

NodeJS

Express

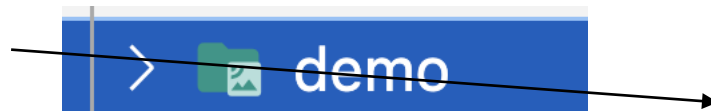
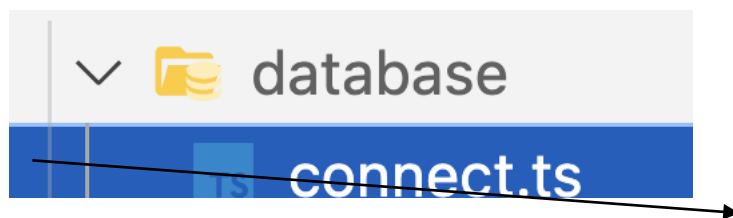
עבודה עם משתני סביבה - .env

עבודה עם Mongodb - Mongoose

עבודה על פרויקט מסכם

פרויקט לשיעור Lec5

העתקנו את הפרויקט משיעור קודם



package.json ×

package.json > name

```
1  {  
2    "name": "lec5",  
3    "version": "1.0.0",  
4    "description": "",  
5    "main": "src/index.ts",
```

פרויקט לשיעור Lec5

TS users.ts ×

src > routes > TS users.ts > ...

```
1 import { Router } from "express";
2
3 const usersRouter = Router();
4
5 export { usersRouter };
6
```

TS users.ts

TS people.ts ×

src > routes > TS people.ts > ...

```
1 import { Router } from "express";
2
3 const router = Router();
4
5 export { router as peopleRouter };
6
```

עבודה עם משתני סביבה:

הספריה env.

לא נרצה לשמור סיסמאות והגדרות כגון שם הדטה-בייס בתוך קוד המקור שלנו.

שם הדטה-בייס.

שינוי סיסמא - סבירות גבוהה שנרצה לעשות את זה בשלב מסויים

לא נרצה להטמיע את הסיסמא שלנו בתוך הלוגיקה בפרוייקט.

כי אחרת - כשנרצה להחליף סיסמא

בפרוייקט יכולים להיות המון קבצים

נצטרך לחפש בכל הקבצים איפה השתמשנו בסיסמא.

אם יהיה לנו קובץ ייעודי להגדרות:

נשנה בקובץ הזה בלבד.

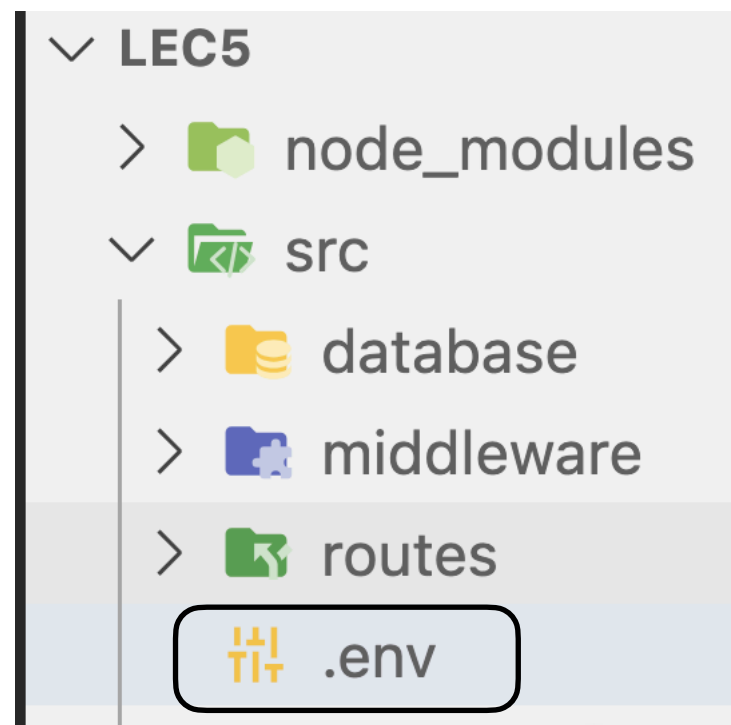
(לא נצטרך לחפש בכל הפרוייקט).

