

@maryemhadjwannes

NodeJS

Start Tutorial



1

Introduction

2

**Tools
Installations**

3

**first Project
Initiation**

4

Postman test

5

**CRUD
Operations**

6

**Database
Connection**

7

**Model
Creation**

8

Routing

9

Upload Files

10

**Account
Creation**

11

Login

12

**Tutorial
Summary**

What is Node.js?

Node.js is a JavaScript runtime built on Chrome's V8 engine, allowing you to run JavaScript on the server side.

Why Node.js?

It is lightweight, efficient, and designed for building fast, scalable network applications.

Key Features:

- Asynchronous and event-driven architecture
- Non-blocking I/O
- High-performance for real-time applications

Common Uses:

- Web servers, APIs, real-time applications (chat, gaming), data streaming, etc.



Visual Studio Code

<https://code.visualstudio.com/>



<https://nodejs.org/en>



POSTMAN

<https://www.postman.com/>

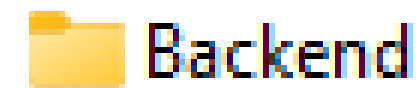


<https://www.mongodb.com/>

●●● First Project Initiation

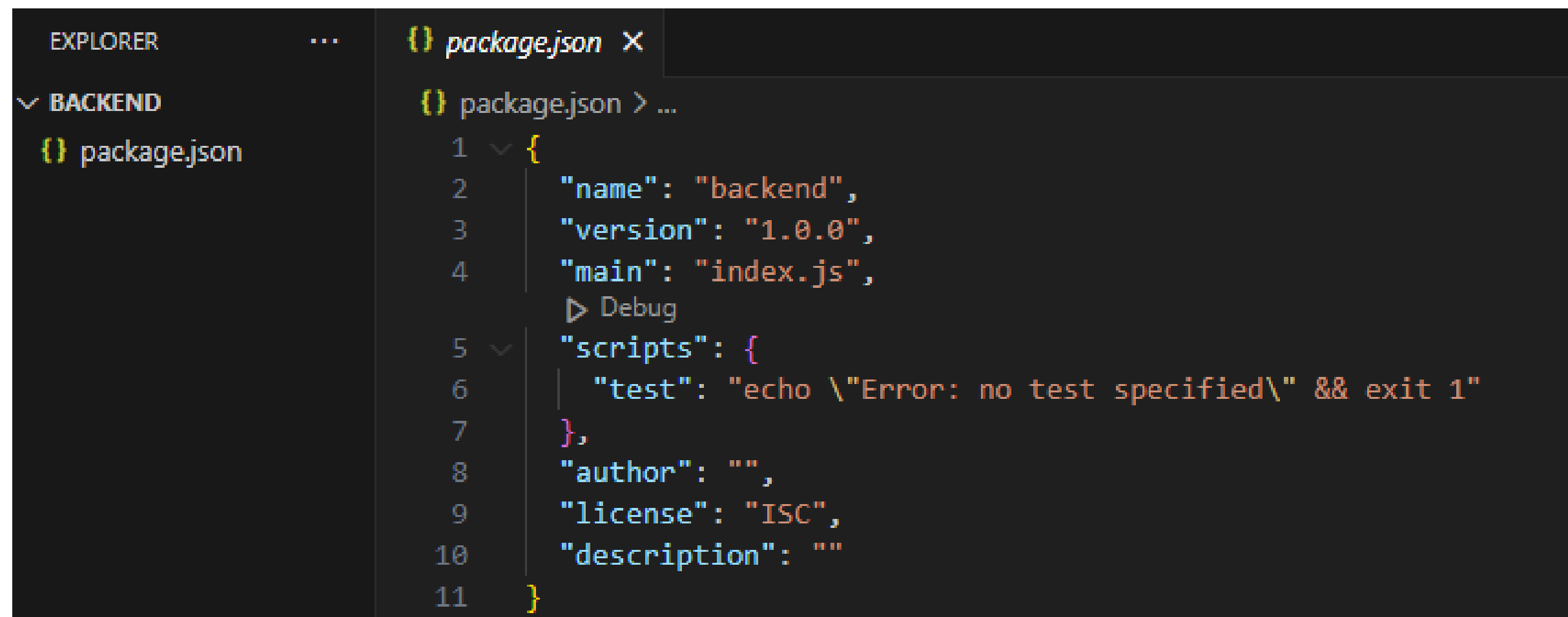
@maryemhadjwannes

1- Create “Backend” Folder



2- Initiate NodeJS project

```
C:\Users\user\Documents\Backend>npm init
```

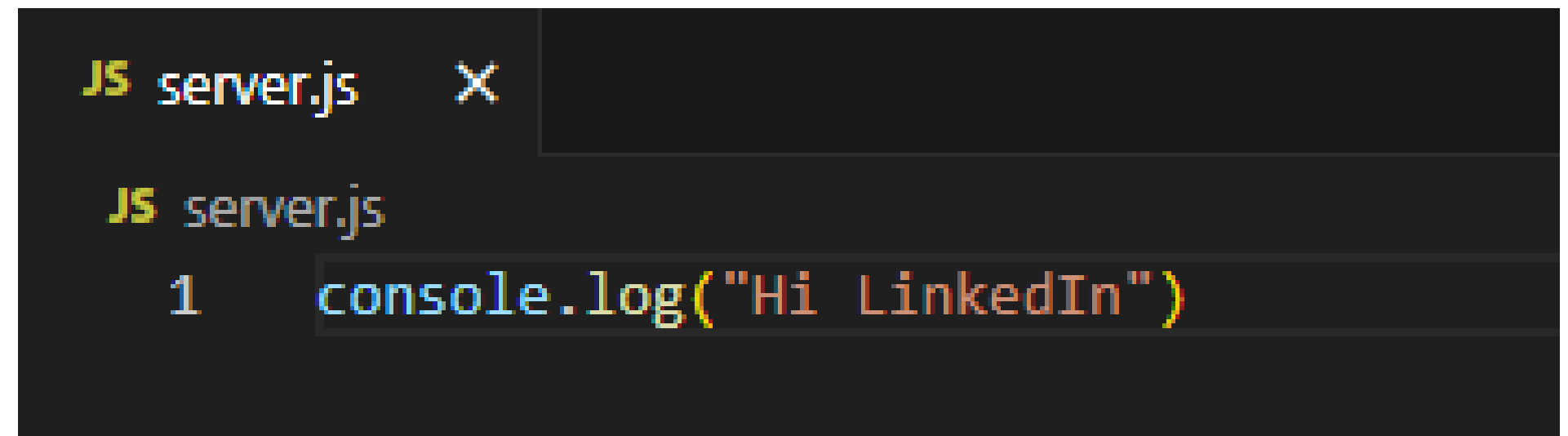
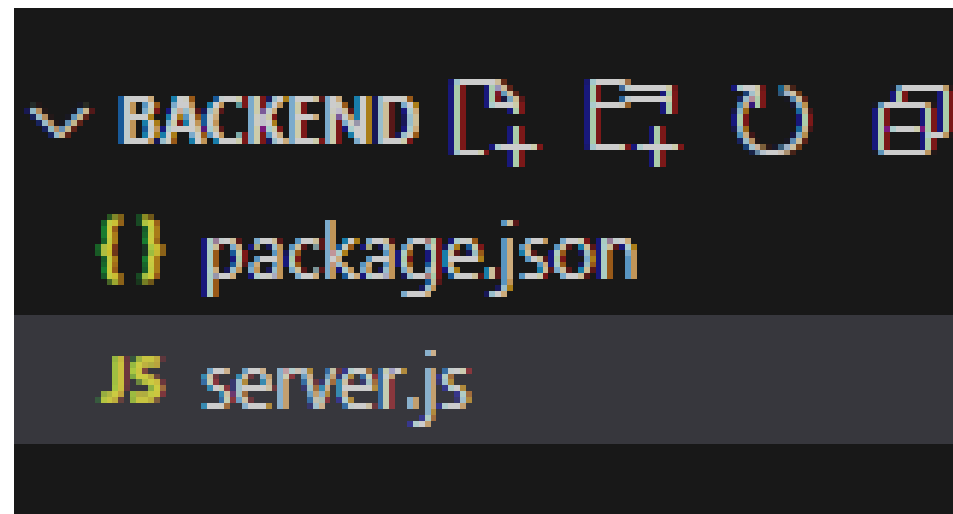
A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows a folder named 'BACKEND' containing a file named 'package.json'. The main editor area displays the content of 'package.json' with the following JSON structure:

```
1 {  
2   "name": "backend",  
3   "version": "1.0.0",  
4   "main": "index.js",  
5   "scripts": {  
6     "test": "echo \"Error: no test specified\" && exit 1"  
7   },  
8   "author": "",  
9   "license": "ISC",  
10  "description": ""  
11 }
```

