PART 1

MASYNCTECH
SCHOOL

FULLSTACK WEB DEV

EXPRESS JS

BACKEND DEVELOPMENT

MASYNCTECH
SCHOOL

WHAT'S EXPRESS?

express JS

GETTING STARTED

OVERVIEW OF EXPRESS.JS

MASYNCTECH SCHOOL

High-Level Explanation

- Express.js: Minimal, flexible Node.js web app framework
- Features for web and mobile apps
- Simplifies web server setup and operation

Basic Explanation

- Analogy: Express as a lemonade stand kit
- Kit provides table, signage, cups, pitcher
- Developers decide location and pricing
- Express handles complexities, enables focus on sales

Important Rules

- **Modularity**: Split routes into separate files.
- **Middleware**: Order matters; use judiciously.
- **Error Handling:** Include error handling middleware last.
- **Security:** Use packages like `helmet` for security.
- **Logging**: Implement logging for app operations and errors.

Deep Dive

- Express.js: Eases web app/API development
- Middlewares, routes, utility functions provided
- Developers focus on app logic, not HTTP details
- Acts as mediator for routing, request handling, responses

When to use?

- Use cases for Express.js:
  - Building web apps in Node.js
  - Creating RESTful APIs
  - Serving static files/assets
  - Developing single-page, multipage, hybrid apps
  - Building performance-centric apps without overhead

Summary

Express.js simplifies Node.js web app and API dev with routing, middleware, and more. It's a ready-made kit for web servers, and best practices ensure efficiency and security.

MASYNCTECH
SCHOOL

NODEJS VS EXPRESS

express JS

GETTING STARTED

NODEJS VS EXPRESS

MASYNCTECH SCHOOL

High-Level Explanation

- Node.js: Server-side JavaScript runtime environment.
- Express.js: Framework for faster web app and API development.

Basic Explanation

- Node.js is like the engine powering a car.
- Express.js is like a built-in GPS, making web app development easier.

Deep Dive

- Node.js is an environment for server-side JavaScript execution.
- It uses the V8 JavaScript engine from Google.
- Express.js is a web application framework for Node.js.
- Express simplifies web development tasks like routing and middleware handling.

When to use?

- Node.js for server-side JavaScript in various applications.
- Express.js streamlines web app and API development on Node.js.

Summary

- Node.js runs server-side JavaScript.
- Express.js is a framework for efficient web app/API development on Node.js.

NODEJS VS EXPRESS

MASYNCTECH
SCHOOL

Node.js	
Environment	-> Enables server-side execution of JavaScript.
Runtime	-> Uses V8 engine to run JS efficiently.
Asynchronous	-> Handles tasks without blocking the main thread.

Express.js	
Framework	-> Built on top of Node.js for web apps & APIs.
Middleware	-> Facilitates handling requests and responses.
Routing	-> Directs incoming requests to specific handlers.

MASYNCTECH
SCHOOL

CREATING EXPRESS APP

express JS

GETTING STARTED

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

REQUEST RESPONSE OBJECT

express JS

GETTING STARTED

REQUEST AND RESPONSE OBJECT

MASYNCTECH
SCHOOL

High-Level Explanation

- **Request** (`req`) stores client request details.
- **Response** (`res`) sends server responses to the client.

Basic Explanation

- Request (`req`) is your restaurant order.
- Response (`res`) is the restaurant's food response.

Important Rules

- **Request (`req`):**
  - Validate and sanitize input.
  - Check property existence before access.
- **Response (`res`):**
  - Always send a response.
  - Use descriptive status codes and messages.
  - Avoid detailed errors in production.

Deep Dive

- Request (`req`) holds HTTP request info.
- Response (`res`) manages Express HTTP responses.

- Request (`req`) holds HTTP request info.
- Response (`res`) manages Express HTTP responses.

When to use?

- Request (`req`) for client's request details.
- Response (`res`) for sending data after processing.

Summary

- Request (`req`) details client's request.
- Response (`res`) sends data to the client.

common properties and methods

Request (`req`) properties and methods:

- `req.params`: Named route parameters.
- `req.query`: Query string parameters.
- `req.body`: Parsed data for `POST`.
- `req.headers`: Request headers.
- `req.method`: HTTP method.

Response (`res`) properties and methods:

- `res.send(body)`: Plain text response.
- `res.json(json)`: JSON response.
- `res.status(code)`: Set response status code.
- `res.download(file)`: Transfer files.
- `res.redirect([status,] path)`: URL redirection.

MASYNCTECH
SCHOOL

ACCESSING

ROUTE & QUERY

PARAMS

express JS

GETTING STARTED

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

PROJECT BOOK API

express JS

GETTING STARTED

MASYNCTECH
SCHOOL

UNDERSTANDING
MIDDLEWARE

express JS

EXPRESS MIDDLEWARE

UNDERSTANDING MIDDLEWARE

MASYNCTECH
SCHOOL

High-Level Explanation

- Middleware serves as checkpoints for requests.
- Each checkpoint modifies the request.
- Request proceeds to the route handler afterward.

Basic Explanation

- Middleware is like amusement park checkpoints.
- Ticket check, height check, item check.
- Ensures a smooth path to your final destination.

Important Rules

- Modularity for maintainability.
- Middleware order is crucial.
- Use `next()` to pass control.
- Place error handling at the stack end.

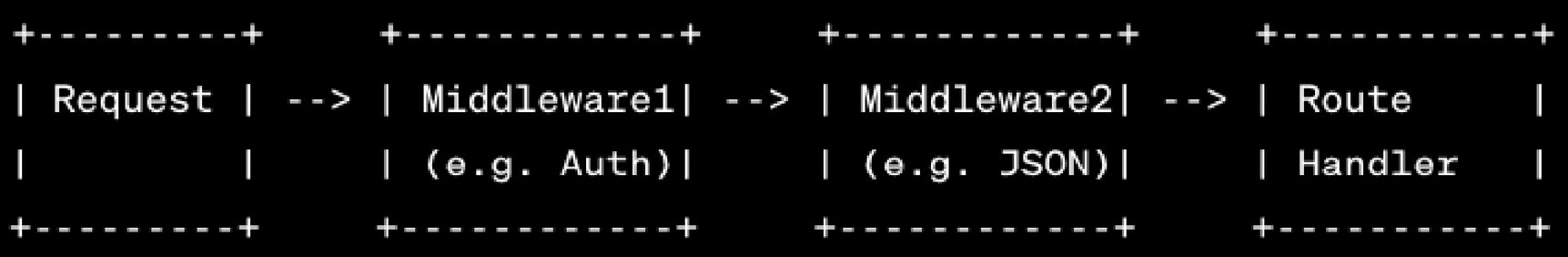
Deep Dive

- Middleware has access to `req`, `res`, and `next`.
- `next` points to the next middleware.
- Middleware can modify, end, or chain the cycle.

When to use?

- Middleware for common logic across routes.
- Transform request data.
- Log requests.
- Handle errors or conditions pre-route.

UNDERSTANDING MIDDLEWARE



MASYNCTECH
SCHOOL

MIDDLEWARE
SYNTAX

express JS

EXPRESS MIDDLEWARE

MASYNCTECH
SCHOOL

Syntax

Creating

```
function middlewareName(req, res, next) {
    // Middleware logic here

    next(); // Proceed to the next middleware or route handler
}
```

- `middlewareName` function (optional name).
- `req`: Request object for data.
- `res`: Response object for setting data.
- `next`: Callback to pass to next middleware.
- `// Middleware logic here`: Where actual middleware logic resides.
- `next();`: Move to next middleware/route handler.

MASYNCTECH
SCHOOL

Syntax

Global usaga

```
app.use(middlewareName);
```

Specific use case

```
app.get('/some-route', middlewareName, routeHandler);
```

UNDERSTANDING MIDDLEWARE

MASYNCTECH
SCHOOL

High-Level Explanation

- Middleware serves as checkpoints for requests.
- Each checkpoint modifies the request.
- Request proceeds to the route handler afterward.

Basic Explanation

- Middleware is like amusement park checkpoints.
- Ticket check, height check, item check.
- Ensures a smooth path to your final destination.

Important Rules

- Modularity for maintainability.
- Middleware order is crucial.
- Use `next()` to pass control.
- Place error handling at the stack end.

Deep Dive

- Middleware has access to `req`, `res`, and `next`.
- `next` points to the next middleware.
- Middleware can modify, end, or chain the cycle.

When to use?

- Middleware for common logic across routes.
- Transform request data.
- Log requests.
- Handle errors or conditions pre-route.

Summary

- Express.js middleware as processing units.
- Inspect, modify, end, or pass requests.
- Fine-grained control for cleaner, modular apps.

MASYNCTECH
SCHOOL

HOW MIDDLEWARE WORKS

express JS

EXPRESS MIDDLEWARE

HOW MIDDLEWARE WORKS

MASYNCTECH
SCHOOL

High-Level Explanation

- Middleware is a chain of tasks for requests.
- Each task can modify and decide flow.
- Controls request journey to the final destination.

Basic Explanation

- **1st middleware:** Check-in desk, ticket and boarding pass.
- **2nd middleware:** Security check, bag and person scan.
- **3rd middleware:** Boarding gate, boarding pass verification.
- Problem at any point may halt progress.

