

EX5 - Authentication

1 הצפנת הסיסמא

רישום משתמש

יצירת המודל משתמש

```
const { sequelize, DataTypes } = require("../sequelize");
const User = sequelize.define("user", {

  name: {
    type: DataTypes.STRING,
    allowNull: false
  },
  username: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true
  },
  email: {
    type: DataTypes.STRING,
  },
  password: {
    type: DataTypes.STRING,
    allowNull: false
  },
  roles: {
    allowNull: false,
    type: DataTypes.ENUM('USER', 'ADMIN'),
    defaultValue: 'USER'
  },
  active: {
```

```

    type: DataTypes.BOOLEAN,
    defaultValue: true
  }
});
module.exports = User;

```

ונעדכן את מסד הנתונים בקובץ index.js שבתקיית המודלים

```
db.users = require('./user')
```

וכבר עכשיו ניתן לראות שהמערכת הוסיפה לנו את הטבלה במסד הנתונים

Info	Columns	Indexes	Triggers	Foreign keys	Partitions	Grants
Column	Type	Default Value				
id	int					
name	varchar(255)					
username	varchar(255)					
email	varchar(255)					
password	varchar(255)					
roles	enum('USER','ADMIN')	USER				
active	tinyint(1)	1				
createdAt	datetime					
updatedAt	datetime					

לא שומרים סיסמאות במסד הנתונים!

לא שומרים את הסיסמאות כטקסט, הסיסמא לפני שהיא נכנסת למסד הנתונים עוברת הצפנה, ההצפנה היא הצפנה "חד סטרית" שזה אומר שאין אפשרות לדעת מתוך ההצפנה מהי הסיסמא המקורית, אלא האפשרות היא רק מול הסיסמא לדעת אם אכן הסיסמא הזו היא הסיסמא המוצפנת

נוסיף קונטרולר בשם authController.js ונוסיף בו 2 פונקציות (הרשמה וכניסה)

```

const db = require('../models/index')
const User = db.users
const login = async (req, res) => {
}

const register = async (req, res) => {
}

module.exports = {login, register}

```

כבר בשלב הזה נוסיף את ROUTE בשם authRoutes.js עם הפניה לקובץ הקונטרולר הנל

```

const express = require('express')
const router = express.Router()
const authController = require("../controllers/authController")

router.post('/register', authController.register)
router.post('/login', authController.login)
module.exports = router

```

נעדכן את קובץ ה-SERVER הראשי על ה-ROUTES

```

app.use("/api/auth", require("../routes/authRoutes"));

```

הצפנת הסיסמאות


נוסיף את חבילת bcrypt

npm i bcrypt

```

"author": "",
"license": "ISC",
"dependencies": {
  "bcrypt": "^5.1.0",
  "cookie-parser": "^1.4.6",
  "cors": "^2.8.5",
  "dotenv": "^16.0.3",
  "express": "^4.18.2",
  "mysql2": "^2.3.3",
  "sequelize": "^6.28.0"
},
"devDependencies": {
  "nodemon": "^2.0.20",
  "sequelize-cli": "^6.5.2"
}

```



נעבוד על פונקציית ההרשמה
בראש הקובץ נבצע שימוש בחבילה

```
const bcrypt= require('bcrypt')
```

נקבל מהמשתמש את השדות

```
const {name, username, email, password} = req.body
```

נסוף ואלידציה חובה על השדות

```
if (!name || !username || !password) { // Confirm data
  return res.status(400).json({ message: 'All fields are required' })
}
```

נבדוק אם קיים משתמש עם אותו שם המשתמש

```
const duplicate = await User.findOne({where:{username:username}})
if(duplicate){
  return res.status(409).json({message:"Duplicate username"})
}
```

נצפין את הסימא על ידי שימוש ב bcrypt

ועכשיו נקבל את הסימא המוצפנת

```
const hashedPwd = await bcrypt.hash(password, 10)
```

ברגע שהצפנו את הסימא יש אפשרות לשמור את המשתמש

```
const userObject = {name,email,username,password:hashedPwd}

const user = await User.create(userObject)
if (user) { // Created
  return res.status(201).json({message:`New user ${user.userName} created`
  })
} else {
  return res.status(400).json({ message: 'Invalid user data received' })
}
```

