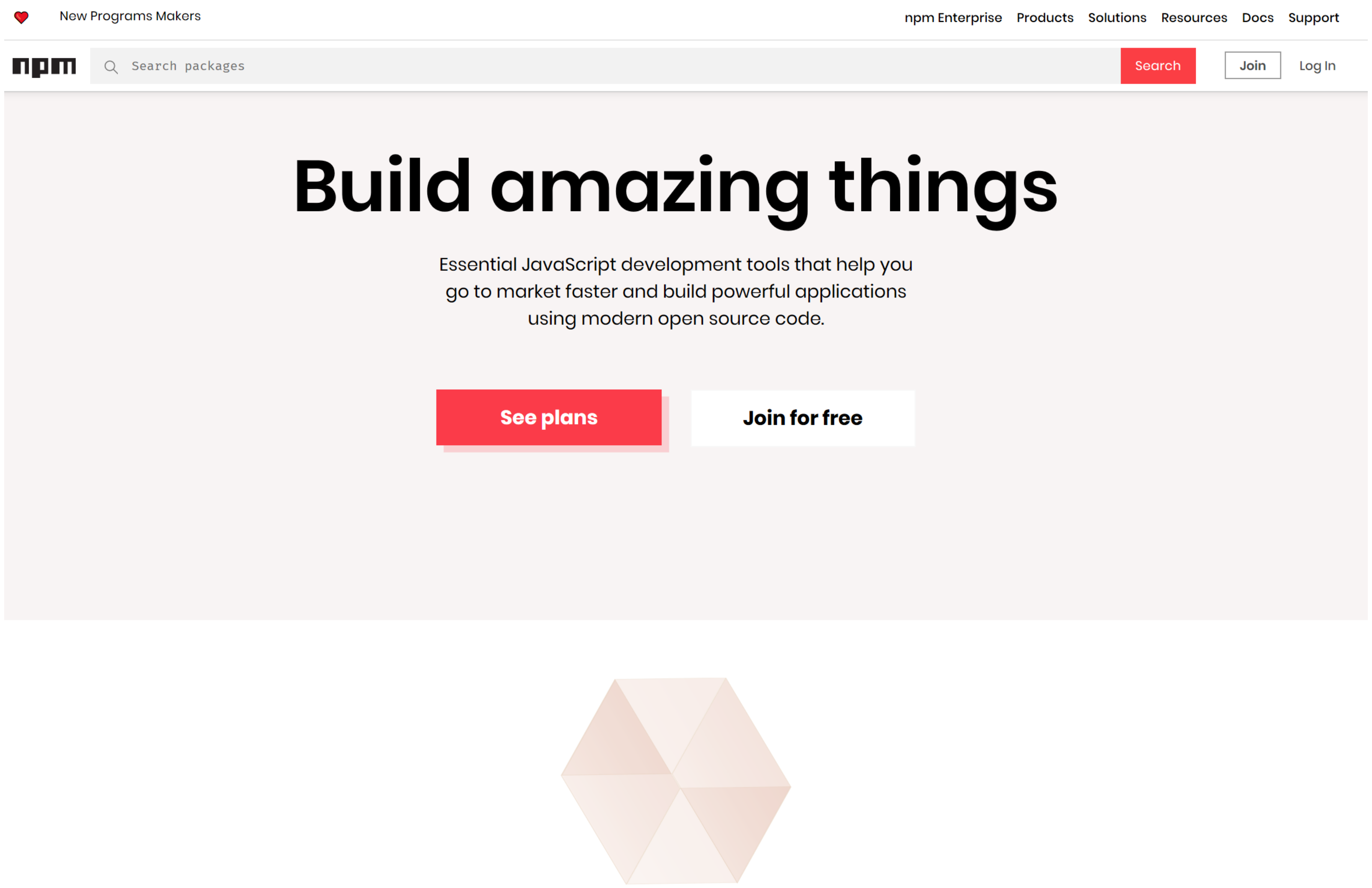
Node Modules



Build amazing things

Essential JavaScript development tools that help you go to market faster and build powerful applications using modern open source code.