עבודה עם משתני סביבה:

הספרייה .env

```
▼ TERMINAL  
○ [ /Users/TomerBu/W160523MR/Node  
pnpm i dotenv
```

1



2

קובץ חדש - שם הקובץ חשוב.

```
.env ×  
src > .env  
1 F00=bar
```

3

עבודה עם משתני סביבה:

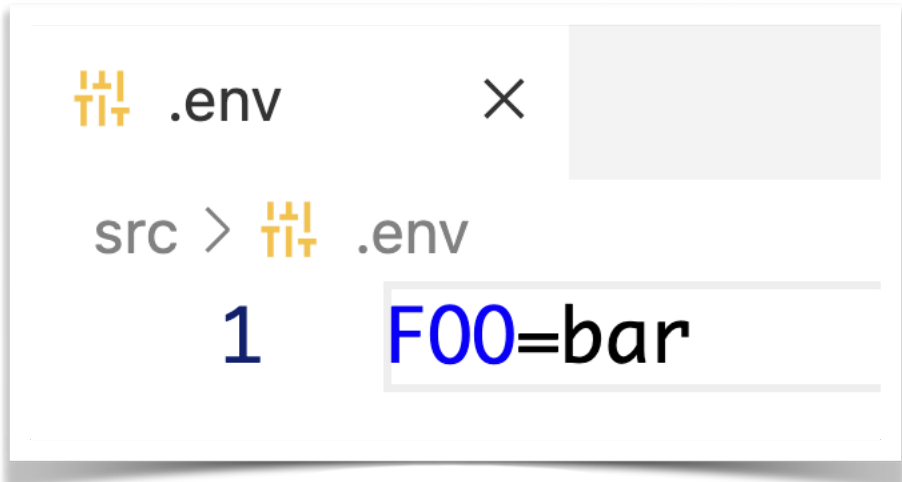
הספריה .env

index.ts

```
import express, { json } from "express";
import { usersRouter } from "../routes/users";
import { logger } from "../middleware/logger";
import { notFound } from "../middleware/not-found";
import { peopleRouter } from "../routes/people";
import { config } from "dotenv";
```

```
//the config function loads our vars from .env file
//to process.env (in memory)
config({ path: ".env" });
const app = express();
```

```
console.log(process.env.F00);
```



```
.env
src > .env
1 F00=bar
```

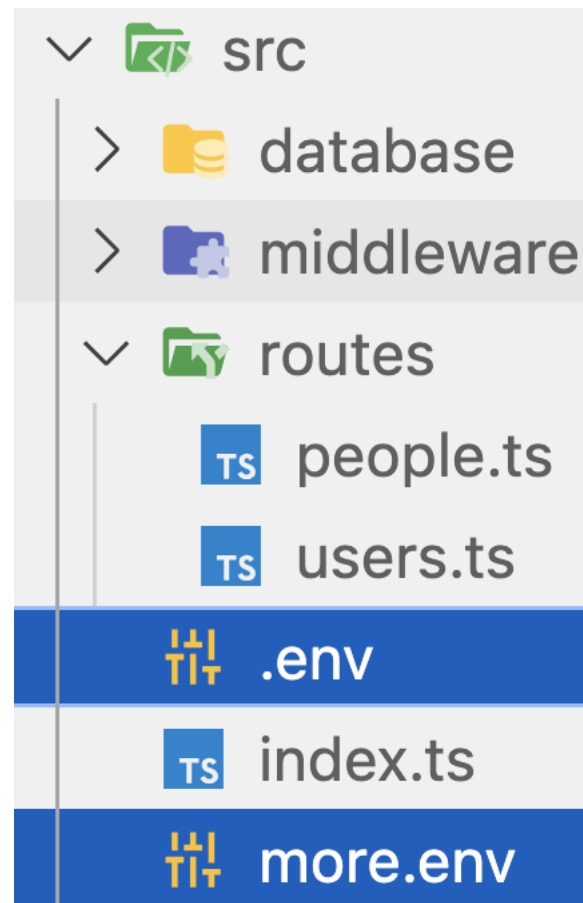
הספריה קוראת את הקובץ .env

הספריה ממלאת את האובייקט process.env

הספריה ממלאת את האובייקט:
`process.env.F00 = "bar"`

עבודה עם משתני סביבה:

אם רוצים להשתמש בתיקיה src:
(בנוסף - אפשר לעבוד עם יותר מקובץ .env אחד)



```
import express, { json } from "express";
import { usersRouter } from "../routes/users";
import { logger } from "../middleware/logger";
import { notFound } from "../middleware/not-found";
import { peopleRouter } from "../routes/people";
import { config } from "dotenv";
```

```
//the config function loads our vars from .env file
//to process.env (in memory)
```

```
config({path: "src/.env"});
config({path: "src/more.env"});
const app = express();
```

```
console.log(process.env.F00);
console.log(process.env.ABC);
```

```
// middleware chain:
app.use(json());
app.use(logger);
app.use("/api/v1/users", usersRouter);
app.use("/api/v1/people", peopleRouter);
app.use(notFound);

app.listen(8080);
```

עבודה עם משתני סביבה:

נרצה קובץ .env (קובץ הגדרות)
לכל סביבה:

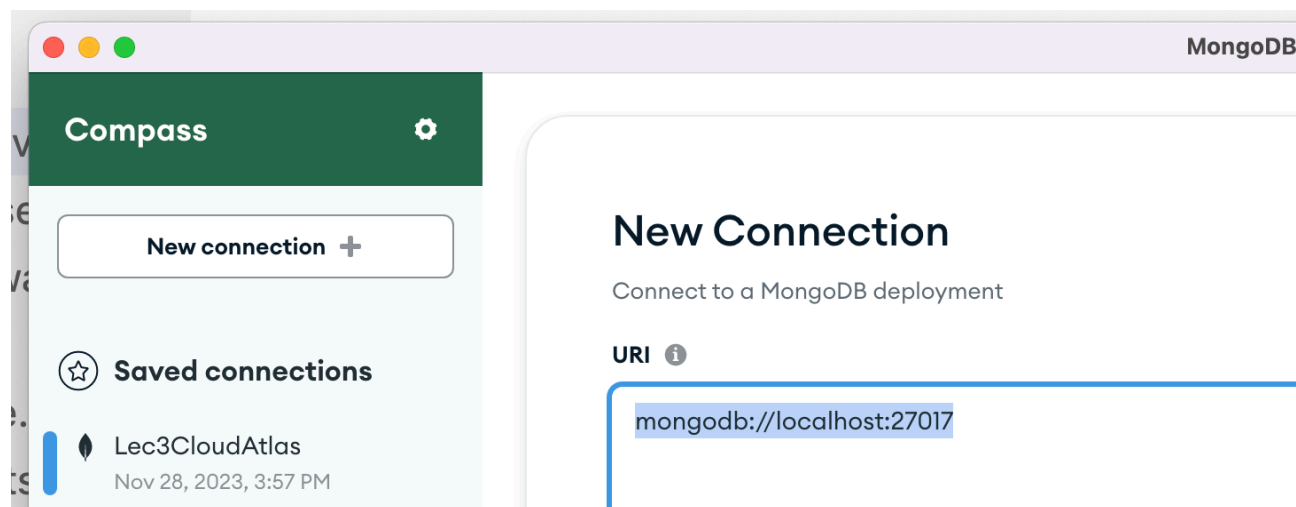
סביבת פיתוח
סביבת בדיקות
production סביבת

כשמפתחים את הפרוייקט - נרצה דטה-בייס למפתחים:
נקרא לדטה-בייס biz_project_dev

כשבודקי תוכנה - בודקים את הפרוייקט - נרצה דטה-בייס לבודקים:
נקרא לדטה-בייס biz_project_tests

כשמעלים את הפרוייקט לענן - נרצה דטה-בייס בענן
(סביבת production)
נקרא לדטה-בייס biz_cards והוא יישב בcloud atlas

עבודה על הקובץ dev.env



שם הדטה-בייס
(לא חייב להיות קיים)



src/config/dev.env

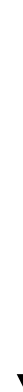
```
DB_CONNECTION_STRING=mongodb://localhost:27017/biz_cards_dev
PORT=8080
CLIENT_URL=http://localhost:3000
```

עבודה על הקובץ test.env

שם הדטה-בייס
(לא חייב להיות קיים)

src/config/test.env

```
DB_CONNECTION_STRING=mongodb://localhost:27017/biz_cards_test
PORT=8081
CLIENT_URL=http://localhost:3000
```



כל הקבצים ביחד:

src/config/dev.env

```
DB_CONNECTION_STRING=mongodb://localhost:27017/biz_cards_dev
PORT=8080
CLIENT_URL=http://localhost:3000
```

יושב מקומית על המחשב שלי

src/config/test.env

```
DB_CONNECTION_STRING=mongodb://localhost:27017/biz_cards_test
PORT=8081
CLIENT_URL=http://localhost:3000
```

src/config/prod.env

```
DB_CONNECTION_STRING=mongo+srv:... YOUR ATLAS Connection string
CLIENT_URL=http://github.com/.... YOURS ....
```

```
# - prod לא צריך לרשום פורט
```

יושב על הענן
המיקום שאליו העליתי צד לקוח
המיקום שאליו העליתי דטה-בייס

קובץ כללי:

src/config/.env

(1) קבועים שמשותפים לכל הסביבות.
(2) נבחר באיזה מצב רוצים להריץ את הפרוייקט

 .env



src > config >  .env

1 PROJECT_NAME=Biz Cards

2 NODE_ENV=dev

שני שלבים בטעינת הגדרות:

1
נטען את הקובץ הראשי:
ממנו נדע מה הקובץ הבא:

 .env



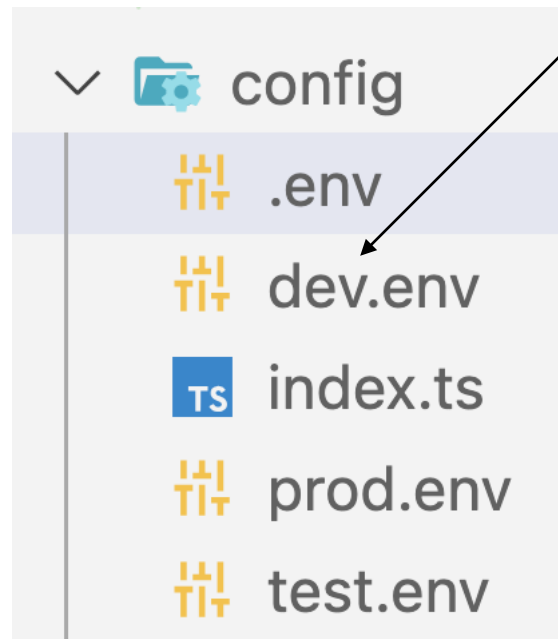
src > config >  .env

1 **NODE_ENV=dev** 

dev|prod|test

2
לפי מה שכתוב בערך של NODE_ENV
נטען את הקובץ המתאים

ויש התאמה לשם הקובץ



הקובץ config/index.ts

config/index.ts

```
import { config } from "dotenv";

// new function
const configDotEnv = () => {
  // load the main/general .env file
  config({path: "src/config/.env"});

  const mode = process.env.NODE_ENV; //dev|test|prod

  config({ path: `src/config/${mode}.env` });
};

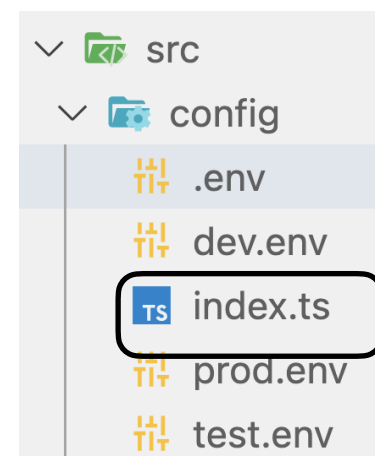
// export the function
export default configDotEnv;
// export the function
export { configDotEnv };

// import {configDotEnv} from 'src/config'
```

1
נטען את הקובץ הראשי:
ממנו נדע מה הקובץ הבא:

2
לפי מה שכתוב בערך של NODE_ENV
נטען את הקובץ המתאים

ויש התאמה לשם הקובץ



הקובץ src/index.ts

```
import configDotEnv from "./config";
configDotEnv();

console.log(process.env.NODE_ENV);
console.log(process.env.DB_CONNECTION_STRING);
```

```
import express, { json } from "express";
import { logger } from "./middleware/logger";
import { notFound } from "./middleware/not-found";
import { peopleRouter } from "./routes/people";
import { usersRouter } from "./routes/users";
```

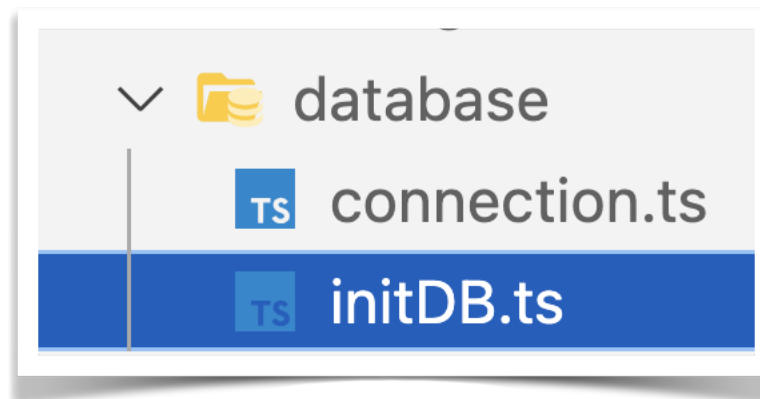
```
const app = express();
```

```
// middleware chain:
```

```
app.use(json());
app.use(logger);
app.use("/api/v1/users", usersRouter);
app.use("/api/v1/people", peopleRouter);
app.use(notFound);
```

```
app.listen(8080);
```

פונקציה למילוי מידע ראשוני בדטא-בייס:



```
TS initDB.ts ×
src > database > TS initDB.ts > ...
1  const initDB = async () => {
2    //TODO: add 3 users
3    //TODO: add 3 cards
4  };
5
6  export { initDB };
```


התחברות לדטה-בייס:

```
// pnpm add mongoose
import { initDB } from './initDB';
import mongoose from 'mongoose';

const connect = async () => {
  //read the connection string from dotenv file:
  const connectionString =
    process.env.DB_CONNECTION_STRING;

  //connect to the database:
  await mongoose.connect(connectionString);

  //init the database:
  await initDB();
};

export { connect };
```

pnpm add mongoose

התחברות

1

נקרא לפונקציה
שממלאת את הדטה-בייס

2

התמודדות עם שגיאות והדפסה לקונסול:

```
import { initDB } from "../initDB";
import mongoose from "mongoose";

const connect = async () => {
  try {
    //read the connection string from dotenv file:
    const connectionString = process.env.DB_CONNECTION_STRING;

    if (!connectionString) {
      console.error("DB_CONNECTION_STRING IS NOT DEFINED IN your .env file");
      return;
    }

    //connect to the database:
    await mongoose.connect(connectionString);

    console.log("Database Connected");
    //init the database:
    await initDB();
  } catch (err) {
    console.error("Cant Connect to database", err);
  }
};

export { connect };
```