והקובץ המלא עם פעולת הרישום

```
const db = require('../models/index')
const bcrypt= require('bcrypt')
const User = db.users

const login = async (req, res) => {
```

```

}

const register = async (req, res) => {
  const {name, username, email, password} = req.body

  if (!name || !username || !password) { // Confirm data
    return res.status(400).json({ message: 'All fields are required'
  })
  }

  const duplicate = await User.findOne({where:{username:username}})

  if(duplicate){
    return res.status(409).json({message:"Duplicate username"})
  }

  //Hash password
  const hashedPwd = await bcrypt.hash(password, 10)

  const userObject = {name,email,username,password:hashedPwd}
  const user = await User.create(userObject)
  if (user) { // Created
    return res.status(201).json({ message: `New user ${user.username}
created` })
  } else {
    return res.status(400).json({ message: 'Invalid user data
received' })
  }
}

module.exports = {login, register}

```

בדיקה בPOSTMAN
 כתובת <http://localhost:3600/api/auth/register>

POST http://localhost:3600/api/auth/register

Params Authorization Headers (9) **Body** Pre-request Script Tests

none form-data x-www-form-urlencoded **raw** binary Grap

```

1  {
2    "name": "Geula Shosha1",
3    "email": "geshtop@gmail.com",
4    "username": "geshtop@gmail.com",
5    "password": "123456"
6  }

```

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2    "message": "New user geshtop@gmail.com created"
3  }

```

ולמרות שהוספתי את אותה הסיסמא ל-2 משתמשים ההצפנה היא לא אחידה

id	name	username	email	password	roles
1	Geula Shosha1	geshtop@gmail1.com	geshtop@gmail.com	\$2b\$10\$NdRsWu5VZVeJh99xV2vBaunN3p/v1FO6qHS1.A3IQy3lLoam7kJSq	SER
2	Geula Shosha1	geshtop@gmail.com	geshtop@gmail.com	\$2b\$10\$N4ZKUo/vhpKdZidNNrpKaeXgR63Ewu1eh4KL25CxQVI8HrcDQCxC	SER
NULL	NULL	NULL	NULL	NULL	NULL

ובהגדלה

Id:	1
Name:	Geula Shosha1
Username:	geshtop@gmail1.com
Email:	geshtop@gmail.com
Password:	\$2b\$10\$NdRsWu5VZVeJh99xV2vBaunN3p/v1FO6qHS1.A3IQy3lLoam7kJSq
Roles:	USER
Active:	1
CreatedAt:	2023-02-13 21:38:38
UpdatedAt:	2023-02-13 21:38:38

לוגין

נבצע את פעולת הלוגין
דבר ראשון נקבל את הנתונים של השם והסיסמא

```
const login = async (req, res) => {
  const { username, password } = req.body
}
```

ואלידציה

```
if (!username || !password) {
  return res.status(400).json({ message: 'All fields are required' })
}
```

נבדוק אם קיים המשתמש לפי username

```
const foundUser = await User.findOne({where:{username:username}})
```

אם המשתמש לא קיים או המשתמש לא פעיל להחזיר הודעת שגיאה

```
if (!foundUser || !foundUser.active) {
  return res.status(401).json({ message: 'Unauthorized' })
}
```

וכאן אנו בודקים האם הסיסמא של המשתמש היא "שווה" לסיסמא המוצפנת, שוב נבצע את זה באמצעות bcrypt

```
const match = await bcrypt.compare(password, foundUser.password)
```

אם הסיסמאות לא שוות גם נחזיר שהמשתמש לא מורשה

```
if (!match) return res.status(401).json({ message: 'Unauthorized' })
```

אם הגענו לשלב הזה זה אומר שהסיסמא אכן נכונה והמשתמש יכול להכנס בשם המשתמש הזה.
נחזיר בינתיים טקסט פשוט של "בוצע בהצלחה"

```
res.send("Logged In")
```

הקוד המלא של הלוגין לבינתיים!!!!

