

NodeJS

**יחסים גומליין -
User Role -**

JWT

סיום הרשמה והתחברות

TomerBu

לכל משתמש שנרשם - נשמר Role של user

```
//api/auth/signup
router.post("/signup", validateSignUp, userAlreadyExists, async (req, res) => {
  const body = _.pick(req.body, "username", "email", "password");

  body.password = await bcrypt.hash(body.password, 12);
  const user = new User(body);
  //before saving the user:

  try {
    //for each user -> save the role id of user
    user.roles = [await (await Role.findOne({name: 'user'}))._id]
    await user.save();
    return res.json({ message: "user saved", id: user._id });
  } catch (e) {
    return res.status(500).json({ message: "Server DB Error", error: e });
  }
});
```

לכל משתמש נוסיף למשתמש roles
את objectId של role של user

בדיקה:

###POST to add user:

Send Request

POST <http://localhost:3001/api/auth/signup>

Content-Type: application/json

```
{  
  "username": "Bruce",  
  "email": "Batman@Batcave.com",  
  "password": "tomerBu@123!g"  
}
```

```
1  _id: ObjectId('63e1fe8d2b9849ed87215da8')  
2  username: "Bruce"  
3  email: "Batman@Batcave.com"  
4  password: "$2a$12$RnzpwpDao9rc403g7qpP6emQW5/2eCFquerqt2x17AgyU8QAdJ6.i"  
5  ▼ roles: Array  
6    0: 63e1fb4ff70ed483ab2fe1ad ← user Role objectid  
7  __v: 0
```

בכל פעם שימוש מתהבר

נחזיר לו JWT (בדומה לKEY API)
ואיתו הוא יוכל להשתמש כדי לשולח בקשות.

אחרת - **בכל שימוש במשאב מוגן -> המשתמש יctrak להזינו שם משתמש וסיסמה.**

נשמר JWT בצד לקוח בlocalStorage

קובץ הגדרות לחתימת JWT JWT Token



ts auth.config.ts U X

src > db > config > ts auth.config.ts > [d] default

```
1 export default {  
2   secret: "My Secret Key"  
3 }
```

עבודה עם הספרייה jsonwebtoken

npm i jsonwebtoken @types/jsonwebtoken

```
import jwt from "jsonwebtoken";  
  
import authConfig from './db/config/auth.config.js';  
  
router.post("/signin", validateSignIn, async (req, res) => {  
  try {  
    const user = await User.findOne({ email: req.body.email });  
  
    if (!user) {  
      return res.status(401).json({ message: "No Such User" });  
    }  
  
    const isPasswordValid = await bcrypt.compare(  
      req.body.password,  
      user.password  
    );  
  
    if (!isPasswordValid) {  
      return res.status(401).json({ message: "Invalid Credentials" });  
    }  
  
    const token = jwt.sign({ id: user.id }, authConfig.secret, {  
      expiresIn: "30d",  
    });  
    return res  
      .status(200)  
      .json({ message: "Sign in succesfull", token: token });  
  } catch (e) {}  
});
```

מי מוחזר את ה`id` כמחרוזת ולא כ`ObjectId`
(Mongoose מבונה בסיסי)

הוספה Role של Admin למשתמש מסויים במנגו:

The screenshot shows the MongoDB Compass interface. On the left, the sidebar lists databases and collections: My Queries, Databases, Search, BizCards (with sub-collections cards, roles, students, users), admin, blogi, business_card_app, config, dbtest2, lec1, and lec2. The 'roles' collection under 'BizCards' is selected and highlighted with a green background. The main pane displays the 'Documents' tab for the 'BizCards.roles' collection. It shows three documents:

- Document 1: _id: ObjectId('63e1fb4ff70ed483ab2fe1ac'), name: "moderator", __v: 0
- Document 2: _id: ObjectId('63e1fb4ff70ed483ab2fe1ad'), name: "user", __v: 0
- Document 3: _id: ObjectId('63e1fb4ff70ed483ab2fe1ab') (highlighted with a red box), name: "admin", __v: 0

נעתיק את התוכן של ה`id`

```
_id: ObjectId('63e20a6c1d9dec44b35255c7')
username: "TomerBu"
email: "Tomerbu@gmail.com"
password: "$2a$12$gNOuT7y77Dew.HW/.dnXTe4EVj7MrYd609oL6Y2jaAISBppl0gMkW"
roles: Array
  0: ObjectId('63e1fb4ff70ed483ab2fe1ad')
  1: ObjectId('63e1fb4ff70ed483ab2fe1ab')
  2: ObjectId('63e1fb4ff70ed483ab2fe1ac')
__v: 0
```