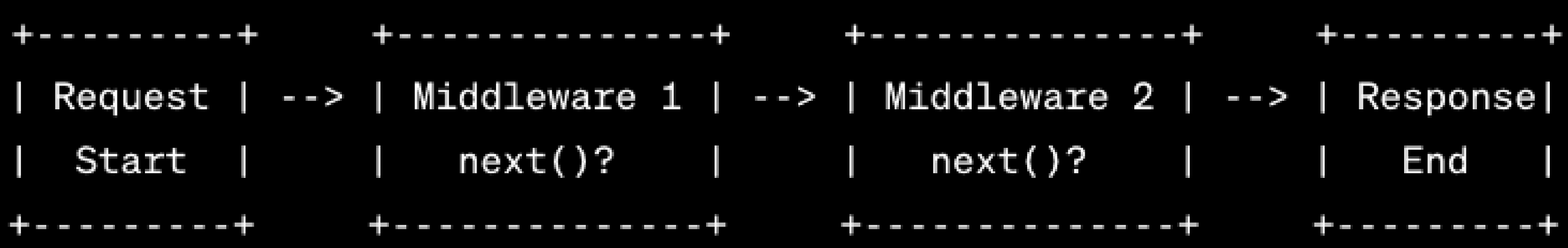
Deep Dive

- Express middleware handles request-response.
- Sequence: Top to bottom in stack.
- Middleware can end or pass control.
- Errors skip to error-handling middleware.

When to use?

- Authentication for sensitive routes.
- Logging for monitoring/debugging.
- Data processing (headers, bodies).
- Error handling centralized.
- Rate limiting to restrict client requests.

HOW MIDDLEWARE WORKS



MIDDLEWARE SYNTAX

MASYNCTECH
SCHOOL

Creating

```
function middlewareName(req, res, next) {
    // Middleware logic here

    next(); // Proceed to the next middleware or route handler
}
```

- `middlewareName` function (optional name).
- `req`: Request object for data.
- `res`: Response object for setting data.
- `next`: Callback to pass to next middleware.
- `// Middleware logic here`: Where actual middleware logic resides.
- `next();`: Move to next middleware/route handler.

Usage

```
app.use(middlewareName);
```

```
app.get('/some-route', middlewareName, routeHandler);
```

MASYNCTECH
SCHOOL

UTILIZING NEXT() FUNCTION

express JS

EXPRESS MIDDLEWARE

UTILIZING NEXT() AND THE MIDDLEWARE STACK

MASYNCTECH
SCHOOL

High-Level Explanation

- `next()` is like passing a relay baton.
- It hands the request to the next middleware.

Basic Explanation

- Middleware is like a cafeteria queue.
- Each counter serves a different purpose.
- "Next counter" is akin to `next()` function.
- Saying "yes" continues; saying "no" ends.

Important Rules

- Always conclude or call `next()`.
- Use `next(err)` for error handling.
- Minimize side effects without `next()`.
- Document complex middleware flows.

Deep Dive

- Middleware stack runs in defined order.
- Middleware can end or continue cycle.
- Missing `next()` leads to request timeout.

When to use?

- Sequential operations.
- Conditional progression.
- Error handling with `next(err)`.

Summary

- `next()` pivotal for middleware flow.
- Ensures sequential, conditional processing.
- Smooth, efficient request handling in Express.js.

MASYNCTECH
SCHOOL

BUILT-IN MIDDLEWARE

express JS

EXPRESS MIDDLEWARE

BUILT-IN MIDDLEWARE

MASYNCTECH SCHOOL

High-Level Explanation

- Express.js has built-in middleware.
- Handles request types and tasks.
- Eliminates external library dependencies.

Basic Explanation

- Express's tools (middleware) unwrap packages.
- Different materials (JSON, static, URL-encoded).
- Default tools for common wrappers.

Important Rules

- Use **express.static()** for static files.
- Set size limits for safety (e.g., `express.json({ limit: '1mb' })`).
- Store static files in a dedicated "public" or "static" folder.

Deep Dive

- **`express.json()`:** Parses JSON request bodies.
- **`express.static()`**: Serves static files.
- **`express.urlencoded()`**: Parses URL-encoded payloads.

When to use?

- **express.json():** JSON data reception.
- **express.static():** Serving static files.
- **express.urlencoded():** URL-encoded form data.

Summary

- `next()` pivotal for middleware flow.
- Ensures sequential, conditional processing.
- Smooth, efficient request handling in Express.js.

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

APPLICATION LEVEL MIDDLEWARE

express JS

EXPRESS MIDDLEWARE

APPLICATION-LEVEL MIDDLEWARE

MASYNCTECH SCHOOL

High-Level Explanation

- App-level middleware: Access request, response, and next.
- Part of request-response cycle.
- Functions in Express for processing.

Analogue

- Express middleware analogy: Park visit steps.
- Each ride is a middleware (step).
- Sequence, decisions, and order matter.
- Application-level middleware explained.

Important Rules

- Order of definition impacts execution.
- Always end cycle or call `next()`.
- Promote reusable logic, not in route handlers.

Deep Dive

- Middleware tasks: Modify req/res, end cycle, call next.
- Sequence matters; executed in order.
- Key part of Express request handling.

When to use?

- Task for multiple routes, global modifications.
- Examples: logging, authentication, headers.

Summary

- Custom logic at request-response stages.
- Crucial for global tasks (auth, logging).

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

ROUTER LEVEL MIDDLEWARE

express JS

EXPRESS MIDDLEWARE

ROUTER-LEVEL MIDDLEWARE

MASYNCTECH SCHOOL

High-Level Explanation

- Router-level middleware focuses on routers.
- Specific to routes bound to the router.
- Different from app-level middleware.
- Granular control over middleware usage.

Analogue

- Amusement park analogy for middleware.
- App-level rules for the entire park.
- Router-level rules for specific sections/routes.
- Example: water park's swimwear rule.

Important Rules

- Router-level middleware for unique route subsets.
- Specific processing needs.
- Always call `next()` (unless cycle ends).

Deep Dive

- Router-level middleware: req, res, and next access.
- Scope is route-specific, not global.
- Differs from app-level middleware.
- Bound to a particular router instance.

When to use?

- Common logic for route sets.
- Group routes by functionality (e.g., `/user`).
- Ensures consistent middleware application.

Summary

-Route-specific, modular.
- Benefits for organized, shared logic.
- Enhances code structure and reusability.

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

ERROR HANDLING MIDDLEWARE

express JS

EXPRESS MIDDLEWARE

ERROR-HANDLING MIDDLEWARE

MASYNCTECH SCHOOL

High-Level Explanation

- Error-handling middleware for handling errors.
- Prevents app crashes, provides feedback.
- Acts as a safety net in Express.

Analogue

- Video game glitch analogy for error handling.
- Middleware prevents app crash.
- Resets and displays error message.
- Ensures app remains functional.

Important Rules

- Error-handling middleware placement.
- After `app.use()` and routes.
- Caution: sensitive info in responses.
- Protect against information leakage.

Deep Dive

- Error-handling middleware has four arguments.
- Arguments: `(**err, req, res, next**)`.
- Place after other middleware/routes.
- Catches errors from previous code.
- Important for graceful error handling.

When to use?

- Error-handling middleware use cases.
- Provide meaningful error messages.
- Log errors for debugging.
- Prevent app crashes on unexpected issues.

Summary

- Safety net for Express app.
- Catches and handles errors.
- Ensures graceful error handling.

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

CREATING CUSTOM MIDDLEWARE

express JS

EXPRESS MIDDLEWARE

CREATING CUSTOM MIDDLEWARE

MASYNCTECH SCHOOL

High-Level Explanation

- Custom middleware as checkpoints.
- Define procedures/rules.
- Requests follow defined rules.

Analogue

- Web app as a nightclub analogy.
- Bouncers (middleware) check requirements.
- Built-in (Express) and custom middleware.
- Ensure compliance before access.

Important Rules

- Call `next()` to avoid freezing.
- Handle errors properly, pass or respond.

Deep Dive

- Custom middleware functions explained.
- Access to req, res, and next.
- `next` passes control to next middleware.
- Integral in request-response cycle.

When to use?

- Log requests.
- Authenticate users.
- Process/transform request data.

Summary

- Tailored functionalities in cycle.
- Logging, auth, data processing.
- Effective handling of operations.

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

EXPRESS ROUTER

express JS

EXPRESS ROUTER

EXPRESS ROUTER OVERVIEW

MASYNCTECH SCHOOL

High-Level Explanation

- Express Router as a mini-app.
- Group route handlers by function.
- Cleaner, maintainable code.
- Enhanced organization in Express.

Analogue

- Analogy: Web app as a book, chapters as related routes
- Express Router organizes app like chapters on topics
- Group and manage related routes efficiently

Important Rules

1. Group related routes.
2. Choose meaningful base paths.
3. Handle errors within routes.
4. Utilize middleware for common tasks.

Deep Dive

- Express.js Router: Split app into route-based modules
- Avoid clutter in main `app` object
- Acts like middleware, mounts on a path
- Partition app into sub-apps by route paths

When to use?

1. Large apps with numerous routes.
2. Need to modularize and structure the app.
3. Different teams or individuals handling app sections.