●●● First Project Initiation

@maryemhadjwannes

3- Create new File “server.js”



4- Console display

```
PS C:\Users\user\Documents\Backend> node server.js  
Hi LinkedIn
```

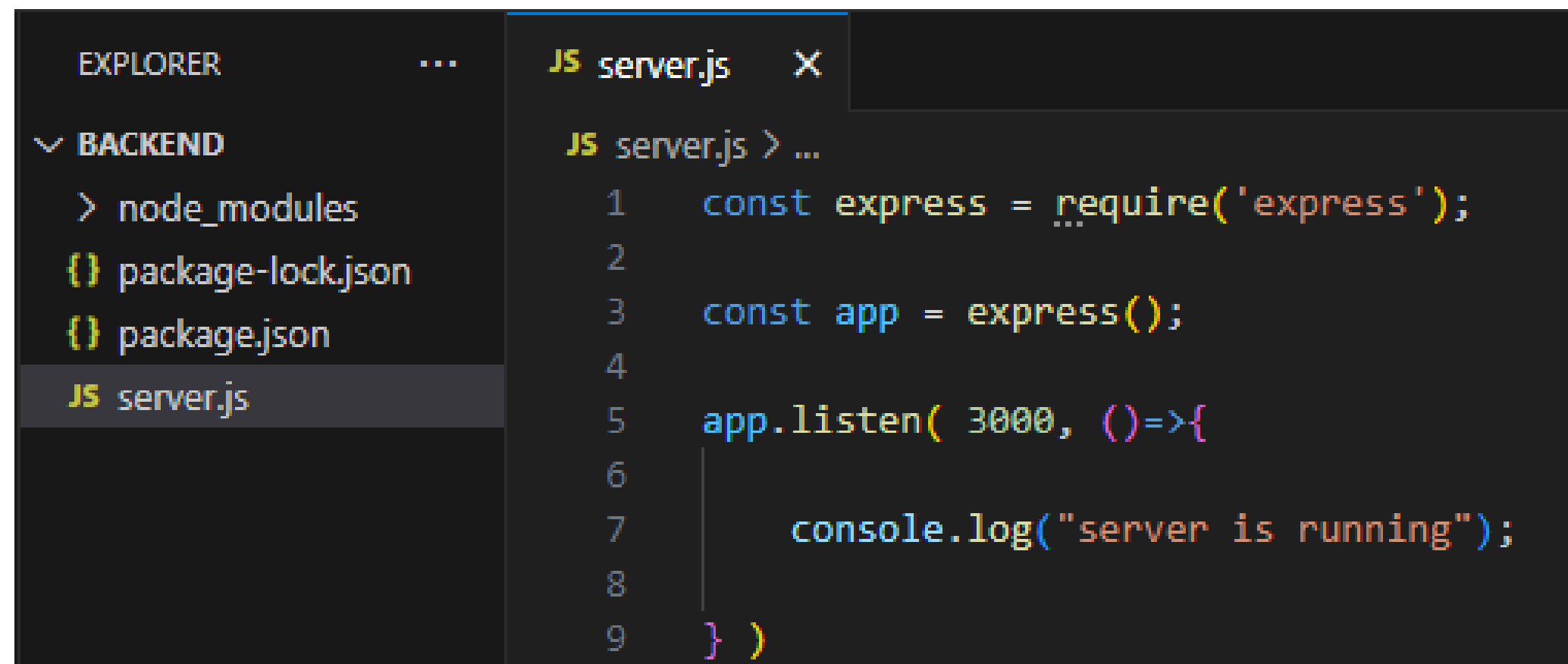
●●● First Project Initiation

@maryemhadjwannes

5- Adding Express library

```
PS C:\Users\user\Documents\Backend> npm install express
```

6- initializing and starting a server using the Express framework.



