

# **NodeJS**

**Node JS API With  
MongoDB  
Bcrypt  
JWT**

**TomerBu**

# וידוא האם האימייל כבר קיים במערכת:

```
const usersRouter = require('express').Router();
const { User, validateUser } = require('../models/users');

usersRouter.post('/', async (req, res) => {
  const err = validateUser(req.body).error;

  if (err)
    return res.status(400).json({
      errors: err.details.map(d => d.message)
    })

  //before trying to save the user => check if user already exists
  let user = await User.findOne({ email: req.body.email });

  if (user)
    return res.status(400).json({ message: `${user.email} already exists` });

  user = new User(req.body);
  await user.save();
  res.json(user);
});

module.exports = usersRouter;
```

Joi => req.body contains ONLY 4 Valid fields

# אבטחת מידע - לא לשמור את הסיסמה בדטה-בייס! לשמור הצפנה של הסיסמה שאיתה ניתן יהיה להשוות

MongoDB Compass - lec7cluster.dlux9.mongodb.net/MongoApi.

lec7cluster.dlux9.mongodb.net

3 DBS 1 COLLECTIONS C

☆ FAVORITE

HOSTS  
lec7cluster-shard-00-00.dl...  
lec7cluster-shard-00-02.dl...  
lec7cluster-shard-00-01.dl...

CLUSTER  
Replica Set (atlas-t2nw9b-...  
3 Nodes

EDITION  
MongoDB 5.0.8 Enterprise

{ My Queries

Databases

Filter your data

MongoApi

users ...

admin

local

Documents MongoApi.users +

## MongoApi.users

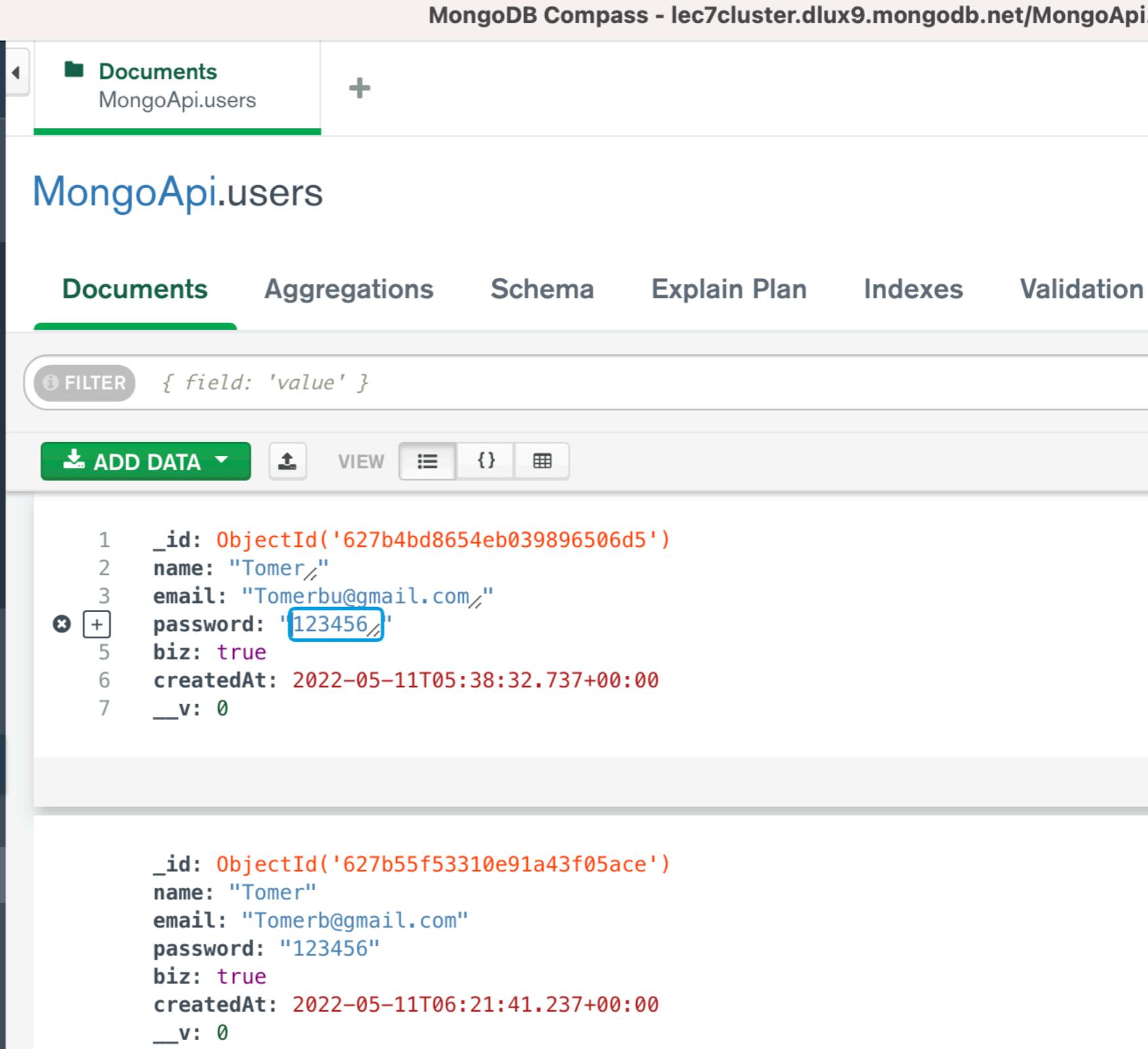
Documents Aggregations Schema Explain Plan Indexes Validation