```
const login = async (req, res) => {  
  
  const { username, password } = req.body  
  
  if (!username || !password) {  
    return res.status(400).json({ message: 'All fields are required'  
  })  
  }  
  
  const foundUser = await User.findOne({where:{username:username}})  
  
  if (!foundUser || !foundUser.active) {  
    return res.status(401).json({ message: 'Unauthorized' })  
  }  
  
  const match = await bcrypt.compare(password, foundUser.password)  
  
  if (!match) return res.status(401).json({ message: 'Unauthorized' })  
  
  res.send("Logged In")  
  
}
```

נבדוק באמצעות POSTMAN
אם נשלח שם או סיסמא שגויים

POST http://localhost:3600/api/auth/login Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies Beautify

raw JSON

```
1
2  ... "username" : "gesdhtop@gmail.com",
3  ... "password" : "123d456"
4
```

Body 401 Unauthorized 9 ms 325 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2  "message": "Unauthorized"
3
```

אם לא נשלח את כל השדות נקבל שגיאת 400

POST http://localhost:3600/api/auth/login Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies Beautify

raw JSON

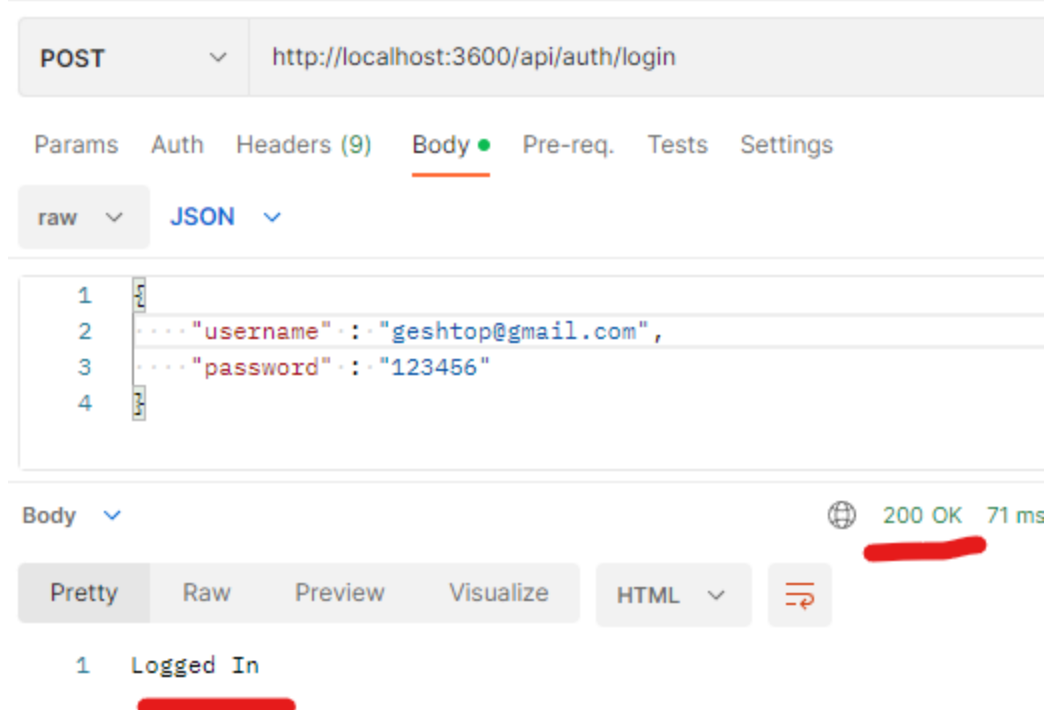
```
1
2  ... "username" : "gesdhtop@gmail.com"
3
```

Body 400 Bad Request 11 ms 335 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2  "message": "All fields are required"
3
```

ואם השם והסיסמא קיימים ונכונים תחזור הודעת משתמש תקין



2 הטמעת JWT


בעיקרון ברגע שמבצעים לוגין לאחר מכן אנו מבצעים קריאה לROUTE נוסף, המערכת לא יודעת שבצענו לוגין והמשתמש הספציפי הזה אכן קיים. לכן כאשר אנו מבצעים לוגין אנו מחזירים TOKEN, בקריאות הבאות המשתמש יעביר את הTOKEN הזה ואז בקריאות המערכת תזהה באיזה משתמש מדובר ואכן תדע איזה משתמש זה.