The screenshot shows the VS Code interface. On the left, the Explorer sidebar is open, showing a project structure with a folder named 'BACKEND'. Inside 'BACKEND', there are files 'node_modules', 'package-lock.json', 'package.json', and 'server.js'. The 'server.js' file is selected and highlighted. The main editor area shows the content of 'server.js', which is a JavaScript file for initializing an Express server. The code is as follows:

```
JS server.js > ...
1  const express = require('express');
2
3  const app = express();
4
5  app.listen( 3000, ()=>{
6
7      console.log("server is running");
8
9  } )
```

- We use the Express library to streamline server setup and request handling.

```
PS C:\Users\user\Documents\Backend> node server.js
server is running
```



HTTP METHODS

@maryemhadjwannes

POST

State-changing or side-effecting

GET

Safe and cacheable

PUT

Idempotent and replaceable

DELETE

Idempotent and destructive

HEAD

No response body

CONNECT

Tunneling

OPTIONS

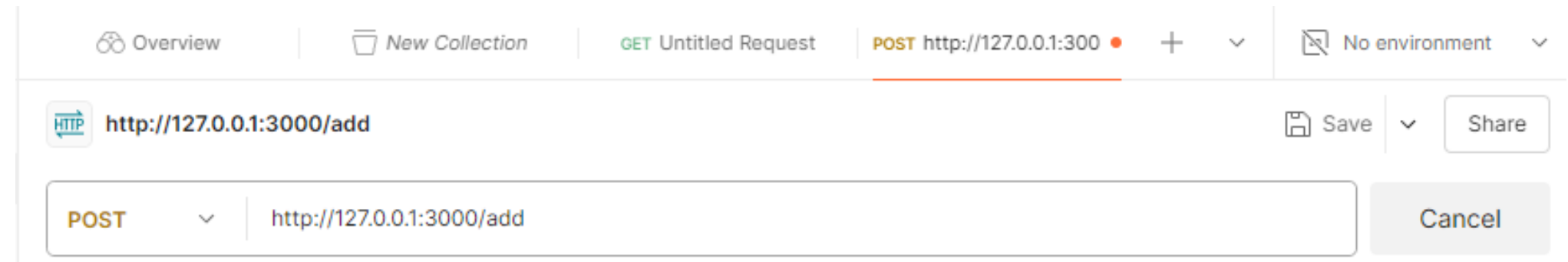
Communication options

TRACE

Message loop-back test

1- Defining a POST request

```
JS server.js x
JS server.js > ...
1  const express = require('express');
2
3  const app = express();
4
5
6  app.post( '/add', ()=>{
7
8      console.log('add post request');
9  })
10
11
12
13  app.listen( 3000, ()=>{
14
15      console.log("server is running");
16
17  } )
```



```
PS C:\Users\user\Documents\Backend> node server.js
server is running
add post request
█
```

●●● Postman Test

@maryemhadjwannes

2- Same for GET, PUT and Delete

```
11 app.get ( '/getall', ()=>{  
12  
13   console.log('add get request');  
14 })  
15  
16 app.put( '/update', ()=>{  
17  
18   console.log('add update request');  
19 })  
20  
21 app.delete( '/delete', ()=>{  
22  
23   console.log('add delete request');  
24 } )  
25
```

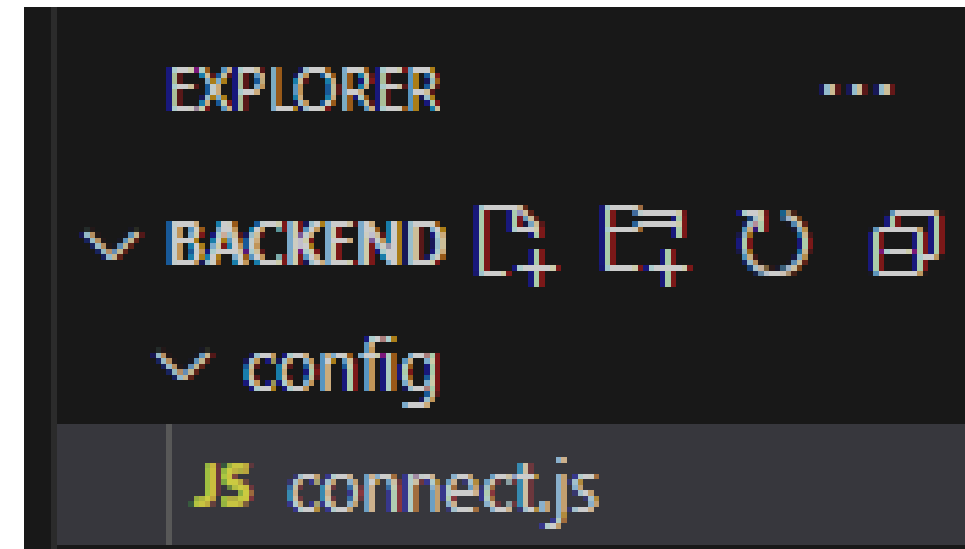
The image displays three sequential screenshots of the Postman application interface, each showing a different HTTP request configuration. The first screenshot shows a GET request to 'http://127.0.0.1:3000/getall' with a 'Send' button. The second screenshot shows a PUT request to 'http://127.0.0.1:3000/update' with a 'Send' button. The third screenshot shows a DELETE request to 'http://127.0.0.1:3000/delete' with a 'Cancel' button. Each interface includes a 'Save' and 'Share' option in the top right corner.

HTTP http://127.0.0.1:3000/getall Save Share
GET http://127.0.0.1:3000/getall Send

HTTP http://127.0.0.1:3000/update Save Share
PUT http://127.0.0.1:3000/update Send

HTTP http://127.0.0.1:3000/delete Save Share
DELETE http://127.0.0.1:3000/delete Cancel

1- Create " **Config > connect.js** "



2- Installing the **mongoose Library**

```
PS C:\Users\user\Documents\Backend> npm install mongoose
```

