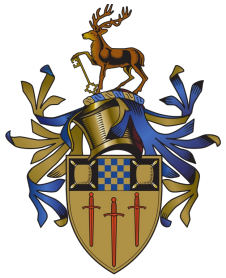


UNIVERSITY OF SURREY

INTRODUCTION TO NODE.JS AND WEB DEVELOPMENT

Lecture 9

Joe Appleton

CSEE

LAST WEEK

- We considered the foundations of how the internet works
- We created some simple JavaScript programs. We ran these programs in the Node.js environment.
  - However, we did not use any Node.js features. We just used JavaScript features.

THIS WEEK

- This week, we will be learning about Node.js features, and why we need them
- Specifically, we will consider:
  - ✓ How to create simple web applications
  - ✓ Using templating languages
  - ✓ Using a package manager
  - ✓ Using a web framework

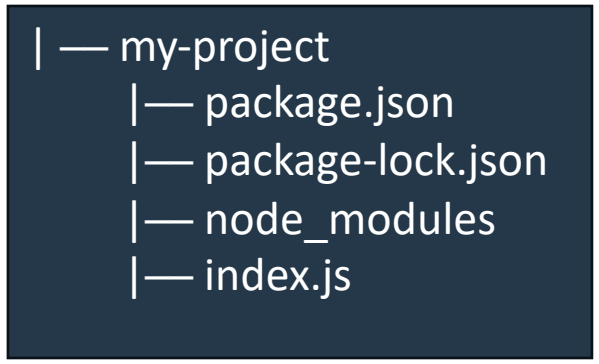
BY THE END OF THE WEEK

You'll be able to complete the following assessment tasks:

- ✓ 8. Application Structure [3 Marks]
- ✓ 10. A Minimum of 4 User-facing Routes Served [4 marks]
- ✓ 11. Construct EJS Views for the 4 Routes [3 marks]