Summary

- Express Router in Express.js: Organize routes in modules
- Promotes clean, modular code
- Enhances maintainability of complex apps

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

ROUTER METHOD CHAINING

express JS

EXPRESS ROUTER

EXPRESS ROUTER METHOD CHAINING

MASYNCTECH SCHOOL

High-Level Explanation

- Express `Router` method chaining: Define routes for multiple HTTP methods
- Same path, concise code
- Supports GET, POST, PUT, DELETE, etc.

Analogue

- Analogy: Method chaining like remote control combo
- Quickly sequence actions (route definitions)
- Efficiently perform multiple actions with one line of code

Important Rules

- Maintain logical order for clarity in method chaining.
- Avoid excessively long chained expressions for readability.
- Properly handle errors in each method of the chain.

Deep Dive

- Method chaining for structured RESTful APIs
- Avoids repetitive route handler definitions
- Enhances code readability
- Central role of `Router.route()` method

When to use?

- RESTful APIs with multiple resource operations
- Organized, less repetitive routing code
- Handling various middlewares for the same path/methods

Summary

- Express `Router` method chaining: Define multiple route handlers
- Ideal for RESTful APIs, code organization, and readability
- Utilize `Router.route()` for streamlined route definitions

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

USING MIDDLEWARE IN ROUTES

express JS

EXPRESS ROUTER

USING MIDDLEWARE IN ROUTES

MASYNCTECH SCHOOL

High-Level Explanation

- Express `Router` method chaining: Define routes for multiple HTTP methods
- Same path, concise code
- Supports GET, POST, PUT, DELETE, etc.

Analogue

- Analogy: Express router middleware as mini security checks
- Different stages (rock, pop, electronic) with unique rules
- Middleware acts as checkpoints for specific sections
- Ensures conditions are met before reaching the main event

Important Rules

- Best practices for Express router middleware:
1. Localize middleware for clarity and management.
2. Use `router.use()` thoughtfully; affects subsequent routes.
3. Avoid duplicating middleware if applied globally.
4. Maintain the order of middleware and routes in routers.

Deep Dive

- Express app: Split route handling into routers
- Routers have their own middleware
- Middleware applies to router's defined routes
- Request passes router middleware before route handler

When to use?

- Use Express router middleware for:
1. Route-specific checks or validations (e.g., admin routes).
2. Modularizing by functionality with unique middleware.
3. Grouping API versions with distinct middleware needs.

Summary

- Express router middleware: Specialized checks for subsets
- Like event security checks for different areas
- Ideal for modularization, specific functionality checks
- Best practices: proper ordering, no duplication, localization

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

ERROR HANDLING MIDDLEWARE

express JS

EXPRESS ROUTER

ERROR HANDLING MIDDLEWARE

MASYNCTECH SCHOOL

High-Level Explanation

- Error-handling middleware in Express.js: Catches application errors
- Acts as a safety net before client response
- Ensures proper handling of issues before responding

Analogue

- Analogy: Error-handling middleware as a "Game Over" screen
- Catches errors, prevents crashes
- Allows choosing next action, like showing an error message

Important Rules

- Best practices for error-handling middleware:

1. Define it at the end, after other middleware and routes.
2. Handle both synchronous and asynchronous errors.
3. Pass the error object with `next(err)` for debugging.
4. Don't suppress errors; log or handle them.
5. Consider third-party logging services for complex apps.

Deep Dive

- Error-handling middleware: `(err, req, res, next)`
- Comes after other middleware and route calls
- Centralized error handling mechanism
- Catches and processes errors, custom responses, logging, etc.

When to use?

- Use error-handling middleware:

1. After all other middleware and routes.
2. Centralize error logging.
3. Customize error message handling.
4. Uniformly format error responses, especially in API apps.

Summary

- Error-handling middleware: Centralized safety net
- Catches and processes errors after other middleware/routes
- Gives control over error responses
- Best practices: place at end, handle all error types

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

PROTECTING ROUTES
BASED ON USER
AUTHENTICATION

express JS

EXPRESS ROUTER

PROTECTING ROUTES BASED ON USER AUTHENTICATION

MASYNCTECH SCHOOL

High-Level Explanation

- Protecting routes via user authentication: Restrict access unless verified
- Analogous to VIP section at a party with wristbands
- Allows only authorized users to access specific routes

Analogue

- Analogy: Authentication middleware as VIP room control
- Check for "VIP sticker" (token or session)
- Allows access to specific "room" (route)
- Without "VIP sticker," stays in general area

Important Rules

- Best practices for authentication middleware:
1. Use established libraries (e.g., Passport.js).
2. Securely store and transmit tokens.
3. Be cautious with error messages; don't reveal too much.
4. Always use HTTPS for sensitive data.
5. Implement rate limiting to prevent brute-force attacks.

Deep Dive

- In Express.js, middleware checks user authentication.
- Verify session, token, or credentials.
- Authenticated: Proceed to route handler.
- Not authenticated: Redirect to login or `401 Unauthorized`.

When to use?

- Use authentication middleware for:
1. Securing sensitive operations (e.g., account settings).
2. Implementing role-based access (e.g., user vs. admin).
3. Protecting API endpoints from unauthorized access.

Summary

- Route protection via authentication: Verified user access
- Middleware checks authentication credentials
- Secures sensitive routes, supports role-based access
- Follow best practices for effectiveness

MASYNCTECH
SCHOOL

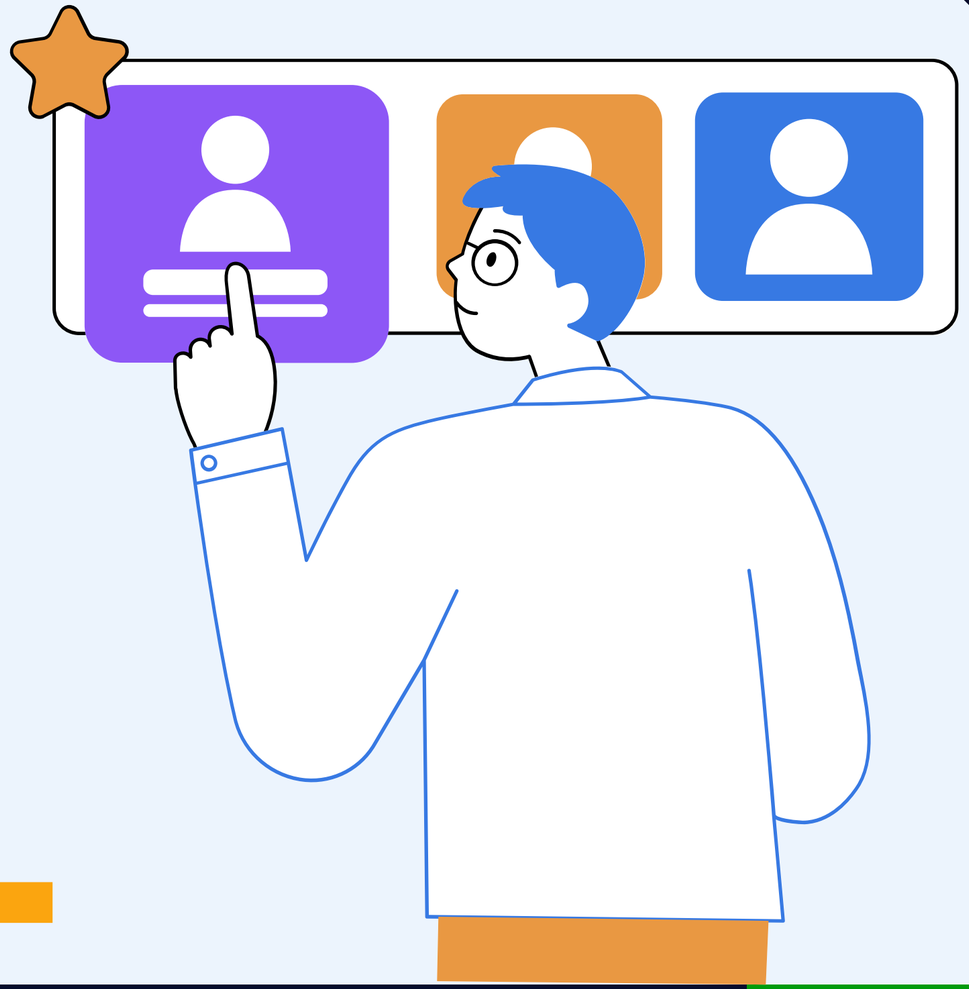
CODE DEMO

node JS

MASYNCTECH
SCHOOL

IMPLEMENTING ROLE-BASED

ACCESS CONTROL



express JS

EXPRESS ROUTER

IMPLEMENTING ROLE-BASED ACCESS CONTROL

MASYNCTECH SCHOOL

High-Level Explanation

- Role-based access control (RBAC): VIP sections with wristbands
- Wristband color determines room access
- Differentiate user roles for specific permissions

Analogue

- Analogy: RBAC as music festival wristbands
- Ticket assigns wristband color for stage access
- 'General Admission' vs. 'VIP' wristbands
- RBAC checks 'digital wristband' (user role) for access

Important Rules

- Best practices for RBAC in Express Router:
1. Clearly define exclusive roles.
2. Avoid hardcoding; define roles as constants or config.
3. Implement role checks as middleware for reuse.
4. Follow the least privilege principle.
5. Respect role inheritance for higher-level roles.