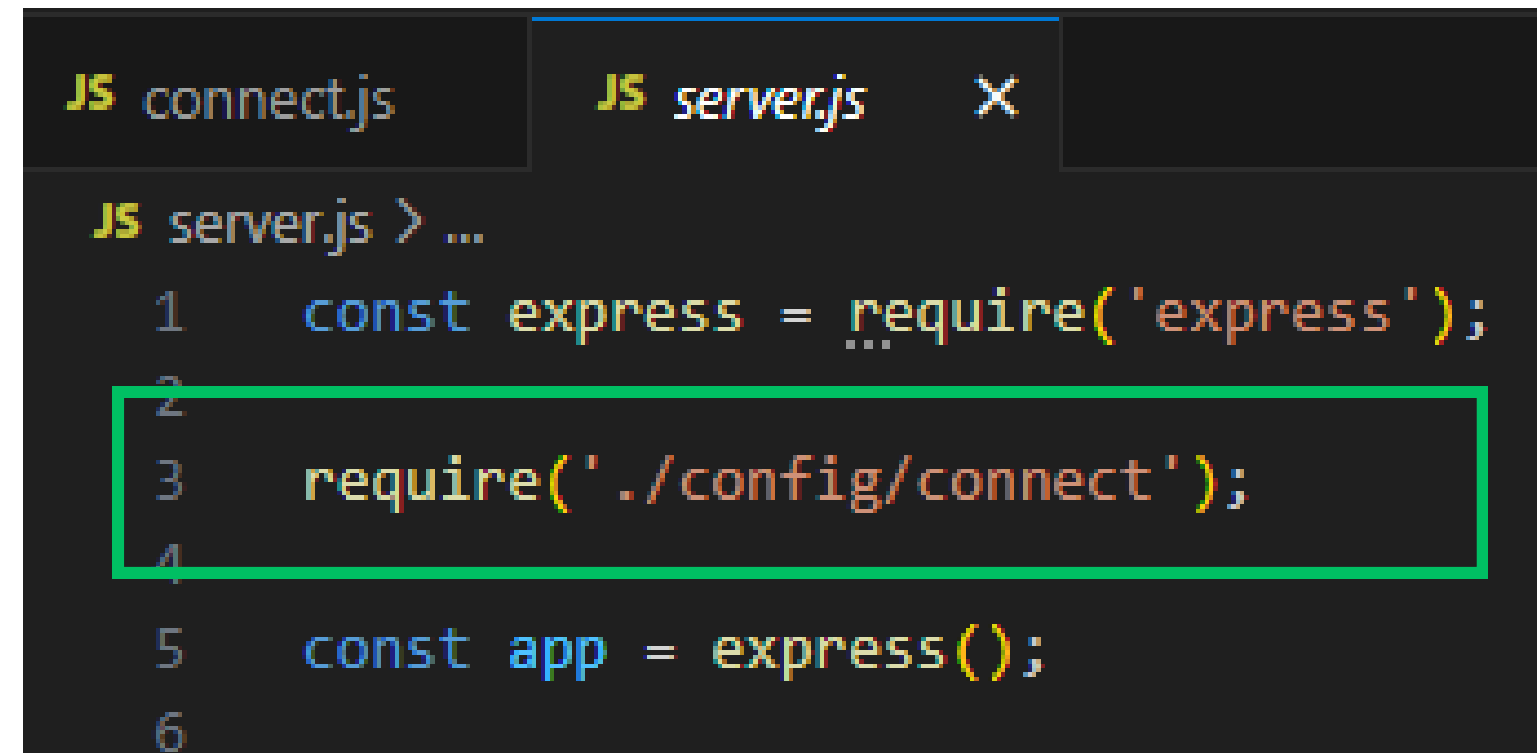
- We use the **Mongoose library** to simplify database interactions.

3- connecting to MongoDB using Mongoose.

```
JS connect.js X
config > JS connect.js > ...
1  const mongoose = require('mongoose');
2
3  mongoose.connect('mongodb://127.0.0.1:27017/firstproject')
4    .then(
5      ()=>{
6        console.log('connected');
7      }
8    )
9    .catch(
10     (err)=>{
11       console.log(err);
12     }
13   )
14
15
16 module.exports = mongoose;
```

- Established a connection to **MongoDB** on **port 27017**, with **firstproject** created automatically if not already existing in the DB.

4- Import the connection setup

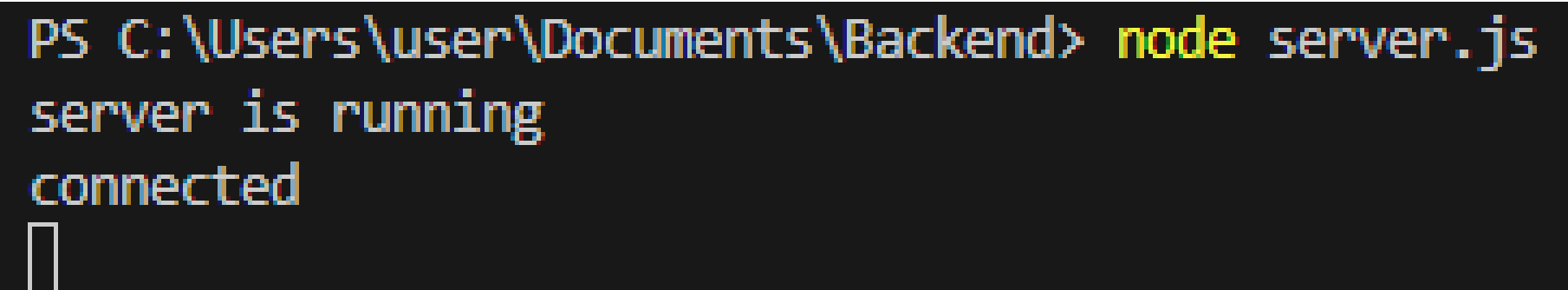


The screenshot shows a code editor with two tabs: 'connect.js' and 'server.js'. The 'server.js' tab is active, and the following code is visible:

```
JS server.js > ...  
1  const express = require('express');  
2  
3  require('./config/connect');  
4  
5  const app = express();  
6
```

Line 3, `require('./config/connect');`, is highlighted with a green rectangular box.

5- establishing the database connection.



The screenshot shows a terminal window with the following text:

```
PS C:\Users\user\Documents\Backend> node server.js  
server is running  
connected  
█
```

1- Create " **models > person.js** "

2- Create a "**Person**" Model

- **Person Model** define the structure of person documents in the database.

▼ models
JS person.js

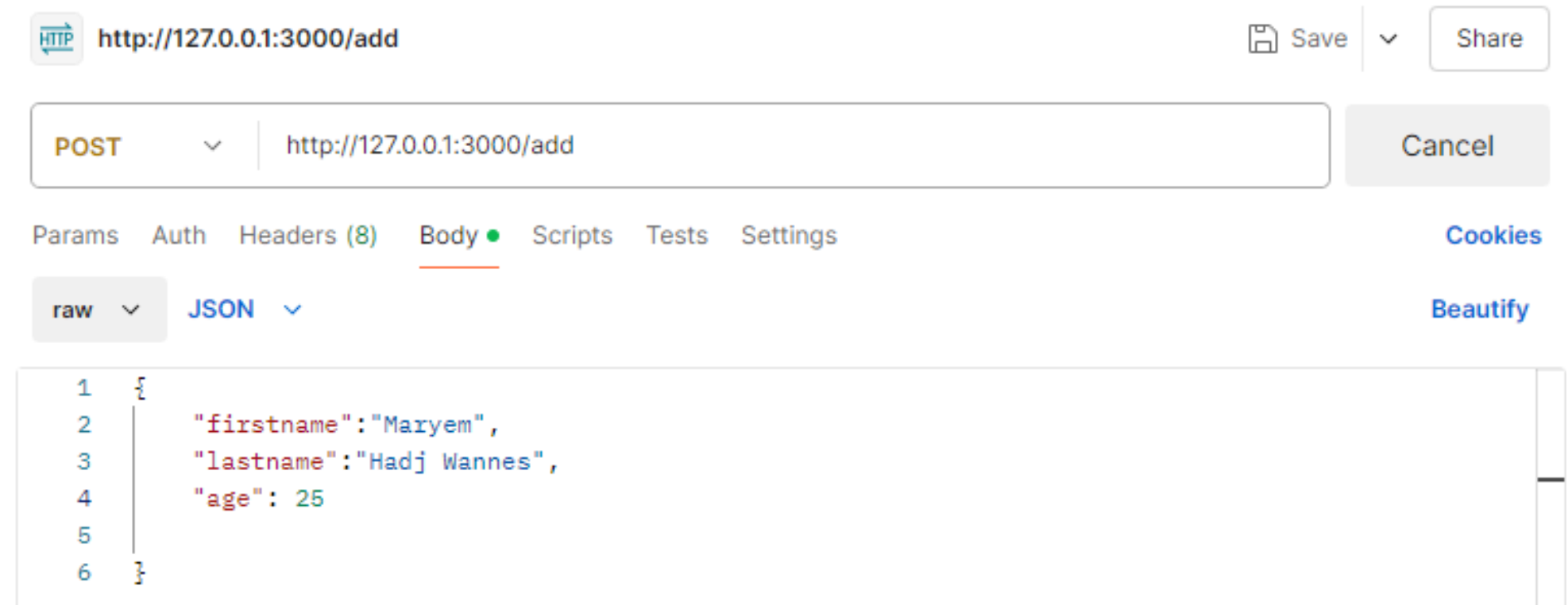
```
models > JS person.js > ...
1  const mongoose = require('mongoose');
2
3  const Person = mongoose.model('Person' , {
4
5      firstname: {
6          type: String
7      },
8      lastname: {
9          type: String
10     },
11     age: {
12         type: Number
13     }
14
15 })
16
17 module.exports = Person;
```

