9. Express.js

Tuesday, June 13, 2023 3:07 PM

====**Introduction**

Express is a minimal Node.js framework, a higher level of abstraction;

Express.js is written in 100% Node.js;

Express contains a very robust set of features:
-Complex routing
-Easier handling of Requests & Response
-Middleware
-Server-side rendering

Express allows for rapid development of Node.js apps:
we don't have to re-invent the wheel

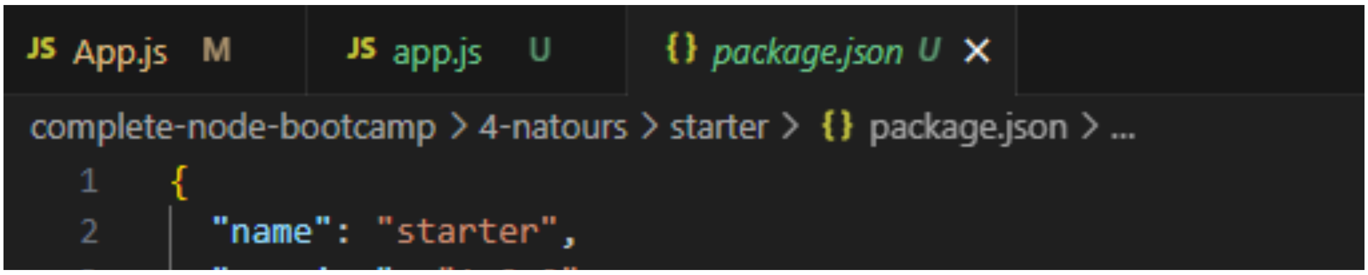
Express makes it easier to organize our app into MVC architecture

====**Install Postman**

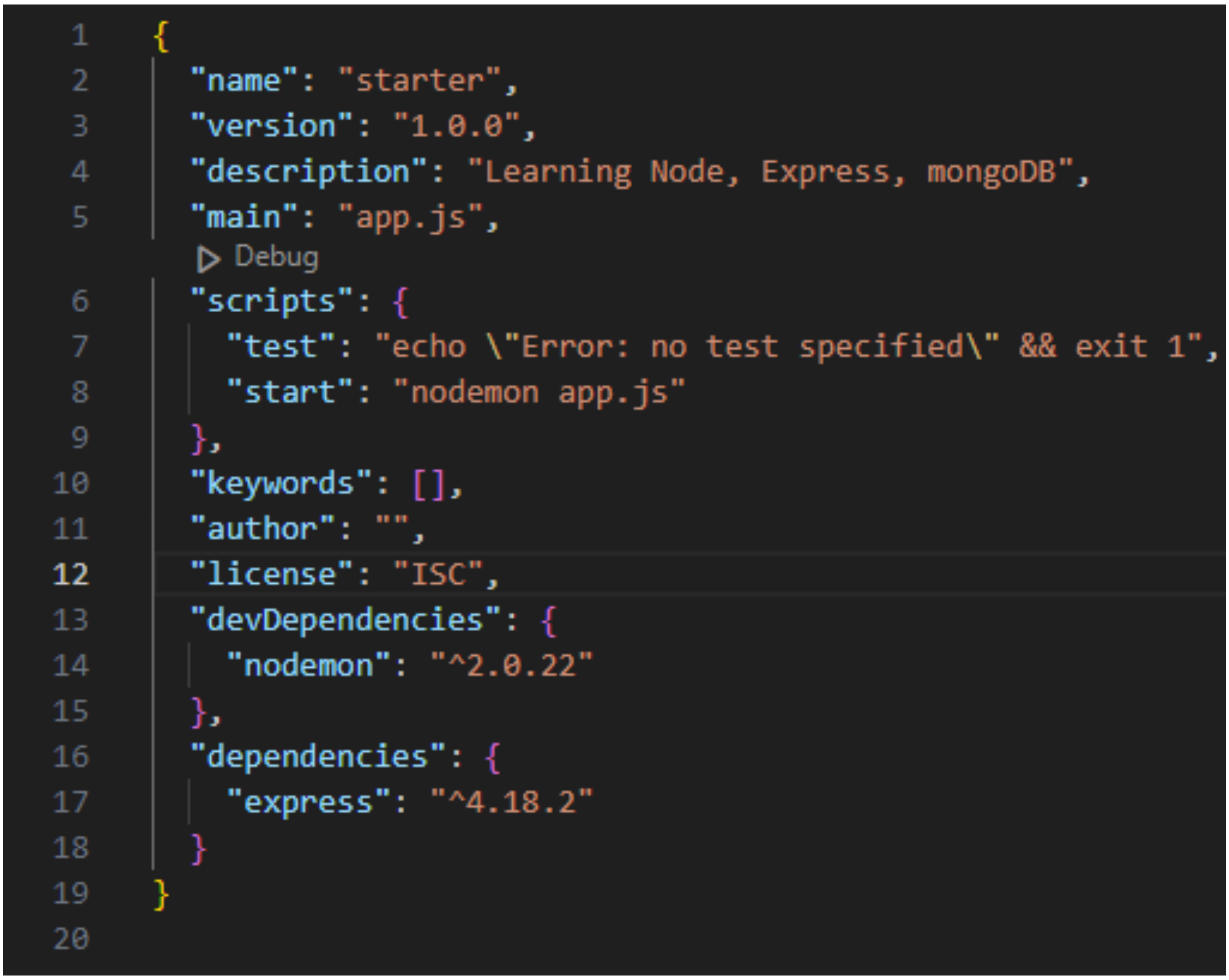
====**Setting up Express & Basic routing**

```
cd /Desktop/Web/vscode/complete-node-bootcamp/4-natours/starter
npm init -y && npm i nodemon --save-dev && npm i express@4
touch app.js
```

