ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Introduction to Node.js

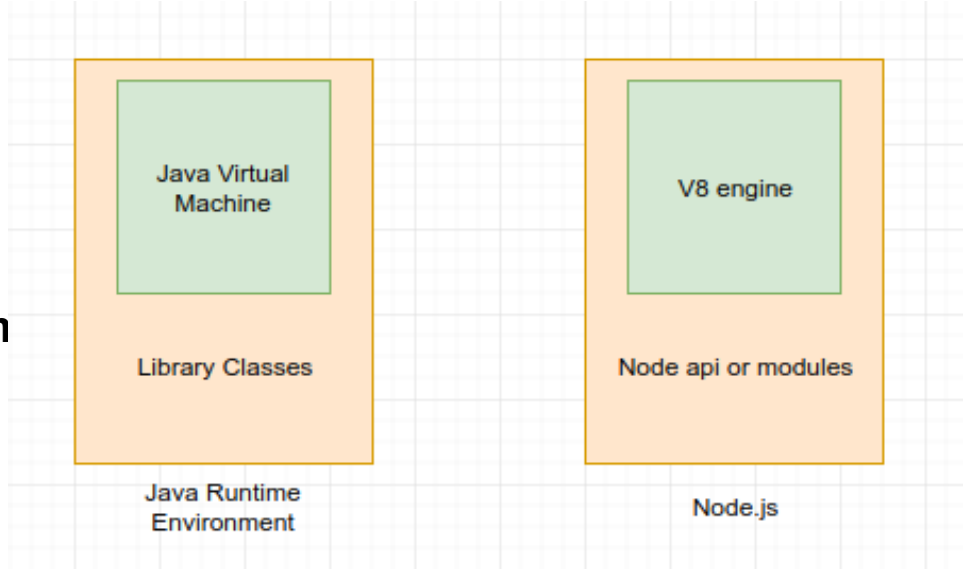
Content

1. Node.js
2. Express
3. Installation and Example

Introduction to Node.js

Basic Nodejs

- Node.js is an open-source and cross-platform JavaScript runtime environment
- Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser.
  - 2009: **Node.js** was created by Ryan Dahl, the first form of **npm** is created
  - 2011: Larger companies adopting **Node.js:** LinkedIn, Uber, (now IBM, Microsoft, Netflix, PayPal, AWS also)
  - 2022: Node.js 19
  - In simple words Node.js is “Server-side JavaScript”.



Node’s goal is to provide an easy way to build scalable network programs – (from nodejs.org)

Basic Nodejs (cont.)

- Node.js represents a "JavaScript everywhere" paradigm

- Unifying web-application development around a single programming language
  => rather than different languages for server-side and client-side scripts.

- Node.js allows the creation of Web servers and networking tools by using JavaScript and a collection of "modules" that handle various core functionalities. Modules are provided for:
  - file system I/O,
  - networking (DNS, HTTP, TCP, TLS/SSL, or UDP),
  - binary data (buffers),
  - cryptography functions,
  - data streams,
  - and other core functions.

Basic Nodejs (cont. 2)

- In 2009, Dahl criticized the limited possibilities of the most popular web server at that time - Apache HTTP Server:
  - to handle a lot of concurrent connections (up to 10,000 and more)
  - the most common way of creating code (sequential programming)
  - code either blocked the entire process or implied multiple execution stacks in the case of simultaneous connections.

- Node.js has an event-driven architecture capable of asynchronous I/O.
  - optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications.

- The most significant difference between Node.js and PHP is that
  - most functions in PHP block until completion (commands execute only after previous commands finish)
  - while Node.js functions are non-blocking (commands execute concurrently or even in parallel, and use callbacks to signal completion or failure).

Basic Nodejs (cont. 3)

- Node.js operates on a single-thread event loop, using non-blocking I/O calls
  ⇒ allowing it to support tens of thousands of concurrent connections without incurring the cost of thread context switching

- To accommodate the single-threaded event loop, Node.js uses the libuv library—which, in turn, uses a fixed-sized thread pool that handles some of the non-blocking asynchronous I/O operations

- A thread pool handles the execution of parallel tasks in Node.js.
  - The main thread function call posts tasks to the shared task queue, which threads in the thread pool pull and execute.
  - Inherently non-blocking system functions such as networking translate to kernel-side non-blocking sockets, while inherently blocking system functions such as file I/O run in a blocking way on their own threads.
  - When a thread in the thread pool completes a task, it informs the main thread of this, which in turn, wakes up and executes the registered callback.

Basic Nodejs (cont. 4)

- A downside of this single-threaded approach is that Node.js does not allow vertical scaling by increasing the number of CPU cores of the machine it is running on without using an additional module, such as cluster, StrongLoop Process Manager, or pm2

- However, developers can increase the default number of threads in the libuv thread pool. The server operating system (OS) is likely to distribute these threads across multiple cores.

Basic Nodejs (cont.)

Example:

- Traditional I/O

```
const fs = require('fs');
const data = fs.readFileSync('/file.md'); // blocks here until file is read
console.log(data);
moreWork(); // will run after console.log
```

- Non-traditional, non-blocking I/O:

```
const fs = require('fs');
fs.readFile('/file.md', (err, data) => {
  if (err) throw err;
  console.log(data);
});
moreWork(); // will run before console.log
```

When can Node.js do?

- Generate dynamic page content
- Create, open, read, write, delete, close files on server
- Collect form data
- Add, delete, modify data in database
- Etc.

When to use Node.js?

- Node.js is good for creating streaming based real-time services, web chat applications, static file servers, etc.
- If high level concurrency and not worried about CPU-cycles.
- If you are great at writing JavaScript code because then you can use the same language at both the places: server-side and client-side

Differences between Node.js and the Browser

- In the browser, most of the time what you are doing is interacting with the DOM, or other Web Platform APIs like Cookies
  - Those do not exist in Node.js
- You don't have the document, window and all the other objects that are provided by the browser
- In the browser, we don't have all the nice APIs that Node.js provides through its modules, like the filesystem access functionality
- Node.js supports both the CommonJS and ES module systems (since Node.js v12):
  - In practice, this means that you can use both require() and import in Node.js, while you are limited to import in the browser.

npm

- In January 2010, a package manager was introduced for the Node.js environment called npm.
- It consists of a command line client, also called npm, and an online database of public and paid-for private packages, called the npm registry
- There is also a website for browsing and searching available packages: https://www.npmjs.com/
- The package manager makes it easier for programmers to publish and share source code of Node.js packages
- Designed to simplify installation, updating, and uninstallation of packages.

Some common npm commands

- npm init: initialize a project
- npm install: add dependency
- npm uninstall: remove dependency
- npm update: update dependency
- npm start / npm run <script_name>: run scripts defined in package.json

package.json

- All npm packages contain a file, usually in the project root, called package.json
  - holds various metadata relevant to the project
- Give information to npm that allows it to identify the project as well as handle the project's dependencies
- Also contain other metadata such as a project description, the version of the project in a particular distribution, license information, even configuration data
  - be vital to both npm and to the end users of the package

package.json fields:

- name
- version
- license
- description
- keywords

package.json fields (cont.):

- **main**: is a direction to the entry point to the module
- **repository**: defines where the source code for the module lives
- **scripts**: defines key/value pairs; key is the name of a command, value is the actual command that is run.
- **dependencies**: defines the other modules that this module uses
  - has the name and version
- **devDependencies**: usually used to define the dependencies the module needs to run in local development and testing

Other

- **nvm**: Node Version Manager
- **npx**: is also a CLI tool whose purpose is to make it easy to install and manage dependencies hosted in the npm registry.
  - Run a locally installed package easily
  - Execute packages that are not previously installed
  - Even allow to run code directly from GitHub
- **eslint**: ESLint is a static code analysis tool for identifying problematic patterns found in JavaScript code

Introduction to Express

Express

- Express.js, or simply Express, is a backend web application framework for building RESTful APIs with Node.js
- Released as free and open-source software under the MIT License.
- It has been called the de facto standard server framework for Node.js.