```
npm i jsonwebtoken
```

```

} package.json > ...
  "dev": "nodemon server"
},
"keywords": [],
"author": "",
"license": "ISC",
"dependencies": {
  "bcrypt": "^5.1.0",
  "cookie-parser": "^1.4.6",
  "cors": "^2.8.5",
  "dotenv": "^16.0.3",
  "express": "^4.18.2",
  "jsonwebtoken": "^9.0.0",
  "mysql2": "^2.3.3",
  "sequelize": "^6.28.0"
},
"devDependencies": {
  "nodemon": "^2.0.20",
  "sequelize-cli": "^6.5.2"
}

```



נוסיף בראש הקובץ

```
const jwt= require('jsonwebtoken')
```

ונמשיך עם פונקציית הלוגין

```

//ניצור אובייקט המכיל את הפרטים ללא הסיסמא
//const userInfo = {password, ...foundUser}

```

```
const userInfo= {id:foundUser.id,name:foundUser.name,
roles:foundUser.roles, username:foundUser.username}
```

נחולל את הTOKEN באמצעות jsonwebtoken
הפונקציה sign מקבלת את האובייקט וסיסמא לערבול

```
const accessToken = jwt.sign(userInfo,"סיסמא לערבול")
```

בקובץ ה.env נוסיף את הסיסמא

```
ACCESS_TOKEN_SECRET=mysecretpassword
```

על מנת לחולל סיסמא ראויה ניתן לכתוב את שורת הקוד הבא בקונסול תחת הרצה של node

ונריץ את הפקודה הבאה

```
require('crypto').randomBytes(64).toString('hex')
```

ואז מתקבלת סיסמא "ראויה" שנעביר אותה לקובץ ה.env

```
ex5 > .env
1  NODE_ENV=development
2  DATABASE_HOST=localhost
3  DATABASE_USER=root
4  DATABASE_PASSWORD=g9095398
5  DATABASE_DB=notes_app
6  ACCESS_TOKEN_SECRET=6aeb1e2b69accbb09a309a0bc69f454213ef110d70547d7edc3
7

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
found 0 vulnerabilities
PS D:\דומיל\node-seqlize\server\ex5> node
Welcome to Node.js v16.13.0.
Type ".help" for more information.
> require('crypto').randomBytes(64).toString('hex')
'6aeb1e2b69accbb09a309a0bc69f454213ef110d70547d7edc3b1c9ec03fcfec43cb2956f76
9045559f86bac1f9a6eb15818a53b578b57207c74f477aced1cc'
> 
```

נחולל את הTOKEN בהתאם לסיסמא השמורה בקובץ ה.env

```
const accessToken= jwt.sign(userInfo,process.env.ACCESS_TOKEN_SECRET)
```

ונחזיר את הTOKEN

```
res.json({accessToken:accessToken})
```

הקוד המלא של פונקצית הלוגין

```
const login = async (req, res) => {
```

```

const { username, password } = req.body

if (!username || !password) {
  return res.status(400).json({ message: 'All fields are required'
})
}

const foundUser = await User.findOne({where:{username:username}})

if (!foundUser || !foundUser.active) {
  return res.status(401).json({ message: 'Unauthorized' })
}

const match = await bcrypt.compare(password, foundUser.password)

if (!match) return res.status(401).json({ message: 'Unauthorized' })

//ניצור אובייקט המכיל את הפרטים ללא הסיסמא
//const userInfo = {password, ...foundUser}
const userInfo= {id:foundUser.id, username:foundUser.username,
roles:foundUser.roles,name:foundUser.name }

//Create the token
const accessToken = jwt.sign(userInfo,process.env.ACCESS_TOKEN_SECRET)
//res.setHeader('Authorization', `Bearer ${accessToken}`)

res.json({accessToken:accessToken})
}

```

נבדוק בPOSTMAN אם מבצעים לוגין מתקבל ACCESS TOKEN

POST http://localhost:3600/api/auth/login Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies


raw JSON Beautify

```

1
2  .... "username" : "geshtop@gmail.com",
3  .... "password" : "123456"
4

```

Body 200 OK 103 ms 515 B Save Response

Pretty Raw Preview Visualize JSON 

