

# Creating a node js server

## Initializing node

- Download and Install Node js
- Check version after installing in terminal: `node --version` and `npm --version` .
- As long as a version is displayed, we are good to go.
- Create a project/folder and open it in vs code. My project name: `backend-tutorial`
- Open VS code terminal or terminal of your choice inside this folder.
- Type and run `npm init`

```
PS E:\trial\backend-tutorial> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help init' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (backend-tutorial)
version: (1.0.0)
description: Creating a basic server
entry point: (index.js)
test command:
git repository:
keywords: backend nodejs
author: Gautam Raj
license: (ISC)
```

- It will ask for the following things with some of them having default values:
  - package name
  - version
  - description: We can describe our app here
  - entry point
  - test command
  - git repository
  - keywords
  - author
  - license
- Just keep pressing `enter` if you don't want to write anything
- We can also run `npm init -y` to skip this menu.
- This will create a `package.json` file where we can change all this if we want.

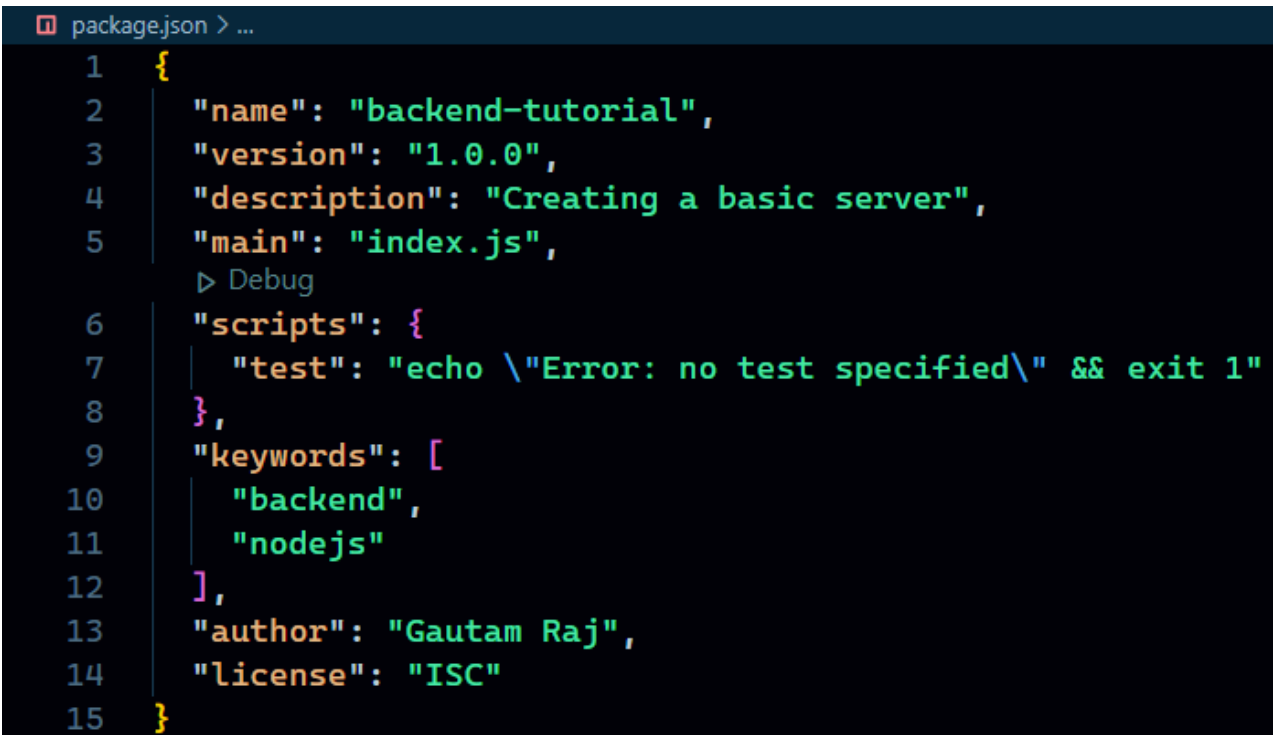
About to write to E:\trial\backend-tutorial\package.json:

```
{
  "name": "backend-tutorial",
  "version": "1.0.0",
  "description": "Creating a basic server",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "backend",
    "nodejs"
  ],
  "author": "Gautam Raj",
  "license": "ISC"
}
```

Is this OK? (yes)

PS E:\trial\backend-tutorial>

- package.json



```
1  {
2    "name": "backend-tutorial",
3    "version": "1.0.0",
4    "description": "Creating a basic server",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [
10     "backend",
11     "nodejs"
12   ],
13   "author": "Gautam Raj",
14   "license": "ISC"
15 }
```

