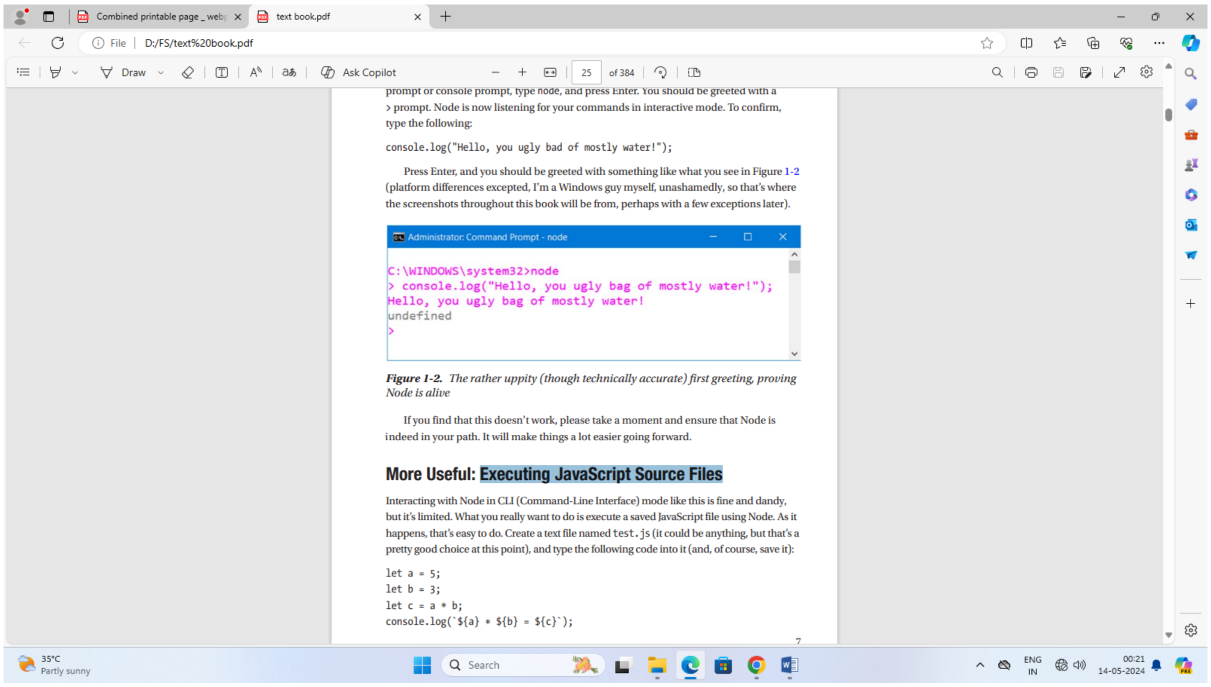
Node :

- Node is a platform for running primarily, server-side code that has high performance
- capable of handling large request loads with ease.
- It is based on the most widely used language: JavaScript
- In Node, almost everything you do is non -blocking, meaning code won't hold up the processing of other request threads.
- Most types of I/O, which is where blocking comes into play most, are asynchronous in Node, whether it's network calls or file system calls or database calls
- Node uses Google's popular and highly tuned V8 JavaScript engine, the same engine that powers its Chrome browser, makes it very high performance and able to handle a large request load.
- Node is event-driven and single-threaded with background workers.
- Node is a runtime environment, meaning that you can do such things as interacting with the local file system, access relational databases, call remote systems, and much more.
- acting as a server is just one capability that Node provides as a JavaScript runtime, and it can provide this functionality only.

To get started, there's only one address to remember: http://nodejs.org

- Once the install completes, you will be ready to play with Node.
- The installer should have added the Node directory to your path.
- go to a command prompt or console prompt, type node, and press Enter.
- You should be greeted with a > prompt.
- Node is now listening for your commands in interactive mode.
- To confirm, type the following: console.log("Hello, you ugly bad of mostly water!")



prompt or console prompt, type node, and press Enter. You should be greeted with a > prompt. Node is now listening for your commands in interactive mode. To confirm, type the following:

```
console.log("Hello, you ugly bad of mostly water!");
```

Press Enter, and you should be greeted with something like what you see in Figure 1-2 (platform differences excepted, I'm a Windows guy myself, unashamedly, so that's where the screenshots throughout this book will be from, perhaps with a few exceptions later).

```
C:\WINDOWS\system32>node
> console.log("Hello, you ugly bag of mostly water!");
Hello, you ugly bag of mostly water!
undefined
>
```

Figure 1-2. *The rather uppity (though technically accurate) first greeting, proving Node is alive*

If you find that this doesn't work, please take a moment and ensure that Node is indeed in your path. It will make things a lot easier going forward.