●●● POST Request

@maryemhadjwannes

1- Defining a POST request handler to receive and log data

```
9 app.post( '/add', (req , res)=>{  
10  
11     data = req.body;  
12  
13     console.log(data);  
14 })
```



```
PS C:\Users\user\Documents\Backend> node server.js  
server is running  
connected  
{ firstname: 'Maryem', lastname: 'Hadj Wannes', age: 25 }  
█
```

●●● POST Request

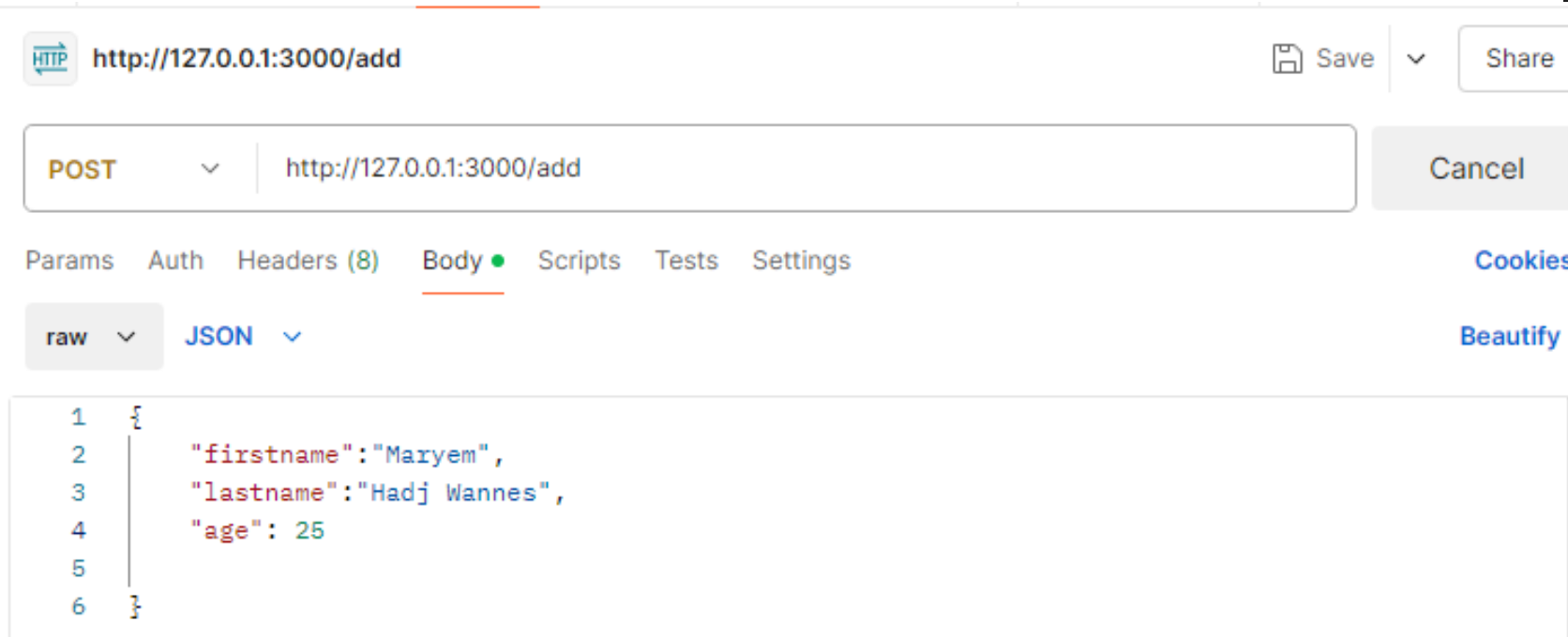
@maryemhadjwannes

2- Importing a Mongoose model

```
2 const Person = require('./models/person');
```

3- Saving data to the database using a Person model.

```
9 app.post( '/add', (req , res)=>{  
10  
11     data = req.body;  
12  
13     person = new Person(data);  
14  
15     person.save();  
16 })
```



4- Send POST request with Postman

5- Refresh Database

The screenshot shows the MongoDB Compass interface. On the left, the 'Connections' panel shows a list of connections: localhost:27017, admin, config, firstproject, people (selected), local, and startup_log. The main panel displays the 'people' collection in the 'firstproject' database on 'localhost:27017'. The 'Documents' tab is active, showing a single document with the following fields: `_id` (ObjectId), `firstname` (Maryem), `lastname` (Hadj Wannes), `age` (25), and `__v` (0). The interface includes a search bar, a query input field, and buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. The document is displayed in a JSON format.