THE ANATOMY OF A NODE PROJECT

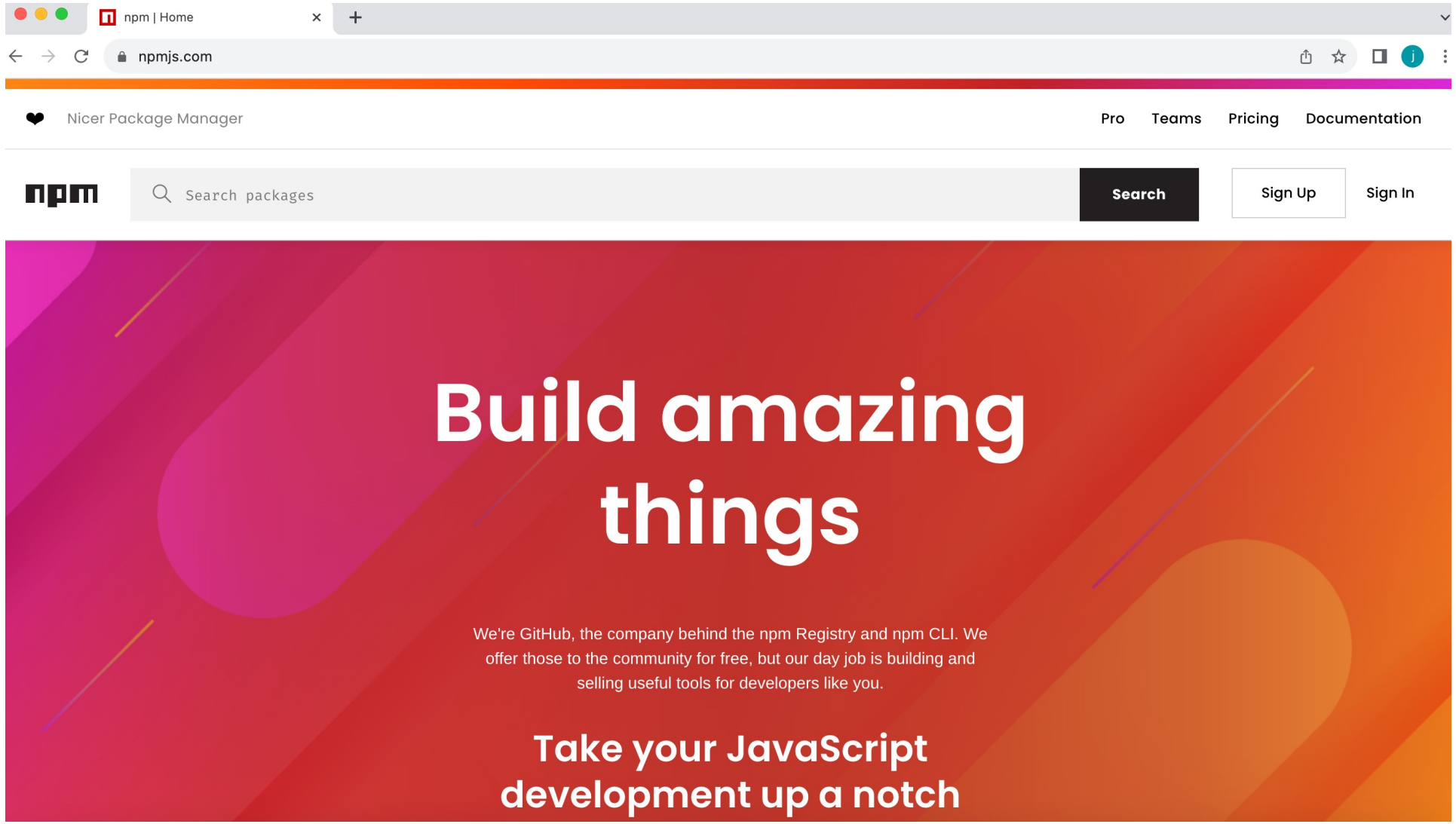
A TYPICAL NODE.JS PROJECT



- A node project is a folder that contains a **package.json** file.
- The **package.json** file contains information about the project, including the packages it uses.
- A **package.lock** file contains information about the packages that are installed in the project
- There should be at least one JavaScript that will be the entry point for your application: **e.g., index.js, app.js**

THE NODE PACKAGE MANAGER

The node package manager can be used to manage node projects
(https://www.npmjs.com/)



THE NODE PACKAGE MANAGER(NPM)

npm is:

- a package manager for JavaScript
- if you have Node.js installed, you have npm installed!
- the world's largest software registry
- it provides a command-line interface (CLI) for managing packages and other tasks
- it can be used to install, manage, and publish packages, manage dependencies, and run scripts

GETTING STARTED WITH NPM

- Ensure your have NodeJS installed
- In the directory of your project run the terminal command: **npm init**
- **You can keep pressing enter to accept the defaults.**

Terminalizer

```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
R2DJ2G5GWM:my_first_website ja0054$
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
R2DJ2G5GWM:my_first_website ja0054$
```

YOU NOW HAVE A PACKAGE.JSON FILE

```
{
  "name": "my_first_website",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
  },
  "devDependencies": {},
  "description": ""
}
```

