Q1. WHAT IS NODE.JS? WHERE CAN YOU USE IT?

NODE.JS IS OPEN-SOURCE, CROSS-PLATFORM, JAVASCRIPT RUNTIME ENVIRONMENT THAT ALLOWS DEVELOPERS TO RUN JAVASCRIPT CODE OUTSIIDE OF A WEB BROWSER.IT IS BUILT ON CHROME’S V8 JAVASCRIPT ENGINE AND PROVIDES AN EVENT-DRIVEN, NON-BLOCKING I/O MODEL THAT MAKES IT LIGHTWEIGHT AND EFFICIENT, PARTICULARLY FOR BUILDING SCALABLE NETWORK APPLICATION.

COMMON USES

1.WEB DEVELOPMENT

2.REAL TIME APPLICATION

3.API DEVELOPMENT

4.MICROSERVICES

5.COMMAND LINE TOOL

6.IOT

7.SINGLE PAGE APPLICATION

Q2. EXPLAIN CALLBACK IN NODE.JS.

A CALLBACK IS SIMPLY A FUNCTION PASSED AS AN ARGUMENT TO ANOTHER FUNCTION WHICH WILL BE INVOKED ONCE THE ASYNCHRONOUS OPERATION COMPLETES OR AN EVENT OCCURS.THE CALLBACK FUNCTION TYPICALLY TAKES TWO PARAMETERS: AN OPTIONAL ERROR PARAMETER AND A RESULT PARAMETER.

Q3. WHAT ARE THE ADVANTAGES OF USING PROMISES INSTEAD OF CALLBACK?

1. READABILITY: PROMISES PROVIDE A MORE READABLE AND UNDERSTANDABLE WAY TO HANDLE ASYNCHRONOUS OPERATION COMPARED TO NESTED CALLBACKS. PROMISES ALLOW YOU TO CHAIN ASYNCHRONOUS OPERATION USING .THEN() AND .CATCH() METHODS, RESULTING IN CODE THAT READS MORE LIKE SYNCHRONOUS CODE, WITH CLEAR AND SEQUENTIAL STEPS.
2. ERROR HANDLING: PROMISES HAVE BUILT IN ERROR HANDLING MECHANISMS THAT MAKE IT EASIER TO MANAGE ERRORS IN ASYNCHRONOUS CODE. INSTEAD ON RELAYING ON ERROR-FIRST CALLBACKS, WHERE ERROR HANDLING LOGIC IS SCATTERED THROUGHT THE CODE, PROMISES ALLOWS YOU TO HANDLE ERRORS IN A CENTRALIZED MANNER USING .CATCH() METHOD AT THE END OF PROMISES CHAIN
3. AVOIDING CALLBACK HELL: PROMISES HELP MITIGATE THE PROBLEM OF CALLBACK HELL, WHICH OCCURS WHEN DEALING WITH DEEPLY NESTED CALLBACK FUNCTION. WITH PROMISES, YOU CAN WRITE CLEANER AND MORE MAINTAINABLE CODE BY CHAINING ASNCHRONOUS OPERATION INSTEAD OF NESTING CALLBACK, LEADING TO CODE THAT IS EASIER TO UNDERSTAND AND DEBUG.
4. COMPOSITION AND REUSABILITY: PROMISES SUPPORT COMPOSITION AND REUSABILITY OF ASYNCHRONOUS OPERATION. SINCE PROMISES CAN BE CHAINED TOGETHER, YOU CAN CREATE MODULAR AND REUSABLE FUNCTION THAT ENCAPSULATE COMPLEX ASYNCHRONOUS WORKFLOWS. THIS MAKES IT EASIER TO REFACTOR AND EXTEND CODE, LEADING TO MORE MAINTAINABLE AND SCALABLE APPLICATION.
5. ERROR PROPAGATION: PROMISES AUTOMATICALLY PROPOGATE ERRORS THROUGH THE PROMISES CHAIN, ALLOWING YOU TO HANDLE ERRORS AND HANDLE THEM APPROPRIATELY, WHETHER AT THE LOCAL LEVEL WITHIN A SPECIFIC PROMISE OR AT THE GLOBAL LEVEL IN THE .CATCH() BLOCK
6. BUILT IN FUNCTIONALITY: PROMISES COME WITH BUILT-IN METHODS LIKE PROMISES .ALL() AND PROMISE.RACE() THAT MAKE IT EASE TO ORCHESTRATE MULTIPLE ASYNCHRONOUS OPERATION THESE METHODS ALLOW YOU TO EXECUTE MULTIPLE PROMISES CONCURRENTLY OR RACE THEM AGAINST EACH OTHER, PROVIDEING MORE FLEXIBILITY AND CONTROL OVER ASYNCHRONOUS WORKFLOWS.