Compass

config local startup_log localhost:27017 people admin +

localhost:27017 > firstproject > people

Documents 1 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE

25 1-1 of 1

```
{
  "_id": "ObjectId('676b1b6e4712a0c5700dba62')",
  "firstname": "Maryem",
  "lastname": "Hadj Wannes",
  "age": 25,
  "__v": 0
}
```

●●● POST Request

@maryemhadjwannes

6- Send response to the Frontend

```
9 app.post( '/add', (req , res)=>{
10
11     data = req.body;
12
13     person = new Person(data);
14
15     person.save()
16     .then(
17         (savedPerson)=>{
18             res.send(savedPerson)
19         }
20     )
21     .catch(
22         (err)=>{
23             res.send(err)
24         }
25     )
26 })
```

The screenshot shows a web browser interface for testing HTTP requests. The URL bar shows `http://127.0.0.1:3000/add`. The request method is set to `POST`. The response status is `200 OK` with a response time of `537 ms` and a size of `332 B`. The response body is displayed in JSON format, showing the following data:

```
{
  "firstname": "Maryem",
  "lastname": "Hadj Wannes",
  "age": 25,
  "_id": "676b1e560d3339df893b43a6",
  "__v": 0
}
```

●●● POST Request

@maryemhadjwannes

```
app.post( '/create', async (req , res)=>{  
  try{  
    data = req.body;  
    person = new Person(data);  
    savedPerson = await person.save();  
    res.send(savedPerson)  
  }catch(error){  
    res.send(error)  
  }  
})
```

Create POST request
with **async await**

●●● GET Request

@maryemhadjwannes

Create get all request

```
48 app.get ( '/getall', (req , res)=>{
49
50     Person.find()
51     .then(
52         (people)=>{
53             res.send(people);
54         }
55     )
56     .catch(
57         (err)=>{
58             res.send(err)
59         }
60     )
61 })
```

HTTP <http://127.0.0.1:3000/getall> Save Share

GET ▼ <http://127.0.0.1:3000/getall> Send ▼

Params Auth Headers (6) Body Scripts Tests Settings Cookies

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (7) Test Results ↺ 200 OK • 22 ms • 433 B 🌐 🔍 ⋮

Pretty Raw Preview Visualize JSON ▼ 🔗 📄 🔍

```
1 [
2   {
3     "_id": "676b1b6e4712a0c5700dba62",
4     "firstname": "Maryem",
5     "lastname": "Hadj Wannes",
6     "age": 25,
7     "__v": 0
8   },
9   {
10    "_id": "676b1e560d3339df893b43a6",
11    "firstname": "Maryem",
```

●●● GET Request

@maryemhadjwannes

```
64 app.get ( '/getall2', async (req , res)=>{
65
66     try{
67         people = await Person.find();
68         res.send(people);
69
70
71     }catch (error){
72         res.send(error)
73     }
74 })
```

**Create GET request
with **async await****

●●● GET Request

@maryemhadjwannes

Create GET by id request

```
77 app.get ( '/getbyid/:id', async (req , res)=>{
78
79     myid = req.params.id;
80
81     Person.findOne({_id: myid})
82         .then(
83             (person)=>{
84                 res.send(person)
85             }
86         )
87         .catch(
88             (err)=>{
89                 res.send(err)
90             }
91         )
92     })
```

```
_id: ObjectId('676b1b6e4712a0c5700dba62')
firstname: "Maryem"
lastname: "Hadj Wannes"
age: 25
__v: 0
```

HTTP <http://127.0.0.1:3000/getbyid/676b1b6e4712a0c5700dba62> Save Share

GET <http://127.0.0.1:3000/getbyid/676b1b6e4712a0c5700dba62> Send

Params Auth Headers (6) Body Scripts Tests Settings Cookies

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (7) Test Results 200 OK 69 ms 332 B 🌐 🔍 ⋮

Pretty Raw Preview Visualize JSON 🔗 📄 🔍

```
1 {
2   "_id": "676b1b6e4712a0c5700dba62",
3   "firstname": "Maryem",
4   "lastname": "Hadj Wannes",
5   "age": 25,
6   "__v": 0
7 }
```

●●● GET Request

@maryemhadjwannes

```
95 app.get ( '/getbyid2/:id', async (req , res)=>{
96
97     try{
98
99         myid = req.params.id;
100
101         person = await Person.findOne({_id: myid})
102
103         res.send(person);
104
105     }catch(error){
106         res.send(error)
107     }
108
109 })
```

**Create GET by id request
with **async await****

●●● DELETE Request

@maryemhadjwannes

```
112 app.delete( '/delete/:id', (req , res)=>{
113
114     myid = req.params.id
115
116     Person.findOneAndDelete({ _id: myid})
117     .then(
118         (deletedPerson)=>{
119             res.send(deletedPerson)
120         }
121     )
122     .catch(
123         (err)=>{
124             res.send(err)
125         }
126     )
127
128 }
```

localhost:27017 > firstproject > people

Documents 1 Aggregations Schema Indexes 1 Validation

🕒 Type a query: { field: 'value' } or [Generate query](#) ✨

➕ ADD DATA ▾ 📄 EXPORT DATA ▾ ✎ UPDATE 🗑️ DELETE

```
_id: ObjectId('676b1e560d3339df893b43a6')
firstname: "Maryem"
lastname: "Hadj Wannes"
age: 25
__v: 0
```

HTTP http://127.0.0.1:3000/delete/676b1e560d3339df893b43a6

Save ▾ Share

DELETE ▾

http://127.0.0.1:3000/delete/676b1e560d3339df893b43a6

Send ▾

●●● DELETE Request

@maryemhadjwannes

```
131 app.delete( '/delete2/:id', async (req , res)=>{
132
133     try{
134         myid = req.params.id
135
136         person = await Person.findOneAndDelete({ _id: myid})
137
138         res.send(person)
139
140     }catch(error){
141         res.send(error)
142     }
143 }
```

Create DELETE request
with **async await**

●●● UPDATE Request

@maryemhadjwannes

```
146 app.put( '/update/:id', (req , res)=>{
147
148     id = req.params.id;
149
150     newData = req.body;
151
152     Person.findByIdAndUpdate({ _id: id} , newData)
153     .then(
154         (updated)=>{
155             res.send(updated)
156         }
157     )
158
159     .catch(
160         (err)=>{
161             res.send(err)
162         }
163     )
164 })
```

HTTP <http://127.0.0.1:3000/update/676b325c5cab7b03cfd8e981> Save Share

PUT ▼ <http://127.0.0.1:3000/update/676b325c5cab7b03cfd8e981> Send ▼

Params Auth Headers (8) **Body** ● Scripts Tests Settings Cookies

raw ▼ **JSON** ▼ Beautify

```
1 {
2   "firstname": "Mohamed",
3   "lastname": "Hadj Wannes",
4   "age": 17
5 }
```

localhost:27017 > firstproject > people

Documents 1 Aggregations Schema Indexes 1 Validation

⌚ ▼ Type a query: { field: 'value' } or [Generate query](#) ⚡

➕ **ADD DATA** ▼ 📄 **EXPORT DATA** ▼ ✎ **UPDATE** 🗑️ **DELETE**

```
_id: ObjectId('676b325c5cab7b03cfd8e981')
firstname : "Mohamed"
lastname : "Hadj Wannes"
age : 17
__v : 0
```

●●● UPDATE Request

@maryemhadjwannes

```
167 app.put( '/update2/:id', async (req , res)=>{
168
169     try{
170         id = req.params.id;
171
172         newData = req.body;
173
174         updated = await Person.findByIdAndUpdate({ _id: id} , newData)
175
176         res.send(updated)
177
178     }catch(error){
179         res.send(error)
180     }
181 })
```