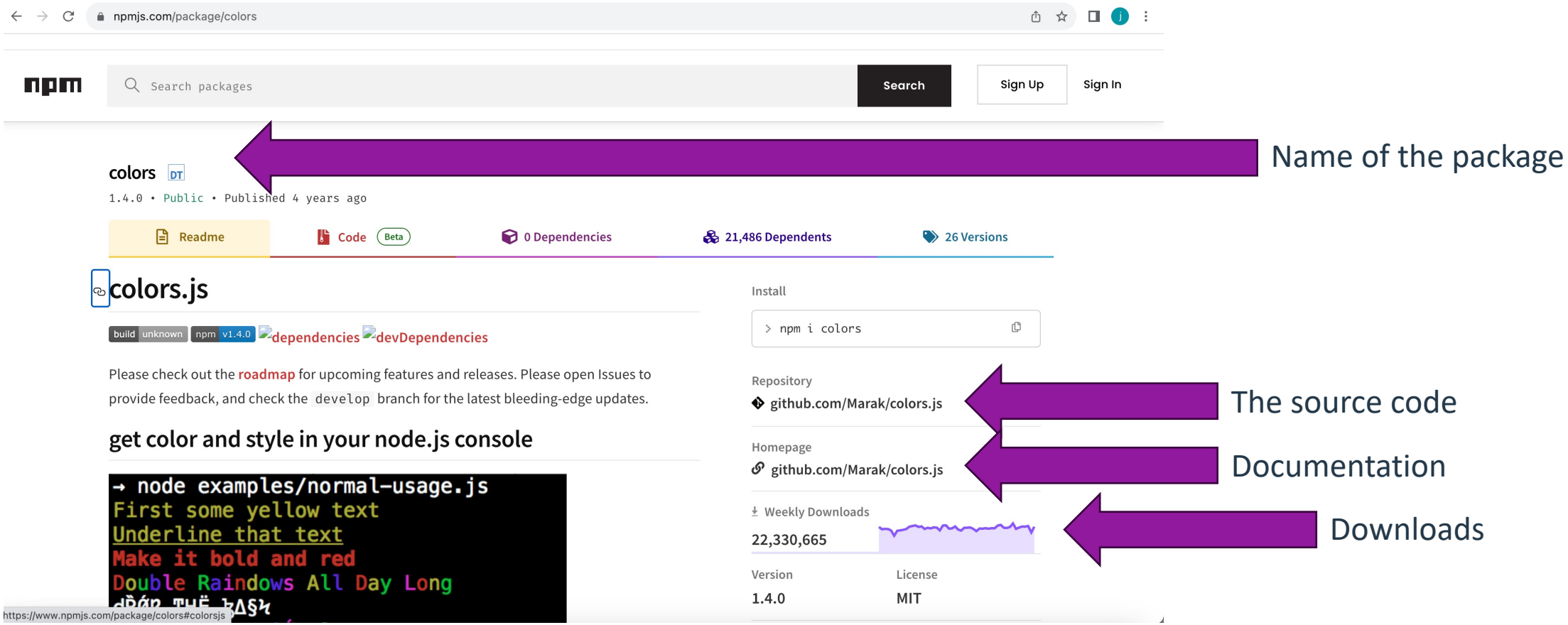
- The `package.json` file contains information about the project and is split into sections.
- For this module, there are only three sections we care about:
- The `devDependencies` field is a list of packages that the project depends on in development mode.
- The `dependencies` field is a list of packages that the project depends on.

GETTING STARTED WITH NPM

- After running “npm init”; we can now install some packages
- You can find packages by searching the https://www.npmjs.com/ Repository
- Packages are open source and free
- There are over 1.2million packages
- Many are downloaded millions of times a week

LET'S INSTALL COLORS (I)

This package lets us print different colors to the console



Name of the package

The source code

Documentation

Downloads

LET'S INSTALL COLORS.JS (II)

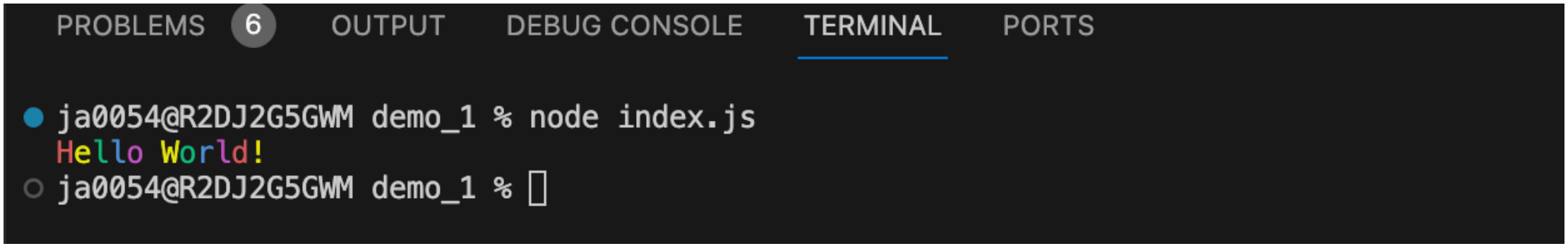
- Dependencies are installed from command line.
  - Ensure your terminal window is pointed to the root of your project.

```
> npm install colors
```

- Packages are stored in the "node_modules" folder
- Dependencies listed in the ”package.json” file
- Packages are stored in the "node_modules" folder

