

NodeJS

Express

בדיקת תקינות גוף המסמך עם Joi

הרשמה - הצפנת הסיסמא עם הספרייה Bcrypt

עבודה עם Service - Code Splitting

התחברות - הנפקת JWT

עוד על Middleware

במידה ושכחנו שרת שרץ על port מסויים

וסגרנו את VSCode מבלי לסגור את השרת

```
npx kill-port --port 8080
```

קובץ עבור תבניות Regex לפרוייקט:



```
const passwordRegex =  
  /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*~])[A-Za-z\d!@#$%^&*~]{6,}$/;  
const phoneRegex = /^(+972|0)([23489]|5[02468]|77)-?[1-9]\d{6})$/;  
  
export { passwordRegex, phoneRegex };
```

קובץ עבור תבניות Regex לפרויקט:

```
import Joi from "joi";
import { IName, IUser, IAddress, IImage } from "../@types/user";
import { passwordRegex, phoneRegex } from "../patterns";

const schema = Joi.object<IUser>({
  address: Joi.object<IAddress>({
    city: Joi.string().min(2).max(50).required(),
    country: Joi.string().min(1).max(50).required(),
    houseNumber: Joi.number().min(0).max(50000).required(),
    street: Joi.string().min(1).max(50).required(),
    zip: Joi.string().min(1).max(20).required(),
    state: Joi.string().max(50).allow(""),
  }).required(),
  email: Joi.string().email().min(5).max(255).required(),
  name: Joi.object<IName>({
    first: Joi.string().min(1).max(50).required(),
    last: Joi.string().min(1).max(100).required(),
    middle: Joi.string().max(100).allow(""),
  }).required(),

  password: Joi.string().pattern(passwordRegex).min(5).max(30).required(),
  phone: Joi.string().pattern(phoneRegex).min(1).max(50).required(),
  image: Joi.object<IImage>({
    alt: Joi.string().min(0).max(100).allow(""),
    url: Joi.string().uri().min(5).max(255).required(),
  }),
  isBusiness: Joi.boolean().required(),
});
export { schema as joiUserSchema };
```

עבודה עם TS - טיפוס לאובייקטים:

```
let obj:{} = {};  
obj.abc = 'efg';
```

אי אפשר להוסיף מפתחות אם הגדרנו כבר טיפוס לאובייקט

אם נרצה להגדיר טיפוס לאובייקט שיש לו מפתחות מסוג string וערכים מסוג כלשהו:

הגדרת טיפוס לKey

הגדרת טיפוס לValue

```
let obj: Record<string, any> = {};  
obj.abc = "efg";  
obj.abc = 11;
```

עבודה עם TS - טיפוס לאובייקטים:

```
import Joi from "joi";  
  
const validation = (schema: Joi.ObjectSchema, userInput: any) => {  
  const { error } = schema.validate(userInput);  
  if (!error) {  
    return null;  
  }  
  
  const { details } = error;  
  let errorObj: Record<string, any> = {};  
  
  for (let detail of details) {  
    let key = detail.path[0];  
    let { message } = detail;  
    errorObj[key] = message;  
  }  
  return errorObj;  
};  
  
export default validation;
```

טיפוס לפרמטרים

טיפוס לפרמטרים

אובייקט גמיש - נתחיל ריק ונוסיף לו מפתחות בלולה

עבודה עם TS - טיפוס לאובייקטים:

בגלל שJOI עוצר אחרי שגיאה אחת שהוא מוצא - אין צורך בלולאה על כל השגיאות

```
import Joi from "joi";

const validation = (schema: Joi.ObjectSchema, userInput: any) => {
  const { error } = schema.validate(userInput);
  if (!error) {
    return null;
  }

  const { message } = error.details[0];
  return message;
};

export default validation;
```

עבודה עם TS - טיפוס לאובייקטים:

שילוב של Joi ישירות בראוטר!