Node Modules

- Node has a small core API
- Most applications depend on third party modules
- Curated in online registry called the Node Package Manager system (NPM)
- NPM downloads and installs modules, placing them into a **node_modules** folder in your current folder.

NPM init

- You can use NPM to manage your node projects
- Run the following in the root folder of your app/project:

```
npm init
```

- This will ask you a bunch of questions, and then create a package.json for you.
- It attempts to make reasonable guesses about what you want things to be set to, and then writes a package.json file with the options you've selected.

Node Modules

- To install NPM modules, navigate to the application folder and run “npm install”. For example :
  npm install express --save
- This installs into a “**node_module**” folder in the current folder.
- The **--save** bit updates your package.json with the dependency
- To use the module in your code, use:
  ```
  import express from 'express';
  ```
- This loads express from local **node_modules** folder.

Global Node Modules

- Sometimes you may want to access modules from the shell/command line.
- You can install modules that will execute globally by including the **'-g'.**
- Example, **Grunt** is a Node-based software management/build tool for Javascript.

```
npm install -g grunt-cli
```

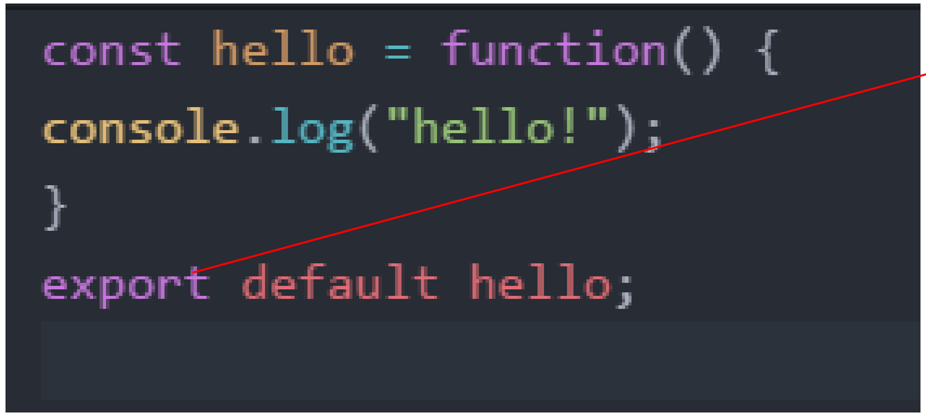
- This puts the **“grunt”** command in the system path, allowing it to be run from any directory.

NPM Common Commands

Common npm commands:
– **npm init** *initialize a package.json file*
– **npm install <package name> -g** *install a package, if –*
g option is given package will be installed globally, ***--save***
and ***--save-dev*** *will add package to your dependencies*
– **npm install** *install packages listed in package.json*
– **npm ls –g** *listed local packages (without –g) or global*
packages (with –g)
– **npm update <package name>** *update a package*

Creating your own Node Modules

- We want to create the following module called **custom_hello.js:**



```
const hello = function() {
console.log("hello!");
}
export default hello;
```