הפעלה ובדיקה:

```
import configDotEnv from "./config";
import express, { json } from "express";
import { logger } from "./middleware/logger";
import { notFound } from "./middleware/not-found";
import { peopleRouter } from "./routes/people";
import { usersRouter } from "./routes/users";
import { connect } from "./database/connection";
```

```
configDotEnv();
connect();
```

```
const app = express();
```

```
// middleware chain:
```

```
app.use(json());
app.use(logger);
app.use("/api/v1/users", usersRouter);
app.use("/api/v1/people", peopleRouter);
app.use(notFound);
```

```
app.listen(8080);
```

יש חשיבות לסדר:

נרצה לקרוא config לפני הפונקציה connect
נרצה להתחבר לדטה-בייס לפני שנמשיך לRoutes

כך יראה משתמש שנשמר בדטא-בייס

האובייקט שצריך להישמר במאגר המידע

```
id: ObjectId('6361a80583262d33d9d4294f')
name: Object
  first: "first"
  middle: ""
  last: "user"
  _id: ObjectId('6361a80583262d33d9d42950')
phone: "050-0000000"
email: "first@gmail.com"
password: "$2a$10$.Do1N9TnpJ4Qj6m7U48TD.Y7qULHi7OOD6w00BbapQPvagTvNQOIe"
image: Object
  url: "https://cdn.pixabay.com/photo/2016/04/01/10/11/avatar-1299805_960_720..."
  alt: "business card image"
  id: ObjectId('6361a80583262d33d9d42951')
address: Object
  state: "not defined"
  country: "israel"
  city: "tel-aviv"
  street: "magnive"
  houseNumber: 5
  zip: 0
  _id: ObjectId('6361a80583262d33d9d42952')
isAdmin: false
isBusiness: true
createdAt: 2022-11-01T23:13:09.180+00:00
__v: 0
```


(1) הגדרה של סכמה

סכמה - מבנה של האוסף:
שדות חובה, ולידציות.

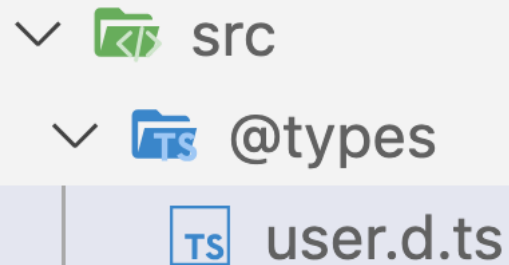
כך לא נשמור טעויות בדטה-בייס.

```
const userSchema = {  
  phone: {type: String, required: true, minLength: 20},  
  email: String,  
  password: String,  
  isAdmin: Boolean,  
  isBusiness: Boolean,  
  createdAt: Date,  
};
```

ולכן אין השלמה אוטומטית
אין כאן הגדרות של typescript



הגדרות עבור Typescript



```
src
├── @types
│   └── user.d.ts
```


הגדרת סכמה עם Typescript - יש השלמה אוטומטית:

```
type User = {
  email: string;
  phone: string;
  password: string;
  isBusiness: boolean;
  isAdmin: boolean;
  createdAt: Date;
};

export { User };
```

```
import { Schema } from "mongoose";

import { User } from "../@types/user";
const userSchema = new Schema<User>({
  phone: {
    required: true,
    type: String,
    minlength: 9,
  },
});
```



הגדרת סכמה עם Typescript - יש השלמה אוטומטית:

בהסתמך על ההגדרה של User

```
import { Schema } from "mongoose";

import { User } from "../../@types/user";
const userSchema = new Schema<User>({
  phone: {
    required: true,
    type: String,
    minlength: 9,
    maxlength: 15,
  },
  email: {
    required: true,
    type: String,
    minlength: 7,
    maxlength: 20,
  },
  password: {
    required: true,
    type: String,
    minlength: 7,
    maxlength: 100,
  },
  isAdmin: {
    type: Boolean,
    required: false,
    default: false,
  },
  isBusiness: {
    type: Boolean,
    required: true,
  },
  createdAt: {
    type: Date,
    required: false,
    default: new Date(),
  },
});
```

סכמה חלקית של משתמש:

הגדרת טיפוסים: (לא חובה - עוזר עם השלמה אוטומטית)

```
type IName = {  
  first: string;  
  middle?: string;  
  last: string;  
};  
  
type IAddress = {  
  street: string;  
  city: string;  
  state?: string;  
  zip?: number;  
  country: string;  
  houseNumber: number;  
};  
  
type IImage = {  
  alt: string;  
  url: string;  
};  
  
type IUser = {  
  name: IName;  
  address: IAddress;  
  image?: IImage;  
  email: string;  
  phone: string;  
  password: string;  
  isBusiness: boolean;  
  isAdmin?: boolean;  
  createdAt?: Date;  
};  
  
export { IUser, IName, IAddress, IImage };
```



```
import { Schema } from "mongoose";

import { IName } from "../../@types/user";

const nameSchema = new Schema<IName>({
  first: {
    required: true,
    type: String,
    minlength: 2,
    maxlength: 20,
  },
  middle: {
    required: false,
    default: "",
    type: String,
    minlength: 0,
    maxlength: 20,
  },
  last: {
    required: true,
    type: String,
    minlength: 2,
    maxlength: 20,
  },
});

export { nameSchema };
```

סכמה עבור שם:

צרו סכמה עבור Image
ועבור Address

user-schema.ts

```
import { Schema } from "mongoose";

import { IUser } from "../../@types/user";
import { nameSchema } from "../name-schema";

const userSchema = new Schema<IUser>({
  name: nameSchema,
  phone: {
    required: true,
    type: String,
    minlength: 9,
    maxlength: 15,
  },
  email: {
    required: true,
    type: String,
    minlength: 7,
    maxlength: 20,
  },
  password: {
    required: true,
    type: String,
    minlength: 7,
    maxlength: 100,
  },
  isAdmin: {
    type: Boolean,
    required: false,
    default: false,
  },
  isBusiness: {
    type: Boolean,
    required: true,
  },
  createdAt: {
    type: Date,
    required: false,
    default: new Date(),
  },
});

export { userSchema };
```

הגדרה של userSchema

מיני-תרגיל - צרו סכמה עבור Image ועבור Address

name-schema.ts

```
import { Schema } from "mongoose";

import { IName } from "../../@types/user";

const nameSchema = new Schema<IName>({
  first: {
    required: true,
    type: String,
    minlength: 2,
    maxlength: 20,
  },
  middle: {
    required: false,
    default: "",
    type: String,
    minlength: 2,
    maxlength: 20,
  },
  last: {
    required: true,
    type: String,
    minlength: 2,
    maxlength: 20,
  },
});

export { nameSchema };
```

@types/user.d.ts

```
type IName = {
  first: string;
  middle?: string;
  last: string;
};

type IAddress = {
  street: string;
  city: string;
  state?: string;
  zip?: number;
  country: string;
  houseNumber: number;
};

type IImage = {
  alt: string;
  url: string;
};

type IUser = {
  name: IName;
  address: IAddress;
  image?: IImage;
  email: string;
  phone: string;
  password: string;
  isBusiness: boolean;
  isAdmin?: boolean;
  createdAt?: Date;
};

export { IUser, IName, IAddress, IImage };
```

address-schema.ts

```
import { Schema } from "mongoose";
import { IAddress } from "../../@types/user";

const addressSchema = new Schema<IAddress>({
  city: {
    type: String,
    required: true,
    minlength: 2,
    maxlength: 50,
  },
  state: {
    type: String,
    required: false,
    default: "",
    minlength: 0,
    maxlength: 50,
  },
  country: {
    type: String,
    required: true,
    minlength: 2,
    maxlength: 50,
  },
  street: {
    type: String,
    required: true,
    minlength: 2,
    maxlength: 100,
  },
  zip: {
    type: String,
    required: false,
    default: "0",
    maxlength: 30,
  },
  houseNumber: {
    type: Number,
    required: true,
    min: 0,
    max: 999999,
  },
});

export { addressSchema };
```

מיני-תרגיל - צרו סכמה עבור IImage ועבור IAddress

image-schema.ts

```
import { Schema } from "mongoose";
import { IImage } from "../../@types/user";

const imageSchema = new Schema<IImage>({
  alt: {
    type: String,
    minlength: 2,
    maxlength: 200,
    required: true,
  },
  url: {
    type: String,
    minlength: 12,
    maxlength: 200,
    required: true,
  },
});

export { imageSchema };
```

```
import { Schema } from "mongoose";
```

```
import { IUser } from "../../@types/user";  
import { nameSchema } from "../name-schema";  
import { imageSchema } from "../image-schema";  
import { addressSchema } from "../address-schema";
```

```
const userSchema = new Schema<IUser>({  
  name: nameSchema,  
  address: addressSchema,  
  image: {  
    type: imageSchema,  
    required: false,  
    default: {  
      alt: "user-profile",  
      url: "https://picsum.photos/200/300",  
    },  
  },  
  phone: {  
    required: true,  
    type: String,  
    minlength: 9,  
    maxlength: 15,  
  },  
  email: {  
    unique: true,  
    required: true,  
    type: String,  
    minlength: 7,  
    maxlength: 20,  
  },  
  password: {  
    required: true,  
    type: String,  
    minlength: 7,  
    maxlength: 100,  
  },  
  isAdmin: {  
    type: Boolean,  
    required: false,  
    default: false,  
  },  
  isBusiness: {  
    type: Boolean,  
    required: true,  
  },  
  createdAt: {  
    type: Date,  
    required: false,  
    default: new Date(),  
  },  
});
```

```
export { userSchema };
```