```

1
2  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImwidXNlcm5hbWUiOiJnZXNodG9wQGdtYWlsLmNvbSIsInJvbGVzIjoivVNFUiIsIm5hbWUiOiJHZXVsYSBTaG9zaGEyIiwiaWF0IjoxNjc2MzYyNDQwfQ.RwPmp4uFXsGyvgvdisI7IdVxIciDJJa0tu0wYLavao"
3

```

3 אבטחת הקריאות

כרגע כאשר המשתמש ביצע לוגין, קיבל את ה-TOKEN או נעביר את ה-TOKEN בתוך ה-HEADER לשאר הקריאות.

אנו מצפים בקריאות מאובטחות שבקריאה יהיה ה-HEADER בשם Authorization והתוכן שלו יהיה

Bearer THE_ACCESS_TOKEN

נוסיף middleware לבדיקה האם הגולש הוא אכן משתמש שביצע לוגין

נוסיף תקיה לפרוייקט בשם middleware ותחתיו נוסף קובץ verifyJWT.js

```

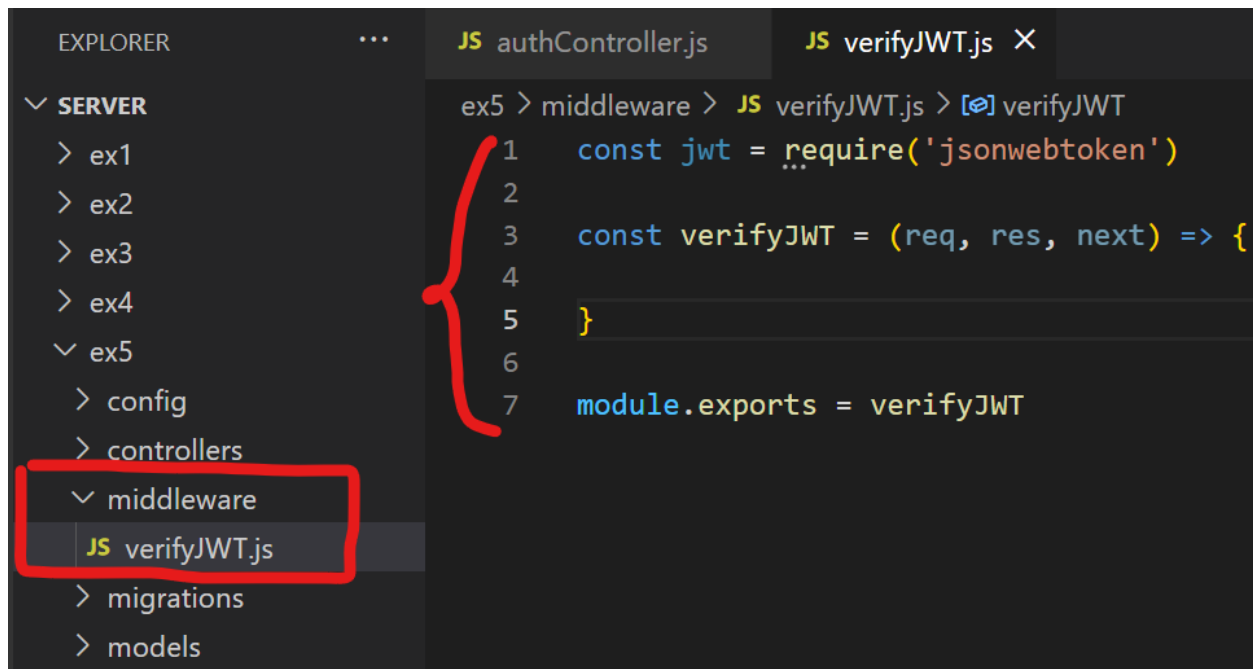
const jwt = require('jsonwebtoken')

const verifyJWT = (req, res, next) => {

}

module.exports = verifyJWT

```



נבדוק שאנו אכן מקבלות את הHEADER ושאלן הHEADER מתחיל בטקסט Bearer

```
const authHeader = req.headers.authorization ||  
req.headers.Authorization  
  
if (!authHeader?.startsWith('Bearer ')) {  
  return res.status(401).json({ message: 'Unauthorized' })  
}
```

במידה ואכן מתקבל נשלוף את הTOKEN

```
const token = authHeader.split(' ')[1]
```

נבדוק שהTOKEN תקין באמצעות jwt.verify שמקבלת 3 פרמטרים

1. הTOKEN
2. הסיסמא איתה הTOKEN הוטמע
3. פונקציית CALLBACK שגם היא מקבלת 2 פרמטרים הפרמטר הראשון אם יש שגיאה לדוגמא הTOKEN לא תקין והפרמטר השני האובייקט אותו הטמענו בTOKEN במידה והכל תקין נכון להכניס את המשתמש לתוך REQ כך ניתן להשתמש בו בכל הקריאה