Export defines what import returns

- To access in our application, **index.js:**

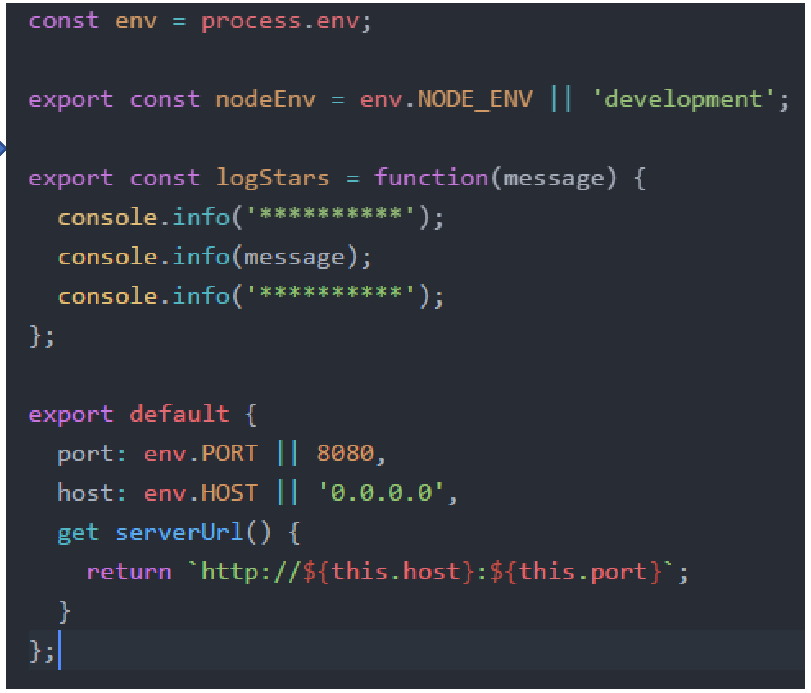
```
import hello from './custom_hello';
hello();
```

Creating your own Node Modules

- Exporting Multiple Properties

- Accessing in other scripts

Config.js

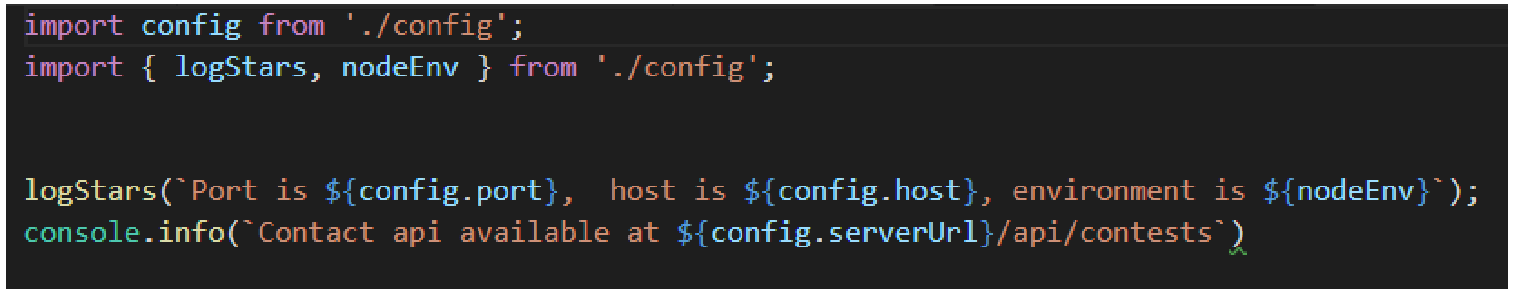


```
const env = process.env;

export const nodeEnv = env.NODE_ENV || 'development';

export const logStars = function(message) {
  console.info('**********');
  console.info(message);
  console.info('**********');
};

export default {
  port: env.PORT || 8080,
  host: env.HOST || '0.0.0.0',
  get serverUrl() {
    return `http://${this.host}:${this.port}`;
  }
};
```

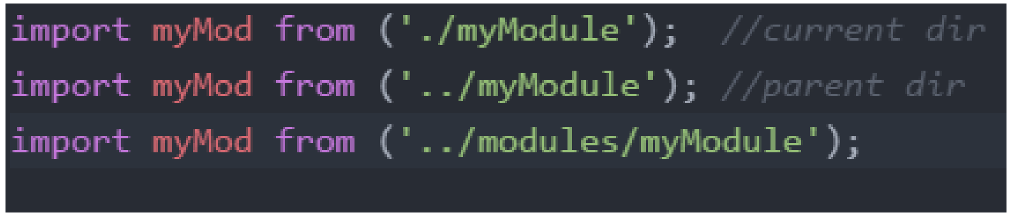


```
import config from './config';
import { logStars, nodeEnv } from './config';


logStars(`Port is ${config.port},  host is ${config.host}, environment is ${nodeEnv}`);
console.info(`Contact api available at ${config.serverUrl}/api/contests`)
```

