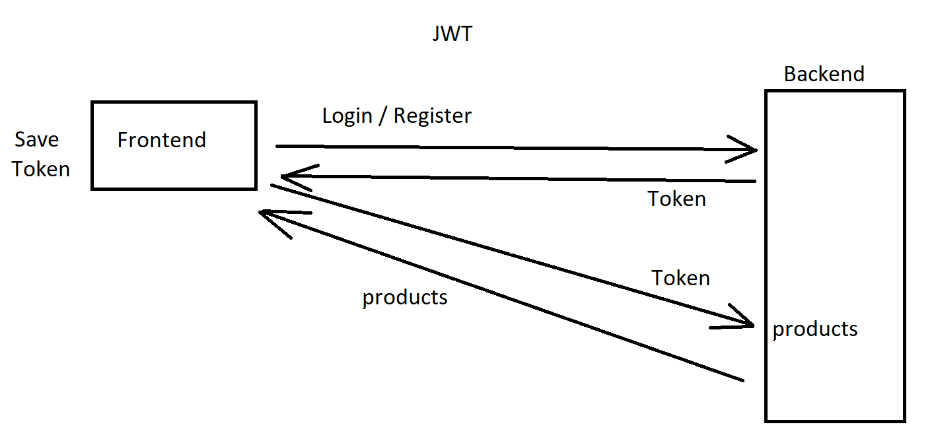
**Secure Coding**

ישנן מגוון פעולות שעל המפתח לבצע עבור בניית קוד מאובטח:

1. חובה לתעד כל שגיאה שהתרחשה ע"י רכיב Logger כלשהו.
2. אין להחזיר את הודעת השגיאה המקורית בסביבת ה-Production אלא רק בסביבת ה-Development.
3. באישור מדיניות CORS יש לאשר רק את כתובת אתר ה-Frontend שלנו ולא את כל העולם, אם המידע דרוש רק עבורנו ולא עבור כלל העולם.
4. עבור מנגנון Auth נהוג להשתמש בטכנולוגיית JWT.

Auth זהו קיצור של Authorization & Authentication (אימות והרשאות)

JWT – Json Web Token



1. אין להחזיר סיסמאות ל-Front בתוך מחרוזת ה-Token.
2. יש לחסום Routes הדורשים Logged In מפני כניסה ללא Token חוקי.
3. אסור לשמור סיסמאות במסד הנתונים כ-Plain Text!

זה לא נכון להצפין סיסמה ולפענח אותה.

סיסמאות שמורות במסד הנתונים ע"י "גיבוב" – "Hash"

זהו אלגוריתם שמערבל את הסיסמה ויוצר ממנה מחרוזת ארוכה כלשהי. זו אינה מחרוזת רנדומלית. אך אין אלגוריתם שמחזיר את ה-Hash בחזרה לסיסמה המקורית. זהו אלגוריתם חד כיווני.

1. חובה לבצע Salt על סיסמאות לפני ביצוע Hash בכדי שהסיסמה לא תתגלה ע"י אתרי Rainbow Tables (שמכילים מאגרים עצומים של Hash-ים).
2. חובה להגן מפני התקפת SQL Injection ע"י Prepared Statements ולא שילוב ערכי האובייקט בתוך השאילתה.
3. חובה להגן מפני התקפת XSS (Cross Site Scripting) ע"י מחיקת תגיות שהתקבלו במחרוזות לפני שמירה ב-DB.
4. חובה להגן מפני DoS Attack!

DoS = Denay of Service.

הגנה מתבצעת ע"י הספרייה:

npm i express-rate-limit

המאשרת רק מספר בקשות למשך חלון זמן מאותו IP Address.

ספרייה זו לא מגנה מפני DDoS Attack (למעשה נכון להיום אין אפשרות להגן 100% מפני DDoS).

1. יש למנוע פריצת IDOR – Insecure Direct Object Reference.

זהו מצב בו חשפנו נתון ללקוח שתוקף משתמש בו בכדי לקבל מידע אחר אם הוא משנה את אותו נתון.

לדוגמה, חשיפת קוד רץ בשורת הכתובת.

פתרון: שליחת UUID במקום id ל-Front לצורך זיהוי המשתמש.

1. חשוב להגן מפני BOT-ים.

פתרון: CAPTCHA – Completely Automated Turing test to tell Computers and Humans Apart.

1. חובה להשתמש בספרייה בשם Helmet להגנה מפני Header-ים שחושפים מידע ומכילים פרצות אבטחה.
2. חובה שהאתר ירוץ על פרוטוקול HTTPS ולא על פרוטוקול HTTP.