Deep Dive

- RBAC in Express Router: Middleware checks user role
- Verify role from token or session data
- Control access to different routes based on roles
- Granular permission system for user-specific access

When to use?

- Use RBAC in Express Router for:
1. Multi-user systems with diverse user types.
2. Admin dashboards for admin or higher roles.
3. Restricting access to sensitive data or endpoints based on roles.

Summary

- RBAC: Limits route access by user roles
- Granular control in multi-user systems
- Useful for admin dashboards, resource restrictions

MASYNCTECH
SCHOOL

CODE DEMO

node JS

MASYNCTECH
SCHOOL

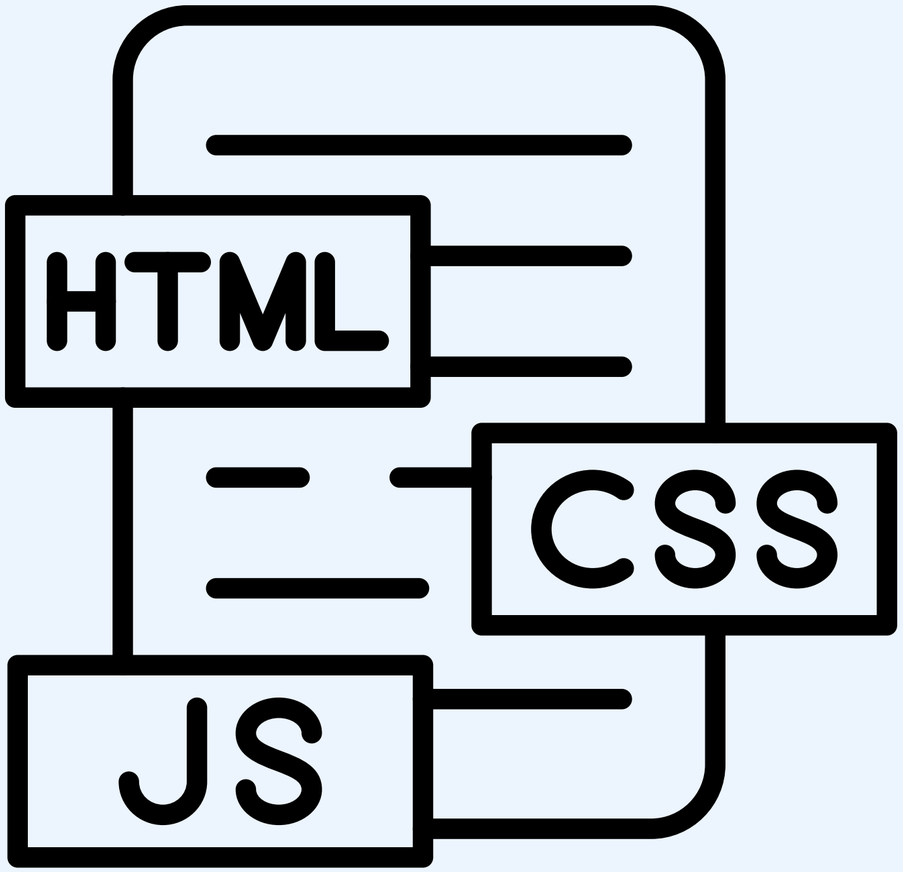
THE EXPRESS.STATIC MIDDLEWARE

express JS

SERVING STATIC ASSETS

MASYNCTECH
SCHOOL

SERVING STATIC ASSETS



express JS

SERVING STATIC ASSETS

THE EXPRESS.STATIC MIDDLEWARE

MASYNCTECH SCHOOL

High-Level Explanation

- `express.static` in Express.js: Middleware for serving static files
- Delivers images, CSS, JavaScript from a directory

Analogue

- Analogy: `Express.static` as a signpost in a bookshelf
- Points to readily accessible books (static files)
- No need for permission or complex routes
- Simplifies direct access to specific files

Important Rules

- Best practices for `Express.static`:
  - Use recognizable directory names like "public" or "assets."
  - Carefully manage cache-control settings.
  - Avoid storing sensitive data in this directory, as it's publicly accessible.

Deep Dive

- `express.static` middleware uses a directory name as its first argument.
- Handles requests for static files in that directory.
- Simplifies serving static resources like images, CSS, and JavaScript.
- Commonly used directory name is "public" for easy access to static files within it.

When to use?

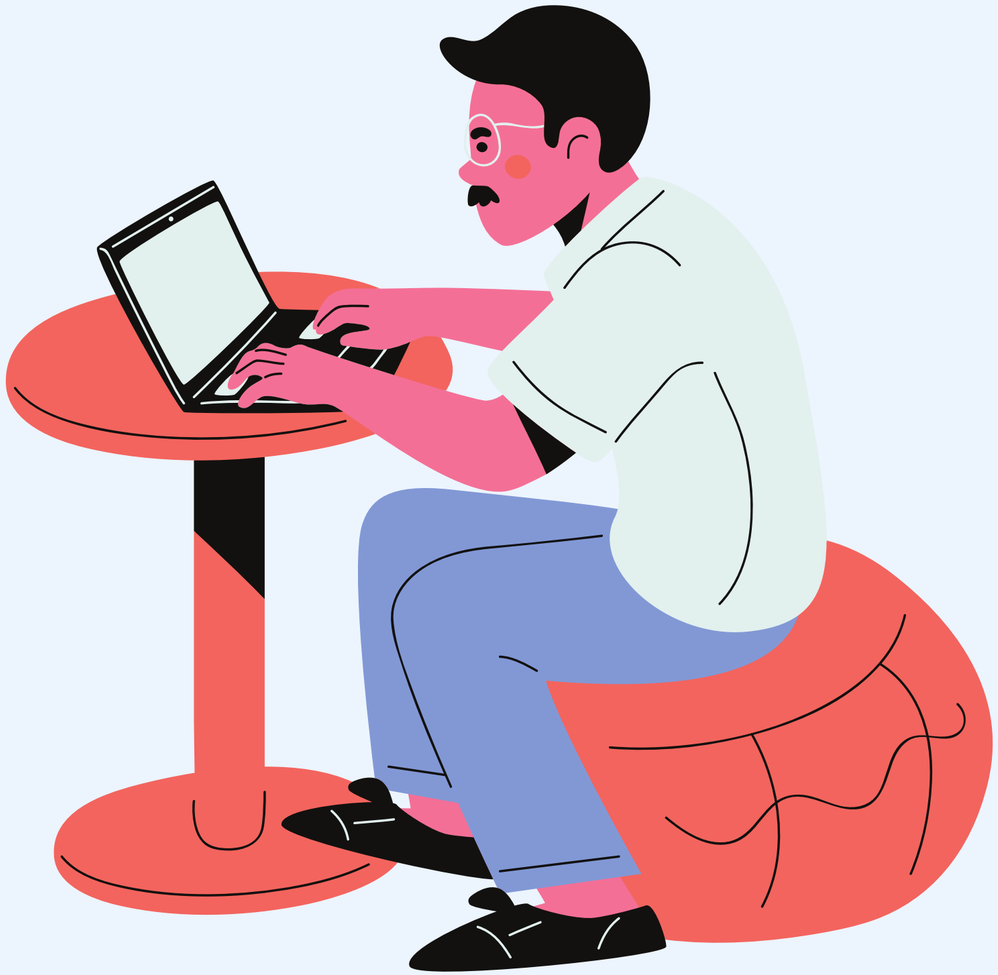
- Use `Express.static` for client-side fetchable files.
- No special permissions or logic needed.
- Ideal for non-sensitive, non-dynamic files like images, stylesheets, scripts.

Summary

- Express.static: Essential for serving static files in Express.js.
- Simplifies routing for static files by specifying a directory.
- Best suited for non-sensitive, client-side files.
- Configure with care for performance and security.

MASYNCTECH
SCHOOL

CODE DEMO



node JS

MASYNCTECH
SCHOOL

UNDERSTANDING TEMPLATE ENGINES

express JS

SERVING STATIC ASSETS

UNDERSTANDING TEMPLATE ENGINES

MASYNCTECH SCHOOL

High-Level Explanation

- Template engine: Inject dynamic data into HTML
- Blueprint for webpages, replace placeholders with content

Analogue

- Analogy: Template engines as sandwich making
- Base HTML with placeholders, like bread and lettuce
- Engine adds specific data, just like fillings
- Customization without building from scratch

Important Rules

- Use one template engine for consistency.
- Keep logic separate from presentation.
- Utilize partials for reusable code (e.g., headers).
- Maintain clean, readable syntax for collaboration.

Deep Dive

- EJS, Handlebars, Pug simplify data transfer server-to-client.
- Create HTML templates with placeholders.
- Replace placeholders with data at runtime.
- Easier dynamic website development with server-side rendering.

When to use?

- Template engines for dynamic web apps with server-side rendering.
- Ideal for UI updates based on backend data.
- Useful in e-commerce, blogs, dashboards, etc.