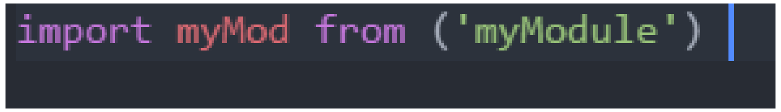
The import search

- Import searches for modules based on path specified:



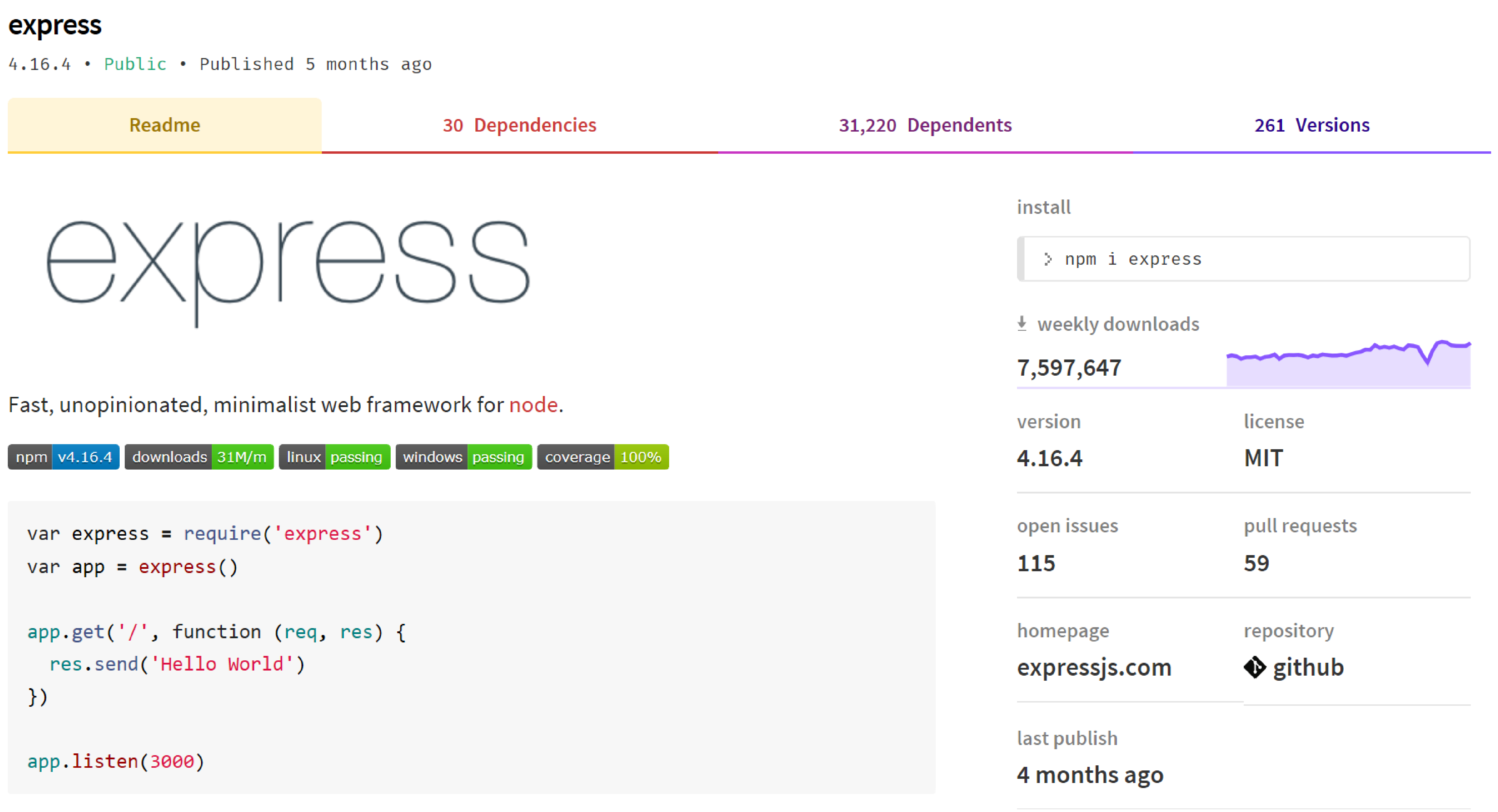
```
import myMod from ('./myModule');  //current dir
import myMod from ('../myModule'); //parent dir
import myMod from ('../modules/myModule');
```

