



ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Backend Node.js & MongoDB

Μ. Καραμπάτσης



CRUD app με Node.js και MongoDB

Δημιουργία νέας εφαρμογής: δημιουργούμε φάκελο productsApp και πληκτρολογούμε `npm init`. Στη συνέχεια εγκαθιστούμε τις βιβλιοθήκες.

Εγκατάσταση βιβλιοθηκών

1. **express:** `npm install express`

Framework του Node.js που παρέχει ένα σύνολο χαρακτηριστικών/δυνατοτήτων για την υλοποίηση web και mobile εφαρμογών. Με το Express μπορούμε να ορίσουμε **ξεχωριστές ενότητες** που έχουν **διαφορετικές ευθύνες** ή να διαχειριστούμε **διαφορετικές κλήσεις** (requests)

2. **nodemon:** `npm install nodemon`

Εργαλείο που επιτρέπει σε μια Node.js εφαρμογή να επανεκκινεί αυτόματα κάθε φορά που γίνεται αλλαγή σε κάποιο από τα αρχεία της εφαρμογής



CRUD app με Node.js και MongoDB

Εγκατάσταση βιβλιοθηκών

3. **dotenv:** `npm install dotenv`

Φορτώνει μεταβλητές περιβάλλοντος από ένα αρχείο `.env` στο `process.env`.

4. **mongoose:** `npm install mongoose`

Το Mongoose είναι ένα MongoDB object modeling tool. Μας παρέχει τη δυνατότητα να σχήματα (models) για να περιγράψουμε τα δεδομένα μας. Περιέχει επίσης δυνατότητες δημιουργίας ερωτημάτων στη MongoDB, validations, δηλώσεις τύπων κ.α

5. **mongoose-unique-validator:** `npm install mongoose-unique-validator`

Προσθέτει ελέγχους ορθότητας στα πεδία ενός mongoose σχήματος έτσι ώστε τα δηλωμένα πεδία να είναι μοναδικά.



CRUD app με Node.js και MongoDB

Εγκατάσταση βιβλιοθηκών

6. **swagger-ui-express:** `npm install swagger-ui-express`

Μας επιτρέπει να web εφαρμογή που περιέχει την τεκμηρίωση του api που έχουμε υλοποιήσει.

7. **mongoose-to-swagger:** `npm install mongoose-to-swagger`

Μετατρέπει τα μοντέλα που έχουν υλοποιηθεί με το Mongoose σε json έτσι ώστε να μπορούν να χρησιμοποιηθούν στο swagger

8. **winston:** `npm install winston`

Πακέτο που μας επιτρέπει να κάνουμε διαχείριση των logs του api. Εμφανίζοντας τα είτε στο console ή αποθηκευοντας τα .



CRUD app με Node.js και MongoDB

Εγκατάσταση βιβλιοθηκών

9. **winston-daily-rotate-file:** `npm install winston-daily-rotate-file`

Παρέχει τη δυνατότητα στο winston να αποθηκεύει τα logs σε αρχεία κάνοντας τα και rotate αν επιθυμούμε.

10. **winston-mongodb:** `npm install winston-mongodb`

Παρέχει τη δυνατότητα στο winston να αποθηκεύει τα logs στη MongoDB.

11. **cors:** `npm install cors`

Επιτρέπει την ρύθμιση των CORS (Cross-Origin Resource Sharing - CORS) παρέχοντας έτσι τη δυνατότητα σε άλλες εφαρμογές να υποβάλουν αιτήσεις, όπως XMLHttpRequest, στο api της εφαρμογής μας.



CRUD app με Node.js και MongoDB

package.json:

```
{
  "name": "productsapp",
  "version": "1.0.0",
  "description": "User and products app",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "test": "node app.js",
    "dev": "nodemon app.js"
  },
  "keywords": [ "products", "users" ],
  ...
}
```

```
"dependencies": {
  "cors": "^2.8.5",
  "dotenv": "^16.3.1",
  "express": "^4.18.2",
  "mongoose": "^7.5.0",
  "mongoose-to-swagger": "^1.4.0",
  "mongoose-unique-validator": "^4.0.0",
  "nodemon": "^3.0.1",
  "swagger-ui-express": "^4.6.1",
  "winston": "^3.8.2",
  "winston-daily-rotate-file": "^4.7.1",
  "winston-mongodb": "^5.1.1"
}
```

Για να τρέξουν οι εντολές του πεδίου scripts -> `npm run όνομα-εντολής` : πχ **npm run dev**



MVC (model view controller)

Στο φάκελο productsApp δημιουργούμε τους υποφακέλους:

1. **controllers**: περιέχει όλους τους controllers της εφαρμογής. Δηλαδή τις διαδικασίες που θα διαβάζουν, γράφουν, τροποποιούν και διαγράφουν documents από τη βάση μας. Τη λογική δηλαδή με την οποία η εφαρμογή χειρίζεται τα εισερχόμενα αιτήματα και τις εξερχόμενες απαντήσεις.
2. **models**: περιέχει όλα μοντέλα της εφαρμογής μας, δηλαδή users και products με τα πεδία τους, τους τύπους των πεδίων, τα validations κλπ.
3. **routes**: εφαρμογή που διαχειρίζεται τα requests (get, post, patch, delete) και ανάλογα τα requests καλεί τους αντίστοιχους controllers.
4. **logger**: περιέχει τις ρυθμίσεις για καταγραφή και αποθήκευση των logs.



environment variables

Environment variables χρησιμοποιούνται για την αποθήκευση ευαίσθητων δεδομένων όπως κωδικούς πρόσβασης και άλλες πληροφορίες που δεν πρέπει να γράφονται απευθείας σε κώδικα.

Αποθηκευόνται σε αρχείο με ονομασία **.env**

Στο Node.js για να διαβάσουμε τις μεταβλητές περιβάλλοντος χρησιμοποιούμε το πακέτο **dotenv** και τη διαδικασία **process.env** για το διάβασμα των μεταβλητών

.env αρχείο:

```
MONGODB_URI =
```

```
"mongodb+srv://username:password@Mycluster.mongod  
db.net/codingfactory?
```

```
retryWrites=true&w=majority"
```

Διάβασμα του **MONGODB_URI** στο node.js

```
require("dotenv").config();  
  
mongoose.connect(  
  process.env.MONGODB_URI  
)
```




app.js (1)

Αρχείο εκκίνησης της εφαρμογής, καλεί όλες τις απαραίτητες βιβλιοθήκες, κάνει τη σύνδεση με την βάση και ξεκινά το server.

Σύνδεση με τη βάση

```
const mongoose = require("mongoose");

mongoose.connect(process.env.MONGODB_URI)
  .then(
    () => { console.log("Connection to MongoDB established") },
    err => { console.log('Failed to connect to MongoDB', err) }
  );

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

1. Καλούμε τη βιβλιοθήκη mongoose
2. Σύνδεση με τη βάση με connection string αυτό που ορίζεται στο process.env.MONGODB_URI
3. Εκκίνηση του Server



app.js (2)

Requests και Middleware συναρτήσεις

```
1 const user = require("./routes/user.route");  
2 const product = require("./routes/product.route");  
3 const userProduct = require("./routes/user.product.route");  
  
app.use('/api/users', user);  
app.use('/api/products', product);  
app.use('/api/user-products', userProduct);
```

Οι ενδιάμεσες συναρτήσεις επιτρέπουν:

1. το τρέξιμο οποιουδήποτε κώδικα
2. την πραγματοποίηση αλλαγών σε request και response αντικείμενα.
3. το τερματισμό του request-response κύκλου
4. να καλέσουν την επόμενη ενδιάμεση συνάρτηση

1. Για requests `/user-products` καλούμε από το φάκελο routes το αρχείο `user.product.route`
2. Για requests `/users` καλούμε από το φάκελο routes το αρχείο `user.route`
3. Για requests `/products` καλούμε από το φάκελο routes το αρχείο `product.route`



Routing

Στο app.js αρχείο καλούμε τις ενδιαμέσες συναρτήσεις που αντιστοιχούν στα path της εφαρμογής μας.

Για παράδειγμα η κλήση `app.use('/user', user);` αντιστοιχεί στο path `/user` και καλεί την ενδιάμεση συνάρτηση `user` από το αρχείο `user.route.js`

app.js

```
const user = require('./routes/user.route');
const product = require('./routes/product.route');
const userProduct = require('./routes/user.product.route');

app.use('/api/users', user);
app.use('/api/products', product);
app.use('/api/user-products', userProduct);
```

user.route.js

```
const express = require('express');
const router = express.Router();

const userController = require('../controllers/user.controller');

router.get('/', userController.findAll);
router.get('/:username', userController.findOne);
router.post('/', userController.create);
router.patch('/:username', userController.update);
router.delete('/:username', userController.delete);

module.exports = router;
```



Controller

Στα αρχεία `όνομα.route.js` ανάλογα με το http request που έχουμε καλούμε τον αντίστοιχο controller. Για παράδειγμα η κλήση `router.get('/', userController.findAll);`

καλεί τον controller `userController.findAll`

Οι controller περιέχουν τις διαδικασίες που διαβάζουν, γράφουν, τροποποιούν και διαγράφουν documents από τη βάση μας. Τη λογική δηλαδή με την οποία η εφαρμογή χειρίζεται τα εισερχόμενα αιτήματα και τις εξερχόμενες απαντήσεις

user.route.js

