Controller

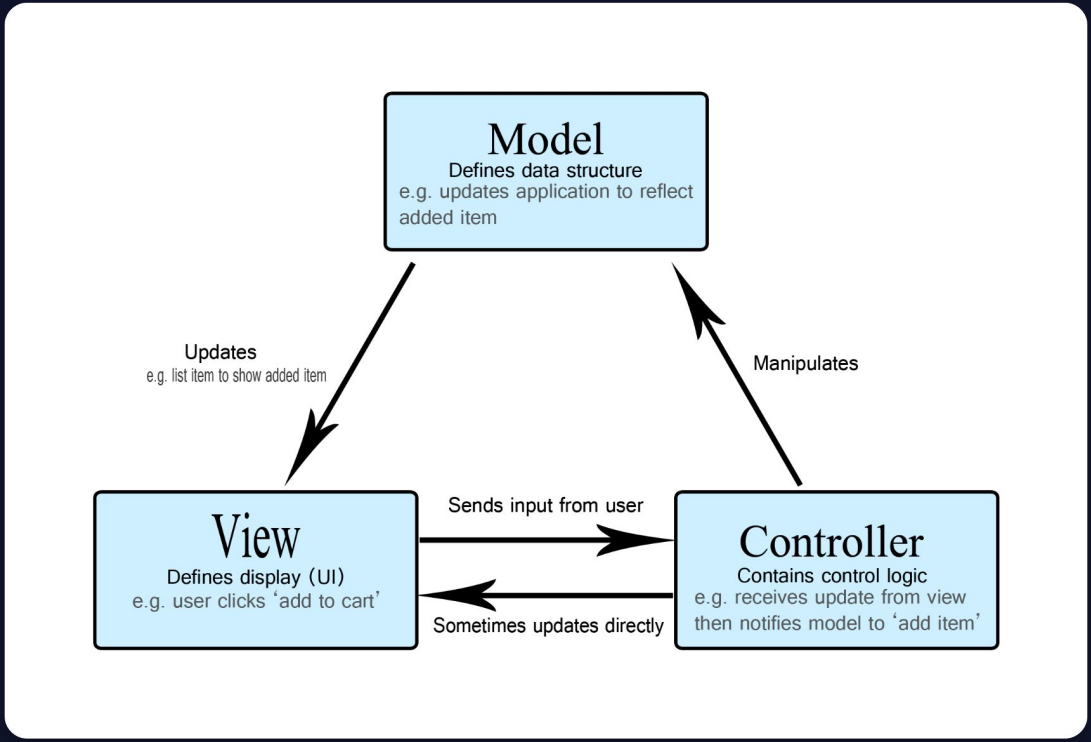
Where Logic Meets the Purpose.

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

Identifying controller

```
// 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}`);
});
```

Refactor as function

```
const express = require("express");
const app = express();
const port = 3000;

// Define a function to handle the "/hello" route
function helloWorldController(req, res) {
  // Set the content type to HTML
  res.setHeader("Content-Type", "text/html");

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

// Define a route for the "/hello" path and
associate it with the controller
app.get("/hello", helloWorldController);

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

???

```
const express = require("express");
const app = express();
const port = 3000;

// Define a function to handle the "/hello" route
function helloWorldController(req, res) {
  // Set the content type to HTML
  res.setHeader("Content-Type", "text/html");

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

// Define a route for the "/hello" path and
associate it with the controller
app.get("/hello", helloWorldController);

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

Understanding routes - 1

Routes in Express are used to define how your application responds to client requests. They determine the behavior of your web server based on the requested URL and HTTP method.

Routes handling

Express supports various HTTP methods, including GET, POST, PUT, DELETE, and more. Each method corresponds to a different type of request and is often used for specific purposes.

You can create routes using app.get(), app.post(), app.put(), and app.delete(). These methods define routes for handling specific HTTP methods.

Route path and parameter

Routes path are defined using URL paths, such as /, /users, or /products.

Route parameters allow you to access dynamic parts of the URL. For example, /users/:id can capture the id value from the URL

Route params example

```
const express = require('express');
const app = express();
const port = 3000;

// Define a route with a parameter ":id"
app.get('/users/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User Profile Page for ID: ${userId}`);
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Route order matters

Express evaluates routes in the order they are defined. Make sure to define more specific routes before more general ones to avoid unexpected behavior.

Bad example

```
const express = require('express');
const app = express();
const port = 3000;

// Route 1: Handle requests for "/users/:id"
app.get('/users/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User Profile Page for ID: ${userId}`);
});

// Route 2: Handle requests for "/users/admin"
app.get('/users/admin', (req, res) => {
  res.send('Admin User Page');
});

// Route 3: Handle requests for "/users"
app.get('/users', (req, res) => {
  res.send('All Users Page');
});

// Route 4: Handle requests for "/"
app.get('/', (req, res) => {
  res.send('Home Page');
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Correct

```
const express = require('express');
const app = express();
const port = 3000;

// Route 1: Handle requests for "/users/admin"
app.get('/users/admin', (req, res) => {
  res.send('Admin User Page');
});

// Route 2: Handle requests for "/users/:id"
app.get('/users/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User Profile Page for ID: ${userId}`);
});

// Route 3: Handle requests for "/users"
app.get('/users', (req, res) => {
  res.send('All Users Page');
});

// Route 4: Handle requests for "/"
app.get('/', (req, res) => {
  res.send('Home Page');
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Chaining routes with middleware

You can chain multiple route handlers together for a single route. This is useful for adding multiple functions to process a request.

Route chaining with middleware

```
const express = require('express');
const app = express();
const port = 3000;

// First route handler: Logging middleware
const logRequest = (req, res, next) => {
  console.log(`Request received for: ${req.url}`);
  next(); // Call next to proceed to the next route
  handler
};

// Second route handler: Route-specific logic
const greetUser = (req, res) => {
  res.send('Hello, User!');
};

// Define a route with chained handlers
app.get('/greet', logRequest, greetUser);

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```