Summary

- Express.js template engines: Inject dynamic data.
- Serve as blueprints filled with dynamic content.
- Ideal for dynamic web apps, maintainability.
- Best practices: code reusability, clean syntax.

MASYNCTECH
SCHOOL

EJS TEMPLATE ENGINE

express JS

EJS

INTRODUCING EJS TEMPLATE ENGINE

MASYNCTECH SCHOOL

High-Level Explanation

- Template engine: Inject dynamic data into HTML
- Blueprint for webpages, replace placeholders with content

Analogue

- EJS analogy: Birthday card template
- Same design, customize name, age, message
- One template with placeholders
- EJS replaces placeholders with actual data

Important Rules

- Isolate business logic from templates.
- Utilize partials for reusable components.
- Sanitize user content to prevent XSS attacks.
- Maintain readable EJS templates with indentation and comments.

Deep Dive

- EJS: Simple and user-friendly.
- No new syntax; uses HTML and JavaScript.
- Embed JavaScript in templates for logic.
- Familiarity with frontend tech eases adoption.

When to use?

- EJS is valuable for quick prototyping.
- Ideal for JavaScript-savvy teams.
- Great for server-side rendering, dynamic data sites.
- Useful in e-commerce, blogs, dashboards.

Summary

- EJS: Inject dynamic data with JavaScript in HTML.
- Great for server-side rendering, JavaScript devs.
- Maintainability and security with best practices.
- Keep logic separate, use partials for reusability.

MASYNCTECH
SCHOOL

EJS COMMON SYNTAX

express JS

TEMPLATE ENGINE (EJS)

COMMON EJS SYNTAX

MASYNCTECH
SCHOOL

EJS template tags:

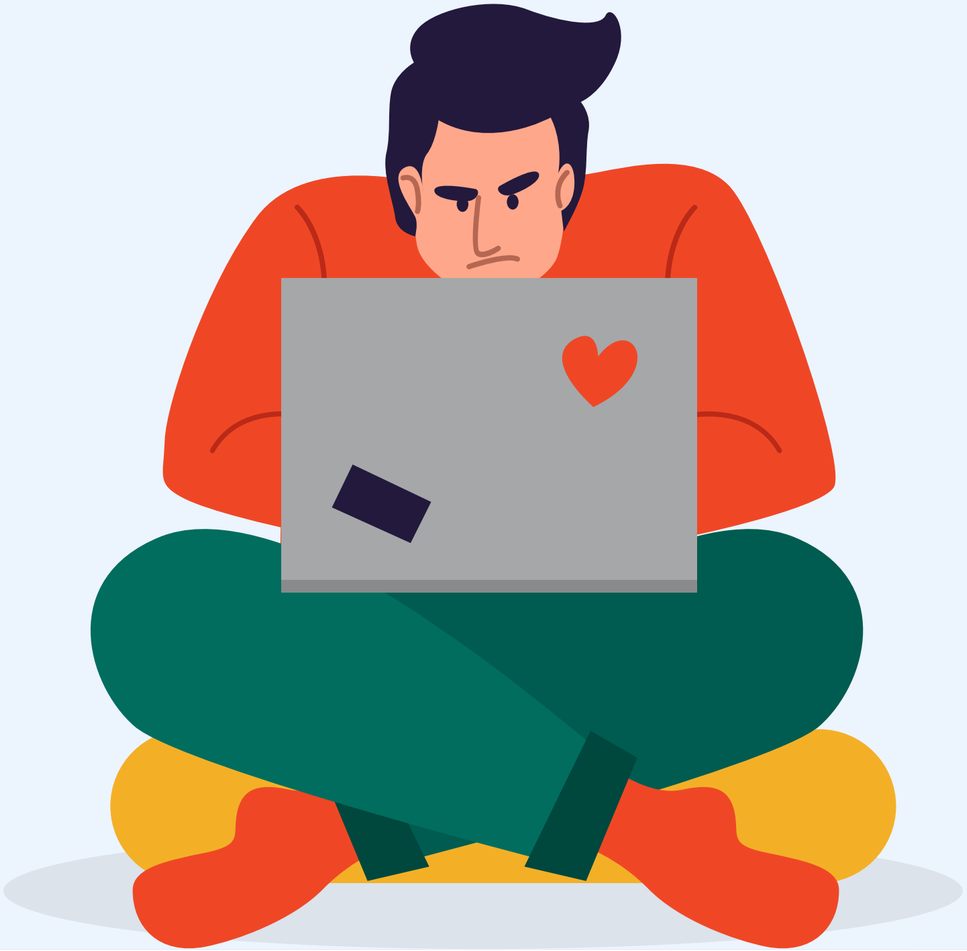
1. `<%= variable %>`: Output value, HTML-escaped.
2. `<%- variable %>`: Output unescaped value.
3. `<% code %>`: Execute JavaScript, no output.
4. `<%_ code _%>`: Same as `<% code %>,` strip whitespace.
5. `<%# comment %>`: Add non-visible HTML comment.
6. `<%%>`: Output literal `<%`.
7. `<% include('file') %>`: Include another EJS file (partials).

MASYNCTECH
SCHOOL

RENDER

TEMPLATES USING

EJS

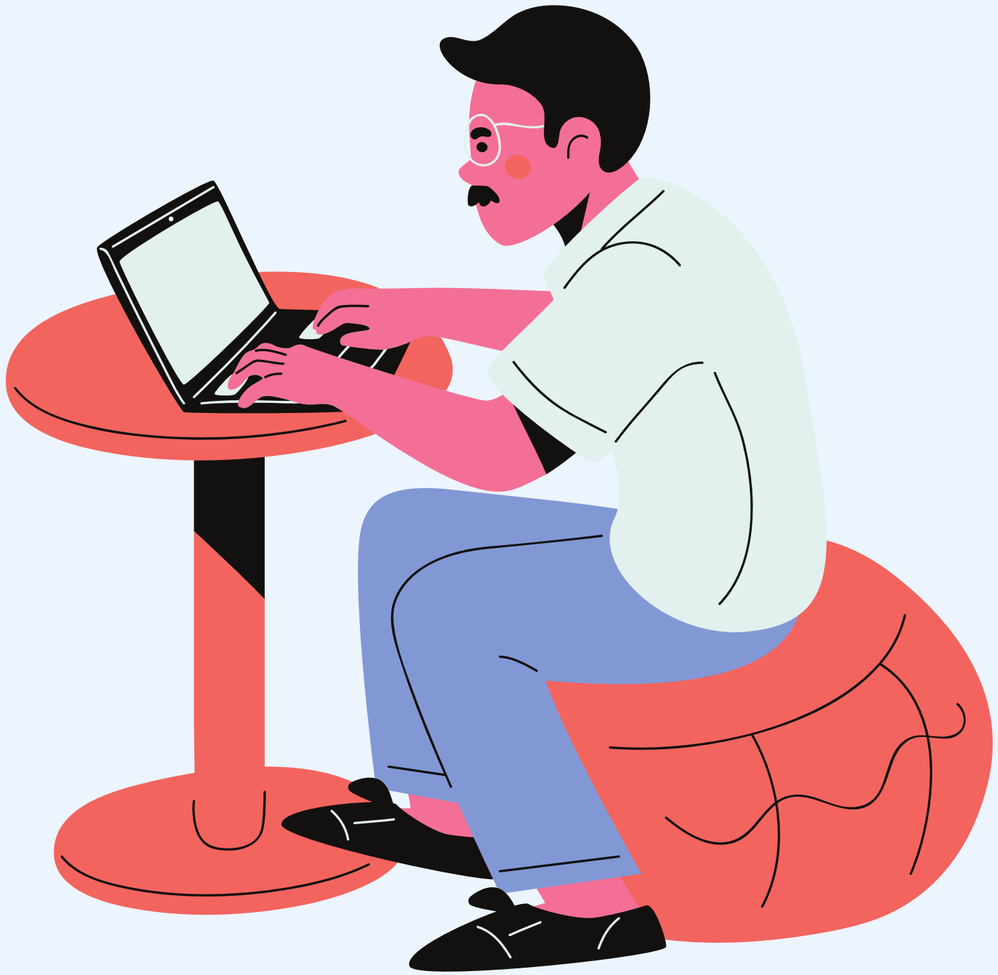


express JS

TEMPLATE ENGINE (EJS)

MASYNCTECH
SCHOOL

CODE DEMO



node JS

MASYNCTECH
SCHOOL

PASSING DATA TO TEMPLATES

express JS

TEMPLATE ENGINE (EJS)

PASSING DATA TO TEMPLATE

MASYNCTECH SCHOOL

High-Level Explanation

- EJS: Send server-side variables to templates
- Generate dynamic content in templates

Deep Dive

- EJS template: Inject dynamic content with `<%= %>`
- Execute JavaScript with `<% %>` tags, no direct output

Analogue

- Analogy: EJS templates like crafting tweets
- `<%= friendName %>` dynamically inserts friend's name
- Change `friendName` variable for automatic updates

When to use?

- EJS use cases:
  1. Display user info
  2. Generate tables/lists from data
  3. Personalize content per user preferences

Important Rules

- EJS best practices:
  1. Sanitize data to prevent XSS attacks.
  2. Keep template logic minimal for maintainability.
  3. Use clear variable names for readability.