הוספה Role של Admin למשתמש מסוים במנג'ר:

1 `_id: ObjectId('63e20a6c1d9dec44b35255c7')` ObjectId
2 `username: "TomerBu"` String
3 `email: "Tomerbu@gmail.com"` String
4 `password: "$2a$12$gNOuT7y77Dew.HW/.dnXTe4EVj7MrYd609oL6Y2jaAISBppl0gMkW,"` String

[] [+] ▼ roles: Array Array

0: 63e1fb4ff70ed483ab2fe1ad
1: 63e1fb4ff70ed483ab2fe1ab
2: 63e1fb4ff70ed483ab2fe1ac

✖ Add item to roles

+ Add field after roles

CANCEL UPDATE

1 `_id: ObjectId('63e1fe8d2b9849ed87215da8')` ObjectId
2 `username: "Bruce"` String
3 `email: "Batman@Batcave.com"` String
4 `password: "$2a$12$RnzpwpDao9rc403g7qpP6emQW5/2eCFquerqt2x17AjyU8QAdJ6.i"` String

▶ roles: Array Array

--v: 0

1 `_id: ObjectId('63e20a6c1d9dec44b35255c7')` ObjectId
2 `username: "TomerBu"` String
3 `email: "Tomerbu@gmail.com"` String
4 `password: "$2a$12$gNOuT7y77Dew.HW/.dnXTe4EVj7MrYd609oL6Y2jaAISBppl0gMkW,"` String

5 ▶ roles: Array Array

0: 63e1fb4ff70ed483ab2fe1ad
1: 63e1fb4ff70ed483ab2fe1ab
2: 63e1fb4ff70ed483ab2fe1ac
3: ""
--v: 0

[] [+]

Document modified.

CANCEL UPDATE

Object Type dropdown menu:
Array
Binary
Boolean
Code
Date
Decimal128
Double
Int32
Int64
MaxKey
MinKey
Null
Object
ObjectId
BSONRegExp
String
BSONSymbol
Timestamp
Undefined

הוסף של users אחרי השינוי:

```
_id: ObjectId('63e1fe8d2b9849ed87215da8')
username: "Bruce"
email: "Batman@Batcave.com"
password: "$2a$12$RnzpwpDao9rc403g7qpP6emQW5/2eCFquerqt2x17AjyU8QAdJ6."
▼ roles: Array
  0: ObjectId('63e1fb4ff70ed483ab2fe1ad')
__v: 0
```

```
▶ _id: ObjectId('63e20a6c1d9dec44b35255c7')
username: "TomerBu"
email: "Tomerbu@gmail.com"
password: "$2a$12$gNOuT7y77Dew.HW/.dnXTe4EVj7MrYd609oL6Y2jaAISBppl0gMkW"
▼ roles: Array
  0: ObjectId('63e1fb4ff70ed483ab2fe1ad')
  1: ObjectId('63e1fb4ff70ed483ab2fe1ab')
  2: ObjectId('63e1fb4ff70ed483ab2fe1ac')
__v: 0
```



הרשמה:

שמירה של משתמש בדטה-ביס. (רַק אם הקלט תקין) (ורק אם אין זה משתמש)

כל משתמש נגדיר עם תפקיד של user

התחברות:

המשתמש מזין אימייל וסיסמא.

נבדוק שהbody תקין

נבדוק שהמשתמש קיים

נבדוק שהסיסמה توامة

נפיק JWT ללקוח שאיתו יוכל לגשת למשתמשים.

פרטים נוספים לגבי JWT

```
const token = jwt.sign({ email: user.email }, authConfig.secret, {  
  expiresIn: "30d",  
});
```

###POST · to · sign · in · user:

Send Request

POST · <http://localhost:3001/api/auth/signin>

Content-Type: · application/json

```
{  
  ...  
  "email": "Batman@Batcave.com",  
  ...  
  "password": "tomerBu@123!g"  
}
```

↙{

```
  "message": "Sign in succesfull",  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJlb  
WFpbCI6IkJhdG1hbkBCYXRjYXZlLmNvbSIssImhdCI6MTY3NTc2MTM  
yMCwiZXhwIjoxNjc4MzUzMzIwfQ.ac71lZz7LwSspX0322patX2fe0  
V7ItZigo0SzIRhLRI"
```

}

פרטים נוספים לגבי ה-JWT

The screenshot shows the jwt.io debugger interface. At the top, there's a warning message: "Warning: JWTs are credentials, which can grant access to resources. Be careful where you paste them! We do not record tokens, all validation and debugging is done on the client side." Below this, the "Algorithm" dropdown is set to "HS256".