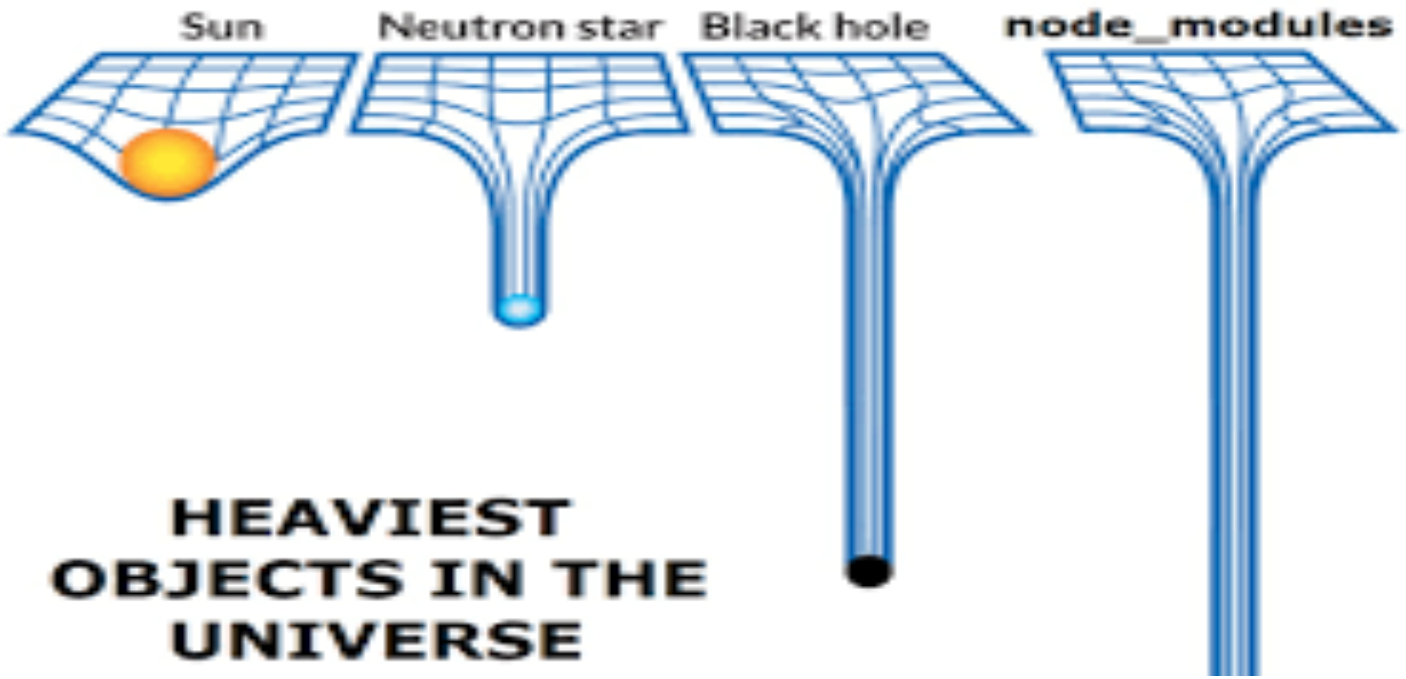
USING PACKAGES

```
const colors = require('colors’);  // save the package to a const
console.log('Hello World!'.rainbow);
```