Express: Basic routing

- Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).
- Each route can have one or more handler functions, which are executed when the route is matched.
- Route definition takes the following structure:

app.METHOD(PATH, HANDLER)

Where:

- app is an instance of express.
- METHOD is an HTTP request method, in lowercase.
- PATH is a path on the server.
- HANDLER is the function executed when the route is matched.

Basic Routing: Examples

- Respond with Hello World! on the homepage:

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

- Respond to a PUT request to the /user route:

```
app.put('/user', (req, res) => {
  res.send('Got a PUT request at /user')
})
```

- Respond to POST request on the root route (/):

```
app.post('/', (req, res) => {
  res.send('Got a POST request')
})
```

- Respond to a DELETE request to the /user route:

```
app.delete('/user', (req, res) => {
  res.send('Got a DELETE request at /user')
})
```

Route Paths

- Route paths, in combination with a request method, define the endpoints at which requests can be made.
- Route paths can be strings, string patterns, or regular expressions.

Route Paths Examples

- This route path will match requests to /about

```
app.get('/about', (req, res) => {
  res.send('about')
})
```

- This route path will match requests to /random.text

```
app.get('/random.text', (req, res) => {
  res.send('random.text')
})
```

- This route path will match acd and abcd

```
app.get('/ab?cd', (req, res) => {
  res.send('ab?cd')
})
```

- This route path will match abcd, abbcd, abbbcd, ...

```
app.get('/ab+cd', (req, res) => {
  res.send('ab+cd')
})
```

Route parameters

- Route parameters are named URL segments that are used to capture the values specified at their position in the URL.
- The captured values are populated in the req.params object, with the name of the route parameter specified in the path as their respective keys.

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

req.query

- A request object that is populated by request query strings that are found in a URL

```
Route: /users/:userId/books?sortBy=name&page=1&pageSize=10
Request URL: http://localhost:3000/users/34/books
req.query: { "sortBy": "name", "page": "1", "pageSize": "10" }
```

Route handlers

- You can provide multiple callback functions that behave like middleware to handle a request.
- Route handlers can be in the form of a function, an array of functions, or combinations of both

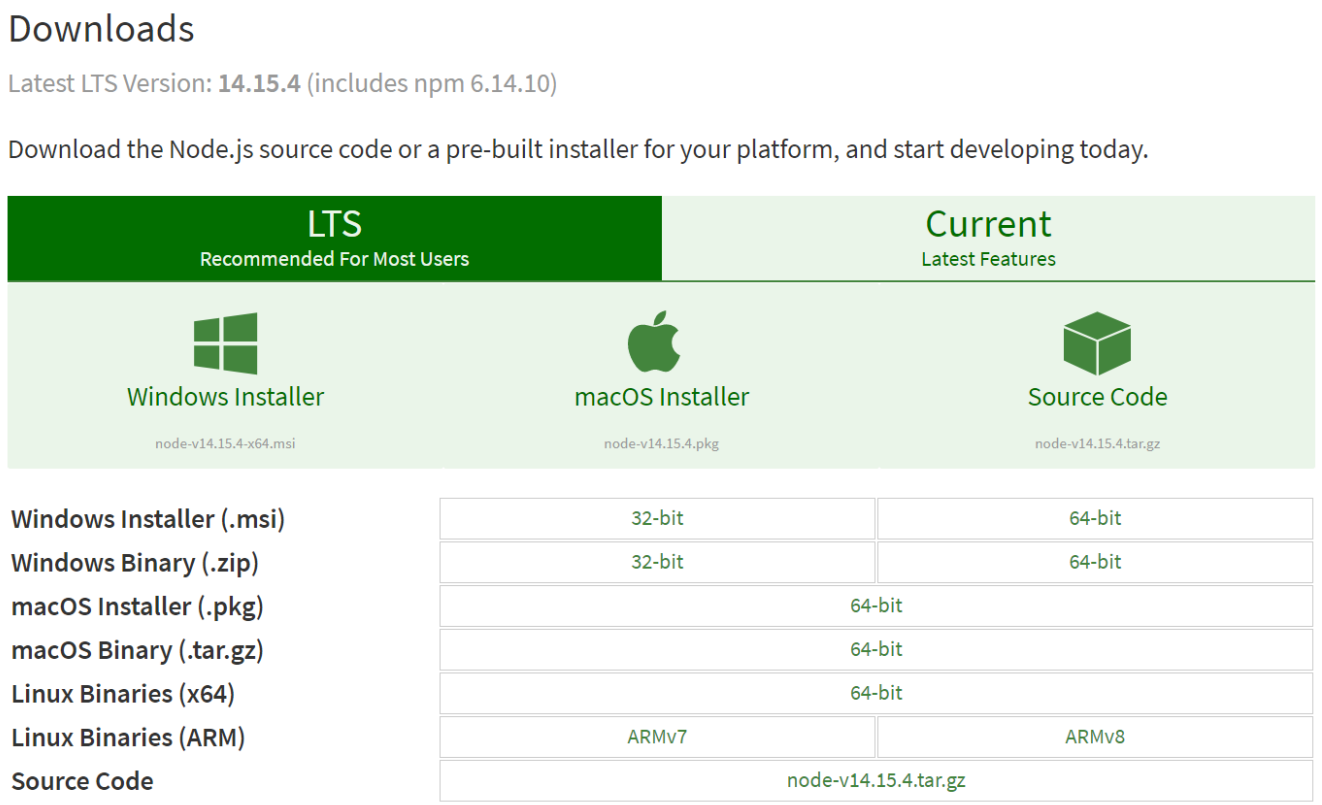
```
app.get('/example/a', (req, res) => {
  res.send('Hello from A!')
})
```

```
const cb0 = function (req, res, next) {
  console.log('CB0')
  next()
}

const cb1 = function (req, res, next) {
  console.log('CB1')
  next()
}

const cb2 = function (req, res) {
  res.send('Hello from C!')
}

app.get('/example/c', [cb0, cb1, cb2])
```

Installation and Example

Installation

- Node.js can be installed in different ways.
  - Download from Node.js website: https://nodejs.org/en/download/



Downloads

Latest LTS Version: **14.15.4** (includes npm 6.14.10)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users		Current Latest Features
Windows Installer node-v14.15.4-x64.msi	macOS Installer node-v14.15.4.pkg	Source Code node-v14.15.4.tar.gz

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit	
macOS Binary (.tar.gz)	64-bit	
Linux Binaries (x64)	64-bit	
Linux Binaries (ARM)	ARMv7	ARMv8
Source Code	node-v14.15.4.tar.gz	

Installation

- Node.js can be installed in different ways.
  - On MacOS: from CLI

```
brew install node
```