- Just providing the module name will search in **node_modules** folder



```
import myMod from ('myModule')
```

The Express Package



express

4.16.4 • Public • Published 5 months ago

express

Fast, unopinionated, minimalist web framework for node.

```
var express = require('express')
var app = express()

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

app.listen(3000)
```

install

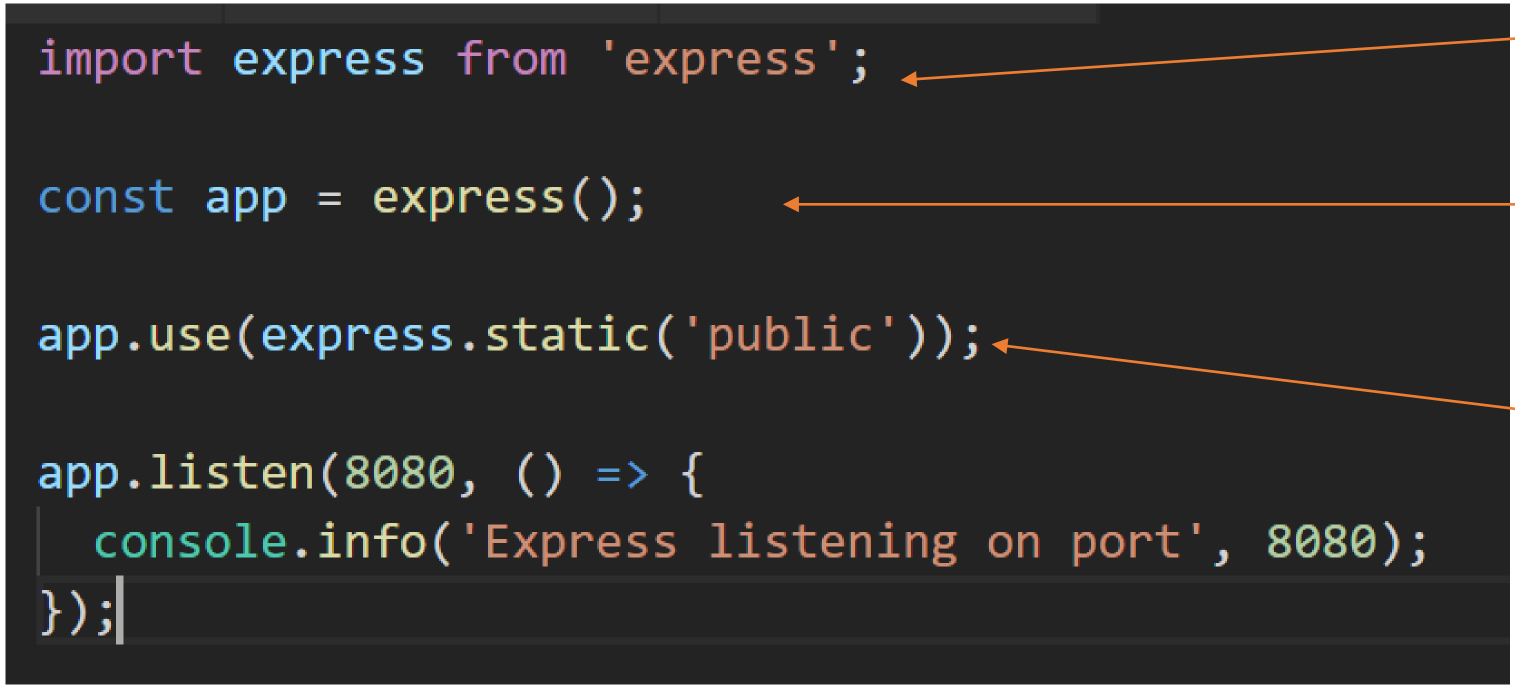
```
> npm i express
```

weekly downloads	7,597,647
version	4.16.4
license	MIT
open issues	115
pull requests	59
homepage	expressjs.com
repository	github
last publish	4 months ago

What Express Gives Us...

- Parses arguments and headers