ראוטר:
ולידציה
דטהבייס
להחזיר תשובה

```
router.post("/", async (req, res) => {  
  try {  
    const userBody = req.body;  
  
    const err = validation(joiUserSchema, req.body);  
    if (err) {  
      return res.status(400).json({ err });  
    }  
  
    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 });  
  }  
});
```


דרך נוספת לשלב Middleware

Express

פונקצית middleware

```
const myFn: RequestHandler = (req, res, next) => {  
  console.log("myFn");  
  next();  
};
```

שימוש בפונקצית middleware

```
router.get("/", myFn, (req, res, next) => {  
  res.json({ message: "Done" });  
});
```

Middleware Joi לבדיקה של סכמה

```
import { RequestHandler } from "express";
import validation from "../validation/validate-schema";
import { joiUserSchema } from "../validation/user.joi";

const validateUserSchema: RequestHandler = (req, res, next) => {
  const err = validation(joiUserSchema, req.body);

  if (err) {
    res.status(400).json(err);
  } else {
    next();
  }
};
```

בעיה: נצטרך לכתוב פונקציה זהה גם לכרטיסיות

מפר את עיקרון DRY

Middleware Joi לבדיקה של סכמה

```
import { RequestHandler } from "express";  
import { ObjectSchema } from "joi";  
import validation from "../validation/validate-schema";
```

הגדרת טיפוס לפונקציה שמקבלת סכמה של Joi ומחזירה RequestHandler

```
type ValidateSchema = (schema: ObjectSchema) => RequestHandler;
```

פונקציה שמקבלת סכמה של Joi ומחזירה RequestHandler

```
const validateSchema: ValidateSchema = (schema) => (req, res, next) => {  
  const error = validation(schema, req.body); ← האם יש שגיאה?  
  
  if (!error) return next(); ← אם אין שגיאה - נמשיך את השרשרת  
  
  res.status(400).json({ error }); ← אם יש שגיאה - נחזיר תגובה  
};  
  
export { validateSchema };
```

עבודה עם TS - טיפוס לאובייקטים:

שילוב של Joi ישירות בראוטר!

ראוטר:
ולידציה
דטהבייס
להחזיר תשובה

```
router.post("/", async (req, res) => {  
  try {  
    const userBody = req.body;  
  
    const err = validation(joiUserSchema, req.body);  
    if (err) {  
      return res.status(400).json({ err });  
    }  
  
    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 });  
  }  
});
```

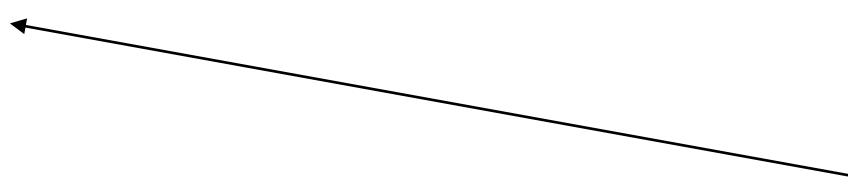
עבודה עם TS - טיפוס לאובייקטים:

הפונקציה validateSchema

ראוטר:
ולידציה
דטהבייס
להחזיר תשובה

2

```
router.post("/", validateSchema(joiUserSchema), async (req, res) => {  
  try {  
    const userBody = req.body;  
  
    const user = new User(userBody);  
    const saved = await user.save();  
  
    res.status(201).json({ message: "Saved", user: saved });  
  } catch (e) {  
    res.status(400).json({ message: "An Error occurred", e });  
  }  
});
```



Middleware Express מאפשרת לנו לכלול פונקציות Middleware לכל Route בצורה פשוטה

קובץ index.ts לתיקיה validation

התפקיד של קובץ index:

להנגיש את מה שיש באותה תיקיה:

validation/index.ts

בדומה לאינדקס במילון.


```
import { joiUserSchema } from "../joi/user.joi";  
import { validateSchema } from "../validate-schema";
```

```
// syntactic sugar
```

```
const validateRegistration = validateSchema(joiUserSchema);
```

```
export { validateRegistration };
```

שימוש במה שהגדרנו בindex.ts



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

אסור לשמור את הסיסמא האמיתית של המשתמש בדטה-בייס

אם הדטה-בייס ייפרץ - תהיה לפורצים את הסיסמא של המשתמש באתרים אחרים
כי המשתמש מזין את אותה הסיסמא בהרבה אתרים

password: "123456aA!/"

פונקצית גיבוב חד-כיוונית



"\$2b\$12\$CnPcYhzNngxCyvHvky9Sx.8wKwjCQw6bVp2B0a5BIG0g4a0kgRrSK"

