View

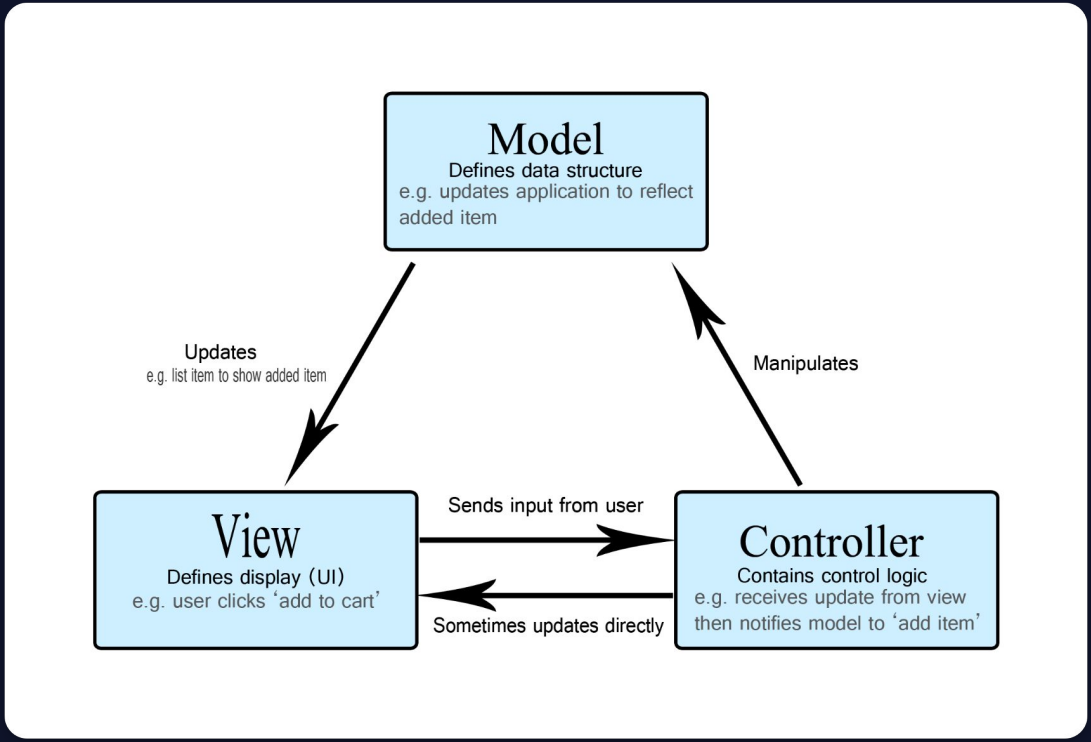
Where Pixels Find Their Purpose.

What is MVC

Model (M): (1) Represents the data and business logic of the application.(2) Responsible for data storage, retrieval, and manipulation.(3) Isolated from the user interface (UI) and user interactions.

View (V): (1) Represents the presentation layer.(2) Displays the data to the user.(3) Responds to user input for presentation purposes.

Controller (C): (1) Acts as an intermediary between Model and View.(2) Handles user input and initiates appropriate actions.(3) Updates both Model and View as needed.(4) Maintains the flow of data and application logic.



https://developer.mozilla.org/en-US/docs/Glossary/MVC

Statically sending html

```
// Import the 'express' module
const express = require("express");

// Create an instance of the Express application
const app = express();

// Define a route for the root path '/'
app.get("/", (req, res) => {
  // Set the response content type to HTML
  res.setHeader("Content-Type", "text/html");

  // Send an HTML response
  res.send(
    "<html><head><title>Hello, World!</title></
    head><body><h1>Hello, World!</h1></body></html>"
  );
});

// Start the server on port 3000
const port = 3000;
app.listen(port, () => {
  console.log(`Server is listening on port ${port}`);
});
```

css and js file?

Public directory

In an Express.js application, the "public" directory is a common convention used to store static assets, such as HTML files, stylesheets (CSS), client-side JavaScript files, images, and other files that are served directly to clients (browsers) without any processing by the server.

Public directory

Static Assets Storage: The "public" directory is where you store files like HTML, CSS, JavaScript, and images that are directly accessed by web browsers.

Client Accessible: Files in "public" are accessible via URLs (e.g., http://localhost:8080/logo.png) without server processing.

Security: Be cautious not to store sensitive information in "public" to avoid exposing it to clients.

Declaring a public directory

```
const express = require("express");
const app = express();

// Serve static files from the "public" directory
app.use(express.static("public"));

// Now, files in the "public" directory are accessible at their relative paths
// For example, 'public/style.css' can be accessed at '/style.css'
```

to do app using node js

Dynamic html rendering?

View engines

Definition: View engines are tools used for dynamic content generation on the server-side.

Purpose: They enable the creation of dynamic HTML pages that can be sent to clients.

Why Use View Engines?

Separation of Concerns: View engines help separate application logic from presentation (following the MVC pattern).

Dynamic Content: They allow embedding data into HTML templates to create customized pages.

Code Reusability: Templates promote code reusability for consistent page layouts.

View engine

Benefits:

1. Templating: View engines use templates to structure HTML and add dynamic data, making code reusable and maintainable.
2. Separation of Concerns: They separate application logic (controllers) from how content looks (views), following the MVC pattern.
3. Code Organization: Views are organized neatly, simplifying design and layout updates.

Use Cases: View engines are ideal for server-side rendering (SSR), generating web pages dynamically from database data. They're great for web apps needing server-generated pages.

HTML string

Benefits:

1. Simplicity: Sending HTML strings directly is straightforward and suitable for serving static pages or simple content that doesn't require dynamic generation.
2. Performance: It can be more efficient for static content because there's no need for server-side template rendering.

Use Cases: This approach is suitable for serving static pages, simple landing pages, or content that doesn't change frequently and doesn't require server-side templating or data injection.

Conclusion?

In summary, the choice between using a view engine like EJS and sending HTML strings directly depends on your project's requirements. View engines are powerful for dynamic content generation and following best practices for web application architecture, while sending HTML strings directly is simpler and more suitable for static or pre-rendered content.