- Easy Routing
    - Route a URL to a callback function
- Sessions
- File Uploads
- Middleware…

Simple Express App (index.js)



```
import express from 'express';

const app = express();

app.use(express.static('public'));

app.listen(8080, () => {
  console.info('Express listening on port', 8080);
});
```

Loads Express module

Instantiates Express server

Define static content for HTTP GET

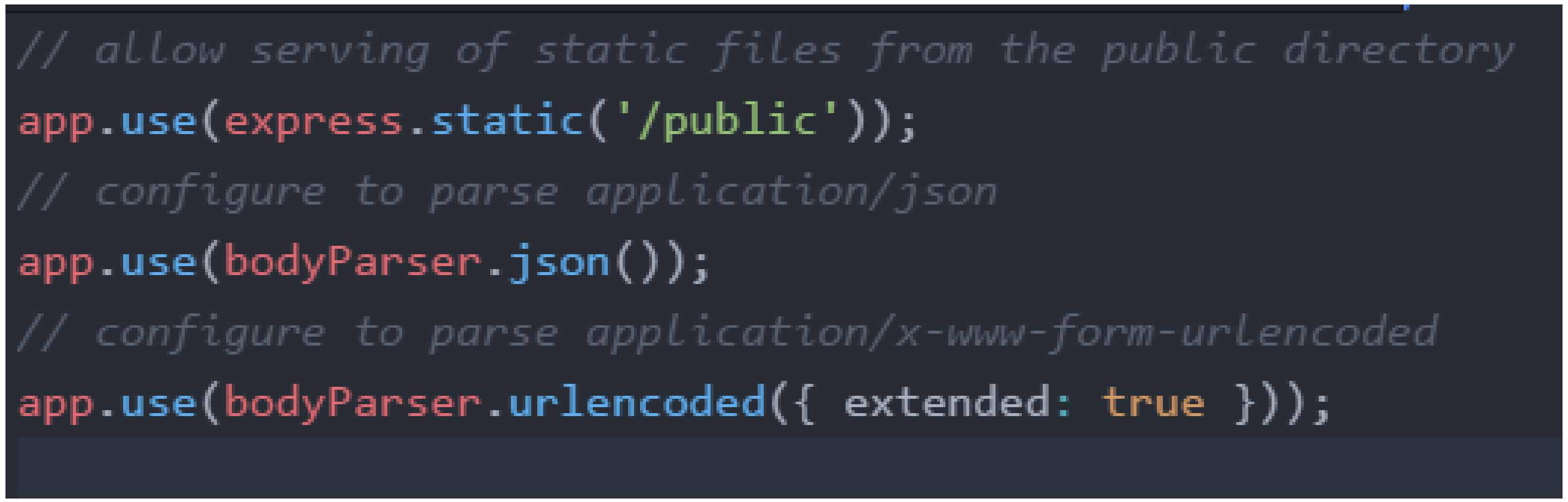
Getting Started with Express

- Installing Express

```
[local install] C:\> npm install express --save
[global install] C:\> npm install express -g
```

Express Configuration

Express allows you to easily configure your web app behaviour...