i FILTER { field: 'value' }

ADD DATA VIEW

1 \_id: ObjectId('627b4bd8654eb039896506d5')  
2 name: "Tomer"  
3 email: "Tomerbu@gmail.com"  
4 password: '123456'  
5 biz: true  
6 createdAt: 2022-05-11T05:38:32.737+00:00  
7 \_\_v: 0

1 \_id: ObjectId('627b55f53310e91a43f05ace')  
2 name: "Tomer"  
3 email: "Tomerb@gmail.com"  
4 password: "123456"  
5 biz: true  
6 createdAt: 2022-05-11T06:21:41.237+00:00  
7 \_\_v: 0



# הצפנה הסיסמה: תהליך כבד

npm install bcrypt

```
const usersRouter = require('express').Router();
const { User, validateUser } = require('../models/users');
const bcrypt = require('bcrypt');

usersRouter.post('/', async (req, res) => {
    const err = validateUser(req.body).error;

    if (err)
        return res.status(400).json({
            errors: err.details.map(d => d.message)
        })

    let user = await User.findOne({ email: req.body.email });

    if (user)
        return res.status(400).json({ message: `${user.email} already exists` });

    user = new User(req.body);

    //before saving - hash the pass:
    user.password = await bcrypt.hash(req.body.password, 12);
    await user.save();
    res.json(user);
});

module.exports = usersRouter;
```

# תהליך לפענוח:

123456

\$2b\$12\$9rXvvNzI0K/yconEHL2990XG10faUns.CnqLP/19HoutU901ZVRnu

# תהליך הצפנה:

מייצרים מחרוזת רנדומלית: SALT

**.\$2b\$12\$0x9HWDyz21UanOEGVGS3C**

משתמשים ב-SALT כדי להצפין את הסיסמה  
ושומרים את ה-SALT הרנדומלי ביחד עם הסיסמה בדטה-ביס

**\$2b\$12\$0x9HWDyz21UanOEGVGS3C.qf7j5OxgD5u4z9XICYypaLKdlx1S4n2**

SALT

תוצר הצפנה: סיסמה + אלגוריתם + מספר סיבובים + SALT

```
//before saving - hash the pass:  
const salt = await bcrypt.genSalt(12);  
console.log(salt);  
user.password = await bcrypt.hash(req.body.password, salt);  
console.log(user.password);
```

# מינি-תרגיל:

**המשתמש מקליד סיסמה: 123456**

הצפינו את הסיסמה ב-12 סיבובי הצפנה:  
יש להשתמש ב-SALT הבא:

**\$2b\$12\$0x9HWDyz21UanOEGVGS3C.**

**\$2b\$12\$0x9HWDyz21UanOEGVGS3C.qf7j5OxgD5u4z9XICYypaLKdlx1S4n2**

**פתרו לתרגיל:**

```
bcrypt.hash('123456', '$2b$12$0x9HWDyz21UanOEGVGS3C.').then(r=> console.log(r))
```

**\$2b\$12\$0x9HWDyz21UanOEGVGS3C.qf7j5OxgD5u4z9XICYypaLKdlx1S4n2**

# המנעות מהחזרה של הסיסמה:

routes/users.js

```
const usersRouter = require('express').Router();
const { User, validateUser } = require('../models/users');
const bcrypt = require('bcrypt');
const _ = require('lodash');

usersRouter.post('/', async (req, res) => {
    const err = validateUser(req.body).error;

    if (err)
        return res.status(400).json({
            errors: err.details.map(d => d.message)
        })

    let user = await User.findOne({ email: req.body.email });

    if (user)
        return res.status(400).json({ message: `${user.email} already exists` });

    user = new User(req.body);

    user.password = await bcrypt.hash(req.body.password, 12);

    await user.save();
    res.status(201).json(_.pick(user, ['name', 'email', 'biz', '_id']));
});

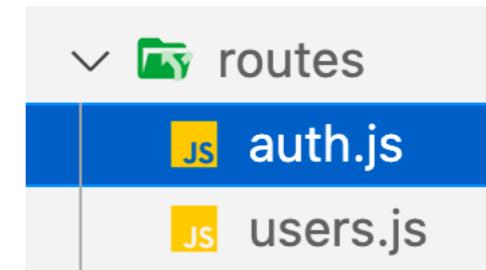
module.exports = usersRouter;
```

הMETHOD pick בוחרת שדות מאוביקט.

סטוס 201 - פירשו

# חדש עבור התחברות:

```
require('dotenv').config();
const express = require('express');
const morgan = require('morgan');
const connect = require('./utils/connect-db');
//import the router
const usersRouter = require('./routes/users');
const authRouter = require('./routes/auth');
```



```
//connect to the db:
connect()
  .then(() => console.log('connected'))
  .catch(e => console.log(e));

const app = express();
app.use(morgan('dev'));
app.use(express.urlencoded({ extended: true })); //middleware that parses body that contain
form data
app.use(express.json()); //middleware that parses body that contain json
```

```
//use the router:
app.use('/api/users', usersRouter);
app.use('/api/auth', authRouter);
```

```
const PORT = process.env.EXPRESS_PORT || 3000;
app.listen(PORT, () => console.log(`http://localhost:${PORT}`));
```

# חדש עבור התחברות: Router

```
const Joi = require('joi');
const bcrypt = require('bcrypt');
const { User } = require('../models/users');
const authRouter = require('express').Router();

const authSchema = Joi.object({
  email: Joi.string().email().max(64).required(),
  password: Joi.string().min(6).max(1024).required(),
});
const validateAuth = (body) => authSchema.validate(body);

//POST /api/auth
//BODY: {email, password}
authRouter.post('/', async (req, res) => {
  const { error } = validateAuth(req.body);
  if (error)
    return res.json({ message: error.details.map(d => d.message) });

  //find user by email:
  const user = await User.findOne({ email: req.body.email });

  if (!user)
    return res.status(400).json({ message: `invalid email or password` });

  //validate password:
  const isValid = await bcrypt.compare(req.body.password, user.password);

  if (!isValid)
    return res.status(400).json({ message: `invalid email or password` });

  res.json({ message: "ok" });
}

module.exports = authRouter;
```

אם מייל לא תקין או סיסמה לא תקינה  
נחזיר תשובה

מציאת משתמש לפי אימייל - כל  
האובייקט של המשתמש חזר  
מהדעתהביביס כולל סיסמא

בדיקת סיסמא שומרה מול מה  
שהמשתמש הzin

אם הסיסמא שגויה - תשובה  
עוממה ללקוח

תשובה חיובית

# בדיקה שהroute עובד

```
### test auth:
```

```
POST http://localhost:3002/api/auth
```

```
Content-Type: application/json
```

```
{  
  "email": "Tomerbu@gmail.com",  
  "password": "123456"  
}
```

# זיהוי באמצעות JWT

כשעושים מנוי לAPI מקבלים  
כך הסיסמה שלנו לא מוחנת בכל בקשה.

מנוי לבריכה:

מקבלים צ'יפ מגנטי ואיתו מזדהים.

באופן דומה לאחר התחברות ראשונה:  
ניתן לשולח ללקוח JWT.

בדומה לתעודת זהות:  
כל אחד יכול לקרוא שם את השם  
רק משרד הפנים  
יכול להנפיק את  
התעודה.  
מנפיקים JWT  
בעזרת SALT  
שמור רק בשרת

The screenshot shows the jwt.io interface. On the left, under 'Encoded', there is a large string of encoded JWT tokens. On the right, under 'Decoded', the token is broken down into its components:

- HEADER:** ALGORITHM & TOKEN TYPE
- {  
  "alg": "HS256",  
  "typ": "JWT"  
}
- PAYOUT:** DATA
- {  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}

Below the decoded payload, there is a section titled 'VERIFY SIGNATURE' containing the following code:

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```

payload:  
כל אחד יכול לקרוא  
(לא נשמר סיסמה או מידע רגיש)