Q4. WHAT IS NPM?

NPM IS A NODE PACKAGE MANAGER. IT’S DEFAULT PACKAGE MANAGER OF NODE.JS. NPM IS USED TO INSTALL, MANAGE, AND SHARE RESUABLE CODE PACKAGES, ALSO KNOWN AS “PACKAGES” OR “MODULES”, WRITTEN IN JAVASCRIPT. THESE PACKAGES CAN INCLUDE LIBRARIES, FRAMEWORK, TOOLS, AND OTHER RESOURCES THAT DEVELOPERS CAN USE TO BUILD NODE.JS APPLICATION

Q5. WHAT ARE THE MODULES IN NODE.JS? EXPLAIN

1. CORE MODULE: NODE.JS COMES WITH A SET OF BUILT-IN MODULES THAT PROVIDE ESSENTIAL FUNCTIONALITY FOR COMMON TAKS SUCH AS FILE I/O, NETWORKING, AND UTILITY FUNCTION. THESE CORE MODULES ARE INCLUDED WOTH THE NODE.JS RUNTIME AND CAN BE IMPORTED INTO YOUR APPLICATION USING THE REQUIRE () FUNCTION WITHOUT THE NEED FOR INSTALLATION.
2. LOCAL MODULES: LOCAL MODULES ARE CUSTOME MODULES CREATED BY DEVELOPERS WITHIN THEIR NODE.JS PROJECTS. THESE MODULES ENCAPSULATE SPECIFIC FUNCATIONALITY OR FEATURES AND ARE STORED IN SEPARATE JAVASCRIPT FILES. LOCAL MODULES CAN BE IMPORTED INTO OTHER PARTS OF THE APPLICATION USING THE REQUIRE() FUNCTION PROVIDING A WAY TO ORGANIZE AND REUSE CODE WITH A PROJECT.
3. THIRD-PARTY MODULES: THIRD PARTY MODULES ARE MODULES CREATED BY EXTERNAL DEVELOPERS AND MADE AVAILABLE FOR USE IN NODE.JS APPLICATION. THESE MODULES ARE PUBLISHED TO THE NPM REGISTRY AND CAN BE INSTALLED INTO A PROJECT USING THE NPM INSTALL COMMAND. THIRD PARTY MODULES COVER A WIDE RANGE OF FUNCIONALITY, INCLUDING FRAMEWORK, LIBRARIES, UTILIES AND TOOLS, ALLOWING DEVELOPERS TO LEVERAGE EXISTING SOLUTIONS TO ACCELERATE DEVELOPMENT AND ENHANCE THEIR APPLICATION.
4. BUILT-IN ES MODULES: STARTING FROM NODE.JS VERSION 12, NODE.JS SUPPORTS FOR ECMASCRIPT, WHICH ARE ANATIVE WAY TO ORGANIZE AND SHARE JAVASCRIPT CODE. ES MODULES USE THE IMPORT AND EXPORT KEYWORD TO DEFINE DEPENDENCIES AND EXPOSE FUNCTIONALITY BETWEEN MODULES. WHILE NODE.JS TRADITIONALLY USED THE COMMONJS MODULE SYSTEM, ES MODULES PROVIDE AN ALTERNATIVE MODULE SYSTEM THAT ALIGNS WITH THE ECMASCRIPT STANDARD.