**Lecture 42**

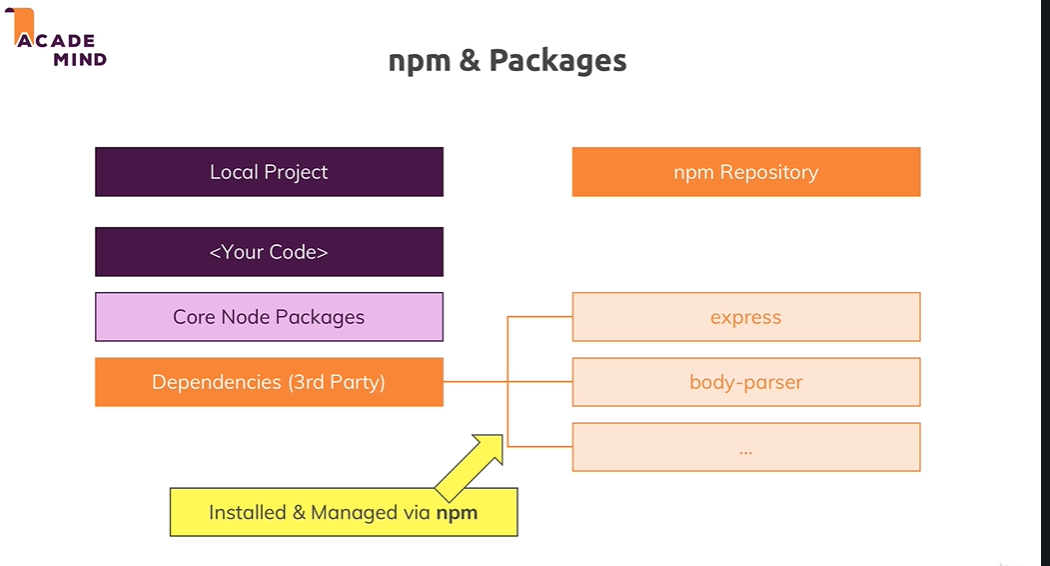
**Understanding NPM Scripts**

* Refer code

**Lecture 43**

**Installing 3rd party packages**

* These packages are available through the npm repository, that is a cloud package repository where all these packages live and you can conveniently install and manage them via npm, remember that tool that shipped with node
* Nodemon is a dev dependency. Installed using –save-dev.
* The package log json file by the way just stores the exact versions I installed today so that if you share your project with others, they can actually get these exact versions too instead of the latest versions but again



Global Features vs Core Modules vs Third-Party Modules

The last lectures contained important concepts about available Node.js features and how to unlock them.

You can basically differentiate between:

* **Global features**: Keywords like const or function but also some global objects like process
* **Core Node.js Modules**: Examples would be the file-system module ("fs"), the path module ("path") or the Http module ("http")
* **Third-party Modules**: Installed via npm install - you can add any kind of feature to your app via this way

**Global features** are **always available**, you don't need to import them into the files where you want to use them.

**Core Node.js Modules** don't need to be installed (**NO npm install** is required) but you **need to import them** when you want to use features exposed by them.

Example:

const fs = require('fs');

You can now use the fs object exported by the "fs" module.

**Third-party Modules** **need to be installed**(via npm install in the project folder) **AND imported**.

Example (which you don't need to understand yet - we'll cover this later in the course):

1. // In terminal/ command prompt
2. npm install --save express-session
3. // In code file (e.g. app.js)
4. const sessions = require('express-session');

**Lecture 45**

**Using nodemon for autorestarts**

* Refer code -- 02-using-nodemon-for-autorestarts
* Simply running “nodemon app.js” will show error, since nodemon is not installed globally.
* Modify start script to “nodemon app.js”. Now when we run “npm start” , it will search for nodemon in the project(local)
* Global & Local npm Packages
* In the last lecture, we added nodemon as a local dependency to our project.
* The good thing about local dependencies is that you can share projects **without the node\_modules** folder (where they are stored) and you can run npm install in a project to then re-create that node\_modules folder. This allows you to share only your source code, hence reducing the size of the shared project vastly.
* The attached course code snippets also are shared in that way, hence you need to run npm install in the extracted packages to be able to run my code!
* I showed that nodemon app.js would **not work** in the terminal or command line because we don't use local dependencies there but global packages.
* You could install nodemon globally if you wanted (this is NOT required though - because we can just run it locally): npm install -g nodemon would do the trick. Specifically the -g flag ensures that the package gets added as a global package which you now can use anywhere on your machine, directly from inside the terminal or command prompt.

**Lecture 47**

**Understanding different error types**

Different error types are :

* Syntax errors – will be indicated by the IDE.
* Runtime errors
* Logical errors

**Lecture 48**

**Finding and fixing syntax errors**

* Indicated by IDE and also while running npm start

**Lecture 49**

**Dealing with runtime errors**

* We cannot send a response after we have already sent one using res.end();

**Lecture 50**

**Logical Errors**

* Select the file which we want to debug. Here app.js , since it is the starting point. It works even if we don’t select app.js , ,becauses app.js is already defined as the entry point in the debug configuration (launch.json)
* Select Start Debugging from Run menu and select Nodejs from the list. Debug mode activated.
* Set breakpoint and goto webpage.

**Lecture 51**

**Using the Debugger**

* We can see the runtime variable values in the Debug Console.

**Lecture 52**

**Restarting the Debugger Automatically after editing code**

* Select Run 🡪 Add Configuration
* Add the following entries to launch.json

1. “restart”:true
2. “runtimeExecutable” : “nodemon” //default value is node
3. “console”: “integratedTerminal” //to use the integrated terminal as the logging console. We can use the Debug Console also

For the 2nd one to work properly , nodemon should be installed globally. When installed globally the nodemon command can be run from any cmd terminal

* The “program” attribute in launch.json indicates the entry file from which execution starts, when we select ‘Start Debugging’
* When we stop the debugger , the nodemon process will not be stopped automatically. We have to stop it using Ctrl-C. This cannot be done in the Debug Console. That is why we had to select the intergrated terminal in launch.json.
* *Note: When I tried to stop the debugger using the stop icon, nodemon process also was stopped*

<https://code.visualstudio.com/docs/nodejs/nodejs-debugging>

**Lecture 54**

**Changing variables in the debug console**

* Refer code – 03-finished.
* We can change runtime variable values during debugging by clicking on the variable in the Variables Panel and modifying it.
* Doing some operations on the variables in the debug console will not change the runtime value.

**Lecture 55**

**Wrapup**

Useful resources:

* More on debugging Node.js: <https://nodejs.org/en/docs/guides/debugging-getting-started/>
* Debugging Node in Visual Studio Code: <https://code.visualstudio.com/docs/nodejs/nodejs-debugging>