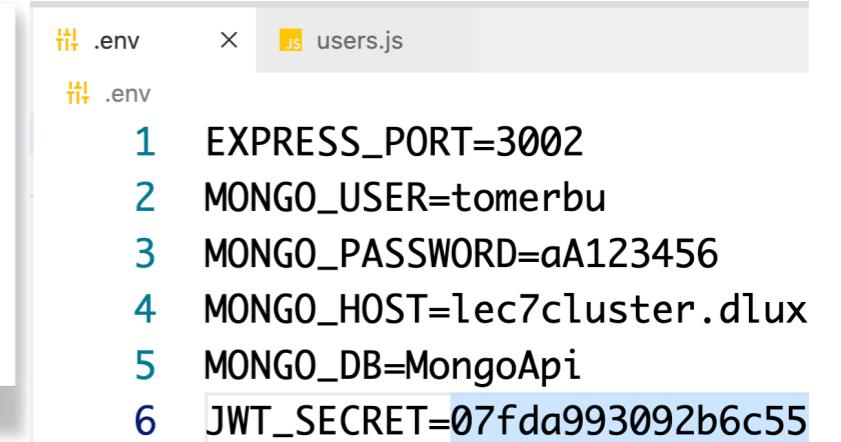
salt:  
saved on the server

# הוספה משתנה לenv

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
(base) Tomers-iMac-6:Lec8 tomerbuzaglo$ node
Welcome to Node.js v16.15.0.
Type ".help" for more information.
> require('crypto').randomBytes(64).toString('hex')
'd9918da7db6303a9805bd6d742aa9f70511f25dacc10b4c64e06d9a0533e6913b68fab997f9a60e9e3bab03a75b0ac'
```

```
node א נט ר
require('crypto').randomBytes(64).toString('hex')
א נט ר
```



The screenshot shows a code editor window with two tabs: '.env' and 'users.js'. The '.env' tab is active, displaying the following content:

```
1 EXPRESS_PORT=3002
2 MONGO_USER=tomerbu
3 MONGO_PASSWORD=aA123456
4 MONGO_HOST=lec7cluster.dlux
5 MONGO_DB=MongoApi
6 JWT_SECRET=07fda993092b6c55
```

## התקנת ספרייה:

```
npm install jsonwebtoken
```

# הוספה מתודה למחילה User

models/users.js

```
const Joi = require('joi');
const mongoose = require('mongoose');
const jwt = require('jsonwebtoken');

//what to check with Mongoose, rules:
const userSchema = new mongoose.Schema({...});

//add methods to User schema: (not an arrow function, since this is referred)
userSchema.methods.generateToken = function () {
    const token = jwt.sign(
        { _id: this._id },
        process.env.JWT_SECRET
    );

    return token;
}

...

```

# הזרת Login מוצלח:

```
const Joi = require('joi');
const bcrypt = require('bcrypt');
const { User } = require('../models/users');
const authRouter = require('express').Router();

const authSchema = Joi.object({
  email: Joi.string().email().max(64).required(),
  password: Joi.string().min(6).max(1024).required(),
});
const validateAuth = (body) => authSchema.validate(body);

authRouter.post('/', async (req, res) => {
  const { error } = validateAuth(req.body);
  if (error)
    return res.json({ message: error.details.map(d => d.message) });

  //find user by email:
  const user = await User.findOne({ email: req.body.email });

  if (!user)
    return res.status(400).json({ message: `invalid email or password` });

  //validate password:
  const isValid = await bcrypt.compare(req.body.password, user.password);

  if (!isValid)
    return res.status(400).json({ message: `invalid email or password` });

  res.json({ token: user.generateToken() });
});

module.exports = authRouter;
```

routes/auth.js

# Middleware JWT לבדיקת JWT אפשר למשתמש להזדהות באמצעות JWT

בדיקות אישור כניסה (-)

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

module.exports = (req, res, next) => {
  const token = req.query.token || req.header('x-auth-token');

  if (!token)
    return res.status(401).json({ message: 'Access denied no token provided.' });

  //No callback, no promise, how to check for failure:
  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET);
    req.user = payload;
    next();
  }
  catch (e) {
    return res.status(401).json({ message: 'Invalid token.' });
  }
}
```

### test bizNumber:

Send Request

POST <http://localhost:3002/api/cards>

x-auth-token: eyJhbGciOiJIUzI1NiIsInR5cCI6Ikpj

# התמונה הגדולה:



אחרי התחברות  
שלוחים ללקוח JWT

כשלקו מנסה לגשת למשאב מסווג:

הלקוח צריך לשולח JWT

רקלקוח רשום יכול להשתמש בAPI

/api/cards

# עבודה מול API

[https://api.themoviedb.org/3/discover/movie?api\\_key=b3b1492d3e91e9f9403a2989f3031b0c](https://api.themoviedb.org/3/discover/movie?api_key=b3b1492d3e91e9f9403a2989f3031b0c)