Encoded: PASTE A TOKEN HERE
The input field contains the following encoded JWT token:
`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6IkJhdG1hbkBcYXRjYXZlLmNvbSIsImhlhdCI6MTY3NTc2MTIyNiwiZXhwIjoxNjc4MzUzMjI2fQ.PGq56Kc9Ah_GgQGX26jGfAnhrrLHy7dGff09RxRbE7U`

Decoded: EDIT THE PAYLOAD AND SECRET
The decoded payload is shown in two sections: HEADER and PAYLOAD.

HEADER: ALGORITHM & TOKEN TYPE
`{
 "alg": "HS256",
 "typ": "JWT"
}`

PAYOUT: DATA
`{
 "email": "Batman@Batcave.com",
 "iat": 1675761226,
 "exp": 1678353226
}`

כתובת האימייל ניתנת לפענוח מתוך הtoken.

ולכן לא נשמר כאן סיסמות.
אפשר גם לא לשמור את האימייל.

אחרי login מוצלח

```
router.post("/signin", validateSignIn, async (req, res) => {
  //email and password:
  try {
    //SELECT * FROM user JOIN Roles ON ...
    const user = await User.findOne({ email: req.body.email }).populate(
      "roles"
    );

    if (!user) {
      return res.status(401).json({ message: "No Such User" });
    }

    const isPasswordValid = await bcrypt.compare(
      req.body.password,
      user.password
    );

    if (!isPasswordValid) {
      return res.status(401).json({ message: "Invalid Credentials" });
    }

    const token = jwt.sign({ id: user.id }, authConfig.secret, {
      expiresIn: "30d",
    });

    const authorities = [];
    for(let i = 0; i< user.roles.length; i++){
      authorities.push(`ROLE_${user.roles[i].name.toUpperCase()}`)
    }

    return res.status(200).json(
      {
        id: user.id,
        username: user.username,
        email: user.email,
        roles: authorities,
        accessToken: token
      }
    );
  } catch (e) {
    return res.status(500).json({ message: "Server error", error: e });
  }
});
```

- נשלח את התפקידים של המשתמש roles
כך בצד לקוח - נוכן להציג עוד אפשרויות

הפעולה populate של mongoose
מקלה על שאלות עם יחס גומליין

מקביל לJOIN בSQL

במקום roles יהיה מערך של ObjectId

Role יהיה מערך של roles

יצרנו מערך:
למערך נוסיף רק את name של כל roles שיש
למשתמש.

"roles": [
 "ROLE_USER"
,

עדכון של Types user.roles כדי יהיה מוכן לTS

```
//SELECT * FROM user JOIN Roles ON ...
const user = await User.findOne({ email: req.body.email }).populate<{
  roles: Array<typeof Role>;
}>("roles");
```

```
username:string
email: string
roles: [{name: 'admin', _id: '2341233244acg'}]
```

סיכום עד כה.

הרשמה:

שמירה של משתמש בדטה-בייס. (רק אם הקלט תקין) (ורק אם אין כזה משתמש)
כל משתמש נגדיר עם תפקיד של user
שומרים הצפנה של הסיסמא

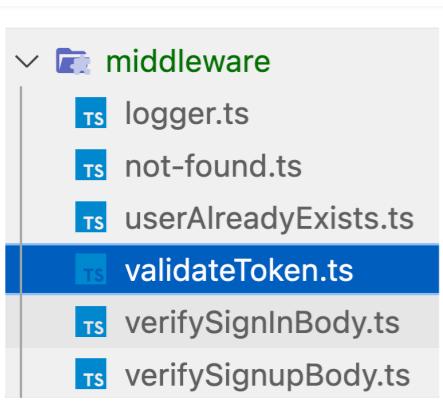
התחברות:

המשתמש מזין אימייל וסיסמא.
נבדוק שהbody תקין
נבדוק שהמשתמש קיים
נבדוק שהסיסמא תואמת
נפיק JWT לckooh שאיתו יוכל לגשת למשאבים.

יצרנו מנגנון של התחברות והרשמה.

נרצה ליצור Routes למשל כרטיסי עסק.
חלק מהroutes יהיו מוגנים - עדכון כרטיס/מחיקת כרטיס JWT ונבדוק אם יש רשות
חלק מהroute לא יהיו מוגנים.