מיני-תרגיל - צרו סכמה עבור Image וAddress

user-schema.ts

- 0 להתחבר
- 1 להגדיר סכמה
- 2 להגדיר מודל**
- 3 ליצור מופע
- 4 לשמור

הגדרת מודל:

database/model/user.ts

```
import mongoose from "mongoose";
import { userSchema } from "../schema/user-schema";

const User = mongoose.model("user", userSchema);

export { User };
```

מודל - מחלקה (לכן באות גדולה)

- (0) להתחבר
- (1) להגדיר סכמה
- (2) להגדיר מודל
- (3) ליצור מופע
- (4) לשמור

הספריה Mongoose יוצרת עבורנו מחלקה
במחלקה יש את כל התכונות של User
כגון שם פרטי ומשפחה וכו'.

הספריה מוסיפה מתודות כגון save/find/findOne
לאובייקטים

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1:27017/test');

const Cat = mongoose.model('Cat', { name: String });

const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

class User{..., ctor, methods}

במחלקה שהספריה יוצרת יש מתודות עזר לשמירה ועבודה עם דטה-בייס.

user Router

```
import { Router } from "express";
import { User } from "../database/model/user";

const router = Router();

router.post("/", async (req, res) => {
  //TODO: try/catch - catch errors
  //TODO: use joi to check the body
  const userBody = req.body;

  const user = new User(userBody);

  const saved = await user.save();

  res.status(201).json({ message: "Saved", user: saved });
});

export { router as usersRouter };
```

בדיקת HTTP

GET All users:

GET http://localhost:8080/api/v1/users

Add a user:

POST http://localhost:8080/api/v1/users

Content-Type: application/json

```
{
  "name": {
    "first": "Bruce",
    "middle": "foobar",
    "last": "Wayne"
  },
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "country": "Israel",
    "state": "anyState",
    "houseNumber": 20,
    "zip": "12345"
  },
  "image": {
    "alt": "user-profile",
    "url": "https://picsum.photos/200/300"
  },
  "phone": "050-8123091",
  "email": "bruce@batcave.com",
  "isBusiness": true,
  "password": "123456aA!"
}
```

GET All users:

GET http://localhost:8080/api/v1/users

Add a user:

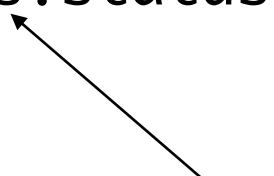
POST http://localhost:8080/api/v1/users

Content-Type: application/json

```
{
  "name": {
    "first": "Bruce",
    "last": "Wayne"
  },
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "country": "Israel",
    "houseNumber": 20,
    "zip": "12345"
  },
  "image": {
    "alt": "user-profile",
    "url": "https://picsum.photos/200/300"
  },
  "phone": "050-8123091",
  "email": "Wayne@batcave.com",
  "isBusiness": true,
  "password": "123456aA!"
}
```

מבוא לטיפול בשגיאות: try/catch

```
router.post("/", async (req, res) => {  
  //TODO: use joi to check the body  
  try {  
    const userBody = req.body;  
  
    const user = new User(userBody);  
  
    //mongo -> save  
    const saved = await user.save();  
  
    res.status(201).json({ message: "Saved", user: saved });  
  } catch (e) {  
    res.status(400).json({ message: "An Error occurred", e });  
  }  
});
```



אם נתפוס את השגיאה - הפרוייקט לא ייעצר בכל פעם שיש שגיאה.
ונחזיר ללקוח הודעה מתאימה.

אם לא נתפוס את השגיאה - השרת יעצר בכל בקשה לא תקינה.

הצגת כל המשתמשים:

```
router.get("/", async (req, res) => {  
  try {  
    const allUsers = await User.find();  
  
    res.json(allUsers);  
  } catch (e) {  
    res.status(500).json({ message: "server error", e });  
  }  
});
```