Summary

- Pass data to EJS templates from server-side code.
- Populate HTML dynamically for personalization or database data.

MASYNCTECH
SCHOOL

Syntax

Passing Data

```
res.render('viewName', { key1: value1, key2: value2, ... });\
```

- `viewName`: Name of EJS template to render (no `.ejs`).
- `{ key1: value1, key2: value2, ... }`: Data for template.
  - Keys = template variables
  - Values = actual data to pass

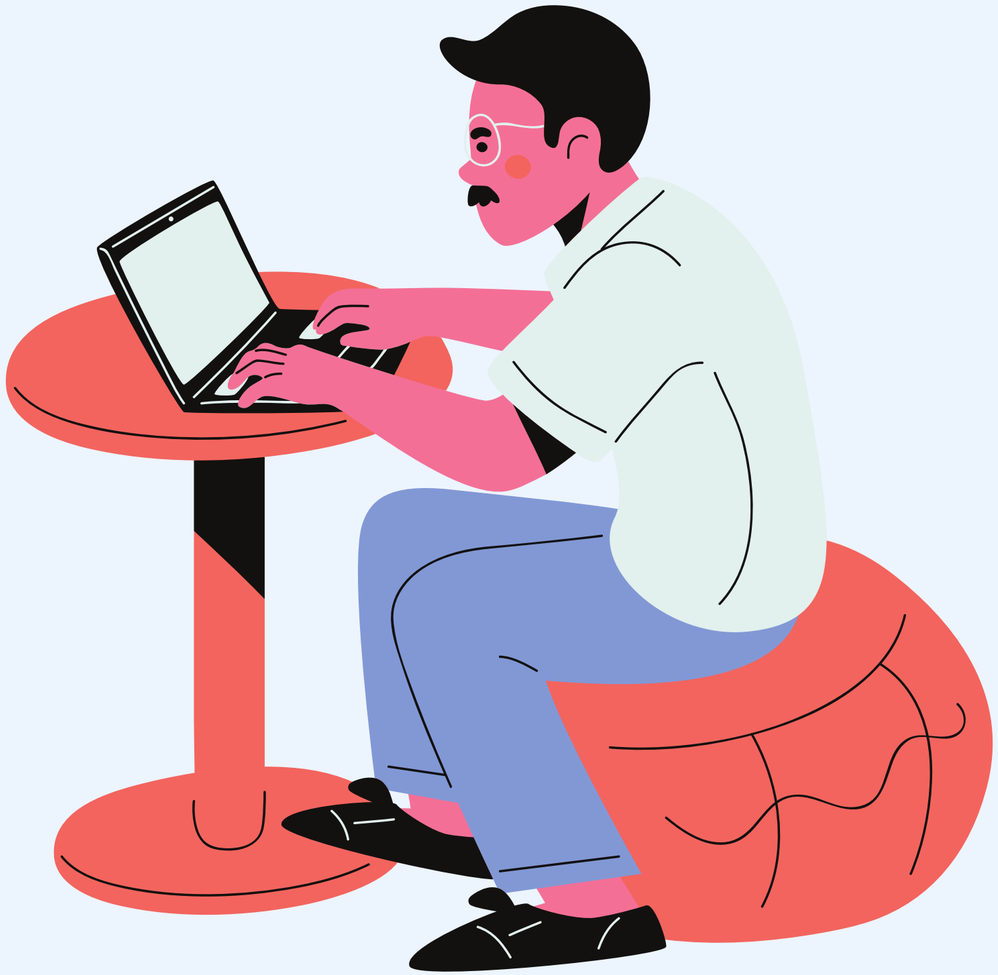
Accessing Data

```
<!-- Using <%= %> to display a variable -->
<h1><%= title %></h1>

<!-- Using <%- %> to insert HTML -->
<div><%- rawHTML %></div>
```

MASYNCTECH
SCHOOL

CODE DEMO



node JS

MASYNCTECH
SCHOOL

UTILIZING CONDITIONALS IN EJS

express JS

TEMPLATE ENGINE (EJS)

Analogue

- Web page scenario: User can be logged in or not.
- Goal: Show "Welcome, [username]!" when logged in, "Please log in" when not.
- Achieved through conditional HTML and CSS classes.

MASYNCTECH
SCHOOL

Syntax

Syntax for If-Else Statements

```
<% if (condition) { %>
  <!-- HTML content when the condition is true -->
<% } else { %>
  <!-- HTML content when the condition is false -->
<% } %>
```

Syntax for If-Else If-Else Statements:

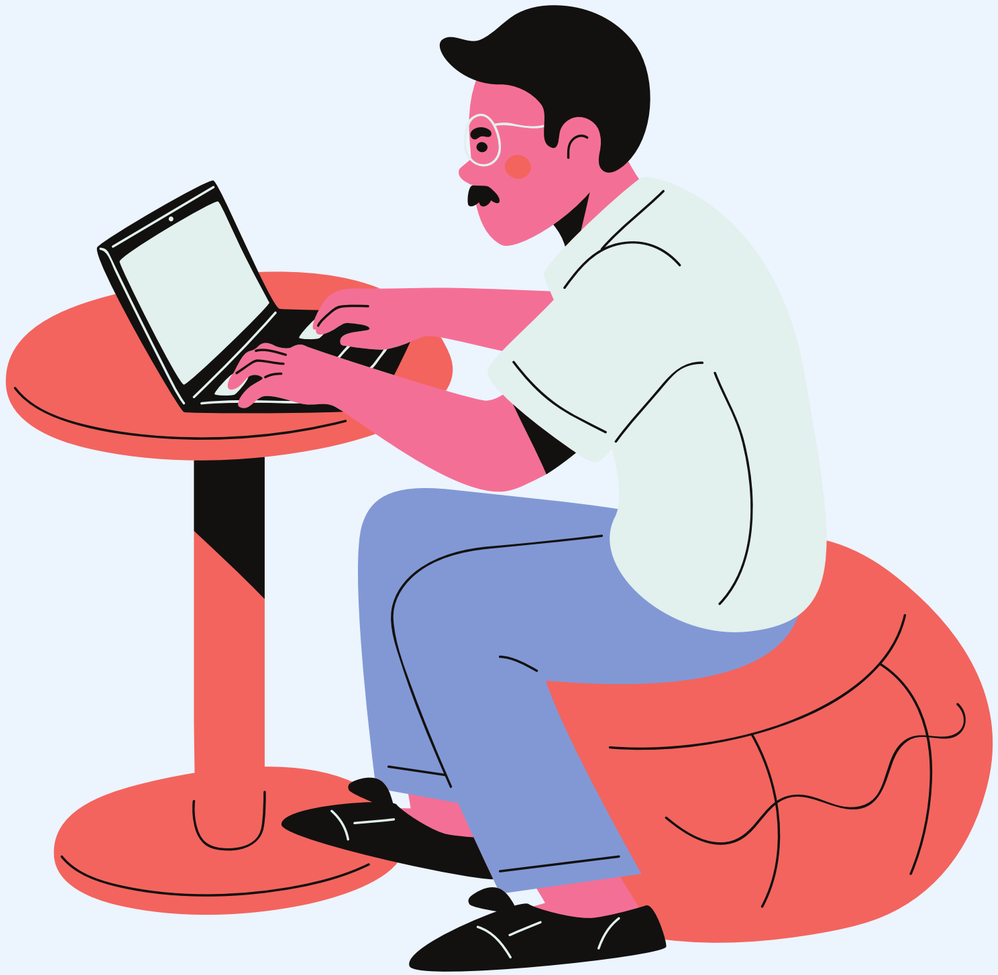
```
<% if (condition1) { %>
  <!-- HTML content when condition1 is true -->
<% } else if (condition2) { %>
  <!-- HTML content when condition2 is true -->
<% } else { %>
  <!-- HTML content when neither condition is true -->
<% } %>
```

Syntax for Ternary Operator:

```
<%= (condition) ? 'HTML or Text when true' : 'HTML or Text when false' %>
```

MASYNCTECH
SCHOOL

CODE DEMO



node JS

MASYNCTECH
SCHOOL

UTILIZING LOOPS IN EJS

express JS

TEMPLATE ENGINE (EJS)

MASYNCTECH
SCHOOL

Syntax

Syntax for For Loop

```
<% for(let i = 0; i < array.length; i++) { %>
  <!-- HTML content -->
  <%= array[i] %>
<% } %>
```