ויזוא של JWT Token Middleware



```
GET http://localhost:3001/api/cards  
Authorization: YOUR_JWT_TOKEN
```

בקשה - יכולים לשלוח Header של Authorization אם המשתמש אכן קיים והJWT תקין ומאושר - יש רשות

```
import jwt from "jsonwebtoken";
import { RequestHandler } from "express";
import authConfig from "../db/config/auth.config.js";

const validateToken: RequestHandler = (req, res, next) => {
  //get the header from the request:
  const token = req.headers.authorization;

  if (!token) {
    //403 = unauthorized
    return res.status(403).json({ message: "No Token Provided" });
  }

  jwt.verify(token, authConfig.secret, (err, payload)=>{
    if(err){
      return res.status(403).json({ message: "Invalid Token" });
    }

    //get the user id from the payload:
    //ts defintion for id in payload
    const jwtPayload = payload as {id: string}
    const id = jwtPayload.id;

    //add the userId to the request ->
    req.userId = id
    next()
  });
};
```

הרחבה של Request לכל הפרויקט: userId?:string בכל request

 @types.d.ts

```
// to make the file a module and avoid the TypeScript error
export {}

declare global {
    namespace Express {
        interface Request {
            userId?: string;
        }
    }
}
```

ויזוא של JWT Token

```
import jwt from "jsonwebtoken";
import { RequestHandler } from "express";
import authConfig from "../db/config/auth.config.js";

//POST api/cards
//AUTHORIZATION: dsflkfjweklrj23lkj423lk4j3
//CONTENT-TYPE

// 
//{
    //...BODY
//}

const validateToken: RequestHandler = (req, res, next) => {
    //get the header from the request:
    const token = req.headers.authorization;

    if (!token) {
        //403 = unauthorized
        return res.status(403).json({ message: "No Token Provided" });
    }

    jwt.verify(token, authConfig.secret, (err, payload: {id:string})=>{
        if(err){
            return res.status(403).json({ message: "Invalid Token" });
        }

        //get the user id from the payload:
        const id = payload.id;

        //add the userId to the request ->
        req.userId = id
        next()
    });
};

export { validateToken };
```

האם המשתמש הוא ?admin

middleware

```
import { RequestHandler } from "express";
import { Role } from "../db/models/role.js";
import { User } from "../db/models/user.js";
const isAdmin: RequestHandler = async (req, res, next) => {
  const userId = req.userId; ← רישיון userId

  try {
    const user = await User.findById(userId); ← מצא את המשתמש מטבלת המשתמשים:
    //user.roles = ['120231005fc', '120231005fa', '120231005fb']
    //~populate
    const roles = await Role.find({ _id: { $in: user.roles } }); ← roles גם למשתמש

    for (let role of roles) {
      if (role.name === "admin") {
        return next();
      }
    }
    return res.status(403).json({ message: "Requires Admin Role" });
  } catch (e) {
    return res.status(500).json({ message: "Requires Admin Role", error: e });
  }
  //find the user role => if admin =>
};

export { isAdmin };
```

middleware יוכנס תמיד אחרי validateToken
ולכן יוכל לסרוק על כך שיש userId

האם המשתמש הוא ?moderator

```
import { RequestHandler } from "express";
import { Role } from "../db/models/role.js";
import { User } from "../db/models/user.js";
const isModerator: RequestHandler = async (req, res, next) => {
  const userId = req.userId;

  try {
    const user = await User.findById(userId);
    const roles = await Role.find({ _id: { $in: user.roles } });

    for (let role of roles) {
      if (role.name === "moderator") {
        return next();
      }
    }
    return res.status(403).json({ message: "Requires moderator Role" });
  } catch (e) {
    return res
      .status(500)
      .json({ message: "Requires moderator Role", error: e });
  }
};

export { isModerator };
```

Protected Routes

```
import { Router } from "express";
import { isAdmin } from "../middleware/isAdmin.js";
import { isModerator } from "../middleware/isModerator.js";
import { validateToken } from "../middleware/validateToken.js";

const router = Router();

// רק משתמש מחובר יוכל לראות את כל הספרים
router.get("/all", validateToken, (req, res) => {
  res.json([{ title: "Hunger Games" }]);
});

// רק אדמין יוכל לראות את ספרי פנטזיה
router.get("/fantasy", validateToken, isAdmin, (req, res) => {
  res.json([{ title: "Harry Potter" }]);
});

// רק מודרטור יוכל לראות את ספרי פנטזיה
router.get("/mod", validateToken, isModerator, (req, res) => {
  res.json([{ title: "Harry Potter" }]);
});

export { router as booksRouter };
```