```
router.get('/', userController.findAll);
```

user.controller.js

```
exports.findAll = async (req, res) => {  
  console.log("Find All system users");  
  try {  
    const result = await User.find();  
    res.status(200).json({ status: true, data: result });  
    console.log('Success in reading all users');  
  } catch (err) {  
    res.status(400).json({ status: false, data: err });  
    console.log(`Problem in reading users: ${err}`)  
  }  
};
```



Model

Στους controller καλούμε τα μοντέλα που πρόκειται να επεξεργαστούμε. Για παράδειγμα

```
const User = require('../models/user.model');
```

Τα μοντέλα περιέχουν το σχήμα των documents με τα πεδία τους, τους τύπους των πεδίων, τα validations κλπ.

user.model.js

```
let userSchema = new Schema({
  username: {
    type: String,
    required: [ true, 'Username is required field' ],
    max: 100,
    unique: true,
    trim: true,
    lowercase: true,
  },
  password: { type: String, required: [ true, 'Password is required field' ], max: 100 },
  name: { type: String, required: [ true, 'Name is required field' ], max: 100 },
  surname: { type: String, required: true, max: 100 },
  email: {
    type: String,
    required: [ true, 'Email is required field' ],
    max: 100,
    unique: true,
    trim: true,
    lowercase: true,
    match: [
      /^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*\\.\\w{2,3}+$/,
      "Email address is not valid",
    ],
  },
  address: addressSchema,
  phone: { type: [ phoneSchema ], null: true },
  products: { type: [ productSchema ], null: true },
}, { collection: 'users', timestamps: true });
```



Swagger

Για να δημιουργήσουμε την ιστοσελίδα της τεκμηρίωσης του API, δημιουργούμε το αρχείο **swagger.js** στο root της εφαρμογής και κάνουμε τις σχετικές ρυθμίσεις στο **app.js** αρχείο.

app.js

```
1 const swaggerUi = require('swagger-ui-express');
2 const swaggerDocument = require('./swagger');

3 app.use(
4   '/api-docs',
5   swaggerUi.serve,
6   swaggerUi.setup(swaggerDocument.options)
7 );
```

1. Καλούμε την βιβλιοθήκη `swagger-ui-express`
2. Καλούμε το αρχείο `swagger.js` που δημιουργήσαμε και συγκρίμμένα τα `export options`
3. Η ενδιάμεση συνάρτηση που ανοίγει την σελίδα τεκμηρίωσης καλώντας το `swaggerUI` με τα απαραίτητα `options`



swagger.js (1)

Περιέχει τα μοντέλα της εφαρμογής από το φάκελο **models** έτσι ώστε αυτά να είναι εμφανή στους χρήστες που επισκέφτονται τη σελίδα.

Στη συνέχεια δημιουργούμε τα options με όλες τις βασικές πληροφορίες της τεκμηρίωσης καθώς και αναλυτικές πληροφορίες όλων των requests και τα response αυτών.

Η βιβλιοθήκη `mongoose-to-swagger` μετατρέπει τα models σε json

swagger.js

```
const m2s = require('mongoose-to-swagger');
const User = require('./models/user.model');
const Product = require('./models/product.model');

exports.options = {
  "components": {
    "schemas": {
      User: m2s(User),
      Product: m2s(Product),
    }
  },
  "openapi": "3.0.1",
  "info": {
    "version": "1.0.0",
    "title": "Products CRUD API",
    "description": "Products Project Application API",
    "license": {
      "name": "MIT",
      "url": "https://opensource.org/licenses/MIT"
    }
  }
}
```



swagger.js (2)

1. Πεδίο components: περιέχει τα μοντέλα της εφαρμογής
2. Πεδίο openapi: την έκδοση του openapi
3. Πεδίο info: γενικές πληροφορίες για την εφαρμογή
4. Πεδίο servers: τους servers στους οποίους μπορεί ο χρήστης να κάνει τα test
5. Πεδίο tags: Χρησιμοποιείται για την ομαδοποίηση των path

```
"servers": [  
  {  
    "url": "http://localhost:3000/",  
    "description": "Local server"  
  },  
  {  
    "url": "https://api_url_testing",  
    "description": "Testing server"  
  },  
],  
"tags": [  
  {  
    "name": "Users",  
    "description": "API for users in the system"  
  },  
  {  
    "name": "Products",  
    "description": "API for Products in the system"  
  },  
  {  
    "name": "Users and Products",  
    "description": "API for users in the system and their products"  
  },  
]
```




swagger.js (3)

1. Πεδίο paths: περιέχει τα requests του api
2. Κάθε path αρχικά περιέχει το path του request
3. Τον τύπο του request πχ get, post κλπ.
4. Το tag στο οποίο ανήκει
5. Μια περιγραφή του request και στη συνέχεια τις παραμέτρους της κλήσεις αν είναι με τα πεδία τους
6. Τέλος, το response της κλήσεις