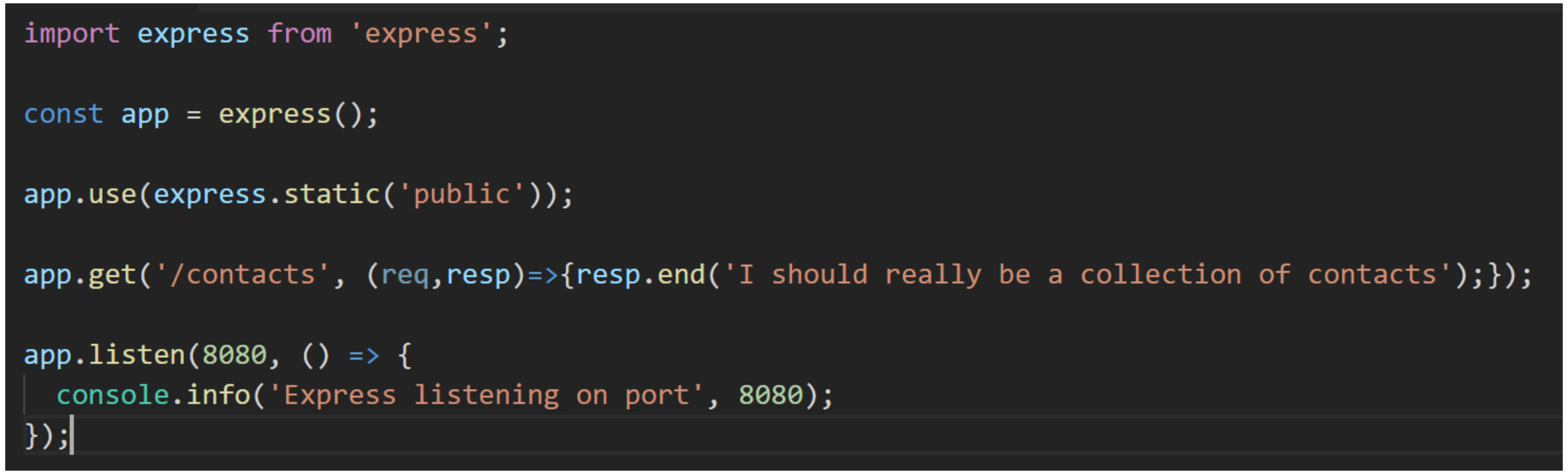
```
// allow serving of static files from the public directory
app.use(express.static('/public'));
// configure to parse application/json
app.use(bodyParser.json());
// configure to parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));
```

(/*)?

Routing Examples

Syntax follows the pattern:

App.[verb](path, (req,res)=>{});



```
import express from 'express';

const app = express();

app.use(express.static('public'));

app.get('/contacts', (req,resp)=>{resp.end('I should really be a collection of contacts');});

app.listen(8080, () => {
  console.info('Express listening on port', 8080);
});
```

```
// Other Route examples
app.post('/contacts', createContact);
app.get('/app/:app', routes.getapp);

//Catch-all
app.all('/private(/*)?', requiresLogin);
```

HTTP POST request

Parametised URL. Accepts :app route argument

Catch-all – works for all HTTP verbs

Node Applications Structure

Structuring Node Apps

- Node Server Code needs to be structured
  - Manage code base
  - Keeps code maintainable
  - Nodes packaging system supports this approach
- Typical Node.js application code:
  - main app code
  - api implementation code
  - helper code

Example Approach:

- Use a “project root” folder is the top level and contains the “entry point” or main server code
  - Always run npm in this folder to ensure just one node_modules folder
  - Use a **public** folder within the node folder for any static content

Basic Node App Structure

```
/projectroot/                    → Root of your actual application
      package.json               → Tells Node and NPM what packages are required
      readme.md
      index.js                   → The main entry point for the Express application
      public/
                /images
                /stylesheets
                /scripts         Static content (if you need it)
                index.html
      node_modules/
      api/
                                 → Output directory for all NPM installations
```