Protected Routes

```
requests.rest M X
requests.rest > ...
11 | ###
12 | Send Request
13 | GET http://localhost:3001/api/books/all
14 | Authorization: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYzZTIwYTZjMWQ5ZGVjNDR
15 | ###
16 | Send Request
17 |
18 | GET http://localhost:3001/api/books/fantasy
19 | Authorization: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYzZTIwYTZjMWQ5ZGVjNDR
20 | ###
21 | Send Request
22 | GET http://localhost:3001/api/books/mod
23 | Authorization: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYzZTIwYTZjMWQ5ZGVjNDR
24 | Response(385ms) X
25 | 
26 | 11 <{
27 | 12   "id": "63e24e8bcc82e9b4b9d888a9",
28 | 13   "username": "Batman",
29 | 14   "email": "Batman@gmail.com",
30 | 15   "roles": [
31 | 16     "ROLE_USER"
32 | 17   ],
33 | 18   "accessToken": "eyJhbGciOiJIUzI1NiI
34 | 19     sInR5cCI6IkpXVCJ9.eyJpZCI6IjYzZTI0ZTh
35 | 20       iY2M4MmU5YjRiOWQ40DhhOSIsImlhdCI6MTY3
36 | 21       NTc3NTcy0CwiZXhwIjoxNjc4MzY3NzI4fQ.QL
37 | 22       MLLY1_upVsULK06FYnVCnHWqlNElKzPfxtU88
38 | 23       j5x0"
39 | 24     }
```

עד ולידציות:

מקבלים בקשה מהלקוח:
נחלץ מהבקשת את השדות: `pick_()`
נשתמש בוועל כדי לוודא תקינות.
נשמר לדטה-בייס.

אפשר להוסיף שכבת ולידציה נוספת:
.Mongoose של

בשביל לוודא עם Mongo
הולכים לדטה-בייס מנסים לשמר ולא מצלחים.

אפשרת הגדרה חוקי ולידציה: Mongoose

```
const breakfastSchema = new Schema({
  eggs: {
    type: Number,
    min: [6, 'Must be at least 6, got {VALUE}'],
    max: 12
  },
  drink: {
    type: String,
    enum: {
      values: ['Coffee', 'Tea'],
      message: '{VALUE} is not supported'
    }
  }
});
```

שלב אקסטרה:

מניעת שmieה של מידע לא תקין:

initDB(){...}

השלמה אוטומטית יותר טובה עם TypeScript

TS @types.d.ts

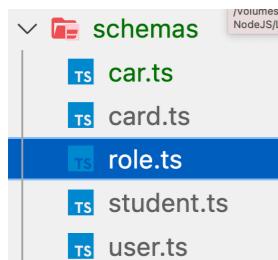
```
export type Car = {  
    vandor:string,  
    model: string,  
    color:string,  
    image?:string  
}
```

```
import { Car } from "../../@types.js";  
import { Schema } from "mongoose";  
const carSchema = new Schema<Car>({  
    vandor: {  
        type: String,  
        required: true,  
        minlength: 2,  
        maxlength: 30,  
    },  
    color: {  
        type: String,  
        required: true,  
        minlength: 2,  
        maxlength: 30,  
    },  
    image: String,  
    model: String,  
});
```

השלמה אוטומטית יותר טובה עם TypeScript בMongoose

TS @types.d.ts

```
export type Role = {
  name: string
}
```



```
import { Role } from "../../../../../@types.d.js";
import { Schema } from "mongoose";

//role has a role name: (user/admin/moderator)
const roleSchema = new Schema<Role>({
  name: { type: String, unique: true },
});

export { roleSchema };
```

הודעות שגיאה עם Mongoose

```
Joi.object({
  age: Joi.number()
    .integer()
    .min(18)
    .messages({ "number.min": "You must be at least 18 years old" })
});
```

הגדרות CORS לצד שרת:

```
//middleware:  
app.use(  
  cors({  
    origin: "http://localhost:3000",  
    allowedHeaders: ["Content-Type", "Authorization"],  
  })  
);
```

פרויקט צד לקוח:

```
npx create-react-app biz-cards --template redux-typescript
```

```
npm i bootstrap sass formik yup react-router-dom react-icons  
react-modal sweetalert2 react-bootstrap react-toastify react-  
loader-spinner axios
```

פרויקט צד לקוח:

```
import { useEffect } from "react";
import "./App.css";

function App() {
  useEffect(() => {
    fetch("http://localhost:3001/api/auth/signin", {
      headers: { "Content-Type": "application/json" },
      method: "POST",
      body: JSON.stringify({
        email: "Batman@gmail.com",
        password: "123456aA!",
      }),
    })
    .then((res) => res.json())
    .then((json) => {
      console.log(json);
    })
    .catch((e) => {
      console.log(e);
    });
  }, []);
  return <div className="App"></div>;
}

export default App;
```

בדיקה שהכל עובד:

שיעור בית:

להתחל ל לעבוד על פרויקט מסכם.