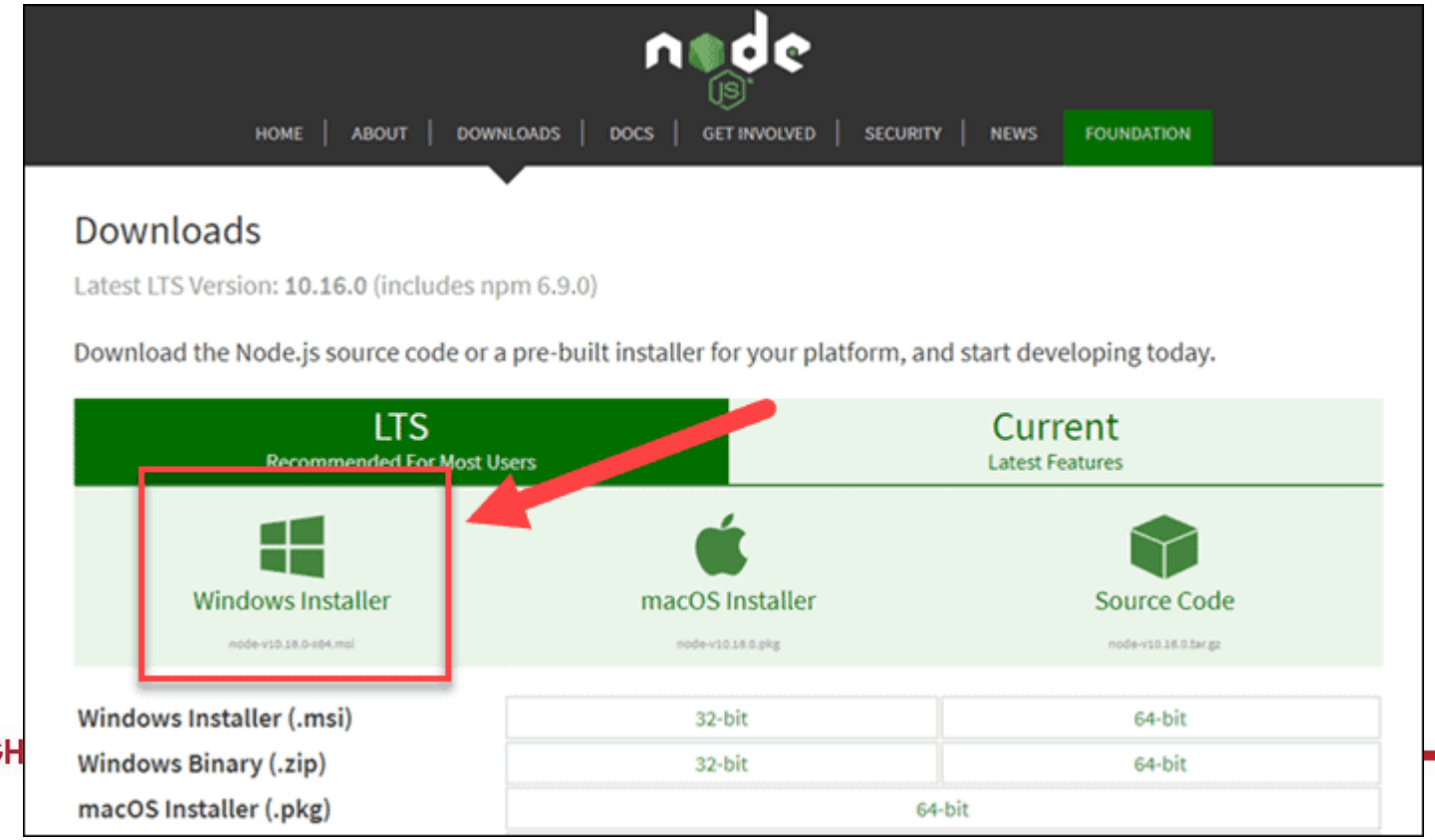
  - On Ubuntu

```
nvm install node
```

Install Node.js and NPM on Windows

Step 1: Download Node.js Installer

navigate to https://nodejs.org/en/download/. Click the **Windows Installer** button to download the latest default version.



Install Node.js and NPM on Windows

Step 2: Install Node.js and NPM from Browser

- Browse to the location where you have saved the file and double-click it to launch > **Run** > **Next**
- On the next screen, review the license agreement. Click **Next** if you agree to the terms and install the software.
- The installer will prompt you for the installation location. Leave the default location, unless you have a specific need to install it somewhere else – then click **Next**.
- The wizard will let you select components to include or remove from the installation. Again, unless you have a specific need, accept the defaults by clicking **Next**.
- Finally, click the **Install** button to run the installer. When it finishes, click **Finish**.

Install Node.js and NPM on Windows

Step 2: Verify installation

Open a command prompt (or PowerShell), and enter the following:

```
node -v
```

The system should display the Node.js version installed on your system. You can do the same for NPM:

```
npm -v
```

Simple Nodejs Express

Simple Nodejs Project Example

Step 1: Create a folder

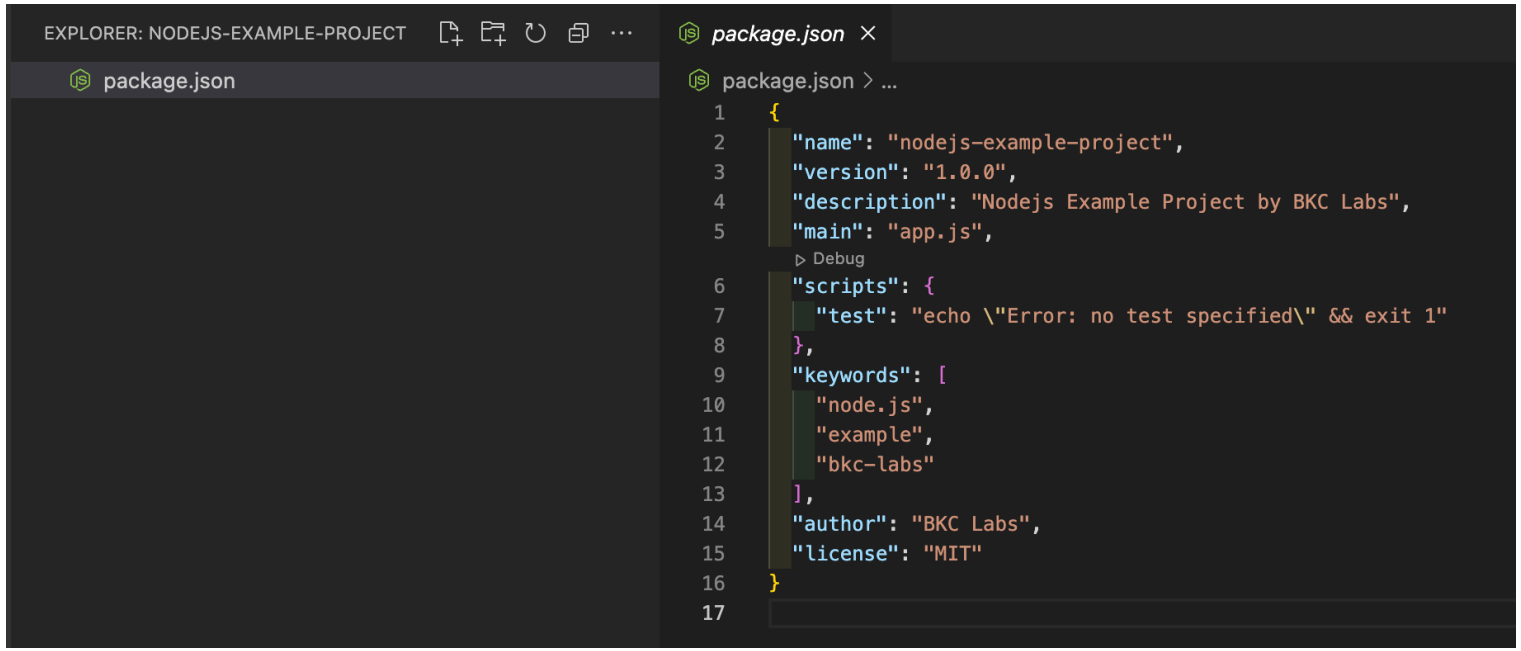
Step 2: Open it by VSCode

Step 3: Open integrated terminal: npm init


```
→  nodejs-example-project 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: (nodejs-example-project)
version: (1.0.0)
description: Nodejs Example Project by BKC Labs
entry point: (index.js) app.js
```