package.json:


JS App.js M
JS app.js U
{} package.json U
complete-node-bootcamp > 4-natours > starter > {} package.json > ...

```
{
  "name": "starter",
```

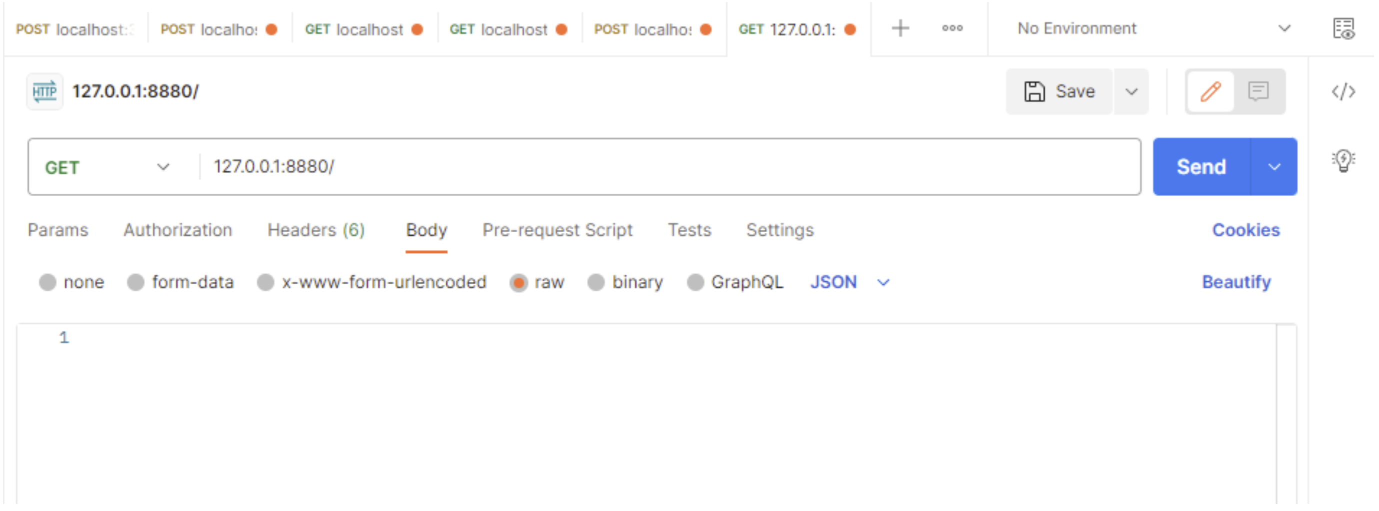


```json
{
  "name": "starter",
  "version": "1.0.0",
  "description": "Learning Node, Express, mongoDB",
  "main": "app.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "nodemon": "^2.0.22"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}

```

app.js:

```javascript
const express = require('express');
const app = express();
app.get('/', (req, res) => {
    console.log('Request received');
    console.log('req.body: \n', req.body);
    res.status(200).send('<h2>Hello~ from server</h2>');
```