```
"paths":{
  "/api/users": {
    "get": {
      "tags": [
        "Users"
      ],
      "summary": "Get all users in system",
      "responses": {
        "200": {
          "description": "OK",
          "schema": {
            "$ref": "#/components/schemas/User"
          }
        }
      }
    }
  },
},
```



```
"phone":{
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "type": { "type": "string" },
            "number": { "type": "number" }
        }
    }
},
"required": ["username", "password", "name", "surname", "email"]
},
},
},
"responses": {
    "200": {
        "description": "New user is created",
    }
}
}
```



winston (1)

Για την καταγραφή των logs στο φάκελο logger δημιουργούμε το αρχείο **logger.js** και το οποίο καλούμε μέσα από τους controllers μας

Απαραίτητες βιβλιοθήκες: **logger.js**

```
const { format, createLogger, transports } = require("winston");
require("winston-daily-rotate-file");
require('winston-mongodb');
require("dotenv").config();
```

winston : για τη δημιουργία του logger

winston-daily-rotate-file : για καταγραφή

των logs σε αρχείο, ρυθμιση για rotate αυτού

winston-mongodb : για καταγραφή των logs στη
mongoDB

```
1 const { combine, timestamp, label, printf, prettyPrint } = format;

2 const fileRotateTransport = new transports.DailyRotateFile({
  filename: "logs/rotate-%DATE%.log",
  datePattern: "DD-MM-YYYY",
  maxFiles: "14d",
});
```

1. Απαραίτητες μέθοδοι για εμφάνιση των στοιχείων στα logs
2. Ρυθμίσης αρχείου log, δήλωση ονόματος αρχείου και ανά πόσες μέρες γίνεται rotate



winston (2)

Ένας logger δημιουργείτε με τη μέθοδο

`createLogger` .

Περιλαμβάνει:

- Το επίπεδο του logging
- Το format των στοιχείων που θα καταγράφονται
- Τον transporter, πίνακας που δηλώνει όλους τα μέσα στα οποία θα γίνεται η καταγραφή των στοιχείων

```
const logger = createLogger({  
  level: "debug",  
  format: combine(  
    label({ label: CATEGORY }),  
    timestamp({  
      format: "DD-MM-YYYY HH:mm:ss",  
    }),  
    // format.json()  
    prettyPrint()  
  ),  
  transports: [],  
});
```



winston (3)

1. Η πρώτη επιλογή δηλώνει ότι η καταγραφή γίνεται σε αρχείο που κάνει rotation
2. Η δεύτερη σε αρχείο `error.log` στα οποία καταγραφονται μόνο επιπέδου error στοιχεία.
3. Η τρίτη καταγραφή όλων των στοιχείων στο Console
4. Η τελευταία καταγραφή στη mongoDB για στοιχεία επιπέδου error στη collection `server_logs`

```
transports: [  
  fileRotateTransport,  
  new transports.File({  
    filename: "logs/example.log",  
  }),  
  new transports.File({  
    level: "error",  
    filename: "logs/error.log",  
  }),  
  new transports.Console(),  
  new transports.MongoDB({  
    level: 'error',  
    //mongo database connection link  
    db : process.env.MONGODB_URI,  
    options: {  
      useUnifiedTopology: true  
    },  
    // A collection to save json formatted logs  
    collection: 'server_logs',  
    format: format.combine(  
      format.timestamp(),  
      // Convert logs to a json format  
      format.json()  
    )  
  })  
],
```



winston (4)

Η καταγραφή των logs στους διάφορους transporters γίνεται στους controller της εφαρμογής.

1. Αρχικά καλούμε το απαραίτητο αρχείο με τον logger.
2. Τα logger.info, logger.warn, logger.error, logger.debug δηλώνουν το επιπεδο του log.
3. Για παράδειγμα τα logs που δηλώνουμε με logger.error θα αποθηκευτούν στους transporter της mongoDB και στο αρχείο error.log

```
1 const logger = require("../logger/logger");  
  
2 logger.info("Success in reading all users");  
3 logger.warn("Success in reading all users");  
4 logger.error("Success in reading all users");  
5 logger.log("debug", "Success in reading all users");  
6 logger.debug("Success in reading all users");
```



CORS

Στο αρχείο `app.js` μπορούμε να ρυθμίσουμε την εφαρμογή μας έτσι ώστε να δέχεται κλήσεις από παντού ή από συγκεκριμένους τομείς

`app.js`

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

app.use(cors({
  origin: '*'
  // origin: ['https://www.section.io', 'https://www.google.com/']
}));
```

Η ρύθμιση `origin: '*'` επιτρέπει requests από παντού.