More Useful: Executing JavaScript Source Files

Interacting with Node in CLI (Command-Line Interface) mode like this is fine and dandy, but it's limited. What you really want to do is execute a saved JavaScript file using Node. As it happens, that's easy to do. Create a text file named `test.js` (it could be anything, but that's a pretty good choice at this point), and type the following code into it (and, of course, save it):

```
let a = 5;
let b = 3;
let c = a * b;
console.log(`${a} * ${b} = ${c}`);
```

7

Executing JavaScript Source Files:

let a = 5;

let b = 3;

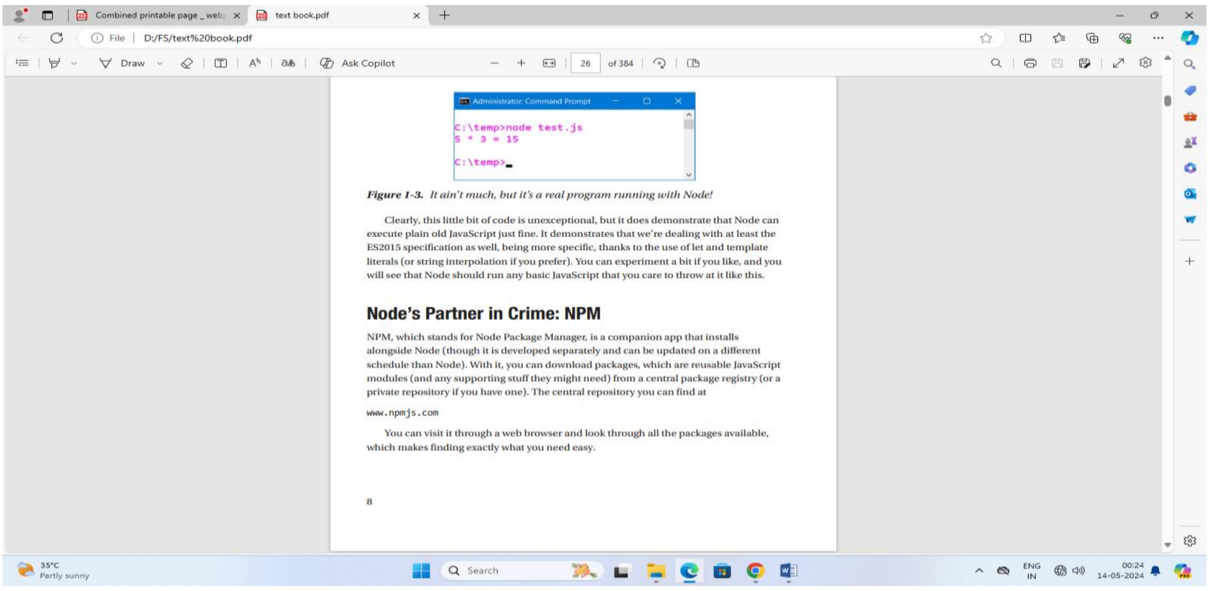
let c = a * b;

console.log(`${a} * ${b} = ${c}`);

Create a text file named test.js), and type the following code into it:

To execute this file, assuming you are in the directory in which the file is located, you simply must type this:

node test.js



```
C:\temp>node test.js
5 * 3 = 15

C:\temp>_
```

Figure 1-3. It ain't much, but it's a real program running with Node!

Clearly, this little bit of code is unexceptional, but it does demonstrate that Node can execute plain old JavaScript just fine. It demonstrates that we're dealing with at least the ES2015 specification as well, being more specific, thanks to the use of let and template literals (or string interpolation if you prefer). You can experiment a bit if you like, and you will see that Node should run any basic JavaScript that you care to throw at it like this.

Node's Partner in Crime: NPM

NPM, which stands for Node Package Manager, is a companion app that installs alongside Node (though it is developed separately and can be updated on a different schedule than Node). With it, you can download packages, which are reusable JavaScript modules (and any supporting stuff they might need) from a central package registry (or a private repository if you have one). The central repository you can find at

`www.npmjs.com`

You can visit it through a web browser and look through all the packages available, which makes finding exactly what you need easy.