Simple Nodejs Project Example

There will be a package.json file generated:

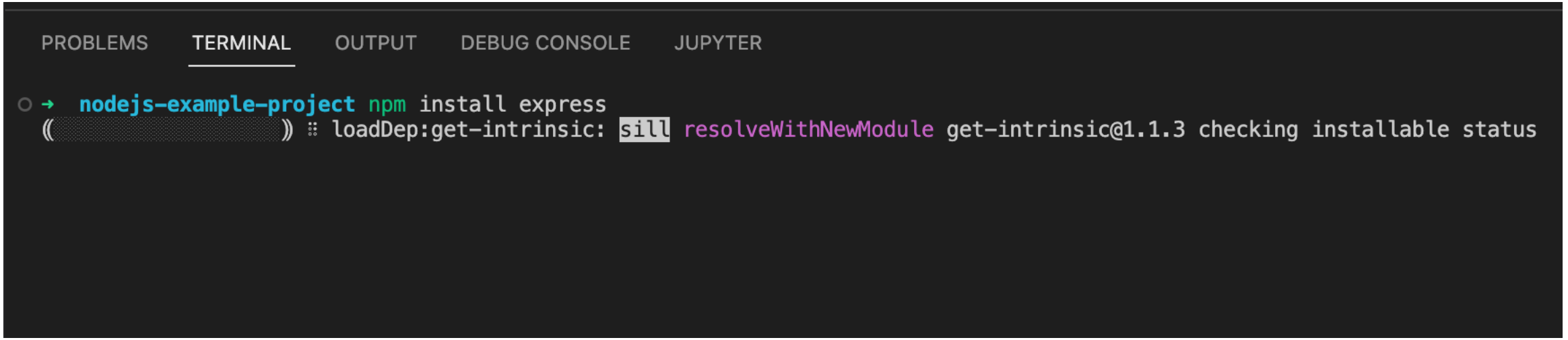


Simple Nodejs Project Example

Step 4: npm install express

Step 5: See changes in package.json

Step 6: See a package-lock.json file and a node_modules folder are generated



Simple Nodejs Project Example

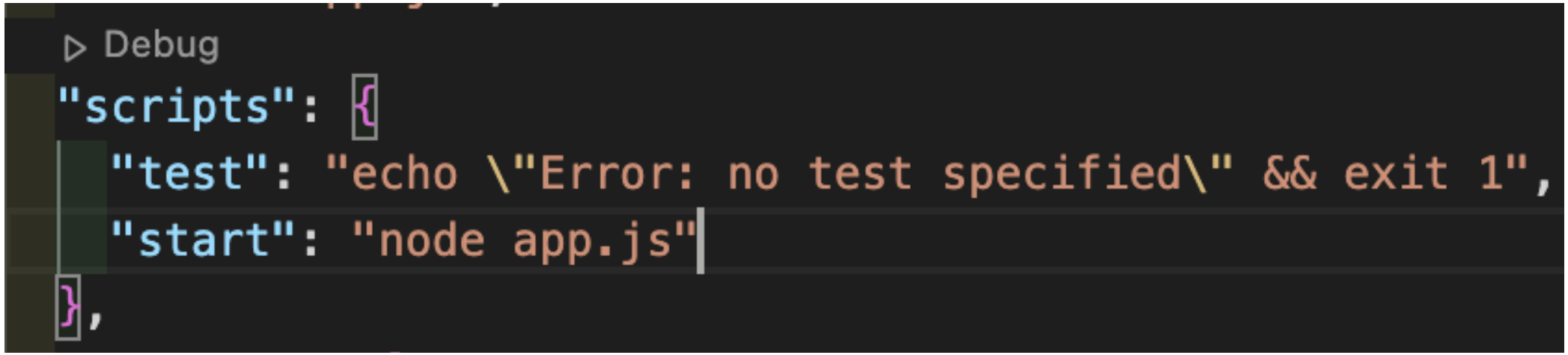
Step 7: create app.js with following code

```
1  const express = require("express");
2  const app = express();
3
4  app.get("/", (req, res) => {
5    res.send("Hello world!");
6  });
7
8  const PORT = process.env.PORT || 8000;
9
10 app.listen(PORT, () => console.log(`Application listening on port ${PORT}!`));
11
```

Simple Nodejs Project Example

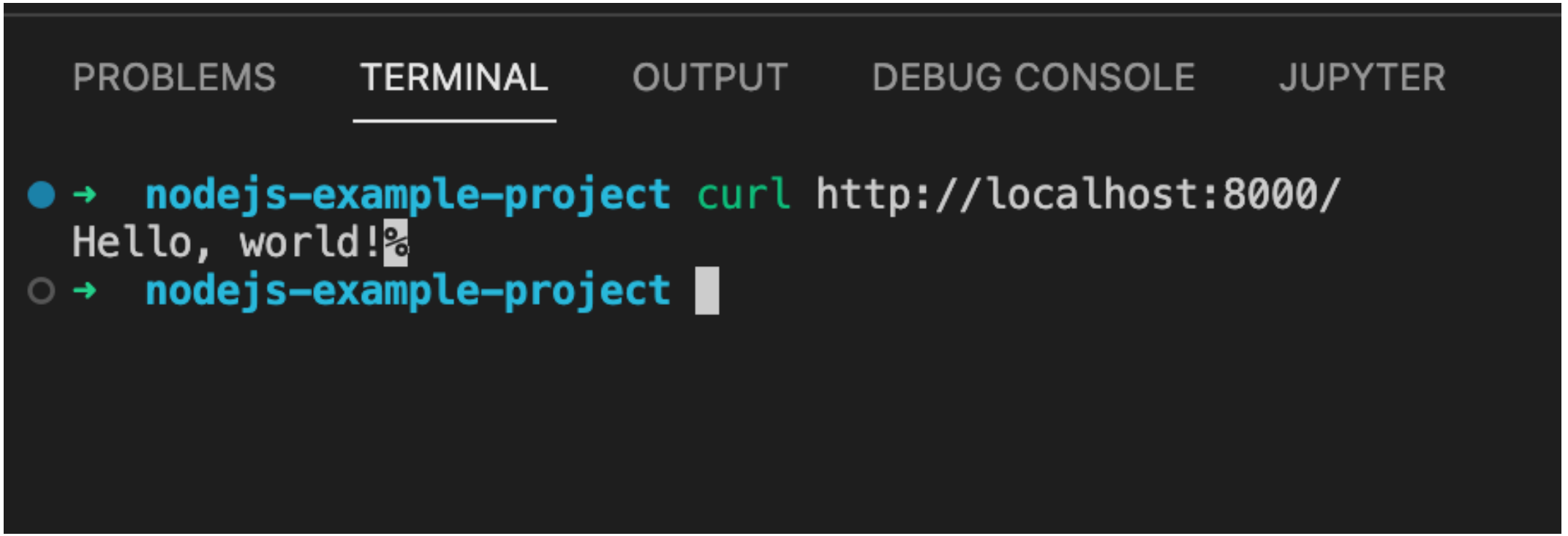
Step 8: Update package.json scripts:

Step 9: Open terminal: npm start



Simple Nodejs Project Example

Step 10: Open another terminal: curl http://localhost:8000/

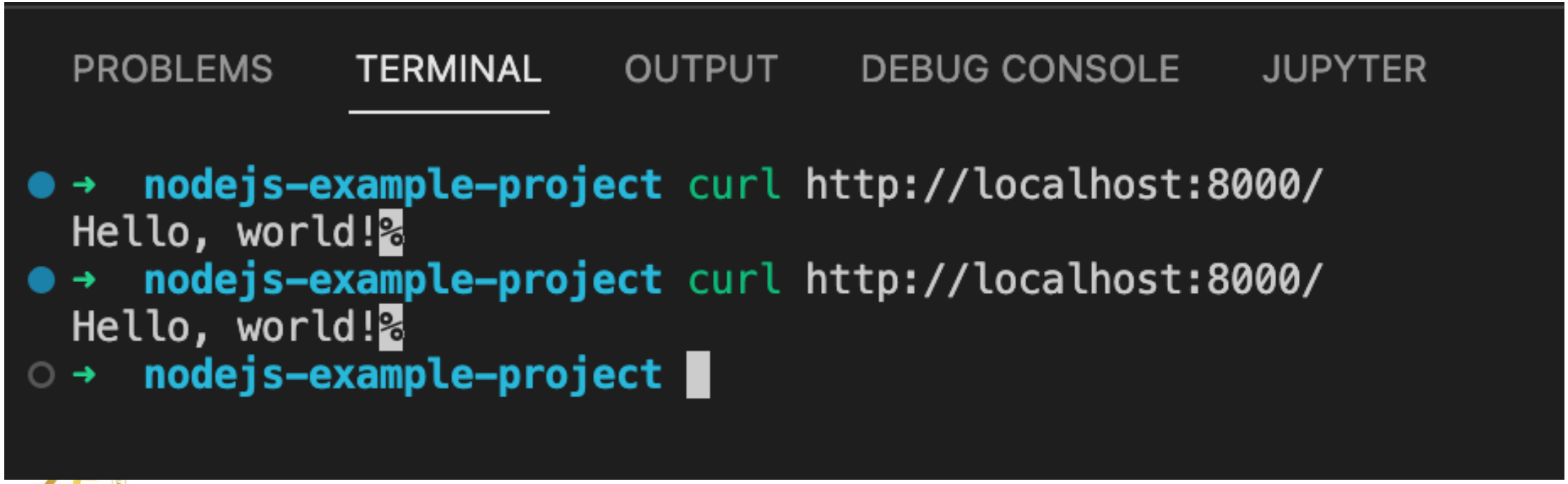


Simple Nodejs Project Example

Step 11: Edit app.js: Change to “Goodbye, world!”, save the file

Step 12: run curl one again:

→ The result has not auto updated



Simple Nodejs Project Example

Step 13: Kill current terminal running app

Step 14: npm install –D nodemon

```
PROBLEMS   TERMINAL   OUTPUT   DEBUG CONSOLE   JUPYTER

→  nodejs-example-project npm install nodemon
npm WARN nodejs-example-project@1.0.0 No repository field.

+ nodemon@2.0.20
added 33 packages from 32 contributors and audited 90 packages in 2.822s

10 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Simple Nodejs Project Example

Step 15: update package.json scripts:

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node app.js",
  "dev": "nodemon app"
```

Simple Nodejs Project Example

Step 16: npm run dev

```
PROBLEMS   TERMINAL   OUTPUT   DEBUG CONSOLE   JUPYTER

→  nodejs-example-project npm run dev

> nodejs-example-project@1.0.0 dev /Users/hoangnam/Desktop/nodejs-example-project
> nodemon app

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Application listening on port 8000!
```