```
jwt.verify(  
  token,  
  process.env.ACCESS_TOKEN_SECRET,  
  (err, decoded) => {
```

```

        if (err) return res.status(403).json({ message: 'Forbidden' })
        req.user = decoded

        next()
    }
}
)

```

הקובץ המלא של middlewares

```

const jwt = require('jsonwebtoken')

const verifyJWT = (req, res, next) => {
    const authHeader = req.headers.authorization ||
req.headers.Authorization

    if (!authHeader?.startsWith('Bearer ')) {
        return res.status(401).json({ message: 'Unauthorized' })
    }

    const token = authHeader.split(' ')[1]

    jwt.verify(
        token,
        process.env.ACCESS_TOKEN_SECRET,
        (err, decoded) => {
            if (err) return res.status(403).json({ message: 'Forbidden' })
            req.user = decoded

            next()
        }
    )
}

module.exports = verifyJWT

```

סיימנו לבנות את middlewares

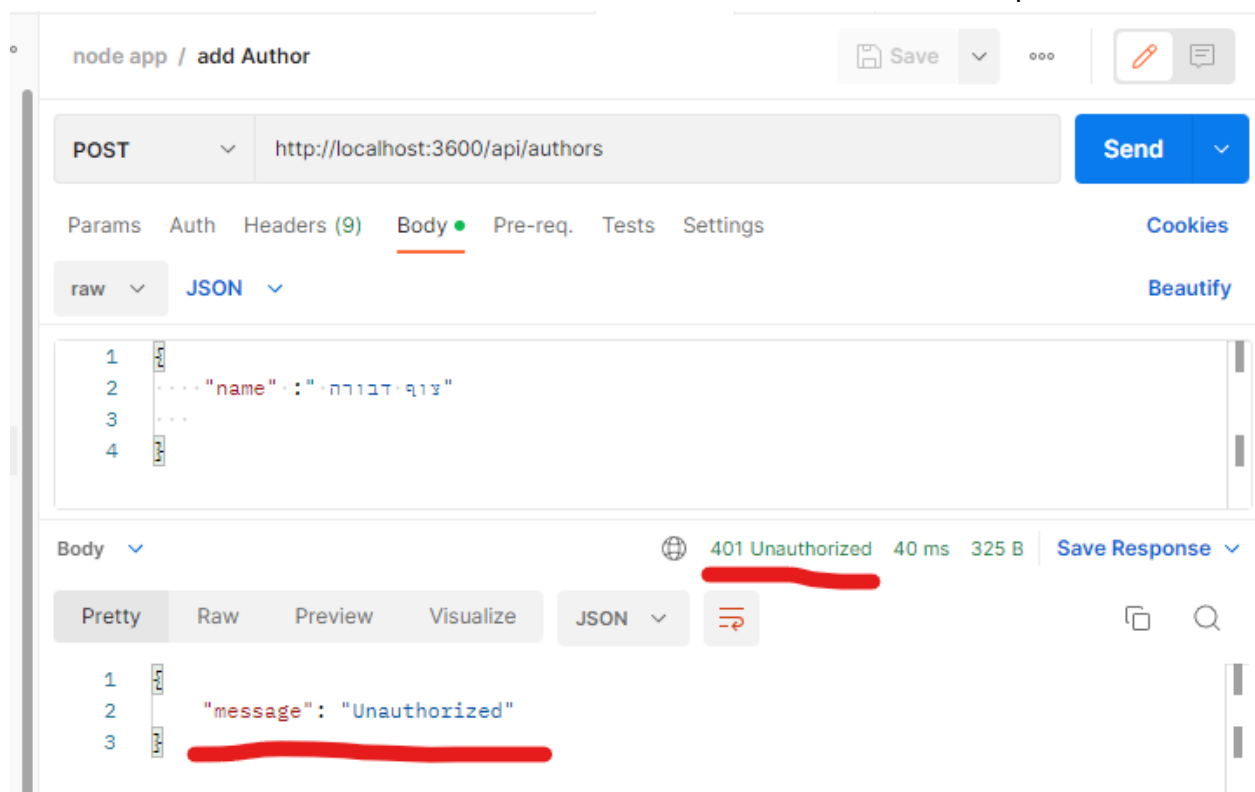
נרצה לדוגמא שאת קריאות הוספת מחבר עדכון מחבר ומחיקת מחבר יהא ניתן רק למשתמשים רשומים

נביא אותו לקובץ ה-Routes, ונוסיף אותו כmiddlewares לקריאות אותן או מעוניינים לחסום