עם async/await

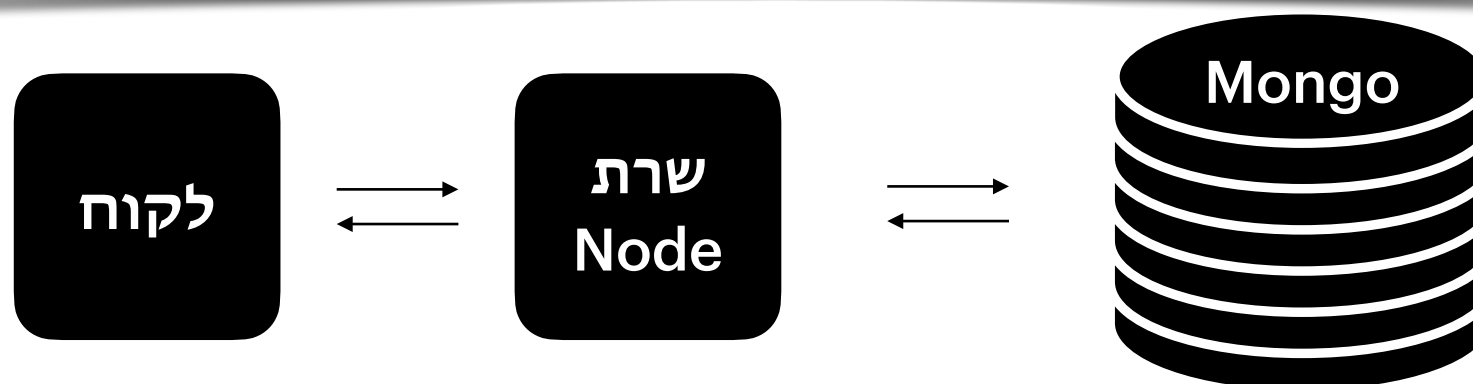
```
router.get("/", (req, res) => {  
  User.find()  
    .then(allUsers => {  
      res.json(allUsers);  
    })  
    .catch(e) => {  
      res.status(500).json({ message: "server error", e });  
    });  
});
```

עם promise chaining

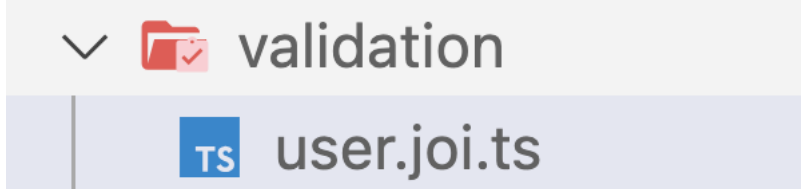
אותו דבר
אבל צריך
לבחור
דרך אחת
שנוחה לכם

בעיה בקוד שלנו:
שולחים שאילתה לדטה-בייס בלי לבדוק את הbody
הדטה-בייס מכשיל שאילתות לא תקינות
(תקשורת עם שרת חיצוני - כגון דטה-בייס עולה כסף וזמן!)

```
router.post("/", async (req, res) => {  
  //TODO: use joi to check the body  
  try {  
    const userBody = req.body;  
  
    const user = new User(userBody);  
  
    //mongo -> save  
    const saved = await user.save();  
  
    res.status(201).json({ message: "Saved", user: saved });  
  } catch (e) {  
    res.status(400).json({ message: "An Error occurred", e });  
  }  
});
```



ולידציה עם Joi



```
import Joi from "joi";
import { IName, IUser } from "../@types/user";

const registerSchema = Joi.object<IUser>({
  name: Joi.object<IName>({
    first: Joi.string().min(2).required()
  })
});
```

עזרה של typescript
להתאים את הסכמה לטיפוס הקיים שלנו
(אם נוסיף <IUser> נקבל השלמה אוטומטית)

מוזמנים להשלים את הסכמה של Joi לפי מה שלמדתם בראקט

שיעורי בית - Mongoose:

צרו פרוייקט Typescript חדש
עם קובץ ראשי בשם index.ts
ומודול ראوتر routes/students.ts

צרו סכמה עבור students
לכל סטודנט: שם פרטי, שם משפחה, מספר סטודנט.
צרו מודל מתאים

השרת יאפשר את הפעולות הבאות:

(1) להציג את כל הסטודנטים - GET api/v1/students

(2) לחפש סטודנטים לפי שם פרטי - GET api/v1/students/search
השתמשו בquery string

(3) להוסיף סטודנט POST /api/v1/students
במידה והכל תקין - הסטטוס יהיה 201

בדקו את תגובת השרת בעזרת קובץ requests.rest
* למה קוראים כך לסיומת של הקובץ *

בדקו את תגובת השרת בעזרת postman

שימו לב שאנו שומרים על עקרון
כתובות אחידות ב REST

rest

lec4.rest

שיעורי בית Joi

validation

TS user.joi.ts

```
import Joi from "joi";
import { IName, IUser } from "../@types/user";

const registerSchema = Joi.object<IUser>({
  name: Joi.object<IName>({
    first: Joi.string().min(2).required()
  })
});
```

נסו להשלים את הסכמה של user.joi.ts
באמצעות הקוד שיש לכם בפרוייקט ראקט
עם מתודה לבדיקה של הbody

בידקו אם הbody תקין לפני נסיון השמירה לדטה-בייס

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

  try {
    const userBody = req.body;

    //TODO: use Joi to check the body
    //if(המידע לא תקין){return res.json({message: "bad request"})}

    const user = new User(userBody);

    //mongo -> save
    const saved = await user.save();

    res.status(201).json({ message: "Saved", user: saved });
  } catch (e) {
    res.status(400).json({ message: "An Error occurred", e });
  }
});
```