Syntax for For..Of Loop

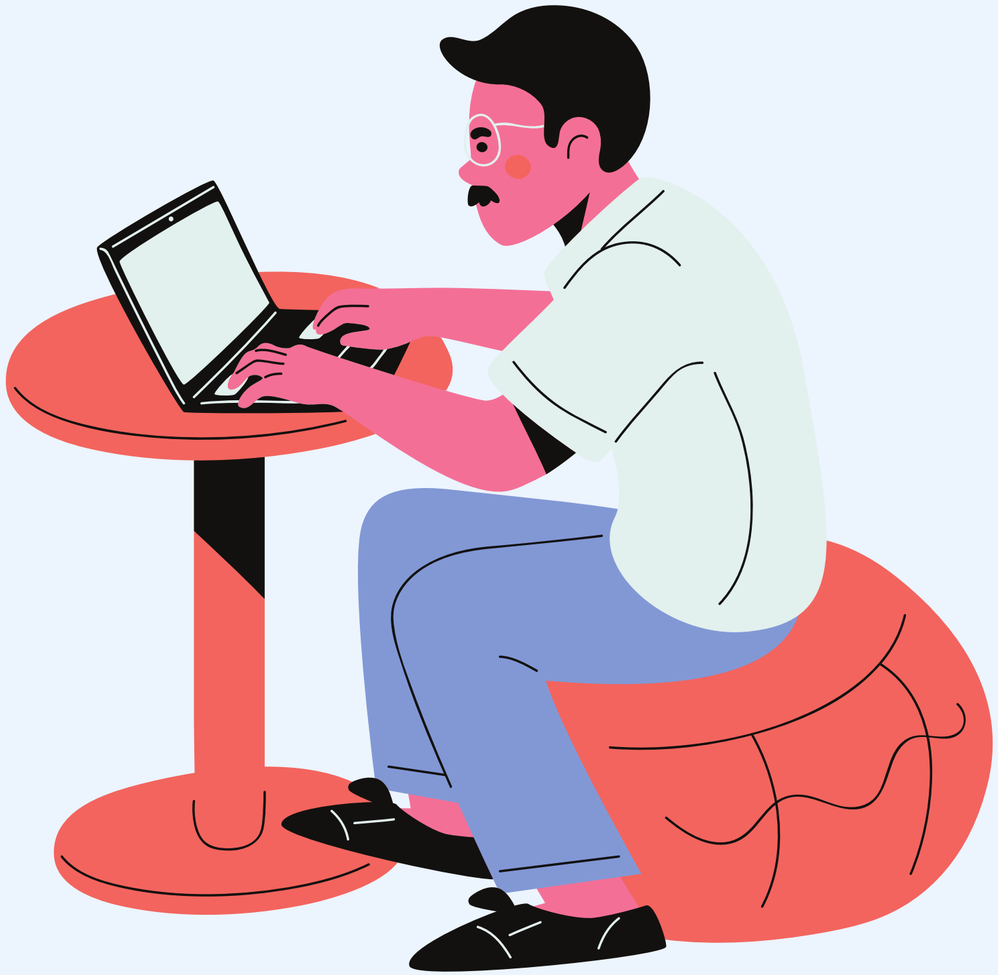
```
<% for(let item of array) { %>
  <!-- HTML content -->
  <%= item %>
<% } %>
```

Syntax for Ternary Operator

```
<% array.forEach(function(item) { %>
  <!-- HTML content -->
  <%= item %>
<% }); %>
```

MASYNCTECH
SCHOOL

CODE DEMO



node JS

MASYNCTECH
SCHOOL

EJS
PARTIALS

express JS

TEMPLATE ENGINE (EJS)

EJS PARTIALS

MASYNCTECH SCHOOL

High-Level Explanation

- EJS Partials: Smaller templates in EJS
- Included in larger EJS templates
- Break down complex pages into reusable components

Analogue

- Analogy: Website as a LEGO castle
- EJS Partials as reusable LEGO rooms
- Build once, add anywhere, like modular building blocks

Important Rules

- Descriptive names for clear partial purposes.
- Relative paths for flexible inclusion.
- Careful variable handling; ensure data availability.

Deep Dive

- EJS Partials: Modular web development approach
- Encapsulate reusable code in separate EJS files
- Include in multiple EJS templates
- Maintain DRY codebase, enhance maintainability, reduce errors

When to use?

- For shared sections on multiple pages (headers, footers, menus).
- To maintain DRY, easily managed code.
- Efficient site-wide updates without altering code in many places.

Summary

- EJS Partials: Mini-templates in EJS
- Ideal for reusable segments like headers, footers
- Enhance code maintainability
- Follow DRY principle

Syntax

```
<%- include('path/to/partial-file') %>
```

```
<%- include('path/to/partial-file', {someData: 'data'}) %>
```

- `<%-` and `%>`: EJS code delimiters for execution. `<%-` renders HTML from included files.
- `include`: EJS function to include partials.
- `'path/to/partial-file'`: Path to the partial EJS file.
- `{someData: 'data'}`: Optional object to pass local variables to the partial.

MASYNCTECH
SCHOOL

EJS
LAYOUTS

express JS

TEMPLATE ENGINE (EJS)

EJS LAYOUTS

MASYNCTECH SCHOOL

High-Level Explanation

- EJS layouts: Base template with dynamic content placeholders.
- Create a master layout with common structure.
- Inject page-specific content into this structure.

Analogue

- Analogy: EJS layouts as PowerPoint master slides.
- Master slide defines common elements.
- Individual slides only require unique content.
- EJS layouts serve as the master template for web pages.

Important Rules

- Descriptive layout file names.
- Include only common elements in the layout.
- Use a separate EJS file for the layout.
- Specify content injection point, e.g., `<%- body %>` in the layout.

Deep Dive

- EJS layout: Blueprint for web pages with common elements.
- Wraps around specific views, adding view-specific content.
- Enhances maintainability, scalability via DRY principle.

When to use?

- For pages with shared structures.
- To make site-wide changes efficiently.
- To ensure scalability and maintainability.

Summary

- EJS layouts: Master templates for common structure.
- Prevent code repetition.
- Insert unique, page-specific content into standardized frame.

MASYNCTECH
SCHOOL

Syntax

```
<!DOCTYPE html>
<html>
<head>
  <title>My App</title>
</head>
<body>
  <%- include('partials/header') %>

  <!-- Content will be inserted here -->
  <%- body %>

  <%- include('partials/footer') %>
</body>
</html>
```

MASYNCTECH
SCHOOL

CREATING DYNAMIC ERROR PAGE

express JS

TEMPLATE ENGINE (EJS)

CREATING DYNAMIC ERROR PAGES

MASYNCTECH SCHOOL

High-Level Explanation

- Error pages are dynamic, vary by error type.
- 404 "Not Found" differs from 500 "Internal Server Error".

Analogue

- Dynamic error pages tailor responses to specific errors.
- Inaccessible movie: "Couldn't find that movie" + alternatives.
- Internal issue: "Something went wrong on our end" page.

Important Rules

- Error middleware last in stack for proper handling.
- Protect sensitive info, no stack traces in production errors.

Deep Dive

- Express.js handles errors with middleware functions.
- Custom error-handling middleware defined after routes.
- Captures errors and renders corresponding dynamic error pages with error details.

When to use?

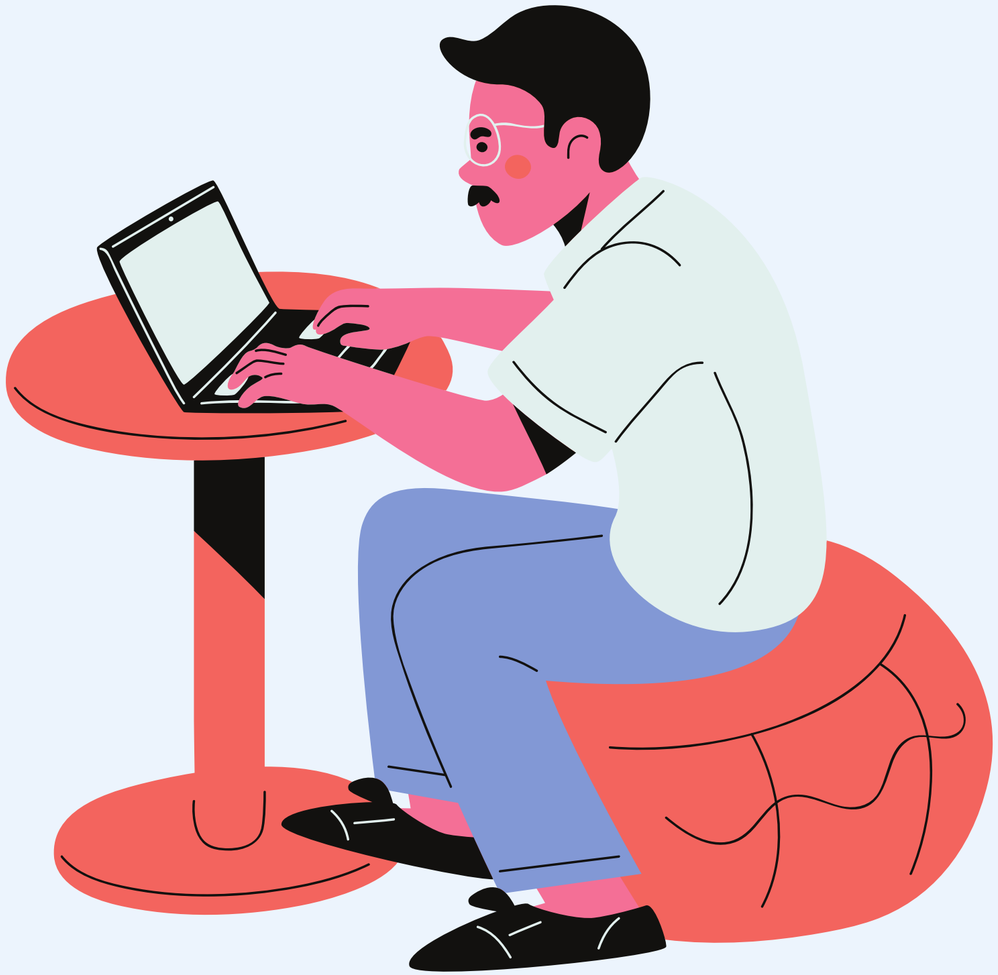
- Enhance UX with informative, customized errors.
- Consistent branding on all pages, including errors.