Simple Nodejs Project Example

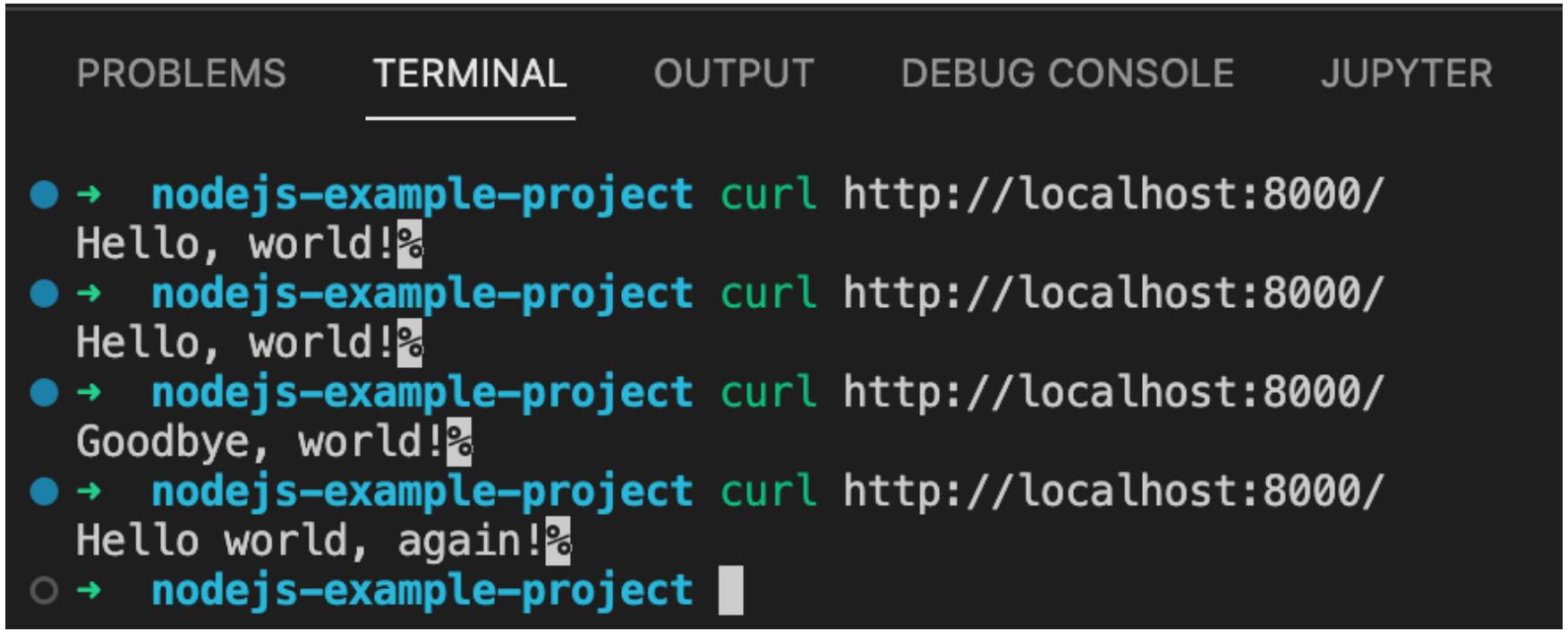
Step 17: curl http://localhost:8000/

Step 18: edit app.js: change to “Hello world, again!” and Save

Step 20: curl http://localhost:8000/

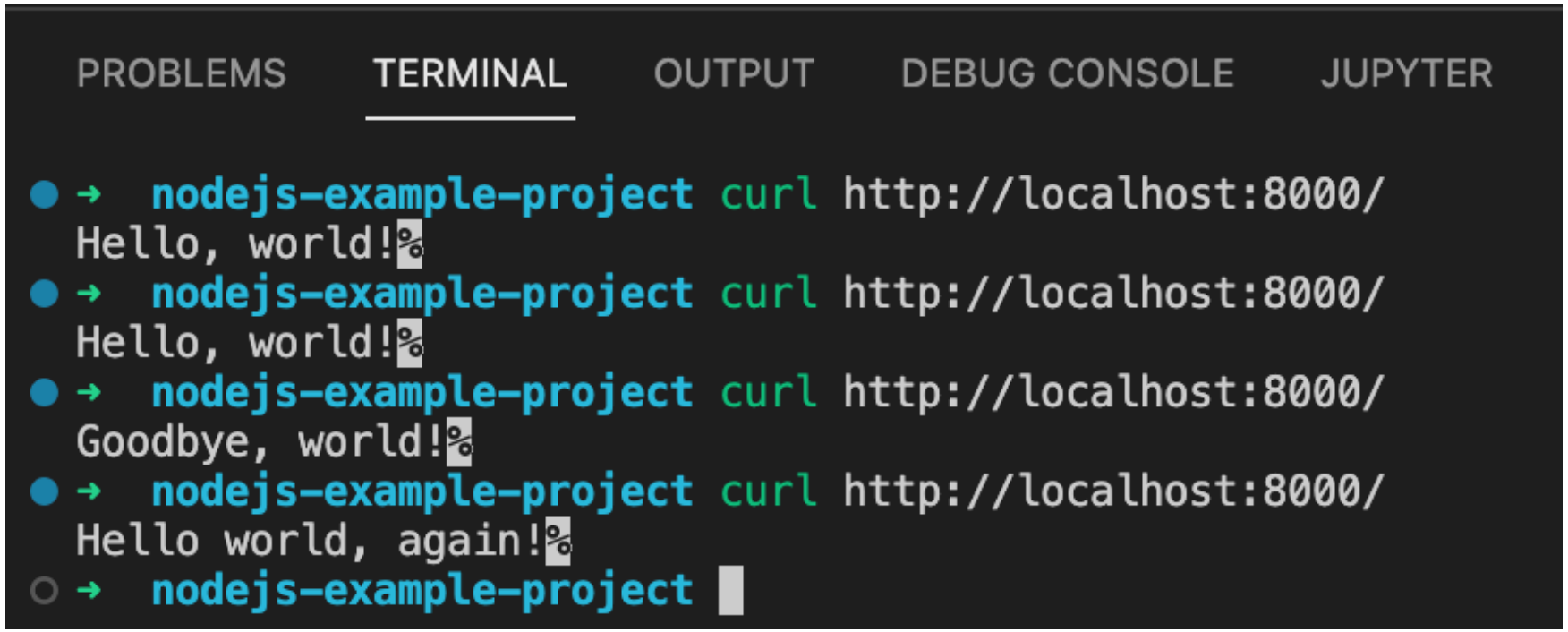
Simple Nodejs Project Example

The result has already auto updated!



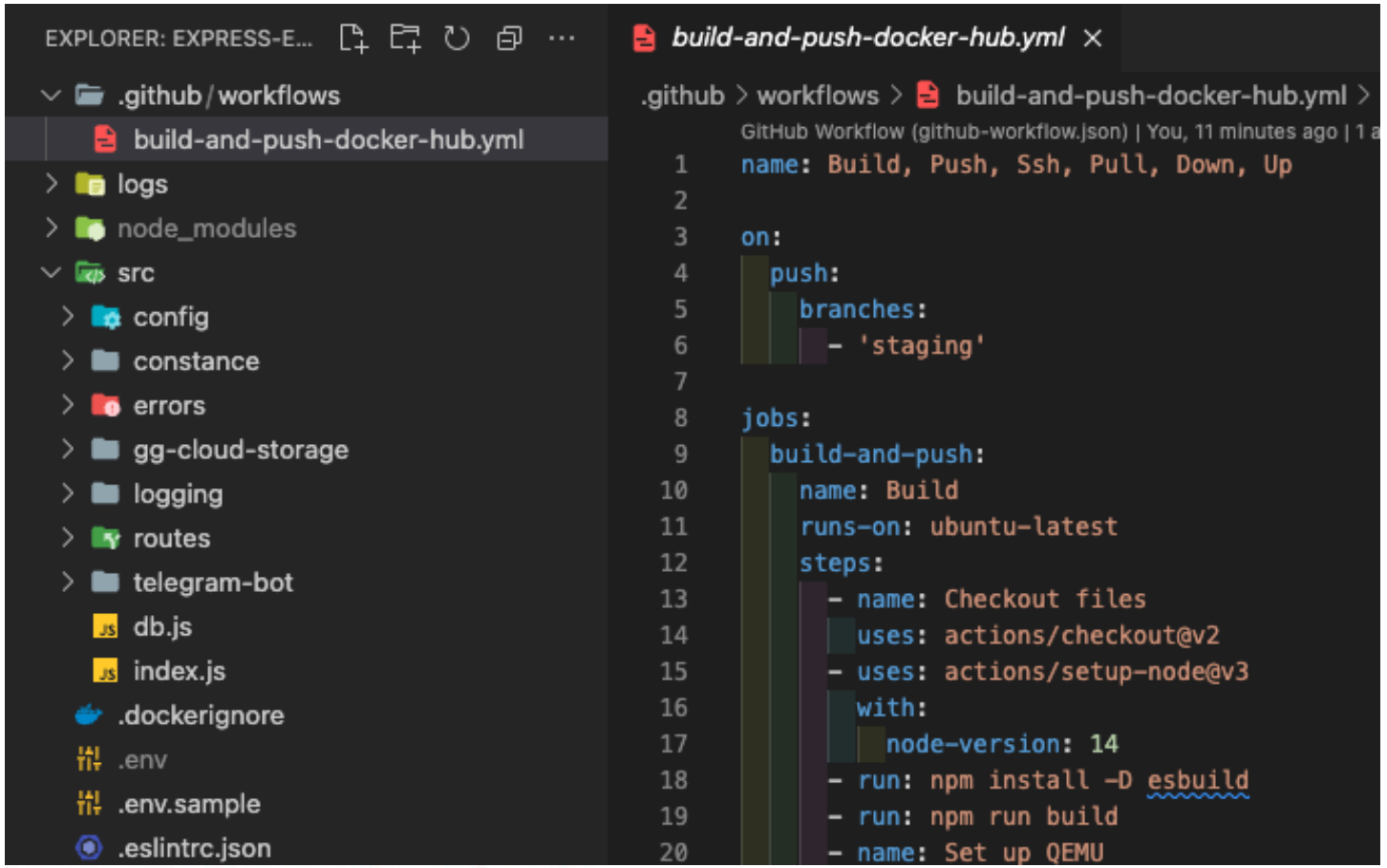
Simple Nodejs Project Example

The result has already auto updated!