# שימוש בMiddleware

routes/users.js

```
const usersRouter = require('express').Router();
const { User, validateUser } = require('../models/users');
const bcrypt = require('bcrypt');
const _ = require('lodash');
const auth = require('../middleware/auth');

usersRouter.get('/me', auth, (req, res) => {
  res.json(req.user);
});

//...
```

# עוד בדיקה של middleware User My Details

```
const usersRouter = require('express').Router();
const { User, validateUser } = require('../models/users');
const bcrypt = require('bcrypt');
const _ = require('lodash');
const auth = require('../middleware/auth');

usersRouter.get('/me', [auth], [async] (req, res) => {
    //req.user._id
    const user = await User.findById(req.user._id).select('-password');
    res.json(user); //security problem...
});
```

# אוסף חדש - כרטיסי עסק: cards

```
use MongoApi  
db.cards.createIndex({"bizNumber": 1}, {unique: true})
```

יצרנו collection חדש  
בשם cards

The screenshot shows the MongoDB Compass interface. On the left, the sidebar displays the connection details: 'lec7cluster.dlux9.mongodb.net', 'HOSTS' (with three hosts listed), 'CLUSTER' (Replica Set with 3 nodes), and 'EDITION' (MongoDB 5.0.8 Enterprise). Below these are sections for 'My Queries', 'Databases', and a search bar. A dropdown menu for 'MongoApi' is open, showing the command history. The main area is titled 'Collections' and lists two collections: 'cards' and 'users'. The 'cards' collection has a storage size of 20.48 kB. The command history at the bottom shows the execution of the indexing command.

```
>_MONGOSH  
> use MongoApi  
< 'switched to db MongoApi'  
> db.cards.createIndex({"bizNumber": 1}, {unique: true})  
< 'bizNumber_1'  
Atlas atlas-t2nw9b-shard-0 [primary] MongoApi>
```

עם אינדקס unique  
על התכונה bizNumber

# אוסף חדש - כרטיסי עסק: cards

```
const Joi = require("joi");
const mongoose = require("mongoose");
const _ = require('lodash');

const cardSchema = new mongoose.Schema({
  bizName: {
    type: String,
    required: true,
    minlength: 2,
    maxlength: 255
  },
  bizDescription: {
    type: String,
    required: true,
    minlength: 2,
    maxlength: 1024
  },
  bizAddress: {
    type: String,
    required: true,
    minlength: 2,
    maxlength: 500
  },
  bizPhone: {
    type: String,
    required: true,
    minlength: 9,
    maxlength: 11
  },
  bizImage: {
    type: String,
    required: true,
    minlength: 11,
    maxlength: 1024
  },
  bizNumber: {
    type: String,
    required: true,
    minlength: 3,
    maxlength: 9_999_999_999
  },
  user_id: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  }
});
```

שיםו לב לא לכלול תמונה ומספר עסק  
נוסיף אותם מהקוד בהמשך.

```
const Card = mongoose.model('Card', cardSchema);

const cardJoiSchema = Joi.object({
  bizName: Joi.string().min(2).max(255).required(),
  bizDescription: Joi.string().min(2).max(1024).required(),
  bizAddress: Joi.string().min(2).max(500).required(),
  bizPhone: Joi.string().min(9).max(11).required().regex(/^0[2-9][-]?[0-9]{7,9}$|  
^05[0-9][-]?[0-9]{7,9}$|^07[7,8][0-9]{7,9}$/)
});

const generateBizNumber = async () => {
  while (true) {
    let randomNumber = _.random(100, 9_999_999_999);
    let card = await Card.findOne({ bizNumber: randomNumber });
    if (!card)
      return String(randomNumber); //`${randomNumber}`
  }
} //uuid

const validateCard = (card) => cardJoiSchema.validate(card);

module.exports = {
  Card,
  validateCard,
  generateBizNumber
}
```

# כתב חדש לכרטיסי עסק:

routes/cards.js

```
const cardsRouter = require('express').Router();
const _ = require('lodash');
const auth = require('../middleware/auth');
const { Card, validateCard, generateBizNumber } = require('../models/cards');

cardsRouter.post('/', auth, async (req, res) => {
  const bizNumber = await generateBizNumber()
  res.json({ bizNumber: bizNumber });
})

module.exports = cardsRouter;
```