POST localhost: | POST localho: ● | GET localhost ● | GET localhost ● | POST localho: ● | GET 127.0.0.1: ● | + | ooo | No Environment

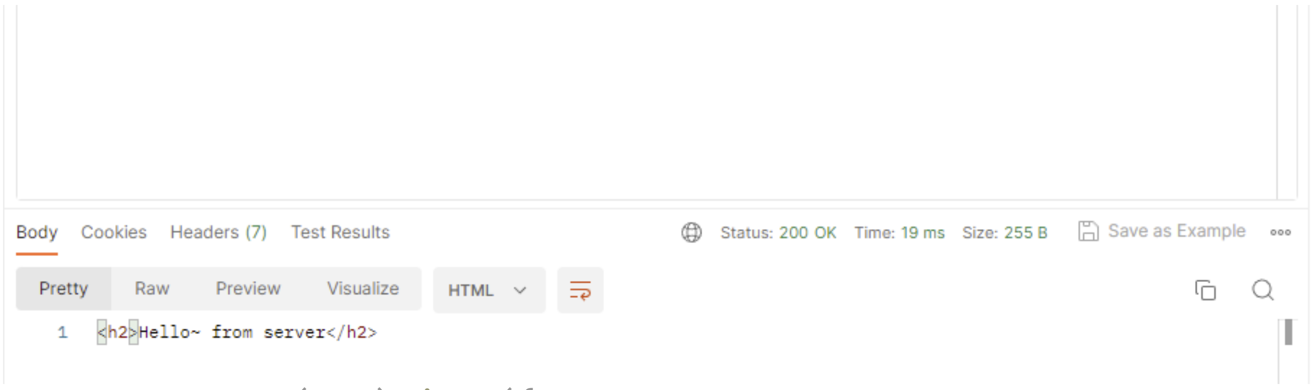
127.0.0.1:8880/ Save

GET 127.0.0.1:8880/ Send

Params | Authorization | Headers (6) | **Body** | Pre-request Script | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON | Beautify