**Socket.io**

זוהי ספרייה המאפשרת ל-Backend לשלוח הודעות ל-Frontend בצורה יזומה.

פרוטוקול HTTP בנוי בצורה כזו שה-Frontend מבקש דף או מידע מה-Backend ורק עקב בקשה זו ה-Back מחזיר את הדף או את המידע.

פרוטוקול Web Socket (ws://) בנוי אחרת. כאן אנו יוצרים קו תקשורת בין ה-Frontend ל-Backend שעל גבי הקו הזה גם ה-Front יכול לשלוח מידע ל-Back וגם ה-Back יכול לשלוח מידע ל-Front – שניהם בצורה יזומה.

זה מתאים למצב בו ה-Back רוצה להודיע דברים ל-Front:

* פרסומות – שרת מעדכן לקוח
* משחקים – שרת שולח מידע שהתרחש אצל לקוחות אחרים בזמן אמת
* מבזקי חדשות בזמן אמת
* מכירה פומבית
* צ'אט
* Push Notifications (לדוגמה WhatsApp Web)

**Socket**

זהו אובייקט הקצה של קו התקשורת. נקודת התקשורת מסתיימת באובייקט Socket דרכו אפשר גם לשלוח מידע וגם להאזין להודעות מהצד השני.

**ארכיטקטורת Socket.io Single-Server Multiple-Clients:**



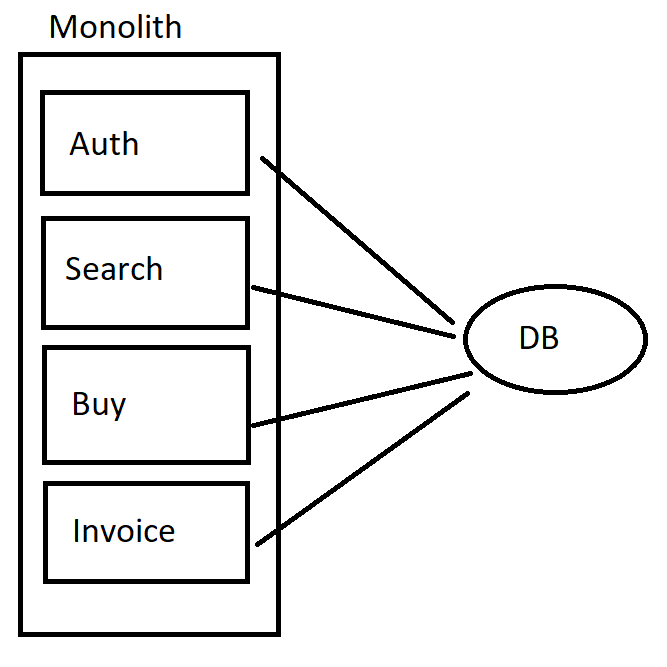
**Monolith**

זו מערכת הבנויה כיחידה אחת / פרויקט אחד המבצע את כלל הלוגיקה של כל המערכת.

לדוגמה REST API אחד ויחיד המאפשר להוסיף/לעדכן/למחוק/לקבל מידע ממספר טבלאות של מסד נתונים כלשהו.

חסרונות:

1. המון קוד, המון קבצים עבור מערכת בינונית ומעלה. קשה לתחזק, קשה לדבג.
2. אם יש לחץ על אזור אחד במערכת, זה משפיע על כלל המערכת.



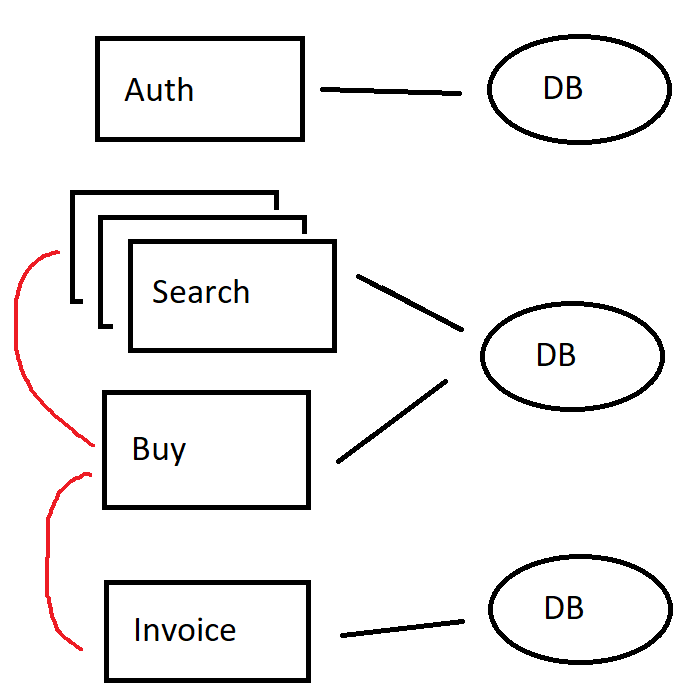
**Microservice**

יחידה קטנה של המערכת המכילה REST API קטן המטפל בתחום אחריות אחד מכלל המערכת.