# שימוש בנתב:

```
require('dotenv').config();
const express = require('express');
const morgan = require('morgan');
const connect = require('./utils/connect-db');
//import the router
const usersRouter = require('./routes/users');
const authRouter = require('./routes/auth');
const cardsRouter = require('./routes/cards');

//connect to the db:
connect()
  .then(() => console.log('connected'))
  .catch(e => console.log(e));

const app = express();
app.use(morgan('dev'));
app.use(express.urlencoded({ extended: true })); //middleware that parses body that contain form data
app.use(express.json()); //middleware that parses body that contain json

//use the router:
app.use('/api/users', usersRouter);
app.use('/api/auth', authRouter);
app.use('/api/cards', cardsRouter);

const PORT = process.env.EXPRESS_PORT || 3000;
app.listen(PORT, () => console.log(`http://localhost:${PORT}`));
```

# הוספה כרטיס עסק חדש:

```
const cardsRouter = require('express').Router();
const _ = require('lodash');
const auth = require('../middleware/auth');
const { Card, validateCard, generateBizNumber } = require('../models/cards');

cardsRouter.post('/', auth, async (req, res) => {

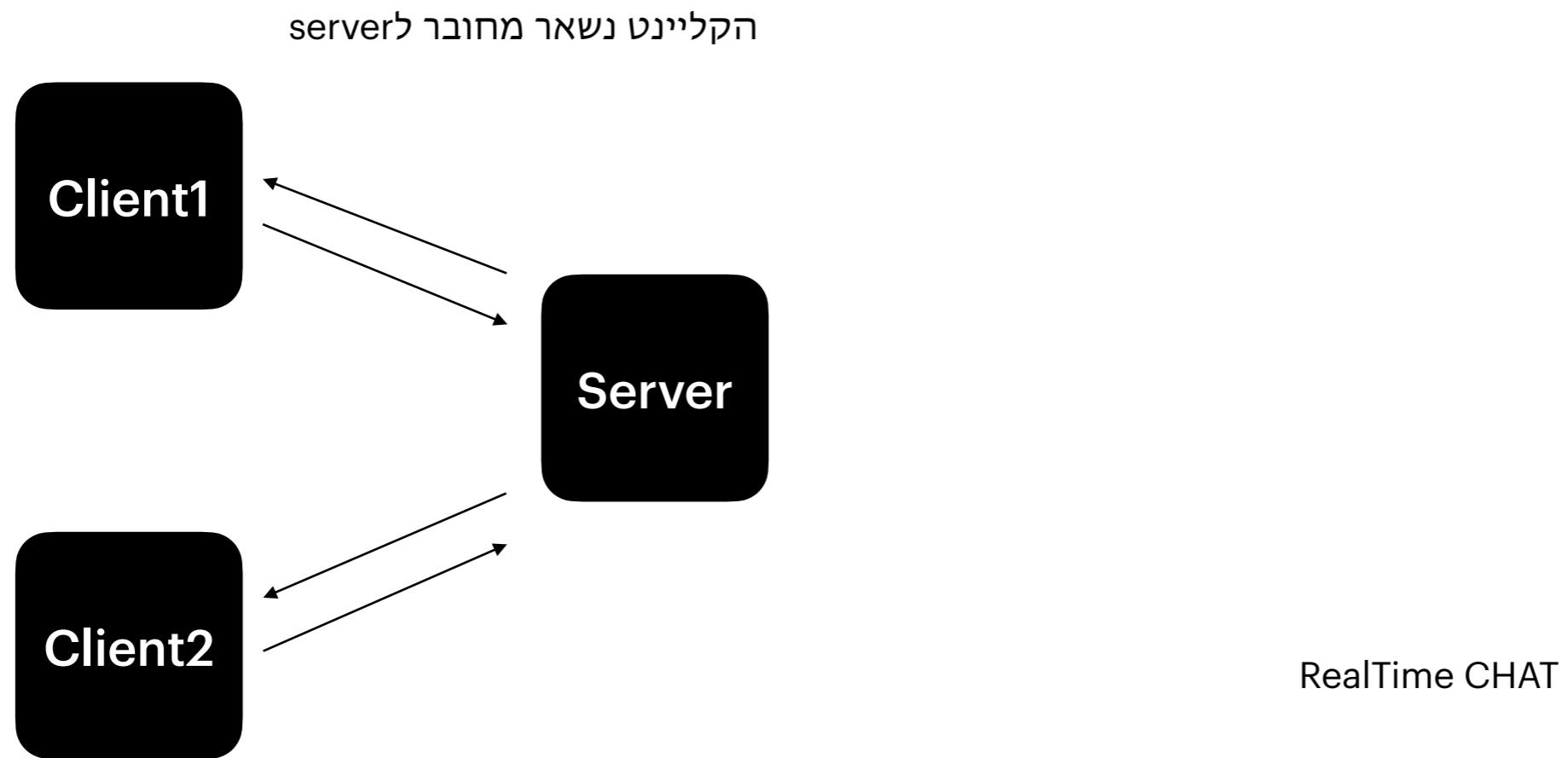
  const { error} = validateCard(req.body);
  if (error)
    return res.status(400).json({ message: error.details.map(d => d.message) });

  let card = new Card({
    bizNumber: await generateBizNumber(),
    user_id: req.user._id,
    ...req.body
  });

  card.bizImage = card.bizImage ?? 'https://cdn.pixabay.com/photo/2015/10/05/22/37/blank-profile-picture-973460_960_720.png'
  card = await card.save();
  res.json(card);
}

module.exports = cardsRouter;
```