Η ρύθμιση `origin: ['https://www.section.io', 'https://www.google.com/']` επιτρέπει κλήσεις μόνο από τις σελίδες `www.section.io` και `www.google.com`

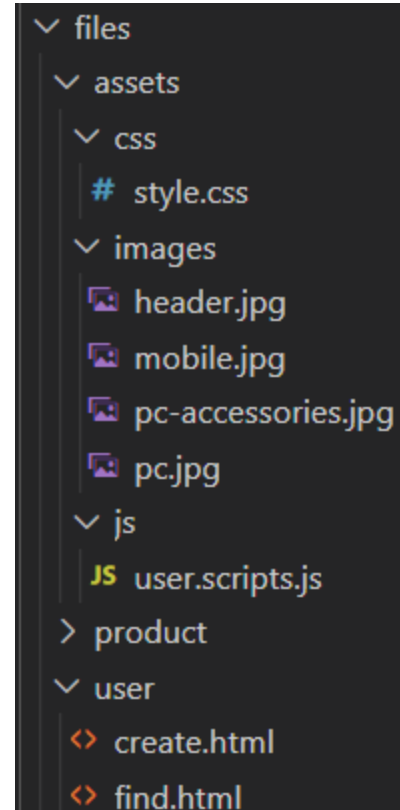


Εμφάνιση στοιχείων (1)

Η εμφάνιση δεδομένων της εφαρμογής, για παράδειγμα των χρηστών, σε ένα πίνακα μιας ιστοσελίδας. Θα μπορούσε να γίνει με AJAX κλήση στο API της εφαρμογής.

1. Αρχείο `index.html` : η κεντρική σελίδα της εφαρμογής.
2. Αρχείο `user\find.html` : αρχείο εμφάνισης στοιχείων χρηστών
3. Αρχείο `user\create.html` : αρχείο με φόρμα για δημιουργία χρηστών
4. Αρχείο `assets\js\user.scripts.js` : αρχείο javascript με τις απαραίτητες διαδικασίες για την εμφάνιση και την δημιουργία των χρηστών

Δομή εφαρμογής

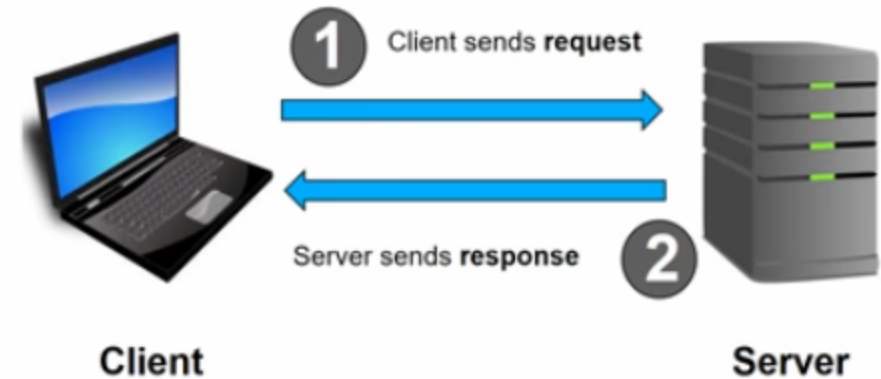




Εμφάνιση στοιχείων (2)

Στο αρχείο `assets\js\user.scripts.js` με χρήση της βιβλιοθηκής jQuery κάνουμε τις παρακάτω διαδικασίες

1. HTTP request με Ajax για να πάρουμε στοιχεία (get) από την εφαρμογή μας για την εμφάνιση των χρηστών.
2. HTTP request με Ajax για να στείλουμε στοιχεία (post) στην εφαρμογή μας για την δημιουργία χρηστών.
3. Διαδικασία για το διαβάσμα των χρηστών από την get κλήση και προσθήκη αυτών στον html πίνακα





Εμφάνιση στοιχείων (3)

find.html

```
<table id="userTable">
  <thead>
    <tr>
      <th>Username</th>
      <th>Όνομα</th>
      <th>Επίθετο</th>
      <th>Email</th>
      <th>Διεύθυνση</th>
      <th>Τηλέφωνο</th>
    </tr>
  </thead>
  <tbody id="tbody"></tbody>
</table>
```

user.scripts.js

```
$(document).ready(function(){
  $.ajax({
    url: 'http://localhost:3000/api/users',
    type: 'get',
    dataType: 'JSON'
  })
  .done(function(response){
    let data = response.data;
    let status = response.status

    if (status) {
      createTbody(data);
    } else {
      alert(false, 'Πρόβλημα στην αναζήτηση των χρηστών ('+ data.message + ')');
    }
  });
});
```