Create UPDATE request
with **async await**

2xx - Success



200 - Success/OK
202 - Accepted
206 - Partial Content

3xx - Redirection



301 - Permanent Redirect
302 - Temporary Redirect
304 - Not Modified

4xx - Client Error



401 - Unauthorized Error
403 - Forbidden
404 - Not Found
405 - Method Not Allowed

5xx - Server Error



501 - Not Implemented
502 - Bad Gateway
503 - Service Unavailable
504 - Gateway Timeout

You can add status for the response

```
res.status(200).send(updated)
} catch (error) {
  res.status(400).send(error)
}
```

●●● New Model + CRUD

@maryemhadjwannes

```
models > JS book.js > [e] <unknown>
 1  const mongoose = require('mongoose');
 2
 3  const Book = mongoose.model('Book' , {
 4
 5      title: {
 6          type: String
 7      },
 8      description: {
 9          type: String
10      },
11      price: {
12          type: Number
13      },
14      image: {
15          type: String
16      }
17
18  })
19
20
21  module.exports = Book;
22
```

Book Model CRUD Same logic
as the **Person model** CRUD.

1- Create "routes > person.js "
and "routes > book.js"

2- This is the Book.js route code,
same logic for the person.js

```
▼ routes  
  JS book.js  
  JS person.js
```

```
routes > JS book.js > ...  
 1  const express = require('express');  
 2  const Book = require('../models/book');  
 3  
 4  const router = express.Router();  
 5  
 6  
 7  //Book crud  
 8  
 9  > router.post( '/createbook', async (req , res)=>{ ...  
23  })  
24  
25  > router.get ( '/getallbooks', async (req , res)=>{ ...  
35  })  
36  
37  > router.delete( '/deletebook/:id', async (req , res)=>{ ...  
49  } )  
50  
51  > router.put( '/updatebook/:id', async (req , res)=>{ ...  
65  })  
66  
67  module.exports = router;
```

2- Update server.js

```
JS server.js > ...
1  const express = require('express');
2
3  const personRoute = require('./routes/person');
4  const bookRoute = require('./routes/book');
5
6  require('./config/connect');
7
8  const app = express();
9  app.use(express.json());
10
11
12  app.use('/book', bookRoute);
13  app.use('/person', personRoute );
14
15
16  app.listen( 3000, ()=>{
17
18      console.log("server is running");
19
20  } )
```

- Simplified request handling by replacing app.post, app.get,.. with **modular routing**.
- Enhanced server.js with a **structured** and scalable **routing approach**.

1- Installing the **multer Library**

```
PS C:\Users\user\Documents\Backend> npm i multer
```

2- Update **routes > book.js** code

- Configured **Multer** to handle file uploads, saving images to the **./uploads** directory with a unique filename based on the current timestamp.
- Used **multer.diskStorage** to define custom storage settings for managing file names and destinations.

```
routes > JS book.js > ...
1  const express = require('express');
2
3  const router = express.Router();
4
5  const Book = require('../models/book');
6
7  const multer = require('multer');
8
9  filename = '';
10
11  const mystorage = multer.diskStorage({
12    destination: './uploads',
13    filename: (req, file, redirect) => {
14
15      let date = Date.now();
16      let fl = date + '.' + file.mimetype.split('/')[1];
17      redirect(null, fl);
18      filename = fl;
19    }
20  });
21
22
23  const upload = multer({storage: mystorage});
```

●●● Upload Files

@maryemhadjwannes


```
router.post( '/createbook', upload.any('image'), async (req , res)=>{  
  
  try{  
    data = req.body;  
    book = new Book(data);  
    this.propfind.image = filename;  
    savedBook = await book.save();  
    filename = '';  
  
    res.status(200).send(savedBook)  
  
  }catch(error){  
    res.status(400).send(error)  
  }  
  
})
```

HTTP <http://127.0.0.1:3000/book/createbook> Save Share

POST ▼ <http://127.0.0.1:3000/book/createbook> Send ▼

Params Auth Headers (8) **Body** ● Scripts Tests Settings Cookies

form-data ▼

	Key		Value	Description	⋮ Bulk Edit
<input checked="" type="checkbox"/>	title	Text ▼	arabic		
<input checked="" type="checkbox"/>	description	Text ▼	level 1		
<input checked="" type="checkbox"/>	price	Text ▼	5000		
<input checked="" type="checkbox"/>	image	File ▼	 Capture d'écran 2024-10-1...		

●●● Account Creation

@maryemhadjwannes

1- Add Authentication attributes to person.js Model (email, password)

```
JS person.js X
models > JS person.js > [Person]
1  const mongoose = require('mongoose');
2
3  const Person = mongoose.model('Person', {
4
5      firstname: {
6          type: String
7      },
8      lastname: {
9          type: String
10     },
11     age: {
12         type: Number
13     },
14     email: {
15         type: String
16     },
17     password: {
18         type: String
19     }
20 }
21 })
22
23 module.exports = Person;
```

2- install **bcrypt** library

```
PS C:\Users\user\Documents\Backend> npm i bcrypt
```

- We Use the **bcrypt** library to securely hash passwords before storing them in the database.