# Intro to Web Sockets



הפרוטוקול מיועד בדיקת נושא המסויים זהה.  
ולכן מיעיל את השירות החיבור פתוח.

מה היי עושים קודם?  
הקלינט פעם בחצי דקה שולח request לשרת...  
...לשרות...

Server Sent Events

# אפליקציה מבוססת express

```
const express = require('express');
```

```
const app = express();
app.use(express.static('public'));
app.get('/home', (req, res) => {
  res.redirect('/index.html')
});
```

```
//regular http server
const server = require('http').createServer(app);
server.listen(3000);
```

```
npm install express socket.io
```

הספריה socket io  
דורשת לעבוד מול http server "רגיל"

כך המרנו את אפליקציית express  
לאפליקציה http רגילה.

# Intro to Web Sockets

## קוד צד שרת:

```
const express = require('express');
const app = express();
const server = require('http').createServer(app);

//io works on top of our http server
const io = require('socket.io')(server);

// runs when a client is connected:
io.sockets.on('connection', (client) => {
  client.on('input', (ms) => {
    io.sockets.emit('new', ms);
  })
})

app.use(express.static('public'));
app.get('/home', (req, res) => {
  res.redirect('/index.html')
});

server.listen(3000);
```

פעולה שרצה ברגע של קווח חדש מתחבר:

ברגע שהלקוח שולח אירוע של "input":

השם input הוא שרירוטי.

נשלח לכל הלקוחות  
אירוע חדש "new"

# Intro To Web Sockets

```
<!DOCTYPE html>
<html>

<head>
    <title>Live chat simple example</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>
    <h1>Live chat example with socket.io</h1>
    <textarea placeholder="Write your message" id="message" cols="50" rows="10"></textarea>
    <br><br>
    <input type="button" value="Send message" id="send-btn"><br><br>
    <div id="ms-holder"></div>

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
    <script src="/socket.io/socket.io.js"></script>
    <!-- http://localhost:3000/socket.io/socket.io.js -->
    <script>

        var socket = io.connect();           ייבוא של הסקריפט מהשרת שלנו

        $('#send-btn').on('click', () => {   בלחיצה על כפתור
            socket.emit('input', $('#message').val());  נשלח אירוע לשרת עם
            $('#message').val('');                      התוכן של input
        });

        socket.on('new', function (ms) {      ונרוקן את input
            $('#ms-holder').append(ms + '<br>');
        });

    </script>
</body>

</html>
```

