Short Answers

1. How to build a server using Node.js

First off, Node.js is great for building fast, scalable network applications, offering benefits in performance, faster development, and other advantages. It is best suited for real-time web applications, streaming applications, messaging apps, chat apps, and more. Now when it comes to building a server, there are many frameworks to choose from, but Express is the one I learned previously and is also the topic of today's lecture.

To build a server using Express, first create a project directory and navigate into it. Then, initialize a `package.json` file by running `npm init -y`, which sets up the project metadata. After that, install Express using `npm install express` or `npm i express` to add it as a dependency. Next, create an `app.js` file and include the following boilerplate code:

```
const express = require("express");  // or import express from "express";
const app = express();
const port = 3000;

app.get("/", (req, res) => {
  res.send('Hello, world!');
});

app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

To run the server, use `node app.js`, and a most basic Express server has been setup.

2. What is Express.js?

Express.js is a minimal and flexible web application framework for Node.js, designed to simplify the development of web and mobile applications. It provides a robust set of features for building single-page, multi-page, and hybrid web applications, as well as RESTful APIs.

Express.js is widely adopted in the development community and serves as the foundation for various popular development stacks, such as MEAN, MERN, and MEVN (MongoDB/Express.js/Vue.js/Node.js). Its popularity stems from its simplicity, flexibility, and the extensive ecosystem of middleware and plugins available, allowing developers to customize and extend their applications as needed.

3. What is the difference between response.send() and response.write() in Express?

In Express, we use `response.send()` to send a final response from the server to the client. `response.send()` automatically determines the content type (JSON, HTML, plain text, etc.) based on the argument provided. If you send an object or an array, it will automatically be converted to JSON. If you send

a string, it will be sent as plain text. It also automatically sets the appropriate headers (`Content-Type`). `response.write()`, on the other hand, is used to send data in chunks. It sends a chunk of data to the client but does not end the request-response cycle (you need to call `response.end()` after `response.write()` to finalize the response). It's useful for sending large amounts of data incrementally, such as for streaming or file downloads. `response.write()` does not automatically convert objects or arrays to JSON. If you want to send an array or object, you need to use `JSON.stringify(data)` to convert it to a JSON-formatted string. It will send strings as plain text. And it does not automatically set the appropriate headers.

But if we use `response.json()`, there is no need to use `response.send()` or `response.write()`.

4. What is the difference between HTTP GET method and HTTP POST method?

`app.get()` handles `GET` requests, while `app.post()` handles `POST` requests. `app.get()` retrieves data from the server through a resource, such as an API endpoint. It appends data to the URL in name-value pairs (query parameters), but it does not modify the server's state (it is stateless and idempotent). The length of the URL is often limited by the browser or environment. `app.get()` is suitable for retrieving data that does not require security and does not involve files or images.

`app.post()` sends data to the server to create or update a resource, such as submitting a form, signing up as a new user, or updating a record. Data sent with a `POST` request is included in the body of the request, not in the URL. This allows sending larger amounts of data and is more secure than `GET` because the data is not exposed in the URL. There is also no limit to the length of the data. `app.post()` is non-idempotent, meaning its requests can result in changes to the server's state.

5. What is Content-Type? What are the different kinds of Content-Type?

`Content-Type` is a header used in HTTP requests and responses to specify the media type (MIME type) of the data being sent or received. It tells the server or client what type of data is being transferred so that it can handle the data appropriately. The `Content-Type` header helps both parties (the client and the server) interpret the body of the request or response correctly.

The most common types of `Content-Type` are: `text/plain`, `text/html`, and `application/json`. But there are also many additional types, including `application/x-www-form-urlencoded`, `multipart/form-data`, `application/xml`, `image/jpeg`, `image/png`, `image/gif`, `audio/mpeg`, `audio/wav`, `video/mp4`, etc.

6. How to enable automatic re-running when files are modified

To enable automatic re-running when files are modified, we can use Nodemon. Use `npm install -g nodemon` to install Nodemon globally, or `npm install -D nodemon` to install it as a local dependency. Now, instead of using `node app.js`, we can use `nodemon app.js`, and this will watch for changes in the current directory and subdirectories. We can also add the line `"start": "nodemon app.js"` to the `scripts` object in the `package.json` file to allow using the command `npm start` to run Nodemon.

7. How to pass parameters in the URL

Well, there are different types of parameters we can pass. For example, with query parameters, which are used to send non-sensitive data in the URL, we would use `request.query` in the route and the corresponding `?` and `&` operators in the URL. The following code demonstrates how query parameters are used with `query` and `sort`:

```
app.get("/search", (req, res) => {
  const query = req.query.query;
  const sort = req.query.sort;
  res.send(`Search for: ${query}, Sort by: ${sort}`);
});
```

The resulting URL would look like something like this:

```
https://example.com/search?query=express&sort=desc
```

For route paramters, which are used to capture dynamic segments in the URL path (IDs), we would use req.param in the route and something like `:id` in the URL. The following code demonstrates how route paramters are used:

```
app.get("/users/:id", (req, res) => {
  const userId = req.params.id;
  res.send(`User ID: ${userId}`);
});
```

The resulting URL would look something like this:

```
https://example.com/users/12
```

There are additional operators and syntaxes we can use to match the route path. For example, we can use `?` after a character, which means it will include everything up to that character and everything else after it is optional, such as `/ab?cd` (which would match both `/abcd` and `/abd`). We can use a `+` to represent one or more occurrences of the preceding character, meaning that the preceding character or group must appear at least once, such as `/ab+cd` (which would match `/abcd`, `/abbcd`, `/abbbcd`, etc.). We can use a `*` to represent any number of characters, including none, such as `/ab*cd` (which would match `/abcd`, `/ab123cd`, `/abRANDOMcd`, etc.). Finally, we can use regular expressions (regex) to define complex matching patterns, such as `/.*fly$/`, which would match `/butterfly`, `/dragonfly`, but not `/butterflyman`.

8. What is the difference between static and dynamic web pages

Static web pages are when a server receives a request for a web page and sends the response to the client without performing any additional processing. In static web pages, the content remains the same until

someone changes it manually. These pages are typically written in languages such as HTML, CSS, and JavaScript. To serve static files, such as images, CSS files, and JavaScript files, we use the `express.static` built-in middleware function in Express. Portfolio websites are often an example of static websites.

Dynamic web pages are web pages that are generated or modified in real time based on user interactions or other factors. Unlike static web pages, the content of dynamic web pages can change depending on user input, database queries, or other conditions. They can display different content for different users or situations. Dynamic web pages often rely on server-side rendering technologies such as Node.js, Express, or Flask to generate the content sent to the client. For example, a dynamic web page might show personalized dashboards, the weather, stocks, or update content without reloading the page. These pages are commonly used for applications like e-commerce sites, social media platforms, and blogs.

9. How to use template engines

Template engines enable you to use static template files in your application. At runtime, the template engine replaces variables in the template file with actual values and transforms the template into an HTML file sent to the client. Some popular template engines that work with Express include Pug (formerly Jade), Mustache, and EJS.

To use Pug, we first install it using the command `npm install pug`. In the `app.js` file, we set Pug as the view engine, create a `views` folder in the same directory as the `app.js file`, and add an `index.pug` file inside the `views` folder. Finally, we render the Pug file in a route by using `res.render()` and passing in the name of the Pug file along with any dynamic data. The following is an example of the Pug boilerplate:

```
// in the app.js file
const express = require("express");
const app = express();
const port = 3000;

// set Pug as the view engine
app.set("view engine", "pug");

// define a route that renders a Pug template
app.get("/", (req, res) => {
    res.render("index", { title: "Home", message: "Welcome to Pug!" });
});

app.listen(port, () => {
    console.log(`Server is running at http://localhost:${port}`);
});
```

We can also use the command `npx express-generator --view=pug myapp` to generate an Express application with Pug as the template engine, or simply `npx express-generator myapp` to generate a basic, bare-bones Express template with the default view engine (which is usually Jade, despite Jade being renamed to Pug). But be aware, `express-generator` is a little outdated and uses `var` in all the variable declarations, along with few other things that may need to be changed.

10. What is SSR?

SSR stands for Server-Side Rendering and refers to the process of rendering web pages on the server instead of the browser. In SSR, when a client makes a request for a web page, the server generates the HTML content for that page and sends it to the client, which then renders it in the browser. SSR is typically used with frameworks like React, Vue.js, or Angular, where the server renders the initial HTML based on the state of the app and then sends it to the browser. Afterward, the client takes over and continues the rendering with JavaScript. Benefits of SSR include faster initial load of the HTML page, more SEO-friendly (search engines can crawl and index fully-rendered HTML content), and improved performance (for users on slower devices or networks).

11. What is CSR?

In Client-Side Rendering (CSR), the browser initially receives a minimal HTML page, often just a skeleton, and then JavaScript takes over to render the content dynamically on the client side. To fetch or update data from the server, two commonly used methods are AJAX and the `fetch()` API. With CSR, the initial page load is a little slower than SSR, but the entire UI does not need to be reloaded every time new data is fetched or updated from the server. Instead, client-side frameworks or libraries (such as React, Vue.js, or Angular) manage the UI and update only the parts of the page that have changed. This is achieved through techniques like the virtual DOM in React, which efficiently updates only the modified elements, improving performance and user experience. It's great for applications that require frequent or real-time updates to the user interface based on user interactions or data changes.

12. How would you structure your node web application

I could use the `express-generator` method mentioned above, but because it's a little outdated, I would probably set it up manually through the following steps:

1. Create project folder
2. Run `npm init -y` to initialize the project
3. Install dependencies: `npm install express`
4. Create `app.js` to set up the Express app
5. Create `/src` folder to organize the application code
6. Set up routes in `/src/routes` folder
7. Add middleware in `/src/middleware` folder (may not be needed)
8. Create `/public` folder for static assets (images, styles)
9. Run the app with `node app.js`, `nodemon app.js`, or `npm start`
10. Test the routes using tools like Postman or the browser