Question Bank : Chapter - 6, 7

Server-side Development with Node.js, Express and MongoDB &

Deployment of MERN Stack Web Applications

➢ Cross-Origin Resource Sharing (CORS)

· The browser's same-origin policy blocks reading a resource from a different origin. This mechanism stops a malicious site from reading another site's data, but it also prevents legitimate uses.

· In a modern web application, an application often wants to get resources from a different origin. For example, you want to retrieve JSON data from a different domain or load images from another site into a <canvas> element.

· In other words, there are public resources that should be available for anyone to read, but the same-origin policy blocks that.

· Enabling CORS lets the server tell the browser it's permitted to use an additional origin.

How does CORS work?

Remember, the same-origin policy tells the browser to block cross-origin requests. When you want to get a public resource from a different origin, the resource-providing server needs to tell the browser "This origin where the request is coming from can access my resource". The browser remembers that and allows cross-origin resource sharing.

Step 1: client (browser) request #

When the browser is making a cross-origin request, the browser adds an Origin header with the current origin (scheme, host, and port).

Step 2: server response #

On the server side, when a server sees this header, and wants to allow access, it needs to add an Access-Control-Allow-Origin header to the response specifying the requesting origin (or * to allow any origin.)

Step 3: browser receives response #

When the browser sees this response with an appropriate Access-Control-Allow-Origin header, the browser allows the response data to be shared with the client site.

· The CORS mechanism supports secure cross-origin requests and data transfers between browsers and servers. Modern browsers use CORS in APIs such as XMLHttpRequest or Fetch to mitigate the risks of cross-origin HTTP requests.

Youtube Link: https://youtu.be/4KHiSt0oLJ0 , https://youtu.be/PNtFSVU-YTI, https://youtu.be/tcLW5d0KAYE

➢ Basic Authentication, Session Based Authentication and Token Based Authentication

HTTP Basic Authentication

HTTP Basic Authentication requires that the server request a user name and password from the web client and verify that the user name and password are valid by comparing them against a database of authorized users. When basic authentication is declared, the following actions occur:

- A client requests access to a protected resource.
- The web server returns a dialog box that requests the user name and password.
- The client submits the user name and password to the server.

The server authenticates the user in the specified realm and, if successful, returns the requested resource.

Session Based Authentication and Token Based Authentication:

https://www.geeksforgeeks.org/session-vs-token-based-authentication/#:~:text=The%20Session%20and%20Token%2Dbased,usually%20used%20for%20different%20purposes.

➢ JSON Web Tokens and its working

- It is a compact and secure way of exchanging information over the network.
- JSON Web Token helps to maintain the integrity and authenticity of the information because it is digitally signed using secret or public/private key pair using RSA or ECDSA.
- An important thing to keep in mind about JWT is that it is a signed token and not an encrypted one.
- Therefore, even though JWT can verify the integrity of the claims contained within it, it cannot hide that information. And because of that, it is advisable not to put any sensitive information within the token.

Why we need JSON Web Token?

HTTP is a stateless protocol that means a new request does not remember anything about the previous one. So for each request, you need to login and authenticate yourself (figure 1).

So, the solution to deal with this is the use of what's called a session. A session is an object stored on the server that helps the user to stay logged in or to save any reference to their account. Figure 2 shows the overall flow of this process.

- First, the user submits a username and a password that are authenticated by the server. If the authentication is successful a session ID is generated for the respective client.
- The generated session ID is returned to the client and is stored on the server-side as well.
- Now, the client just needs to send its session ID along with the request to authenticate itself and retrieve necessary information.
- The server will then check if the session ID is valid or not. If the session is still valid, it will respond with the requested webpage/data. And if not, the server will respond with an error message stating that the request made is unauthorized.

How does a JSON Web Token work?

- When a user sends his credentials to the server to login, the server authenticates the user. If the authorization is successful, the server sends a JSON Web Token to the user.
- The user can use the JWT to request any protected services/resources from the server by including the JWT in the Authorization header using the Bearer schema.
- Authorization: Bearer <token>
- When the server gets a request from the user to access any protected content, the protected routes of the server will look for a valid JWT in the Authorization header. If the token is present and is valid the server will allow access to the user.
- The JWT contains necessary information about the user that can be used to identify the user, know the user's privilege, and serve the user accordingly.
- Because of JWT, the server does not need to query the database every time a request comes in to check if the user has the necessary rights or not.

Figure 3

- Every token assigned by the server is signed by a secret key known to the server only.
- Therefore, only the server can use the secret key to verify the token and to check if the token has tampered. If an attacker tries to make any changes in the token (like granting admin privileges), the signature of the token needs to be calculated again that will require the secret key.
- Since the attacker does not have a secret key making any changes to the token will invalid it. The server will discard such requests to prevent unauthorized access.

3. Fundamentals of Secure Deployment

Following are the principles for secure deployment:

Secure development: Genuine security benefits can only be realised when delivery teams weave security into their everyday working practices.

Updating security knowledge: Creating code that is capable of withstanding attack requires an understanding of attack types and of defensive security practices. Your level of understanding in these areas must be regularly updated if it's to remain useful.

Produce clean & maintainable code: If your code lacks consistency, is poorly laid out and undocumented, you're adding to the overall complexity of your system.

Secure your development environment

There is sometimes a perceived conflict between security and usability. This situation is highlighted in the case of end user devices and the environments used to support software development.