SOME TIPS

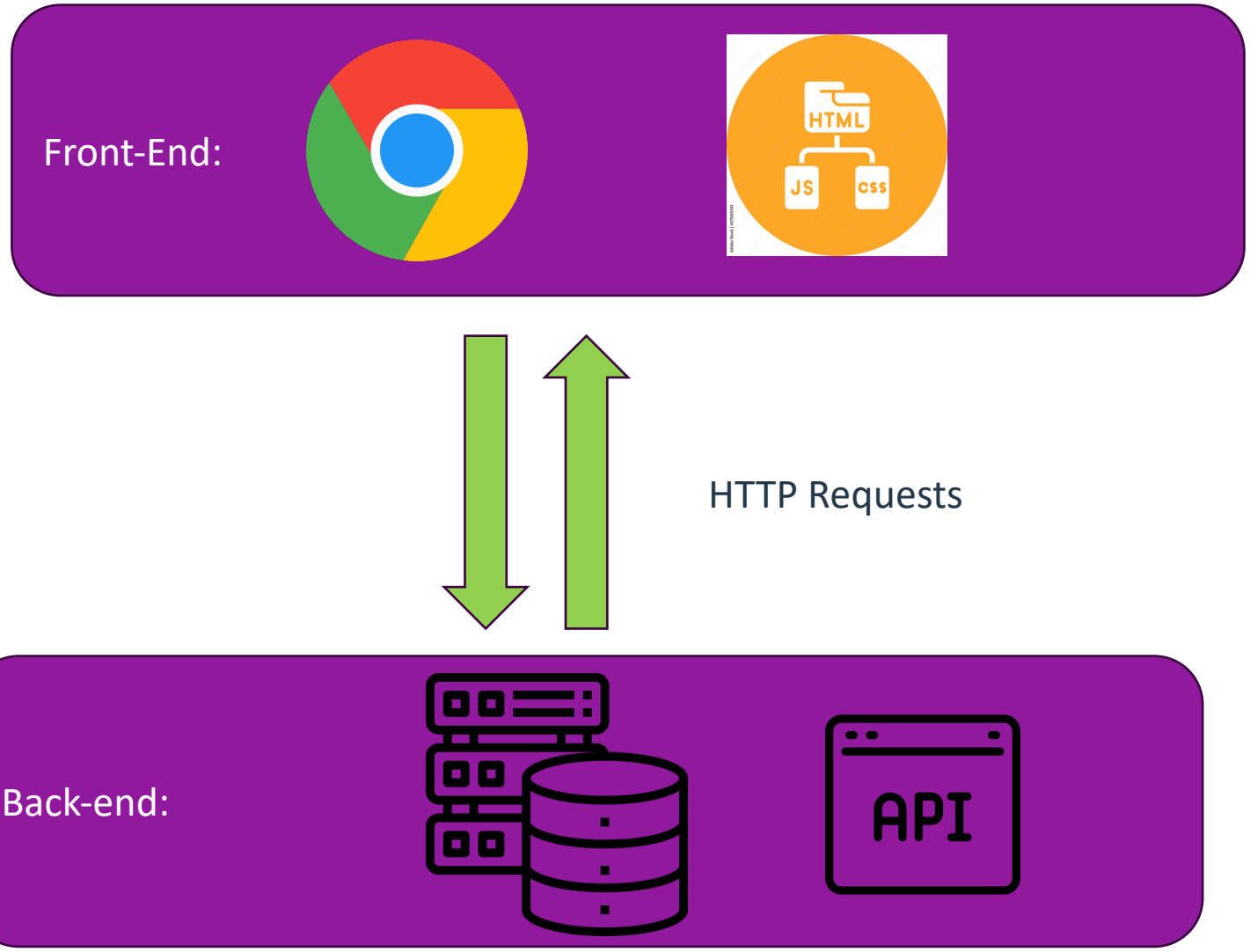
- Since packages have dependencies the ”**node_modules**” folder gets very heavy
- Use .gitignore to exclude "**node_modules**" from version control
- Delete the ”**node_modules**” folder before submitting your assessment
- We can install the dependancies by running “**npm install**”



D E M O

WEB DEVELOPMENT FRAMEWORKS

FRONT END VS BACK-END DEVELOPMENT



WEB FRAMEWORKS

- Web Frameworks Simplify the process of web development
- They abstract low-level details
- Common web frameworks include:
  - Express.js (Node.js)
  - Django (Python)
  - Ruby on Rails (Ruby)
  - Flask (Python)
  - Laravel (PHP)
  - Spring Boot (Java)

This is what we will use

- ✓ A minimal and flexible Node.js web application framework
- ✓ Express includes a routing system that enables developers to define routes for different HTTP methods and URLs.
- ✓ Makes it easy to create a structured and organized API or web application with specific endpoints.
- ✓ Express can be integrated with various template engines, such as **EJS** or Pug.

CREATING AND EXPRESS APPLICATION

```
> npm install express
```

```
const express = require("express");
const app = express();
const port = 3000;

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

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`);
});
```

- The above example uses the express package to create a new web-server that listens on port 3000. We can then set up listeners to respond to any given HTTP request.

SERVING HTML FILES

```
const express = require("express");
const path = require("path"); // this is a built in node package

const app = express();
const port = 3000;

app.get("/", (req, res) => {
  res.sendFile(path.resolve(__dirname, "index.html"));
});

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`);
});
```

- Above, we return an index.html file when a request is received to the root of our website.
- path.resolve is a Node.js method that resolves a sequence of path segments into an absolute path
  - e.g., if your current directory is “/user/example” the output would be “/user/example/index.html”

USING THE STATIC MIDDLEWARE

- Currently we cannot serve static assets such as CSS and images in our HTML pages.
  - We are only sending a single HTML file, not linked assets (e.g., images, css, and JavaScript)
- To configure this functionality, we need to use some middleware.
- Middleware can is bult into express, and there is also third-party modules you can use
- We can use middleware to handle concerns such as

Express middleware

The Express middleware modules listed here are maintained by the Expressjs team.