דרישות לפרויקט המסכם מצורפות בעמודים הבאים

דף אפ"ן פרויקט גמר

למידה עצמית היא Skill מאוד מרכזי במקצוע שלנו, ראוי להשתמש בו ככל האפשר במהלך הפרויקט. עם זאת, חשוב את הזרים מראש (עד Skill חשוב במקצוע שלנו) וראו לסימן את הפרויקט ולהגישו בזמן. השתדל לא להיגר ללמידה ושימוש עוד ועוד בנושא אם זה יפריע לכם להגיש את הפרויקט בזמן.

נדרשת השקעה של כ 120 שעות פיתוח ויש עד 3 חודשים להגשה, המרצה/ הרכזות יגידו לכם את תאריך ההגשה

לטובת הפרויקט תבנה אתר צד לקוח עם צד שרת תומך בנושא שתבחר.

על הפרויקט להיראות כמו אתר/**אפליקציה** אמיתית עם תמונות ותוכן טקסטואלי שלא כוללים **lorem ipsum** וכן כודומה. כמו כן הפרויקט צריך להיות רספונסיבי ומתאים לכל גודל המסך האפשרי.

דרישות כלליות:

- יש לשמר על קוד נקי ומסודר: לנוקוט `log` וקטעי קוד שהפכו להערות.
- מומלץ לבנות קוד שמספר סיפור ולתת לפונקציות ולמשתנים שמות משמעותיים.
- יש לחלק את הפרויקט למודולים לפי נושאים
- יש לשימוש את כל הקשרו לעיצוב בקובץ `css`, את התמונות בתיקיית `images` וכו'.
- כמו כן יש להקפיד על המוסכמות לכתיבת קוד.
- עיצוב הוא חלק בלתי נפרד מהצגת הפרויקט והיכולות של הפיתוח. גם אם אין מעצב באופי, הקפיד על עיצוב נקי ורספונסיבי לגדים שונים של מסכים!

חשוב לציין שפרויקט זה יהיה חלק מתיק העבודות שלכם וייצג אותך בכבוד מול מעסיקים פוטנציאליים
ולכן יש לשמר על אסתטיקה של קוד ועיצוב.

דרישות טכנולוגיות צד לקוח:

- עיצוב ורספונסיביות מומלץ להשתמש בספרייה bootstrap או Material Design
- קובץ העיצוב הראשי אם קובץ העיצוב (`css`) הוא מעל 100 שורות, יש לחלק אותו לקובציים נפרדים לפי הנושאים השונים. לצורך העניין מומלץ להשתמש בספריית scss
- אייקונים מומלץ להשתמש באילוקונים לדוגמה מספרית google material fontawsom או fontawsom
- דף כניסה צריך לכלול כותרת ראשית, כותרת משנה, טקסט תמורה שיתאמו לאופי האתר/ האפליקציה. אם מדובר באתר של חנות אינטרנטית כלשהי, יש להציג בדף הפתיחה שדה חיפוש ולפחות שלשה כרטיסי מוצר. מדף הפתיחה צריך להיות ברור לאיזה סוג של האתר/ אפליקציה הגענו וצריך להיות מעוצב בצורה עצמאית שתזמין את הגולש להירשם.
- תפריט ניווט על האתר/ אפליקציה להכיל תפריט ניווט דינامي שימושות לכלי דפי האתר
- Footer על האתר / אפליקציה להכיל footer עם לוגו, זכויות יוצרים ואמצעי ליצור קשר עם האתר. במידת הצורך ניתן להוסיף גם בתפריט הניווט, קישורים למדיה חברתית או כל דבר אחר שיתאים לאזור זה באפליקציה/אתר
- נגישות יש לשימוש שם האפליקציה בתגית ה – `title` בקובץ ה – `index.html` הראשי, וכן תמורה לוגו – `favicon:link`. כל תמורה חייבת לכלול את האטירבות `alt` עם כתוב שיתאר את התמונה.