Εμφάνιση στοιχείων (4)

user.scripts.js

```
function createTbody(data){

    $("#userTable > tbody").empty();

    // console.log("CreateTBody", data);
    const len = data.length;
    for (let i=0; i<len; i++){
        let username = data[i].username;
        let name = data[i].name;
        let surname = data[i].surname;
        let email = data[i].email;
        let address = data[i].address.area + ", " + data[i].address.road;
        let phone = "";
        for (let x=0; x<data[i].phone.length; x++) {
            phone = phone + data[i].phone[x].type + ":" + data[i].phone[x].number + "<br>"
        }

        // console.log(username, name);

        let tr_str = "<tr>" +
            "<td>" + username + "</td>" +
            "<td>" + name + "</td>" +
            "<td>" + surname + "</td>" +
            "<td>" + email + "</td>" +
            "<td>" + address + "</td>" +
            "<td>" + phone + "</td>" +
            "<td>" +
                "<button class='btnUpdate btn btn-primary' value='"+username+"'>Τροποποίηση</button>" +
                "<button class='btnDelete btn btn-primary' value='"+username+"'>Διαγραφή</button>" +
            "</td>" +
            "</tr>";

        $("#userTable tbody").append(tr_str);
    }
}
```

Η διαδικασία createTbody παίρνει ως είσοδο τα αποτελέσματα από την get κλήση. Η κλήση αυτή περιέχει όλους τους χρήστες της εφαρμογής.

Στη συνέχεια διασχίζει τα αποτελέσματα και για κάθε document που διαβάζει δημιουργεί μια γραμμή του πίνακα `userTable`.

Αφου δημιουργήσει την γραμμή την κάνει append στο tbody του πίνακα



Δημιουργία χρηστών (1)

create.html:

```
<form id="frmUser">
  <div class="mb-3">
    <label for="username" class="form-label">Username</label>
    <input type="text" class="form-control" name="username" id="username" aria-describedby="usernameHelp">
    <div id="usernameHelp" class="form-text">Πληκτρολογήστε το username που επιθυμείτε.</div>
  </div>
  <div class="mb-3">
    <label for="password" class="form-label">Password</label>
    <input type="password" class="form-control" name="password" id="password">
  </div>
  <div class="mb-3">
    <label for="name" class="form-label">Όνομα</label>
    <input type="text" class="form-control" name="name" id="name">
  </div>
  <div class="mb-3">
    <label for="surname" class="form-label">Επίθετο</label>
    <input type="text" class="form-control" name="surname" id="surname">
  </div>
  <div class="mb-3">
    <label for="email" class="form-label">Email</label>
    <input type="email" class="form-control" name="email" id="email">
  </div>
  <div class="mb-3">
    <fieldset>
      <legend>Στοιχεία Διεύθυνσης</legend>
      <label for="area" class="form-label">Περιοχή</label><br>
      <input type="text" class="form-control" name="area" id="area"><br>
      <label for="road" class="form-label">Οδός</label><br>
      <input type="text" class="form-control" name="road" id="road">
    </fieldset>
  </div>
</form>
```

Φόρμα δημιουργίας χρηστών

Με την υποβολή της φόρμας καλείται η αντίστοιχη διαδικασία απο το user.scripts.js αρχείο.

Η διαδικασία αυτή κάνει μια AJAX κλήση τύπου post στο API της εφαρμογής για την δημιουργία του χρήστη

```
<div class="row">
  <div class="col text-center">
    <button class="btnSubmit btn btn-primary text-center" value="insert">Υποβολή</button>
    <button class="btnReset btn btn-primary text-center">Εναφορά</button>
  </div>
</div>
```



Δημιουργία χρηστών (2)

user.scripts.js:

```
$('.row').off('click', '.btnSubmit').on('click', '.btnSubmit', function () {  
  
    let username = $("#username").val();  
    let password = $("#password").val();  
    let name = $("#name").val();  
    let surname = $("#surname").val();  
    let email = $("#email").val();  
    let area = $("#area").val();  
    let road = $("#road").val();  
  
    const item = {  
        'username': username,  
        'password': password,  
        'name': name,  
        'surname': surname,  
        'email': email,  
        'address': {  
            'area': area,  
            'road': road  
        }  
    }  
    $.ajax({  
        ...  
    })  
  
    return false  
});
```

AJAX κλήση

```
// console.log($('.btnSubmit').val(), item);  
$.ajax({  
    url: "http://localhost:3000/api/user/create",  
    type: "post",  
    data: item,  
    dataType: "JSON",  
    // encode: true,  
})  
.done( function(response) {  
    // console.log(">>", response);  
  
    let data = response.data;  
    let status = response.status  
  
    if (status) {  
        console.log(true, 'Επιτυχής εισαγωγή του χρήστη');  
        alert(true, 'Επιτυχής εισαγωγή του χρήστη');  
        $('#frmUser')[0].reset();  
    } else {  
        console.log(false, 'Πρόβλημα στην εισαγωγή του χρήστη ('+ data.message + ')');  
        alert(false, 'Πρόβλημα στην εισαγωγή του χρήστη ('+ data.message + ')');  
        $('#frmUser')[0].reset();  
        // console.log(data.message);  
    }  
});
```