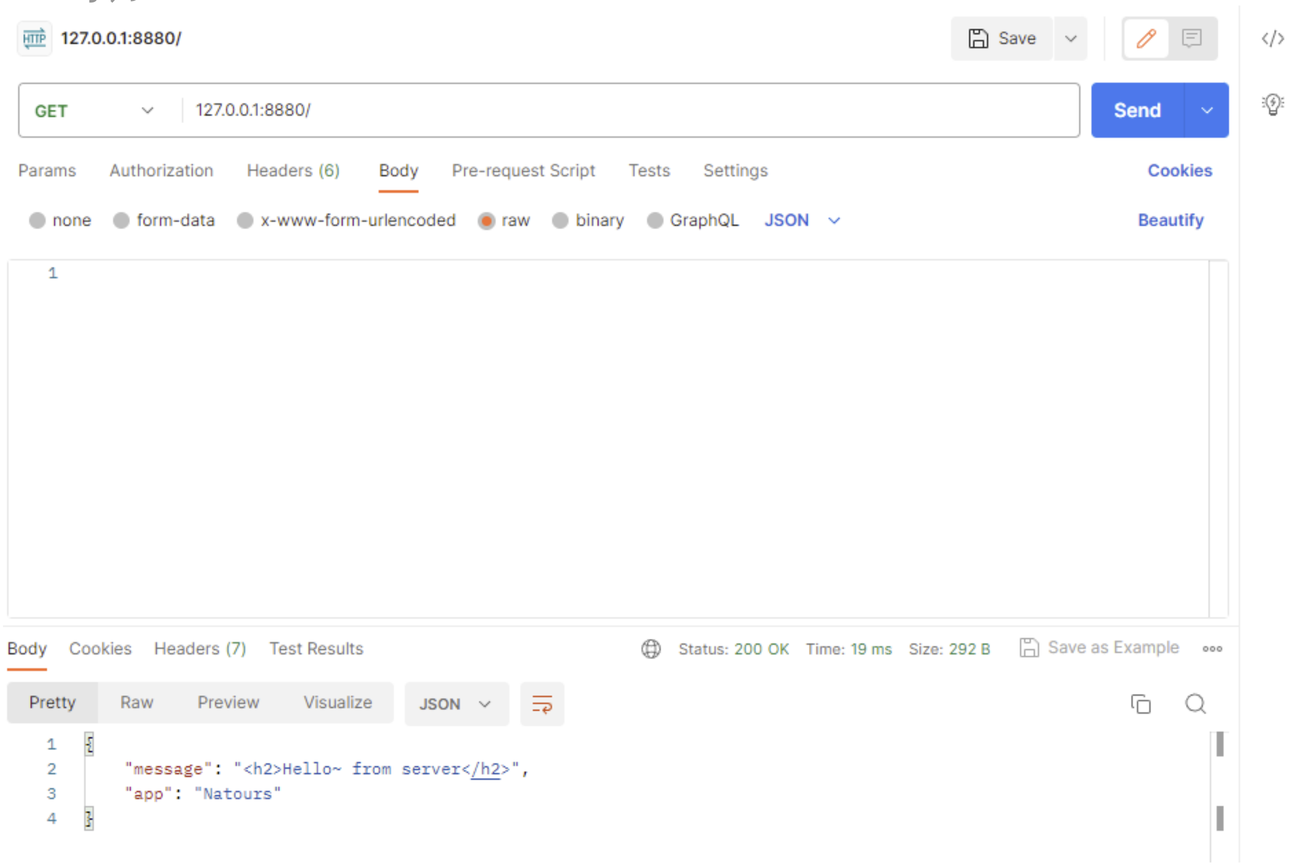
```
1
```



Status: 200 OK Time: 19 ms Size: 255 B

```
<h2>Hello~ from server</h2>
```

```
    res.status(200).json({
        message: '<h2>Hello~ from server</h2>',
        app: 'Natours'
    });
```



127.0.0.1:8880/

GET 127.0.0.1:8880/

Status: 200 OK Time: 19 ms Size: 292 B

JSON

```
{
    "message": "<h2>Hello~ from server</h2>",
    "app": "Natours"
}
```

```
})

// To start a web server
const port = 8880;
const localhost = '127.0.0.1';
app.listen(port, localhost, () => {
    console.log(`Server is listening on ${localhost}:${port}`);
})
```

====APIs & RESTful API design

A software can be used by another software.

allowing to talk to each other.

Database => JSON data => API => Browsers, iOS, Android etc...

Node.js' fs or http APIs ("node APIs");
Browser's DOM JavaScript API;
With OOP, when exposing methods to the public,
we're creating an API;

REST architecture:
Represtation State Transfer

1. Separate API into logical **resources**
2. Expose structured, **resource-based URLs**
3. Use **HTTP methods** (get, post, put, delete)
4. Send data as **JSON**
5. **Stateless**

Resource:
Object / representation of something,
which has data associated to it.
Any info that can be **named** can be a resource.

tours
users
reviews

https://www.natours.com/**addNewTour** **(Endpoint)**
CRUD
/addNewTour --> POST /tours (**Create**) --> Data in --> Database
/getTour --> GET /tours/7 (**Read**) --> Data out
/updateTour --> PUT /tours/7 (**Update**) --> Data in --> Database

/deleteTour --> DELETE /tours/7 (**Delete**) --> Data --> Database

login/search are not CRUD
/login
/search?

```
originalData: {
	"id": 5,
	"tourName": "The Park Camper",
	"rating": "4.9",
	"guides": [
		{
			"name": "Steven Miller",
			"role": "Lead Guide"
		},
		{
			"name": "Lisa Brown",
			"role": "Tour Guide"
		}
	]
}
```

Response Formatting:
-JSend
-JSOPN:API
-OData JSON Protocol

```
JSend: {
	"status": "success",
	"statusCode": "200",
	"data": {
		"id": 5,
		"tourName", "The Park Camper"
```

```
            "tourName": "The Park Camper",
            "rating": "4.9",
            "guides": [
                {
                    "name": "Steven Miller",
                    "role": "Lead Guide"
                },
                {
                    "name": "Lisa Brown",
                    "role": "Tour Guide"
                }
            ]
    }
}
```

Stateless RESTful API:
All state is handled on **Client**.
Each request must contain **all** info
necessary to process a certain request.
The server should **NOT** have to remember previous requests.

Examples of state:
loggedIn
currentPage

GET /tours/nextPage [Bad]
currentPage=5
GET /tours/nextPage --> Web Server --> State on server:
nextPage = currentPage + 1;
send(nextPage); [Bad practice]

GET /tours/page/6 (State coming from client) --> Web Server -->
send(6)