8

NPM:

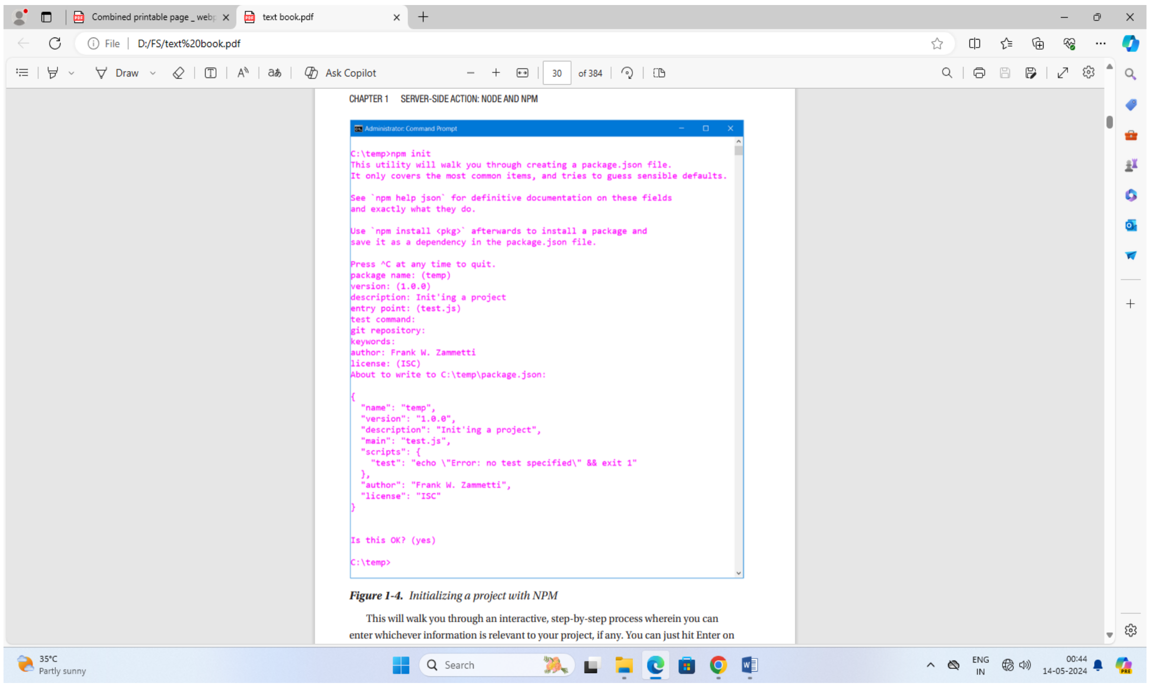
- NPM, which stands for Node Package Manager, is a companion app that installs alongside Node. NPM is just a JavaScript application that runs on Node
- The central repository you can find at www.npmjs.com
- A command to run from a command prompt, just like Node.js.
- Create a directory named MyFirstNodeProject.
  In it, you execute the following: ***npm install express***
  a directory called node-modules has been created
- NPM takes care of fetching all those dependencies for you. You'll also notice a file named package-lock.json has been created.
- The modules you name are installed in the current directory, and this is referred to as the local cache, or project cache.
- You can also install the module into what's called the global cache by adding an argument to the command: ***npm install -g express***
- NPM Commands: **npm ls**

- If you want to see what's installed in global cache instead, you can do
  npm -g ls
- You can also update a given module: ***npm update express***
  npm uninstall express
  npm help

Initializing a New NPM/Node Project

With most Node/NPM projects, a file named package.json is found in the root directory of the project. This file is the project manifest file, and it provides metadata information to NPM.

npm init


Combined printable page _ web
text book.pdf
File | D:/FS/text%20book.pdf
Draw
Ask Copilot
30
of 384
Administrator: Command Prompt
35°C
Partly sunny
Search
ENG
IN
00:44
14-05-2024

CHAPTER 1 SERVER-SIDE ACTION: NODE AND NPM

```
C:\temp>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 json` 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: (temp)
version: (1.0.0)
description: Init'ing a project
entry point: (test.js)
test command:
git repository:
keywords:
author: Frank W. Zammetti
license: (ISC)
About to write to C:\temp\package.json:

{
  "name": "temp",
  "version": "1.0.0",
  "description": "Init'ing a project",
  "main": "test.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Frank W. Zammetti",
  "license": "ISC"
}


Is this OK? (yes)

C:\temp>
```

Figure 1-4. *Initializing a project with NPM*

This will walk you through an interactive, step-by-step process wherein you can enter whichever information is relevant to your project, if any. You can just hit Enter on

Adding Dependencies:

To add the Express package.