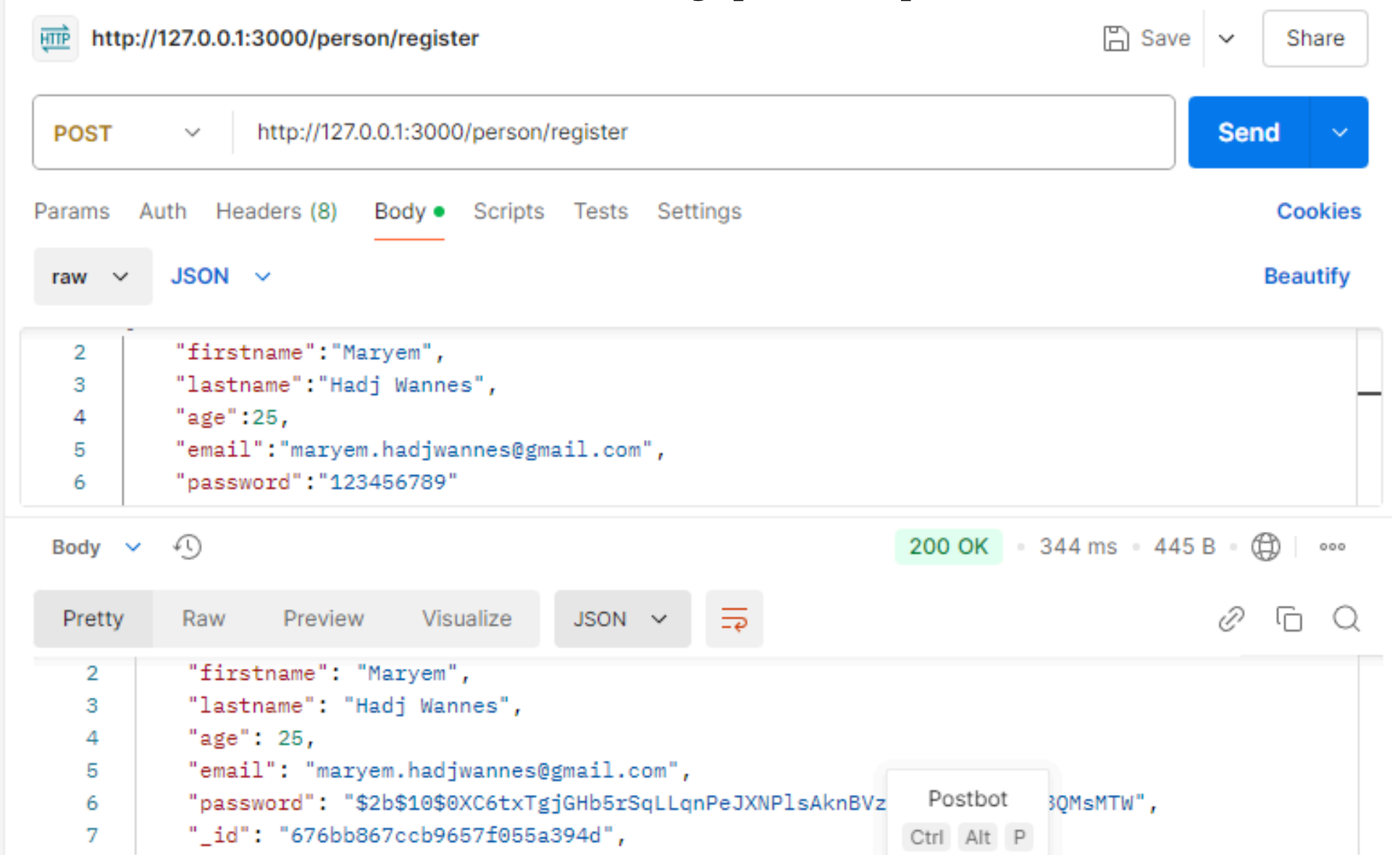
●●● Account Creation

@maryemhadjwannes

2- create POST request (Register) and crypt password using **bcrypt**

```
routes > JS person.js > router.post('/register') callback
4
5 const bcrypt = require('bcrypt');
6
7
8 router.post('/register', async(req, res)=>{
9
10   data = req.body;
11
12   person = new Person(data);
13
14   salt = bcrypt.genSaltSync(10);
15   cryptedException = await bcrypt.hashSync(data.password, salt);
16
17   person.password = cryptedException;
18
19   person.save()
20     .then(
21       (savedPerson)=>{
22         res.send(savedPerson)
23       }
24     )
25     .catch(
26       (err)=>{
27         res.send(err)
28       }
29     )
30 }
31 })
```

3- Postman shows cryptedException



http://127.0.0.1:3000/person/register

POST http://127.0.0.1:3000/person/register

Params Auth Headers (8) Body ● Scripts Tests Settings Cookies Beautify

raw JSON

```
2 "firstname": "Maryem",
3 "lastname": "Hadj Wannes",
4 "age": 25,
5 "email": "maryem.hadjwannes@gmail.com",
6 "password": "123456789"
```

Body 200 OK • 344 ms • 445 B • 🌐 ⋮

Pretty Raw Preview Visualize JSON 🔗 📄 🔍

```
2 "firstname": "Maryem",
3 "lastname": "Hadj Wannes",
4 "age": 25,
5 "email": "maryem.hadjwannes@gmail.com",
6 "password": "$2b$10$XC6txTgjGHb5rSqLLqnPeJXNPlsAknBVz",
7 "_id": "676bb867ccb9657f055a394d",
```

Postbot Ctrl Alt P

1- create POST request (Login)

```

34
35 const jwt = require('jsonwebtoken');
36
37 router.post('/login', async (req , res)=>{
38
39     data = req.body;
40
41     person = await Person.findOne({email: data.email})
42
43     if(!person){
44         res.status(404).send('email or password invalid !')
45     }else{
46         validPassword = bcrypt.compareSync(data.password, person.password)
47
48         if(!validPassword){
49             res.status(401).send('email or password invalid !')
50         }else{
51             payload = {
52                 _id: person._id,
53                 email: person.email,
54                 firstname: person.firstname
55             }
56             token = jwt.sign( payload , '12345' )
57
58             res.status(200).send({mytoken: token})
59         }
60     }
61 })

```

2- install jsonwebtoken Library

```
PS C:\Users\user\Documents\Backend> npm i jsonwebtoken
```

3- Postman return mytoken

The screenshot shows a REST client interface. At the top, the URL is `http://127.0.0.1:3000/person/login`. The method is set to `POST`. The request body is a JSON object: `{ "email": "maryem.hadjwannes@gmail.com", "password": "123456789" }`. The response status is `200 OK` with a response time of `1.49 s` and a size of `478 B`. The response body is a JSON object: `{ "mytoken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NzZiYjg2N2NjYjk2NTdmMDU1YTM5NGQlLCJlbWVpbCI6Im1hcnllbS5oYWwRd2FubmVzQGdtYWlsLmNvbSIsImZpcnN0bmFtZSI6IkhcnllbSIsImh0dCI6MTczNTExNDA2MH0.1ID2-EpSkbsgTu7AK9fD2MqCzQgDpvZ6@L3lw8I-0J8" }`.

4- Try invalid email or password

The screenshot shows a REST client interface with the following components:

- URL Bar:** `http://127.0.0.1:3000/person/login` with **Save** and **Share** buttons.
- Method and Path:** **POST** `http://127.0.0.1:3000/person/login` with a **Send** button.
- Tabs:** Params, Auth, Headers (8), **Body** (selected), Scripts, Tests, Settings, Cookies, Beautify.
- Body Format:** **raw** (selected) and **JSON**.
- Request Body (JSON):**

```
1 {  
2     
3   "email": "maryem.hadjwannes@gmail.com",  
4   "password": "1234567"  
5 }  
6
```
- Response Status:** **401 Unauthorized** • 93 ms • 265 B • [Globe icon] • [More icon]
- Response Body (Pretty):**

```
1 email or password invalid !
```

- In this tutorial, we started by setting up a Node.js project and connecting it to MongoDB using Express.
- We explored how to handle requests, perform CRUD operations, and test our endpoints with Postman.
- We then structured the project with routing, created models for data, and implemented account creation and login functionality.



@maryemhadjwannes

I hope you found this tutorial helpful!