יכול לתקשר עם Microservices אחרים, עם מסד נתונים משל עצמו או מסד נתונים משותף

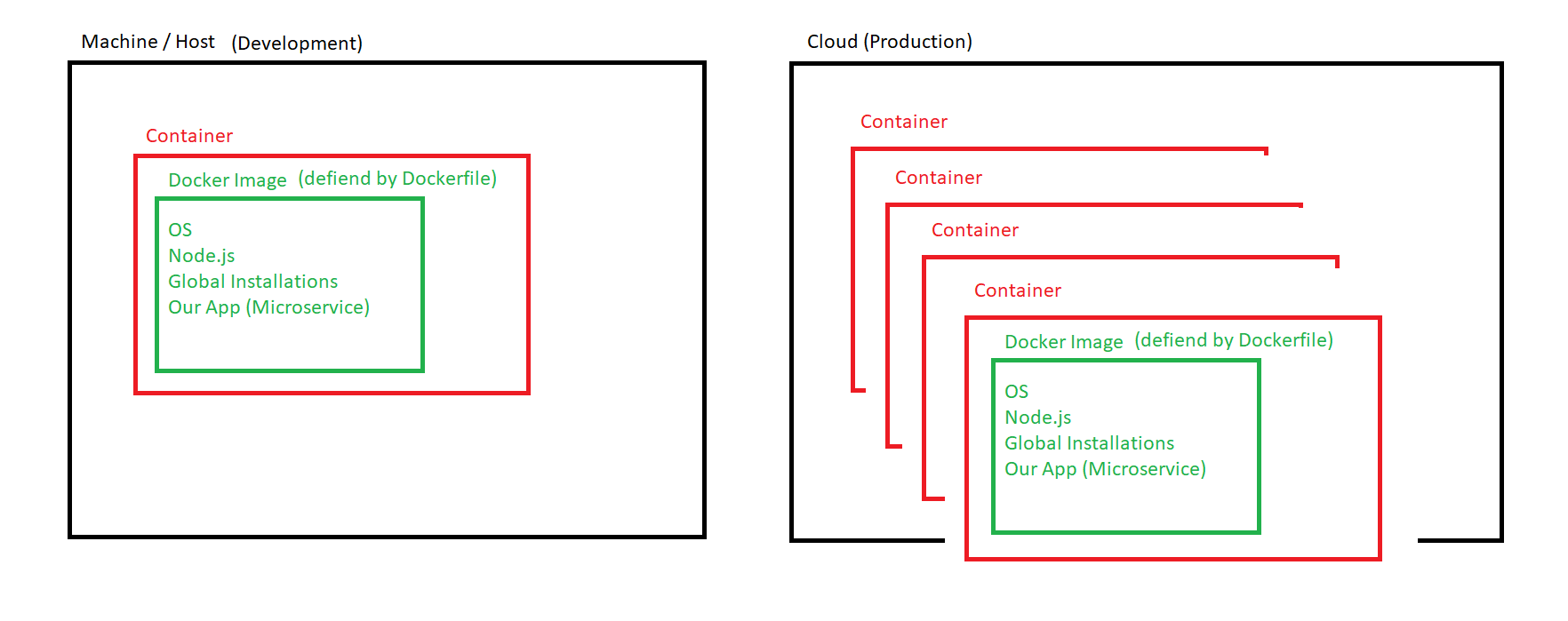
יתרונות:

1. קוד קטן, ניתן לתחזוקה בקלות, לטיפול בבאגים...
2. ניתן ע"י מערכות כגון Kubernetes לשכפל Microservices המכילים מספר רב של פניות לצורך יעילות כלל המערכת.



Docker

מערכת המאפשרת להריץ Container (כמו מכונה וירטואלית מצומצמת) בתוך המחשב שלנו (Host) המריץ בתוכו את האפליקציה ואת כל מה שהיא דורשת. זה מאפשר להעביר לאחר מכן את המערכת לשרת מרוחק שם היא תרוץ על Container זהה. בנוסף, ניתן להריץ ב-Containers מספר Microservices כך שאם אחד מהם מכיל דרישה גבוהה – ה-Container ישוכפל לצורך מענה עבור הדרישה הגבוהה מהלקוחות.



בהצלחה 😊