פונקצית גיבוב חד-כיוונית

אם ידועה הסיסמא - אפשר ליצור ממנה את הHash

אבל בהנתן הHash אי אפשר למצוא את הסיסמא

עבודה עם הספרייה bcrypt

```
pnpm add bcrypt
```

← התקנה של הספרייה

```
pnpm add -D @types/bcrypt
```

← התקנה של טיפוסים עבור הספרייה
מאפשר השלמה אוטומטית



Nabbing Pleasant Monads

npm

🔍 Search packages

bcrypt 

← כאן כתוב שצריך להתקין גם @types/bcrypt

5.1.1 • Public • Published 4 months ago



Readme



Code

Beta

הספריה משתמשת ב Salt/Secret כדי להצפין את הסיסמא

 .env



src > config >  .env

1 NODE_ENV=dev

2 BCRYPT_SECRET=BUzX2_x@QjiJhxUAgiVw_KTQmBy/rJ9

הספריה משתמשת ב Salt/Secret כדי להצפין את הסיסמא

```
import bcrypt from "bcrypt";
```

הספריה יכולה ליצור עבורנו SALT

```
const demo = async () => {  
  const secret = await bcrypt.genSalt(12); // $2b$12$Kt8xsmnIGJvq5myvGylKFe
```

```
  const password = "123456";
```

אם נשתמש באותו SALT ונצפין את אותה הסיסמא - נקבל את אותה תוצאה

```
  // הצפנה של הסיסמא עם המפתח
```

```
  const hash = await bcrypt.hash(password, secret);  
  // $2b$12$Kt8xsmnIGJvq5myvGylKFevnpK9iL54sB5vD5J/RyLxAyxkzxanF0
```

```
  // הצפנה נוספת של הסיסמא עם אותו המפתח
```

```
  const hash2 = await bcrypt.hash(password, secret);  
  // $2b$12$Kt8xsmnIGJvq5myvGylKFevnpK9iL54sB5vD5J/RyLxAyxkzxanF0
```

```
  console.log(hash);  
  console.log(hash2);
```

```
};
```

```
demo();
```

הספריה משתמשת ב Salt/Secret כדי להצפין את הסיסמא

```
import bcrypt from "bcrypt";
```

```
const demo = async () => {  
  const password = "123456";
```

```
  const hash1 = await bcrypt.hash(password, 12);  
  const hash2 = await bcrypt.hash(password, 12);
```

```
  // $2b$12$FpHy.BJ1U4Jk.aXPd7TIG004L7tdnd7HQtdFDmydP7CNqecN38Y0m  
  console.log(hash1);
```

```
  // $2b$12$DLKGSIN/hxmINRoQRCzwB.LSGLumPVLFFus3He/stmZZfJCF7nQlq  
  console.log(hash2);
```

```
};
```

```
demo();
```

הספריה יוצרת SALT חדש בעת הצפנת הסיסמא
וכך לא משתמשים באותו SALT פעמיים.

כתוצאה מכך - הסיסמאות המוצפנות שונות אחת מהשניה
אף על פי שמדובר באותה סיסמא "123456"

הספריה משתמשת ב Salt/Secret כדי להצפין את הסיסמא

```
import bcrypt from "bcrypt";
```

```
const demo = async () => {  
  const password = "123456";
```

```
  const hash1 = await bcrypt.hash(password, 12);  
  const hash2 = await bcrypt.hash(password, 12);
```

```
  // $2b$12$FpHy.BJ1U4Jk.aXPd7TIG004L7tdnd7HQtdFDmydP7CNqecN38Y0m  
  console.log(hash1);
```

```
  // $2b$12$DLKGSIN/hxmINRoQRCzwB.LSGLumPVLFFus3He/stmZZfJCF7nQlq  
  console.log(hash2);
```

```
};
```