Protect your code repository

Your code is only as secure as the systems used to create it. As the central point at which your code is stored and managed, it's crucial that the repository is sufficiently secure.

Secure the build and deployment pipeline

Continuous integration, delivery and deployment are modern approaches to the building, testing and deployment of IT systems.

Continually test your security

Security testing can be manual, but it can also be automated.

Plan for security flaws

All but the very simplest software is likely to contain bugs, some of which may have a security impact.

➢ Node.js Modules

In Node.js, Modules are the blocks of encapsulated code that communicates with an external application on the basis of their related functionality. Modules can be a single file or a collection of multiples files/folders. The reason programmers are heavily reliant on modules is because of their re-usability as well as the ability to break down a complex piece of code into manageable chunks.

Modules are of three types:

- Core Modules
- local Modules
- Third-party Modules

Core Modules: Node.js has many built-in modules that are part of the platform and comes with Node.js installation. These modules can be loaded into the program by using the require function.

Syntax:

```
var module = require('module_name');
```

The require() function will return a JavaScript type depending on what the particular module returns. The following example demonstrates how to use the Node.js Http module to create a web server.

```
var http = require('http');

http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/html'});

  res.write('Welcome to this page!');

  res.end();

}).listen(3000);
```

In the above example, the require() function returns an object because the Http module returns its functionality as an object. The function http.createServer() method will be executed when someone tries to access the computer on port 3000. The res.writeHead() method is the status code where 200 means it is OK, while the second argument is an object containing the response headers.

The following list contains some of the important core modules in Node.js:

http : creates an HTTP server in Node.js.

assert : set of assertion functions useful for testing.

fs : used to handle file system.

path : includes methods to deal with file paths.

process : provides information and control about the current Node.js process.

os : provides information about the operating system.

querystring : utility used for parsing and formatting URL query strings.

url : module provides utilities for URL resolution and parsing.

Local Modules: Unlike built-in and external modules, local modules are created locally in your Node.js application. Let’s create a simple calculating module that calculates various operations. Create a calc.js file that has the following code:

Filename: calc.js

```
exports.add = function (x, y) {
   return x + y;
};
exports.sub = function (x, y) {
   return x - y;
};
exports.mult = function (x, y) {
   return x * y;
};
exports.div = function (x, y) {
   return x / y;
};
```

Since this file provides attributes to the outer world via exports, another file can use its exported functionality using the require() function.

Filename: index.js

```
var calculator = require('./calc');
var x = 50, y = 10;
console.log("Addition of 50 and 10 is " + calculator.add(x, y));
console.log("Subtraction of 50 and 10 is " + calculator.sub(x, y));
console.log("Multiplication of 50 and 10 is " + calculator.mult(x, y));
console.log("Division of 50 and 10 is " + calculator.div(x, y));
```

Third-party modules: Third-party modules are modules that are available online using the Node Package Manager(NPM). These modules can be installed in the project folder or globally. Some of the popular third-party modules are mongoose, express, angular, and react.

Example:

```
npm install express
```

npm install mongoose

npm install -g @angular/cli

➢ What is npm?

npm is two things: first and foremost, it is an online repository for the publishing of open-source Node.js projects; second, it is a command-line utility for interacting with said repository that aids in package installation, version management, and dependency management.

NPM (Node Package Manager) is the default package manager for Node.js and is written entirely in Javascript. Developed by Isaac Z. Schlueter, it was initially released in January 12, 2010. NPM manages all the packages and modules for Node.js and consists of command line client npm. It gets installed into the system with installation of Node.js. The required packages and modules in Node project are installed using NPM.

A package contains all the files needed for a module and modules are the JavaScript libraries that can be included in Node project according to the requirement of the project.

NPM can install all the dependencies of a project through the package.json file. It can also update and uninstall packages. In the package.json file, each dependency can specify a range of valid versions using the semantic versioning scheme, allowing developers to auto-update their packages while at the same time avoiding unwanted breaking changes.

Installing NPM:

To install NPM, it is required to install Node.js as NPM gets installed with Node.js automatically.

Install Node.js.

Checking and updating npm version:

Version of npm installed on system can be checked using following syntax:

Syntax:

npm -v

Checking npm version

If the installed version is not latest, one can always update it using the given syntax:

Syntax:

npm update npm@latest -g

As npm is a global package, -g flag is used to update it globally.

- **Expressjs**

  https://expressjs.com/

  What is routing in Expressjs?

  https://expressjs.com/en/guide/routing.html

  Routing Methods in expressjs

  https://expressjs.com/en/guide/routing.html

  Error Handling in Expressjs.

  https://expressjs.com/en/guide/error-handling.html

- **API - Application Programming Interface**

  https://aws.amazon.com/what-is/api/

  How do APIs work?

  https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

- **NodeJs http Module.**

  https://nodejs.org/en/knowledge/HTTP/servers/how-to-create-a-HTTP-server

  https://www.w3schools.com/nodejs/nodejs_http.asp

- **Building a REST API with Node and Express.**

  https://stackabuse.com/building-a-rest-api-with-node-and-express/

  https://www.geeksforgeeks.org/node-js-building-simple-rest-api-in-express/

- **Https and Secure Communication.**

  https://www.cloudflare.com/learning/ssl/what-is-https/

  https://www.javatpoint.com/https

  https://www.tutorialsteacher.com/https/what-is-https

- **Secure Communication Protocols**

  https://www.educba.com/internet-security-protocols/

  https://www.geeksforgeeks.org/types-of-internet-security-protocols/