- **דף אודות** יש ליצור דף אודות בו תספקו הסבר מעמיק על האתר ודרך ההתמכשות עמו.
- **הירשות והתחברות** על מנת仄ן כדי ל��וח להציג דף התחברות ודף הרשמה הכלולים ולידציות על שדות הטפסים השונים. צריך להיות חיוני מתחת לשדה הרלוונטי אם המשתמש עומד או לא עומד בדרישות השדה או תוך כדי ה הקלדה או focus change so לשדה אחר. יש לאפשר שליחה של טופס אך ורק לאחר שכל שדות החובה מלאים. בלחיצה על כפתור השלח בטופס, יש לעדכן את האולש בהצלחה או כישלון שליחת הנתונים. כדי להשתמש לצורך העניין בספרית react-tostify או בספריה דומה. מומלץ לאפשר התחברות גם באמצעות חשבון Google, Facebook וכו.
- **Regex** יש להשתמש ב – `Regex` לשדה הסיסמה בטפסים, שיחיב הכנסת סיסמה עם לפחות אות אחת גדולה ואות אחת קטנה באנגלית, לפחות ארבעה מספרים ולפחות סימן מיוחד מבין הסימנים הבאים (!@#\$%^&*_) וascal הסיסמה תהיה לפחות בת 8 תווים. כמו כן יש להשתמש ב `Regex` לשדה של טלפון שיחיב הכנסת מס' טלפון תקין אם אפשרות לרוח אום מקף אמצעי (-) לאחר שניים או שלושה המספרים הראשונים. וכן `Regex` על שדה המייל שיחיב הכנסת מייל תקין. יש אפשרות ליצור פונקציות הצד שרת לצורך העניין או להשתמש בספרית jio-browser.
- **Token** לאחר התחברות יש ליצור token עם ערך מוצפן תוך שימוש בספרית jwt. ה – `token` צריך להישמר ב – `localStorage` ולוודор בקביעת סטאטוס המשתמש ואם הוא מחובר או לא. רק במקרה ה – `token` יכולם להיות פרטיים על הלוקום באופן מוצפן.
- **Crud** לאחר התחברות יש לאפשר למשתמש את פעולות ה – `crud`, קרי: קריאה, יצרה, עדכון ומיקפה של תוכן. התוכן שיצרים צריך להיות זמין בחלקים שונים של האתר. לדוגמה אם מדובר בחנות אינטרנטית, לאחר שימושים מוצר הוא צריך להופיע בדף הראשי או בדף מוצר. יש להשתמש בספרית react-tostify או בספריה דומה, כדי לעדכן את האולש בהצלחה/ כישלון ביצוע פעולות ה – `crud`.
- **מודדים/ הכנסה לסל קניות** יש לחת אפשרות לאולש לשמר תוכן (כרטיס/ מוצר/ משתמש וכו') במועדפים. יש לחת חיוני ויזואלי לכך שהתוכן הוא מועדף על ידי האולש. יש לשמר את העדפות הלוקום ברשותה התוכן במאגר המידע כך שלא משנה מאיזה מכשיר המשתמש "יכנס לאתרא/ אפליקציה התוכן שהוא העדיף ימשיך להיות מועדף". יש ליצור דף של תוכן מועדף, בו האולש יוכל לראות את כל הפרטים שהוא סימן כמועדפים ואם ירצה יכול ולהסרר מהדף פרטיים מועדפים
- **דף פרטי תוכן** בלחיצה על כרטיס/ משתמש/ תוכן, האולש יעבור לדף דינامي בו ינתנו פרטיים נוספים על פרט התוכן עליו לחץ האולש
- **שדה חיפוש** יש ליצור שדה חיפוש לתוכן (כרטיס/ מוצר/ משתמש וכו')
- **הרשאות** יש לאפשר יצירה של לפחות שני סוגי משתמשים, כאשר הרישון הוא משתמש רגיל והשני הוא admin. רק משתמש שמו גדר C – `admin` יכול ליצור, לעורר ולמחוק תוכן, בעודו משתמש רגיל יוכל רק לראות או לסמן תוכן כמועדף עליו.
- **קריאות http** יש לבצע קריאות `http` לצד שרת מצד לקוח ובאמצעות ושלוח ולקבל מידע מהשרת. מומלץ להשתמש כר ספריית `axios`. יש להשתמש במנגן `try & catch` בקריאה אסינכרונית לצד שרת במקורה ומשתמשים בפונקציות אסינכרוניות, או לחליפין במנגן `().then().catch()`. עם בוחרם להשתמש ב – `promises` וזאת על מנת שהקובד לא ישבר במקרה ותחזר שגיאה קריטית מהשרת.
- **ארQUITקטורה** יש לשמר על סדר הגיוני ומקובל בתעשייה של קבצים. יש להשתמש בתיקייה services, על הקובד להיות נקי וקריא, עם חלוקה נכונה לתיקיות וקומפוננטות.