```
const verifyJWT = require("../middleware/verifyJWT")
```

```
JS authController.js JS verifyJWT.js JS authorRoutes.js •
ex5 > routes > JS authorRoutes.js > ...
1  const express = require('express')
2  const router = express.Router()
3  const authorController = require("../controllers/authorController")
4  const verifyJWT = require("../middleware/verifyJWT") ← 1
5  router.route('/')
6    .get(authorController.getAll)
7    .post(verifyJWT, authorController.create)
8    .patch(verifyJWT, authorController.update)
9    .delete(verifyJWT, authorController.deleteOne)
10 router.get('/:id', authorController.getOne)
11 module.exports = router
12
```

עכשיו אם ננסה להוסיף מחבר



גם אם נוסיף את ה-HEADER אך הוא לא תקין נקבל שגיאה

node app / add Author

POST http://localhost:3600/api/authors

Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

<input checked="" type="checkbox"/>	Accept	*/*
<input checked="" type="checkbox"/>	Accept-Encoding	gzip, deflate, br
<input checked="" type="checkbox"/>	Connection	keep-alive
<input checked="" type="checkbox"/>	Authorization	Bearer WRONG_TOKEN
	Key	Description

Body 403 Forbidden 6 ms 319 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Forbidden"
3 }
```

אבל אם נבצע לוגין תקין ונקבל TOKEN

node app / auth login

POST http://localhost:3600/api/auth/login

Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies Beautify

raw JSON

```
1 {
2   "username": "geshtop@gmail.com",
3   "password": "123456"
4 }
```

Body 200 OK 77 ms 515 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImwidXNlcm5hbWUiOiJnZXNodG9wQGdtYWlsLmNvbSI6InJvbGVzIjoiaVVNFUiIsIm5hbWUiOiJHZXVsYSBTaG9zaGEyIiwiaWF0IjoxNjc2MzY1MDYzfQ.xYpDAuQXHpW0gP-oE4Z1GHcHhkdecC3w7oQthT8CTc0"
3 }
```

נכניס את הTOKEN לתוך הHEADER אז הפעולה תוכל להתבצע

node app / add Author

POST http://localhost:3600/api/authors

Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

Key	Value	Description
<input checked="" type="checkbox"/> Postman-Token	<calculated when request is sent>	
<input checked="" type="checkbox"/> Content-Type	application/json	
<input checked="" type="checkbox"/> Content-Length	<calculated when request is sent>	
<input checked="" type="checkbox"/> Host	<calculated when request is sent>	
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.30.1	
<input checked="" type="checkbox"/> Accept	/*/*	
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/> Connection	keep-alive	
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImwidXNlcm5hbWUiOiJnZXNodG9wQGdtYWlsLmNvbSIsInJvbGVzIjoiaVNFUilslm5hbWUiOiJHZXVsYSBTaG9zaGEuX2liwWF0IjoxNjc2MzY1MDYzfQ.xY	
Key	PDauQXHpW0gP-oE4Z1GHcHhkdecC3w7oQthT8CTc0	

Body

201 Created 34 ms 326 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "message": "New author created"
3 }

```

במידה ויש לנו קובץ ROUTES שאנו מעוניינים שכל הROUTES ישתמשו בו ניתן להוסיף אותו לROUTER וזה יחול על כל הROUTES

```

const verifyJWT = require('../middleware/verifyJWT')

router.use(verifyJWT)

```

שימוש בreq.user בכל הקריאות המאובטחות
 כמו שיצרנו לעיל הוספת מחבר מאובטח
 אם נכנס לקונטרולר בהוספת מחבר נראה שכבר קיים req.user אותו הטמענו במiddleware

```
15  const create = async (req, res) => {  
16    console.log(req.user)  
17  
18    const { name } = req.body
```

PROBLEMS

OUTPUT

TERMINAL

DEBUG CONSOLE

```
{  
  id: 2,  
  username: 'geshtop@gmail.com',  
  roles: 'USER',  
  name: 'Geula Shosha1',  
  iat: 1676365063
```