## Creating a basic server

- We will be using `express` framework to create a basic server.
  - Express is a *fast, unopinionated, minimalist* web framework for node js.
  - We can do anything we want to do with express js using standalone node js but express js makes it easier to write node js webapp.
  - Install express using `npm install express` command in terminal
  - Create a file `index.js` in the root directory. Here is the code for basic server setup.
-

```

index.js > ...
1  const express = require('express');
2  const app = express();
3
4  const PORT = process.env.PORT || 3000;
5
6  app.get('/', (req, res) => {
7    res.send('Hello World');
8  });
9
10 app.get('/login', (req, res) => {
11   res.send('<h2>Login Here</h2>');
12 });
13
14 app.listen(PORT, () => {
15   console.log(`Serving on port ${PORT}`);
16 });

```

- **Explanation**

- `const express = require('express');`
  - Imports the Express.js library. This allows you to use the functionalities provided by Express in your application.
- `const app = express();`
  - Creates an instance of the Express application. This instance (app) is where you define routes, middleware, and other settings for your web application.
- `const PORT = process.env.PORT || 3000;`
  - Defines the port number the server will listen on. It uses the environment variable PORT if available (commonly set by hosting services like Heroku) or defaults to port 3000.
- `app.get('/', (req, res) => { res.send('Hello World'); });`
  - Defines a route for handling HTTP GET requests to the root URL ('/'). When a user accesses the root URL, the server responds with the plain text message 'Hello World'.
- `app.get('/login', (req, res) => { res.send('<h2>Login Here</h2>'); });`
  - Defines another route for handling GET requests to the '/login' URL. When a user accesses '/login', the server responds with an HTML message `<h2>Login Here</h2>`.
- `app.listen(PORT, () => { console.log(Serving on port ${PORT}); });`
  - Makes the Express application listen on the specified port (PORT). When the server starts, it logs a message indicating that it is serving on the specified port. The callback function is optional and runs once the server has started listening.

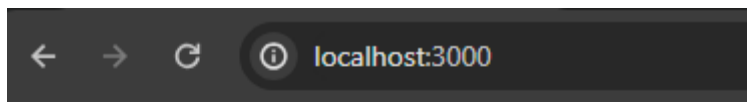
- In the terminal type and run: `node index.js`

```

PS E:\trial\backend-tutorial> node index.js
Serving on port 3000

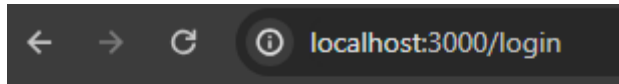
```

- And now if we go to `localhost:3000` in our browser



Hello World

- And now if we go to `localhost:3000/login` in our browser



Login Here

- **Important:** Running our app using `node index.js` is not a good way instead we can define a **start** script inside `package.json` file to run our app. This is how we define a start script.

```
{
  "name": "01_basic_server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^16.3.1",
    "express": "^4.18.2"
  }
}
```

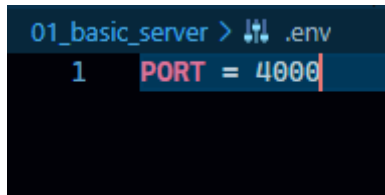
- Now we can simply run our app using `npm run start` or `npm start` in terminal.
- We should do this because:
  - `node any.js` - This will simply run the javascript file "any.js". So if there is no code in there to start a server, we will get error.
  - `npm start` - It will run the start command in the `package.json`.

## Optimizing our basic server code

- Sensitive information, such as API keys or database credentials, should not be hardcoded into the source code. By using a `.env` file, which is typically added to `.gitignore`, we can keep sensitive

information separate from the codebase and prevent it from being unintentionally exposed or committed to version control.

- We should add the `port` number inside `.env` file.
- Different environments (development, testing, production) may require different port configurations. By keeping the port number in a `.env` file, we can easily customize it for each environment.
- Placing the port number in a dedicated configuration file makes it simple to change without modifying the source code. This is particularly important when deploying your application to different servers or platforms. Each environment can have its own `.env` file with the appropriate port number.
- To do this we first need to install `dotenv` package. `npm install dotenv`
- Then create a `.env` file in the root directory.
- In the `.env` file, specify all the environment variables.



```
01_basic_server > .env
1 PORT = 4000
```

- Using the environment variable in `.env` in `index.js`



```
require('dotenv').config()
const express = require('express')
const app = express()

const PORT = process.env.PORT || 3000

app.get('/', (req, res) => {
  res.send('Hello World')
})

app.get('/login', (req, res) => {
  res.send('<h2>Login Here</h2>')
})

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

- **Explanation:**

- `require('dotenv').config();`
  - The `require` function in Node.js is used to import modules. In this case, it's importing the `dotenv` module.
  - The `config` function is a method provided by the `dotenv` module. When called, it reads the contents of a file named `.env` in the root directory of the project and loads the environment variables defined in that file into the `process.env` object.
- `const PORT = process.env.PORT || 3000;`
  - `process.env.PORT` is accessing the `PORT` property of the `process.env` object. The `process.env` object in Node.js provides access to the environment variables of the current process. In this case, it's specifically looking for an environment variable named `PORT`.
  - `||`: This is the logical OR operator. It's used here to provide a default value (3000) if the `process.env.PORT` is falsy (undefined, null, false, 0, etc.).
  - 3000: This is the default value assigned to `PORT` if `process.env.PORT` is falsy.