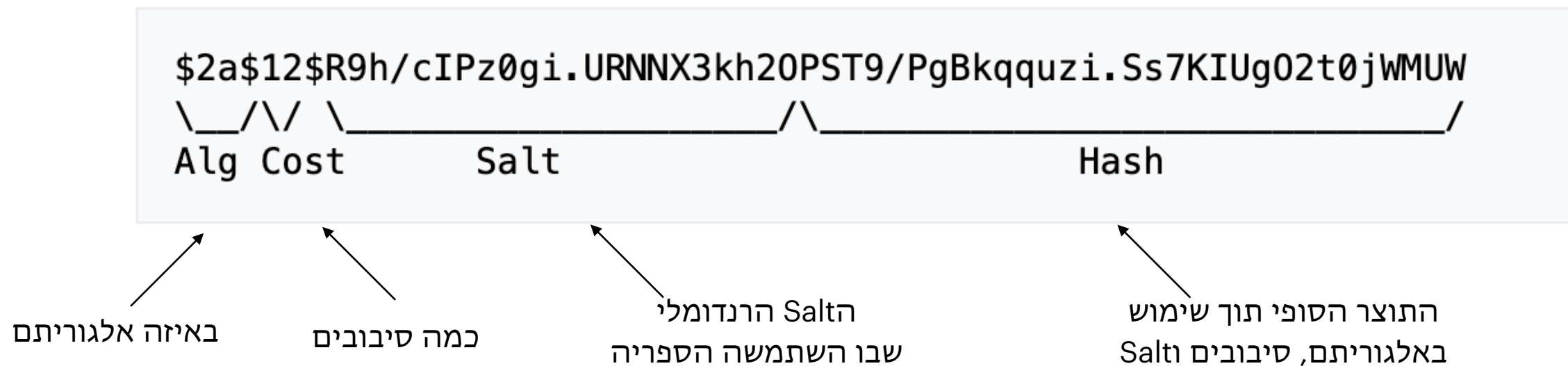
```
demo();
```

הספריה יוצרת SALT חדש בעת הצפנת הסיסמא
וכך לא משתמשים באותו SALT פעמיים.

כתוצאה מכך - הסיסמאות המוצפנות שונות אחת מהשניה
אף על פי שמדובר באותה סיסמא "123456"

איך Bcrypt בודקת אם הסיסמא נכונה?

הספרייה שומרת את ה Salt ביחד עם התוצאה



123456
↓
12 סיבובים
ואותו SALT

```
$2<a/b/x/y>${cost}${22 character salt}[31 character hash]
```

```
const hash = await bcrypt.hash(password, 12);
```

אנחנו שומרים בדטה-בייס את ה SALT ביחד עם התוצאה של ההצפנה.

Bcrypt שומרת גם את הSALT וגם את התוצר אבל לא את הסיסמא.

בהרשמה: מזינים סיסמא:
המערכת מגרילה Secret
המערכת מצפינה את הסיסמא עם Secret
שומרת אותו בדטה-בייס

123456

סיסמא מוצפנת	Secret
--------------	--------

בהתחברות:

123456

המערכת שולפת מהדטה-בייס את הSecret

מצפינה שוב את הסיסמא שקיבלה
אם התוצר זהה - סימן שהסיסמא נכונה.

הרשמה של משתמש:

```
router.post("/", validateRegistration, async (req, res) => {  
  try {  
    const userBody = req.body;  
  
    const user = new User(userBody);  
  
    //a 12 salt hash  
    user.password = await bcrypt.hash(user.password, 12);  
  
    const saved = await user.save();  
  
    res.status(201).json({ message: "Saved", user: saved });  
  } catch (e) {  
    res.status(400).json({ message: "An Error occurred", e });  
  }  
});
```


Joi - ולידציה עבור התחברות:

```
type ILogin = {  
  email: string;  
  password: string;  
};
```

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

```
import Joi from "joi";  
import { passwordRegex } from "../patterns";  
import { ILogin } from "../@types/user";  
  
const schema = Joi.object<ILogin>({  
  email: Joi.string().email().required(),  
  password: Joi.string().pattern(passwordRegex).required(),  
});  
  
export { schema as joiLoginSchema };
```

Express Middleware for User Login

```
import { JoiLoginSchema } from "../../joi/login.joi";
import { JoiUserSchema } from "../../joi/user.joi";
import { validateSchema } from "../validate-schema";

const validateRegistration = validateSchema(JoiUserSchema);
const validateLogin = validateSchema(JoiLoginSchema);

export { validateRegistration, validateLogin };
```

Route עבור התחברות

```
router.post("/login", validateLogin, async (req, res) => {
  //check the pass
  const { email, password } = req.body as ILogin;

  const user = await User.findOne({ email });

  if (!user) {
    return res
      .status(400)
      .json({
        message: "Login failed Check your username and password and try again",
      });
  }

  const isValidPassword = await bcrypt.compare(password, user.password);

  if (!isValidPassword) {
    return res.status(400).json({
      message: "Login failed Check your username and password and try again",
    });
  }
  //TODO: JWT
  res.json({ message: "Logged in" });
});
```

הנפקה של JWT

צד לקוח רוצה לשמור את המשתמש שמחובר - בלי לשמור את הסיסמא.