First, you could edit package.json yourself, adding this element:

"dependencies": { "express": "^4.16.1" }

The module isn't installed at this point.

npm install

- fetch the Express package from the central repository, will determine all the dependencies it needs, and will download and install all of them in the node_modules directory under the current directory.

npm install express –save

Semantic Versioning:

The most common dependency versions in package.json, using Express as an example:

• “express” : “1.2.3” – NPM will grab this specific version only.

• “express”: “~1.2.3” – NPM will grab the most recent patch version.

• “express”: “^1.2.3” – NPM will grab the most recent minor version.

“express”: “*” – NPM will grab the newest version available.

Example:

“My First Node Web Server”:

- small bit of code is all it takes in Node to write a web server

```
require("http").createServer((inRequest, inResponse) =>
 { inResponse.end("Hello from my first Node Web server");
 }).listen(80);
```

Steps: create a directory and use NPM to init it as a project.

- type that code into a file, save it as server.js.
- Open the generated package.json file, and in the scripts section, add a new attribute to the object:
- "start": "node server.js"
- npm start
- create your own modules too just by adding other .js files to your project and require()-ing them
- require() function returns an object that is essentially the API provided by the module.
- createServer():This method creates a web server instance and returns a reference to it. This function handles all incoming HTTP request.
  • Interrogate the incoming request to determine the HTTP method.
  • Parse the request path.
  • Examine header values

To begin, let’s add a dependency to our project.

We’re going to use request module will provide to our server an elementary HTTP client for it to use to make remote calls: **npm install request --save**

copy that server.js file and name it server_time.js,

then replace its contents with the following code:

```
require("http").createServer((inRequest, inResponse) => { const requestModule =
require("request");

requestModule( "http://worldtimeapi.org/api/timezone/America/New_York", function (inErr,
inResp, inBody) { inResponse.end( `Hello from my first Node Web server: ${inBody}` ); } );
}).listen(80);
```

start the app: **npm start**

NPM knows what to do now and will launch Node and tell it to execute server.js. Now fire up your favorite web browser and visit http://127.0.01. You'll be greeted with the text "Hello from my first Node Web server".

NPM: More on package.json
name – We start with a simple one: the name of the thing you're coding!
author – The author is a single person and is defined by an object with three potential attributes: name, email, and url (where name is required, and both email and url are optional)
bin – Some packages require executables to be installed to do their work and added to the path.
browser –
bugs – If your project has an issue tracker, then you can reference it with the bugs element. The value of this is an object with two attributes, url and email, and you can specify either or both (but you must specify at least one, or NPM will complain).
• **bundledDependencies** – Some projects need to preserve NPM packages locally or through a single download.
• **config** – If you need to have parameters available in the environment when your package is used, then the config element might do the trick.
• **contributors** – The contributors element is just like the author element except that this is an array of people who helped with the project.
• **cpu** – If your code is only meant to run on certain system architectures, you can specify which as an array of strings with the cpu element.
• **dependencies** – You saw the dependencies element in the previous chapter, but I'll also mention that in addition to specifying a package name and optionally a version to be pulled down from the NPM registry, you can also specify a URL to a tarball to be downloaded or a Git/GitHub URL or a local file system path. Chapter 2 A Few More Words: Advanced Node and NPM 23
• **description** – A freeform string that describes your package. It's as simple as that!
• **devDependencies** – Again, one I mentioned in the previous chapter, and it's simply the same as dependencies, but it names packages that are only needed during development.
• **directories** – This element allows you to describe the structure your package, things like the location of library components binary content, man pages, Markdown documentation, examples, and tests.
• **engines** – This element allows you to specify what version(s) of Node your package works on.. • homepage – If your project has a web site, then you can specify the URL of its homepage with this element.
• **keywords** – The keywords element is an arbitrary array of strings that can be used to help people find your package (more on this in the next section). • license – The value of the license element is the license your package is released under. T