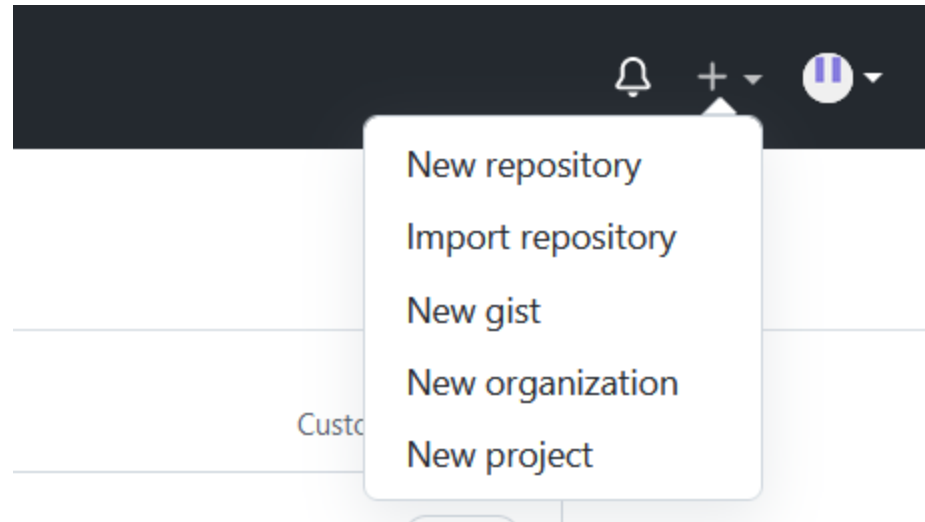


Ανέβασμα εφαρμογής στο Github

1. Πρέπει να έχει εγκατασταθεί το περιβάλλον git στον υπολογιστή μας.
2. Κάνουμε Sign in ή Sign up στην σελίδα του GitHub (<https://github.com/>)



3. Πάνω δεξιά επιλέγουμε το κουμπί με στο σύμβολο + και στην συνέχεια `new repository`



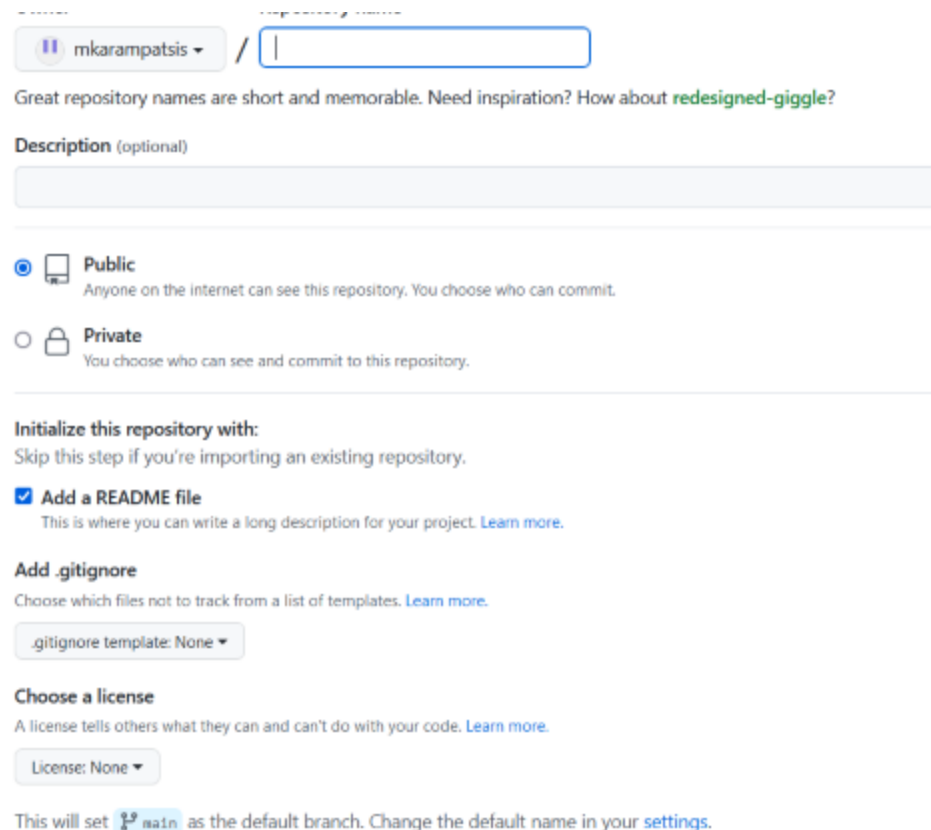


Ανέβασμα εφαρμογής στο Github

1. Στο πεδίο `Repository name` πληκτρολογούμε το όνομα του repository
2. Στο πεδίο `Description` (προαιρετικό πεδίο) πληκτρολογούμε μια περιγραφή
3. Αφήνουμε επιλεγμένο το `Public`
4. Επιλέγουμε το `Add a README file`
5. Στην επιλογή `Add .gitignore` διαλέγουμε `Node`
6. Πατάμε το `Create repository`

 You are creating a public repository in your personal account.