Middleware module	Description	Replaces built-in function (Express 3)
body-parser	Parse HTTP request body. See also: body, co-body, and raw-body.	express.bodyParser
compression	Compress HTTP responses.	express.compress
connect-rid	Generate unique request ID.	NA
cookie-parser	Parse cookie header and populate `req.cookies`. See also cookies and keygrip.	express.cookieParser
cookie-session	Establish cookie-based sessions.	express.cookieSession
cors	Enable cross-origin resource sharing (CORS) with various options.	NA
errorhandler	Development error-handling/debugging.	express.errorHandler
method-override	Override HTTP methods using header.	express.methodOverride
morgan	HTTP request logger.	express.logger
multer	Handle multi-part form data.	express.bodyParser
response-time	Record HTTP response time.	express.responseTime
serve-favicon	Serve a favicon.	express.favicon
serve-index	Serve directory listing for a given path.	express.directory
serve-static	Serve static files.	express.static
session	Establish server-based sessions (development only).	express.session
timeout	Set a timeout period for HTTP request processing.	express.timeout
vhost	Create virtual domains.	express.vhost

SERVING STATIC ASSETS

- Below, we instruct express to serve assets from a folder in our root directory called **public**
- **Note,** use “express.use” to set up middleware

```
const express = require("express");
const path = require("path");
const app = express();
app.use(express.static(path.join(__dirname, "public")));
```

- Above, express looks up the files relative to the static directory public:
  - the name of the static directory is not
  - if we place the image foo.jpeg in the public folder we would reference it in index.html like this, .

D E M O

WHAT'S WRONG WITH JUST HTML

- Serving plain HTML files provides us with a means to present a website
- How do we inject data into our html pages?
- Shared components or layouts must be replicated across different HTML files
- As web applications grow in complexity, maintaining a large codebase written in pure HTML becomes increasingly challenging.
- The lack of a clear separation of concerns makes it difficult to manage the presentation layer independently from the application logic

TEMPLATING TO THE RESCUE

- Templating allows for the dynamic generation of content in web applications
- Templating promotes code reusability by separating the HTML structure from the data
- Shared components
- Templating ensures consistent layouts across different pages of a website.
- Some popular templating languages include: EJS, Handlebars, Mustache , and Pug

EXPRESS SUPPORTS TEMPLATING

- We'll use EJS
- EJS is simple to use. First, we need to install it:

```
> npm install ejs
```

- We can now tell express to render our html pages using ejs. Below is a full example.

USING EJS (I)

```
const express = require("express");
const path = require("path");
const app = express();
const port = 20000;
app.set("view engine", "ejs");
app.use(express.static(path.join(__dirname, "public")));

app.get("/", (req, res) => {
  res.render("index");
});

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`);
});
```

USING EJS (II)

- Express will assume that we have an index.ejs file in a views folder which lives in the root directory of your project.
- EJS, is a superset of HTML. This means, to get the above example to work, we can simply rename our any html file to index.ejs and move it to a views folder.
- Using EJS, we now have some dynamic capabilities within our HTML views.

EJS ALLOWS HTML TO BE CHUNKED

```
<!-- views/common/header.ejs -->
<ul>
  <li><a href="home">Home</a></li>
  <li><a href="about"></a>About</li>
  <li><a href="contact">Contact</a></li>
</ul>
```

```
<!-- views/index.ejs -->
...
<body>
  <%- include('common/header'); %>
  <h1>Home Page</h1>
  <img src="test.jpeg" alt="wtf" />
</body>
...
```

DIFFERENT TYPES OF TAGS

- We will investigate more dynamic EJS tags next week
- You can get a full list of tags from the EJS docs:

Tags

- `<%` 'Scriptlet' tag, for control-flow, no output
- `<%_` 'Whitespace Slurping' Scriptlet tag, strips all whitespace before it
- `<%=` Outputs the value into the template (HTML escaped)
- `<%-` Outputs the unescaped value into the template
- `<%#` Comment tag, no execution, no output
- `<%%` Outputs a literal '<%'
- `%>` Plain ending tag
- `-%>` Trim-mode ('newline slurp') tag, trims following newline
- `_%>` 'Whitespace Slurping' ending tag, removes all whitespace after it

D E M O

END OF LECTURE