יצירת חיבור socket לשרת שלנו

בלחיצה על כפתור נשלח אירוע לשרת עם התוכן של input ונרוקן את input

ברגע שמגיע מידע מהשרת - נוסיף אותו לdiv

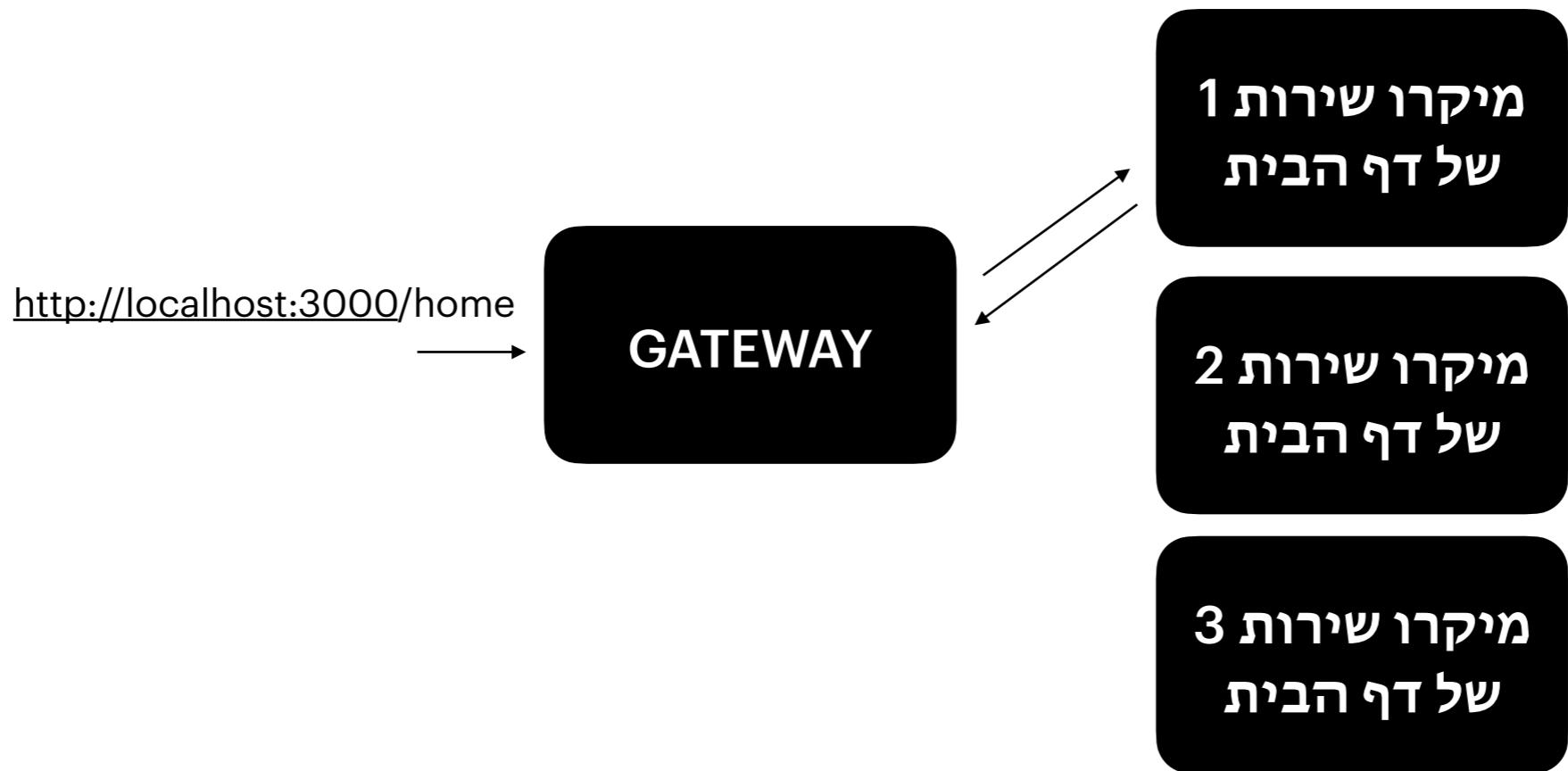
# mongoo עם אטלס - יודע לשגר אירועים ברגע שימושה המידי בטבלה מסוימת.

The screenshot shows a developer's workspace with several open tabs and panels:

- 1** Top-left: A code editor tab for `app.js` containing Node.js code. A callout highlights the line: `require("./models/user").User.watch().on("change", console.log);`
- 2** Top-right: A panel showing an `examples.http` file with a POST request to `/api/users`. The request body is a JSON object with fields: `"name": "daniel", "email": "a@cca.com", "password": "123456", "biz": true`. Below it, another POST request is shown to `/api/auth`.
- 3** Bottom: A terminal window showing the results of a `node` command. It displays the user document created in step 1, followed by a success message from the API, and a welcome message from Node.js v12.18.1.

3  
מופעל callback כי היה  
שינויים(users collection)

# מבוא למיקרו-שירותים



load balancing

# מבוא למיקרו-שירותים

אפליקציות קטנות למשל טיפול בתמונה.

שרת אחד לכיווץ תמונות:

שרת ליציאת:

תחזוקה: מיקרו-שירות אחד נפל ולא כל האתר.

תחזוקה: מפתחים ג'וניורים יכולים להשתלב בפרויקט קטן של מיקרו-סרביס מסוים.

תחזוקה: מפתחים מתעניינים בפרויקט קטן של מיקרו-סרביס מסוים ושולטים בו הרבה יותר.

אפשר לשלב שפות תכנתות.

אחד המיקרו-שירותים נכתב בJAVA

אחד המיקרו-שירותים נכתב בC++

אחד המיקרו-שירותים נכתב בPHP

אחד המיקרו-שירותים נכתב NODE