- **Console** יש להקליד על עבודה נכונה עם ה – `console`. על הקונסול להיות נקי מהעroot אזהרה, שגיאות ותוכן, כך שייהי ניתן לראות בקלות שגיאות קritisיות מהשרת.
- **סינון תוכן** יש לחת לגולש אפשרות לסנן את התוכן המוצג בדף מסוים לפי פרמטרים שונים **בונוס לצד לקוח**
- **Logout** יש לאפשר לאפליקציה/אתר לנתק את המשתמש במידה ולא השתמש באתר/אפליקציה במשך יותר מ - 4 שעות
- **הגבלת בקשות** יש להגביל את מספר הקיריאות לשרת שימוש יכול לבצע ב – 24 שעות לצורך הגנה על השרת ממתקפה שנועדה להאט/להקריס אותו.
- **ממשק ניהול משתמשים** דף ניהול משתמשים שיציג את מספר המשתמשים הרשומים במאגר המידע באתר ויזג בטבלה את פרטייהם. בדף זה יהיה ניתן למחוק משתמשים או לשנות את הסטאטוס שלהם משתמש רגיל `admin`.
- **ניהול הזמנות/מודדים** דף שיראה לאדמין את התוכן לפי כמות האנשים שהגדירו את אותו התוכן כמועדף עליהם.
- **ניהול מלאי** במקורה של חנות אינטרנטית, דף לניהול מלאי של חנות. וברגע שמוחזר מסויים נגמר מהמלאי יש לחת חוווי לגולש על כך שהמוחזר אזל.
- **תמונת פרופיל משתמש** עדכן פרטי משתמש עם תמונה פרופיל שתועלה לתוך תיקייה במחשב עליו נמצא הפרויקט
- **עדכן סיסמה** יש ליצור דף שיאפשר למשתמש שוכח את הסיסמה שלו להחליף סיסמה זואת רק לאחר וידוא שakan מדבר במשתמש שמנסה לשנות את סיסמתו ולא האкар. לצורך העניין מומלץ לשלוח מייל למשתמש עם 링ק לדף לשינוי הסיסמה.

דרישות טכנולוגיות צד שרת:

- **package.json** בקובץ `package.json` יש לשים ב – `devDependencies` את `nodemon`, וכן שייהי בתוך מפתח ה – "main" עם שם הקובץ להפעלת האפליקציה.
- **האזנה לבקשת HTTP** יש לבנות ממשק `api` שמאפשר לקבל בקשות, ליצירה, עריכה, הצגה ומחיקת מידע ממAGER המידע בהתאם לבקשת צד לקוח. לצורך העניין מומלץ להשתמש בספרית `express`
- **אותנטיקציה** יש להעביר את הבקשה תהליך של אוטנטיקציה על מנת לוודא שakan הגולש הוא זה שלח את הבקשה ולא האкар שמנסה לפורץ למאגר המידע דרך השרת.
- **אוטוריזציה** יש לאפשר בצד שרת רק למשתמש מחובר ומוגדר כ – `admin` לבצע הוסף, מחיקה או עריכה של מידע ממAGER המידע.
- **מאגר מידע** את המידע בפרויקט יש לשמר בסיס הנתונים `MongoDB` או `MySQL` בצורה לוקלית או על ענן. כמו כן יש לאפשר הוספה, עריכה ומחיקה של פרטיים ממAGER המידע. בקובץ `shartet` מגישים חייב להיות קובץ ה – `config` במידה ושמרתם בו את המפתחות להתחברות עם שרת הענן
- **ולידציות צד שרת** יש לעשות ולידציות צד שרת עם ספריית `joon` או ספרייה דומה, ובמקורה של שגיאות יש ולוודר את הפונקציות בטרם שליחת האובייקט לוולידציה של `mongoose` ושמירה במאגר המידע.

- יש לחלק את הקוד למודולים לשומר על קוד נקי וקריא
- **Logger** יש להשתמש בספרייה לניהול בקשות http כדוגמת morgan או לחליפין ליצור logger משלכם שידפיס בקונסול קריאות מצד לקוח לצד שרת
- **הערות על** שמות המשתנים והפונקציות להיות האגינאים ושיספרו סיור על הקוד. במידה הצורך יש להוסיף הערות תמציתיות למתכנתים במידה יש פונקציות מורכבות או שיש דף עם ריבוי פונקציות בתוכו.

הערה כללית
לא ניתן להציג פרויקט סוף מודול כפרויקט גמר

אופן הגשת הפרויקט:
את הפרויקט יש להעלות ל – git ללא תיקיות ה – node_modules, ושלחו במיל נפרדקובץ **ספר פרויקט/ מסמך אפיון** שיכלול קובץ טקסט ReadMe שמסביר על הפרויקט, תוכולתו, הפונקציונליות ודרך ההתמכשות עמו. אם יש קבצי env. או config עם פרטיים אישיים יש לצרף גם אותם למיל, אחרת יש להעלות אותם ל – git. יש לרשום במיל שם מלא של התלמיד, שם הקורס, מספר הקורס ומס' טלפון שנייתן להציג את הסטודנט בעזרתו.