צד לקוח יקבל JWT שמהווה מסמך JSON חתום.

.env

```
NODE_ENV=dev
JWT_SECRET=BUzX2_x@QjiJhxUAgiVw_KTQmBy/rJ9
```

Algorithm HS256

Encoded

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),  
    your-256-bit-secret  
)
```

☐ secret base64 encoded

 **Signature Verified**

SHARE JWT

הספרייה jsonwebtoken

```
pnpm add jsonwebtoken
```

```
pnpm add -D @types/jsonwebtoken
```

הנפקת JWT אחרי התחברות מוצלחת:

```
import JWT from "jsonwebtoken";
```

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

```
  //check the pass
```

```
  const { email, password } = req.body as ILogin;
```

```
  const user = await User.findOne({ email });
```

יש כאן יותר מדי קוד בנושאים שונים - לא SRP

בשיעור הבא נמשיך Code Splitting

```
  if (!user) {
```

```
    return res.status(400).json({
```

```
      message: "Login failed Check your username and password and try again",
```

```
    });
```

```
  }
```

```
  const isValidPassword = await bcrypt.compare(password, user.password);
```

```
  if (!isValidPassword) {
```

```
    return res.status(400).json({
```

```
      message: "Login failed Check your username and password and try again",
```

```
    });
```

```
  }
```

```
  const secret = process.env.JWT_SECRET!;
```

חילוץ ה-Secret מהקובץ .env

```
  const jwt = JWT.sign({ email: user.email }, secret);
```

נשתמש בפעולה sign

payload, secret

```
  res.json({ jwt });
```

נחזיר את ה-JWT בתגובה ללקוח:

```
});
```

סידור מחדש של הקוד:

@types/user.d.ts

```
type IJWTPayload = {  
  email: string;  
};  
  
export { IUser, IName, IAddress, IImage, ILogin, IJWTPayload };
```

Code Splitting:

כל המתודות בנושא התחברות והרשמה במקום אחד

```
import jwt from "jsonwebtoken";
import bcrypt from "bcrypt";
import { IJWTPayload } from "../@types/user";

const authService = {
  hashPassword: (plainTextPassword: string, rounds = 12) => {
    return bcrypt.hash(plainTextPassword, rounds);
  },

  validatePassword: (plainTextPassword: string, hash: string) => {
    return bcrypt.compare(plainTextPassword, hash);
  },

  generateJWT: (payload: IJWTPayload) => {
    const secret = process.env.JWT_SECRET!;
    return jwt.sign(payload, secret);
  },

  verifyJWT: (token: string) => {
    const secret = process.env.JWT_SECRET!;
    const payload = jwt.verify(token, secret);
    return payload as IJWTPayload & { iat: number };
  },
};

// export the entire object:
export { authService as auth };
```


השלמה בנושא Typescript

```
type Person = {firstName: string}
```

```
type PersonWithDog = Person & {dogName:string}
```

```
const jhonny: PersonWithDog = {  
  firstName: "Jhonny",  
  dogName: "Barky"  
}
```

שיעורי בית:

הוסיפו ולידציות עם Joi לשיעורי הבית מהשיעור הקודם:
לכל Student יש שם פרטי, שם משפחה ומספר סטודנט

(לנוחיותכם רצ"ב שיעורי הבית משיעור קודם בעמוד הבא)

צרו סכמה (מונגוס) למשתמש: שם, אימייל וסיסמא
צרו מודל למשתמש: על בסיס הסכמה
צרו סכמה של JOI עבור ולידציות של פרטי משתמש.

הוסיפו Router עבור משתמשים:

צרו מתודת הרשמה שתשמור את המשתמש עם הסיסמא מוצפנת
צרו מתודת התחברות שתקבל מייל וסיסמא ותחזיר JWT במידה והפרטים שקיבלה תקינים.

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