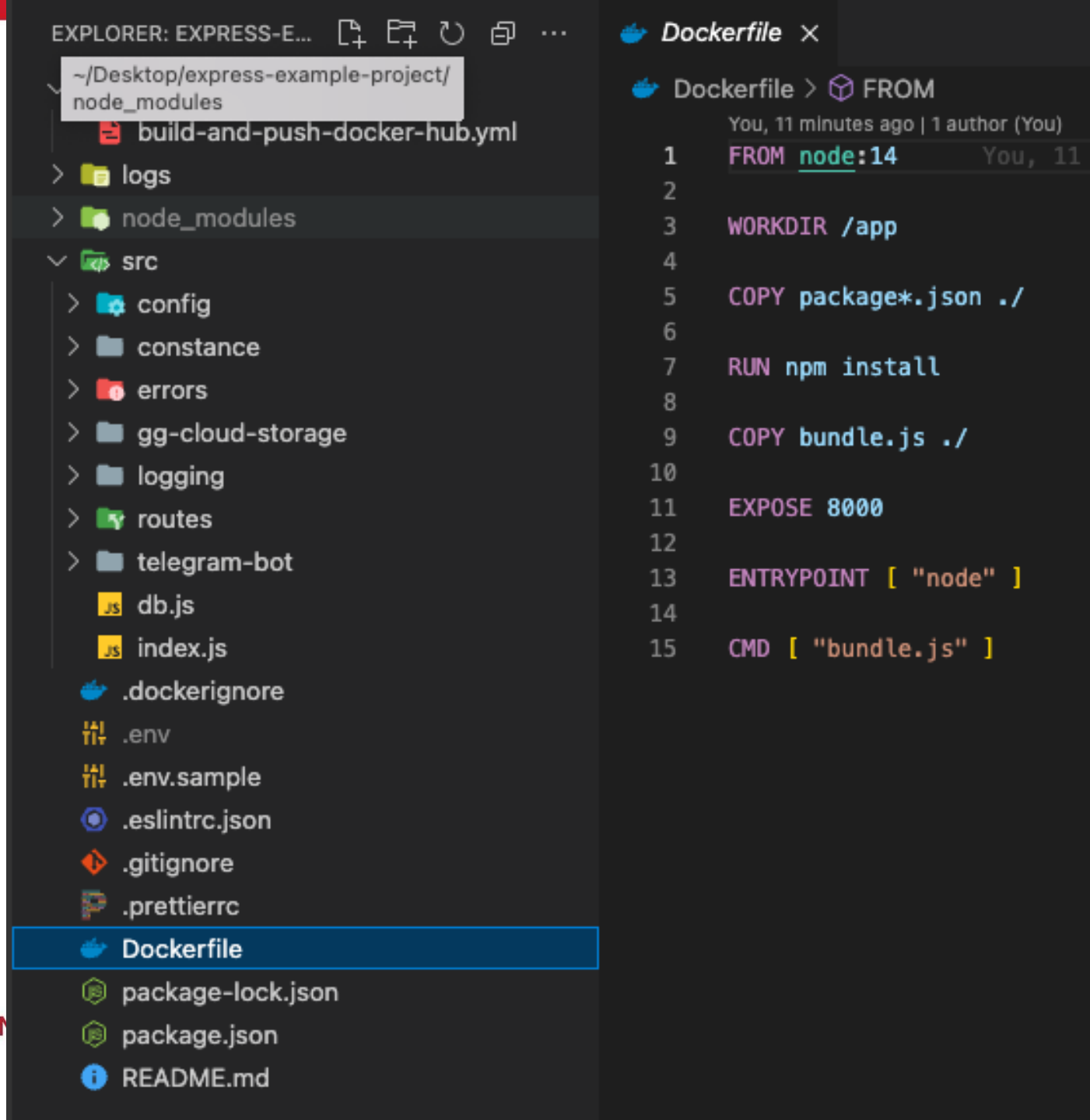


Advanced Express

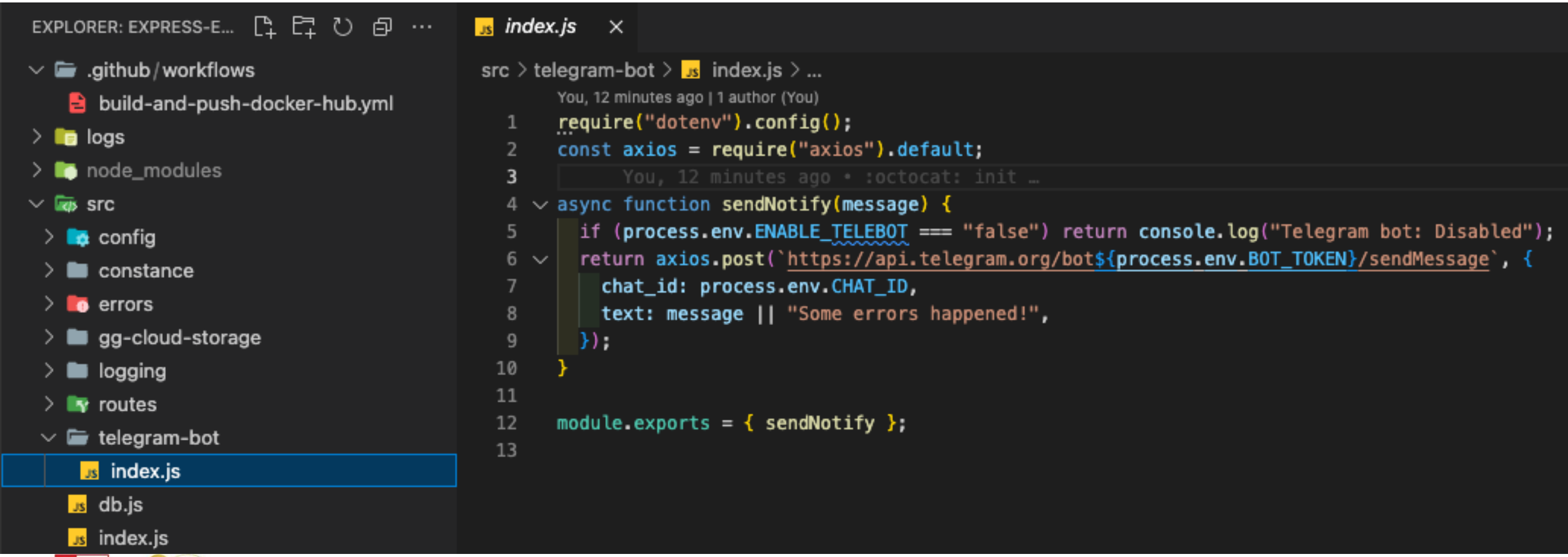
CI/CD


EXPLORER: EXPRESS-E...
.github / workflows
build-and-push-docker-hub.yml
logs
node_modules
src
config
constance
errors
gg-cloud-storage
logging
routes
telegram-bot
db.js
index.js
.dockerignore
.env
.env.sample
.eslintrc.json
build-and-push-docker-hub.yml
.github > workflows > build-and-push-docker-hub.yml
GitHub Workflow (github-workflow.json) | You, 11 minutes ago | 1 a
1 name: Build, Push, Ssh, Pull, Down, Up
2
3 on:
4 push:
5 branches:
6 - 'staging'
7
8 jobs:
9 build-and-push:
10 name: Build
11 runs-on: ubuntu-latest
12 steps:
13 - name: Checkout files
14 uses: actions/checkout@v2
15 - uses: actions/setup-node@v3
16 with:
17 node-version: 14
18 - run: npm install -D esbuild
19 - run: npm run build
20 - name: Set up QEMU