Create repository




mkarampatsis /

Great repository names are short and memorable. Need inspiration? How about [redesigned-giggle?](#)

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: **None**

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: **None**

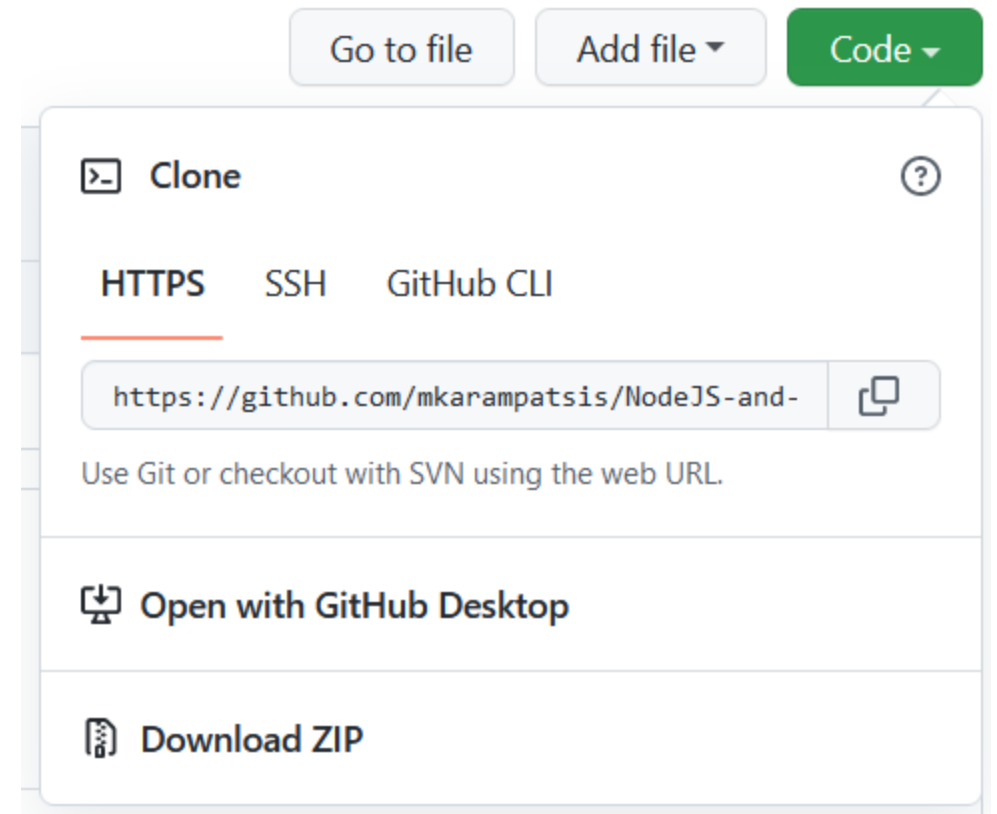
This will set **main** as the default branch. Change the default name in your [settings](#).



Ανέβασμα εφαρμογής στο Github

1. Στο repository που δημιουργήσατε πατήστε την επιλογή `code` και κάντε copy το link του repository
2. Με command line πλοηγηθείτε στο φάκελο της εφαρμογής και τρέξτε τις εντολές

```
git init -b main
git remote add origin {το link που αντιγράψατε}
git pull origin main
git add .
git commit -m "Initialize project"
git push origin main
```





Άσκηση (1)

1. Χρησιμοποιώντας ως οδηγό τα αρχεία εφαρμογής που έχουν ανέβει στην πλατφόρμα edudz να δημιουργήσετε CRUD κλήσεις για τα requests των προϊόντων.

Συγκεκριμένα για τις κλήσεις:

- get: `/api/products` τον controller `productController.findAll`
- get: `/api/products/:id` τον controller `productController.findOne`
- post: `/api/products` τον controller `productController.create`
- patch: `/api/products/:id` τον controller `productController.update`
- delete: `/api/products/:id` τον controller `productController.delete`

Δείτε τα αρχεία `user.route.js`, `user.controller.js`, `user.model.js` καθώς και την `app.use` κλήση στο αρχείο `app.js`



Άσκηση (2)

2. Χρησιμοποιώντας ως οδηγό τα αρχεία εφαρμογής που έχουν ανέβει στην πλατφόρμα edudz να δημιουργήσετε τις ιστοσελίδες εμφάνισης των προϊόντων και την φόρμα εισαγωγής αυτών

Δείτε από το φάκελο files της εφαρμογής τα αρχεία `user/find.html`, `user/create.html` και `assets/js/user.scripts.js`

ΠΡΟΣΟΧΗ

Η τελική εφαρμογή πρέπει να ανέβει στο github και στη πλατφόρμα θα πρέπει να ανεβάσετε το link της εφαρμογής στο github.