

1

	SPA & Working with NPM		
	Pros for using SPA:		
	Faster transitions between views.		
	 Better User Experience. Reduces server load.(partial page Update) earlier we use to implement that AJAX 		
	Cons:		
	SEO challenges		
	_	itial load ime might be higher.	
	Requires client side routing. (dependency on client browser)		
	Past practices to work with SDA:		
	Best practices to work with SPA: 1. Use client-side routing libraries(Angular Router, react router)		
	2. Lazy-load components.		
	3. Handle loading states efficiently		
	Following are the benefits of using NPM: • Access to a vast ecosystem of packages. • Easy dependency management.		
	Version control.		
Day 17			
	Cons:		
	Dependency bloat.		
	Security vulnerability in third party packages.		
	Best practices :		
	 Use Package-lock.json for version control Regularly audit (npm audit) for vulnerabilities. Keep your dependencies updated 		
		Var-hoisting: var declarations are hoisted to the top of the scope.	
		Functional Scope: variable declared with var are scoped to the function	
	Type Script JSX	Classes & Methods & Constructor	
		Maps : A key Value pair collection maintaining insertion order.	
		Iterators	
		Interfaces	
		Declarations and Annotations	
		Anytype	
		Enumeration: it is a way of defining a constant.	
		Decorator , Arrays & tuples Introduction to NPM	
Day 18		Initializing a project with npm init.	
	Working with NPM	Installing, updating, and removing packages:	
	& Node JS	npm install, npm update, npm uninstall.	
		Understanding package.json and package-lock.json.	
		Creating a simple Node.js application, Installing, updating, and removing	



packages: POWER A
Node.js Core Modules : fs (File System),http/https (HTTP/HTTPS Servers and
Clients)
"path (File and Directory Paths)
events (EventEmitter), Understanding package.json and package-lock.json."
"Asynchronous Programming in Node.js : Callback Functions , Promises ,
Async/Await , More Built in Modules :Events and Errors, What is Express.js?
Why use Express.js? Features and advantages
Getting started with Express.js"
"Creating a new Node.js project
Installing Express.js in your project
Basic Routing: Understanding routes in Express.js
Creating routes for different HTTP methods (GET, POST, PUT, DELETE), Handling
dynamic routes and parameters, Using route middleware"

	var	let
scope	Function-scoped or global	Block -scoped{}
hoisting	hoisted(initialized as undefined)	Hoisted(not initialized)
Reassignment	Allowed	allowed
Use cases	Legacy code(avoid in modern JS/TS)	Variables that needs reassignment

	async/await	Promises(.then/.catch)
Syntax	Cleaner and more readable	verbose with chaining
Error handling	Easier with trycatch	Handle using .catch()
Browser support	Modern browsers(ES2017+)	Supported in all ES6- compatible browser
Nested Logic	Easier to follow	Can lead to "callback hell" in complex flow

For mastering javascript topics:

- 1. Project 1: Student Registration Portal
 - a. Bootstrap: Forms, Cards, Layout grid
 - b. JavaScript: Data types, branching, arrays, string functions, validation
 - c. DOM Manipulation: Fetch input, show error messages
 - d. JSON: Create structured payload on submit

Great Learning

Features:

- Form with fields: name, Email, Phone, Course, gender
- Validation using javascript
- Display confirmation
- Output using data as JSON
- 2. Demo Project 2: Task Manager Web App
 - Bootstrap: Cards, Badges, Buttons, List groups
 - JavaScript: Arrays, loops, string manipulation, functions
 - Advanced JavaScript: Closures, Date objects
 - DOM: Add/Delete DOM elements dynamically
 - jQuery (optional): Event delegation

Features:

- 1. Adding tasks with description and due date.
- 2. Mark task as done using a button
- 3. Deleting task from the list
- 4. Show task count(Pending/completed)

Case Study: Building a Smart E-Learning Platform

Context:

You're part of a development team building a modern E-Learning Platform that delivers personalized learning content, live sessions, and assessments to students. The platform must be fast, scalable, and deliver a seamless user experience, especially on single-page applications (SPAs).

Problem Statement

You are tasked with designing the frontend and backend structure of a modular E-Learning platform using:

- SPA architecture
- TypeScript for robust typing
- NPM for package management
- Node.js and Express.js for server-side development

Objectives



Participants need to:

- 1. Plan a SPA layout for the E-Learning portal.
- 2. List and explain essential npm packages required.
- 3. Demonstrate how asynchronous features like Promises and Async/Await can improve responsiveness.
- 4. Explain how TypeScript features (Interfaces, Classes, Enums, Tuples, etc.) help in building scalable architecture.
- 5. Configure a Node.js backend using Express.js, including routing logic for different user interactions.

Tasks with Hints

Task 1: SPA Design Strategy

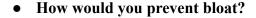
Hint: Sketch a layout of different sections of the platform (e.g., dashboard, course list, course detail, quizzes) and map them to routes. Consider lazy loading and client-side routing.

- What challenges might SEO face in your SPA?
- How will you show a loading indicator when navigating between modules?

Task 2: Managing Project with NPM

Hint: Think of what dependencies would be essential for:

- Authentication
- Frontend form validations
- Backend routing and database connection
- How will package-lock. json help in a team setting?





Task 3: Implementing Async Logic

Hint: Use an example where the system fetches quiz results or course progress.

- Where would you prefer using Promises over Callbacks?
- When should you use async/await to avoid callback hell?

Task 4: Using TypeScript Features

Hint: Identify scenarios where you would use:

- Interfaces (e.g., defining user model or course structure)
- Enums (e.g., course status: ACTIVE, COMPLETED, EXPIRED)
- Tuples (e.g., returning a question and options)
- Decorators (optional, for advanced features)
- How can TypeScript help avoid runtime errors in your platform?

Task 5: Backend Design using Express.js

Hint: Consider the following routes:

- GET /courses
- POST /enroll
- PUT /progress
- DELETE /unenroll



- What middleware would you use for authentication?
- How do you handle dynamic routes for accessing /courses/:courseId?

Reflection Questions

- 1. How would you secure API endpoints on your Node.js server?
- 2. What are potential performance bottlenecks in an SPA?
- 3. How can NPM vulnerabilities impact your users?
- 4. What are trade-offs between using TypeScript over plain JavaScript?

Deliverables

Participants should submit:

- Architecture flow of SPA and routes
- List of npm packages and their usage
- TypeScript interface/enum/class sketches
- Express.js route structure (pseudocode or flowchart)
- Notes on async handling (Promise vs. async/await)