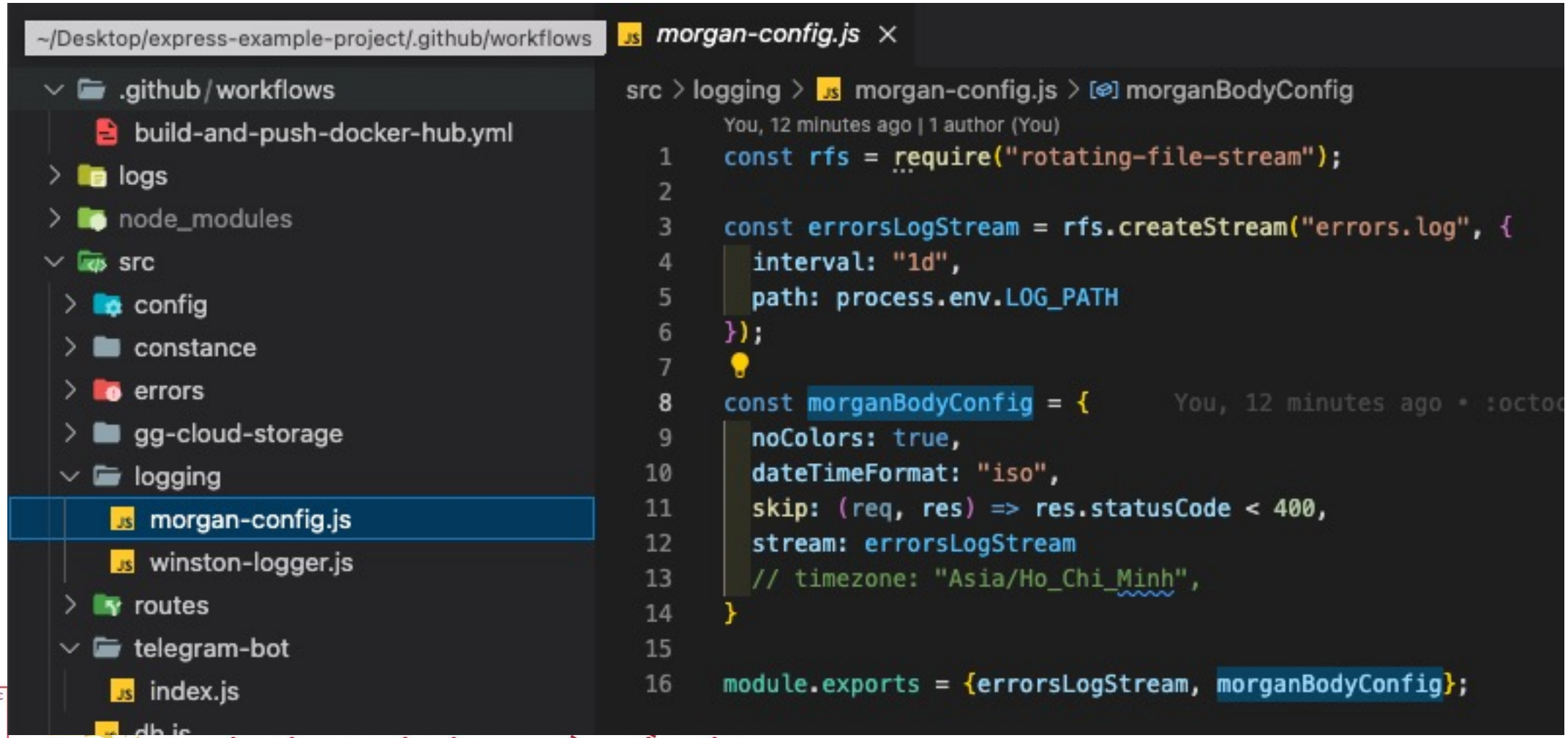
Dockerization



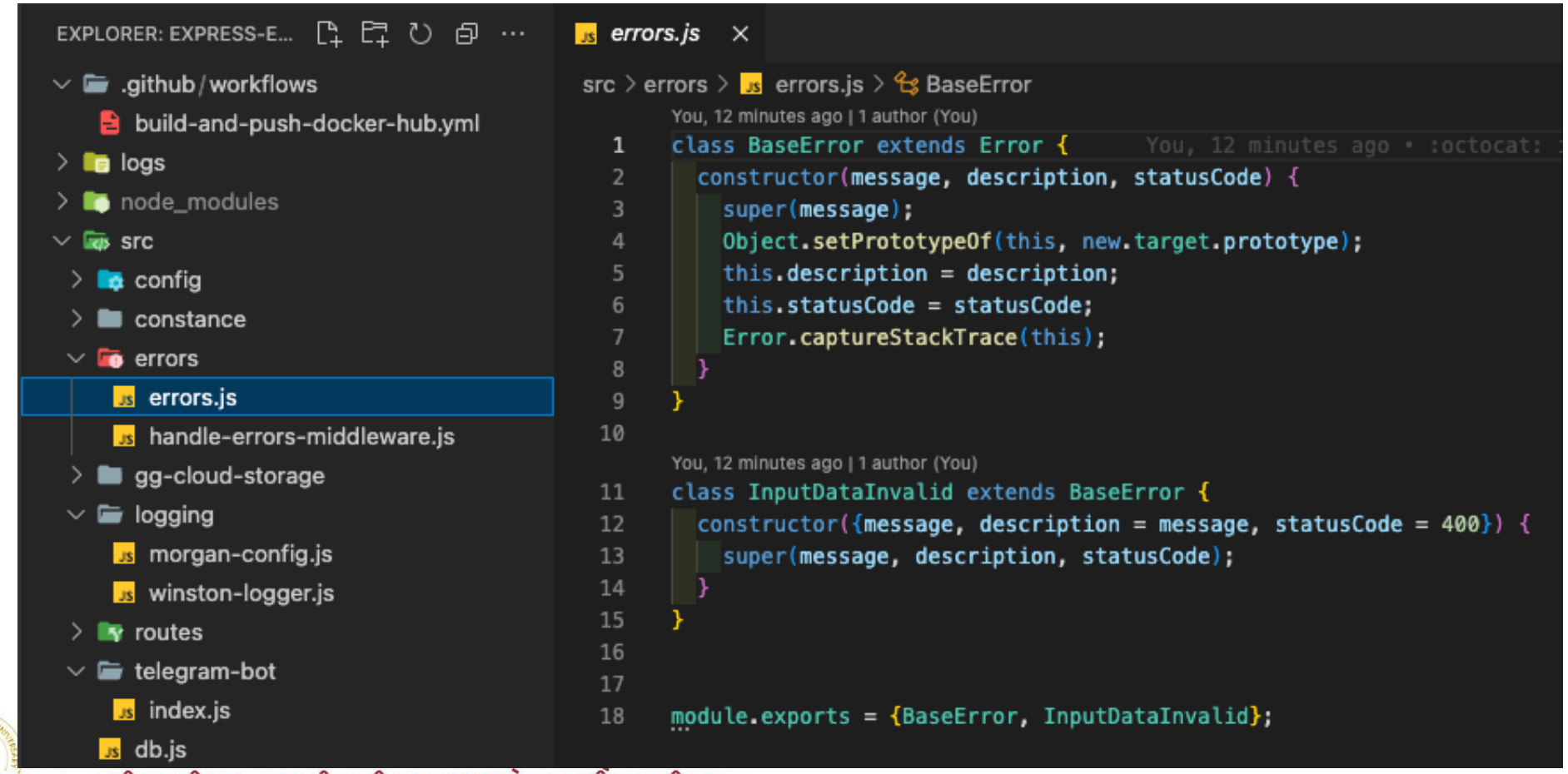
Notifying


EXPLORER: EXPRESS-E...
.github/workflows
build-and-push-docker-hub.yml
logs
node_modules
src
config
constance
errors
gg-cloud-storage
logging
routes
telegram-bot
index.js
db.js
index.js
index.js
src > telegram-bot > index.js > ...
You, 12 minutes ago | 1 author (You)
require("dotenv").config();
const axios = require("axios").default;
You, 12 minutes ago • :octocat: init …
async function sendNotify(message) {
if (process.env.ENABLE_TELEBOT === "false") return console.log("Telegram bot: Disabled");
return axios.post(`https://api.telegram.org/bot${process.env.BOT_TOKEN}/sendMessage`, {
chat_id: process.env.CHAT_ID,
text: message || "Some errors happened!",
});
}
module.exports = { sendNotify };

Logging


~/Desktop/express-example-project/.github/workflows
morgan-config.js
.github / workflows
build-and-push-docker-hub.yml
logs
node_modules
src
config
constance
errors
gg-cloud-storage
logging
morgan-config.js
winston-logger.js
routes
telegram-bot
index.js
src > logging > morgan-config.js > morganBodyConfig
You, 12 minutes ago | 1 author (You)
const rfs = require("rotating-file-stream");
const errorsLogStream = rfs.createStream("errors.log", {
interval: "1d",
path: process.env.LOG_PATH
});
const morganBodyConfig = {
noColors: true,
dateTimeFormat: "iso",
skip: (req, res) => res.statusCode < 400,
stream: errorsLogStream
// timezone: "Asia/Ho_Chi_Minh",
}
module.exports = {errorsLogStream, morganBodyConfig};

Error Handling



GG Cloud Storage

Modularization routes