Summary

- Express.js dynamic error pages customized via middleware.
- Improve UX, ensure branding consistency.

MASYNCTECH
SCHOOL

CODE DEMO



node JS

MASYNCTECH
SCHOOL

AUTHENTICATION AND AUTHORIZATION

express JS

AUTHENTICATION

MASYNCTECH
SCHOOL

AUTHENTICATION

VS

AUTHORIZATION

express JS

AUTHENTICATION

AUTHENTICATION VS AUTHORIZATION

MASYNCTECH
SCHOOL

AUTHENTICATION VS AUTHORIZATION

MASYNCTECH SCHOOL

High-Level Explanation

- **Authentication**: Verify who you are (e.g., showing ID at a nightclub).
- **Authorization**: Verify what you can do (e.g., wristband for drink purchases).

Analogue

- Analogy: Authentication as cover charge and ID check.
- Verify legitimacy and entry.
- Authorization as wristbands for VIP and alcohol access.
- Dictate actions inside once you're in.

Important Rules

Authentication:

- Secure channel (HTTPS) for data transmission.
- Implement multi-factor authentication (MFA).
- Store passwords in hashed formats.

Authorization:

- Principle of least privilege (minimal permissions).
- Regular permission and role audits.
- Role-based or claims-based authorization.

Deep Dive

- **Authentication**: Verify user/device/system identity.
- **Methods**: Username/password, OTPs, smart cards, biometrics, etc.
- **Authorization**: After authentication, handles permissions.
- Defines resource access and actions.
- Managed via roles, permissions, claims in tokens or databases.

When to use?

- **Authentication**: Required for systems restricting access.
- Examples: Email login, online banking access.
- **Authorization**: Essential for varying access levels.
- Examples: Company portal, HR, financial data access.

Summary

- Authentication: Verify user identity.
- Authorization: Define user actions.
- Essential for security and functionality, often work together.

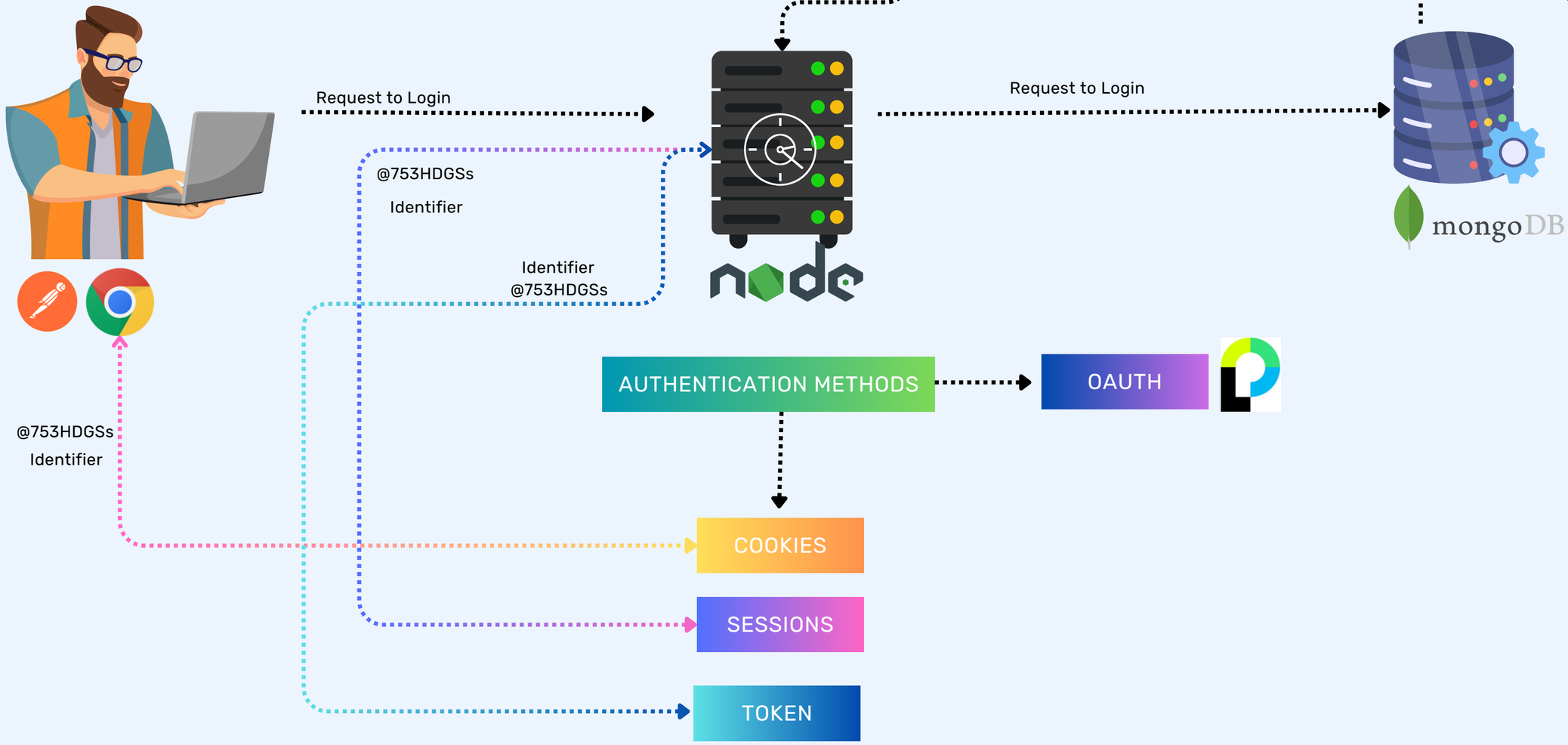
MASYNCTECH
SCHOOL

GENERAL AUTHENTICATION FLOW

express JS

AUTHENTICATION

AUTHENTICATION FLOW



GENERAL AUTHENTICATION FLOW

MASYNCTECH
SCHOOL

High-Level Explanation

- General Authentication Flow: Steps to access secure resource
- Analogous to proving identity to open a locked door

Deep Dive

- Typical web app authentication flow:
  - User submits username/password
  - Credentials checked against database
  - Valid: Create session/token for authentication
  - Session/token identifies user for interaction
  - Access protected resources using it

Analogue

- Analogy: Logging into an online game
- Username/password -> "gamer badge"
- Authentication flow: Login -> Badge -> Access privileges

Summary

- General Authentication Flow: Verify user identity
- Start: User provides credentials (e.g., username, password)
- System verifies credentials
- Authenticated: User gets session/token
- Use session/token to access secured resources

MASYNCTECH
SCHOOL

COOKIE BASED AUTHENTICATION

express JS

AUTHENTICATION

HOW COOKIE-BASED AUTH WORKS

HOW COOKIE AUTHENTICATION WORKS

MASYNCTECH SCHOOL

High-Level Explanation

- Cookie authentication: Hand stamp at a festival
- 1st entry: Show ticket, get hand stamp
- Later entries: Show stamp, skip ticket process

Analogue

- Logging into Netflix: Email, password -> "cookie"
- Cookie sent with each request to identify user
- No need to log in again for each action

Important Rules

- Transmit cookies over HTTPS for security.
- Utilize secure, HttpOnly, and SameSite flags for added protection.
- Implement session expiration for stolen cookie mitigation.
- Regularly rotate session identifiers for security.

Deep Dive

- Cookie authentication: Verify credentials on 1st login
- Successful login: Server sends a unique "cookie" to browser
- Cookie stored locally, sent with every subsequent request
- Server uses cookie to confirm identity, grant access

When to use?

- Cookie authentication ideal for server-rendered apps
- Maintains logged-in state across requests
- Provides seamless user experience, no repeated logins

Summary

- Cookie authentication: Maintains user session
- After login, server issues a cookie
- Cookie sent with each request
- Identifies user, keeps session active

MASYNCTECH
SCHOOL

SESSION BASED AUTHENTICATION

express JS

AUTHENTICATION

HOW SESSION AUTHENTICATION WORKS?

HOW SESSION AUTHENTICATION WORKS?

MASYNCTECH
SCHOOL

High-Level Explanation

-Session authentication like a theme park wristband
- Wristband indicates paid access
- Staff checks wristband for ride access

Analogue

- Analogy: Festival entrance with wristband
- Show ticket, get wristband
- Wristband = session ID in web browser
- Easier access to different areas, no ticket needed

Important Rules

- Implement session timeout for security.
- Securely store session data to prevent hijacking.
- Use HTTPS to prevent interception.
- Rotate session IDs after login to prevent "session fixation" attacks.

Deep Dive

- Session authentication: Validate credentials
- Server creates unique session ID on valid login
- Session ID stored server/client (usually in a cookie)
- Client sends session ID in subsequent requests
- Server retrieves session data, confirms authentication
- Eliminates multiple logins for user

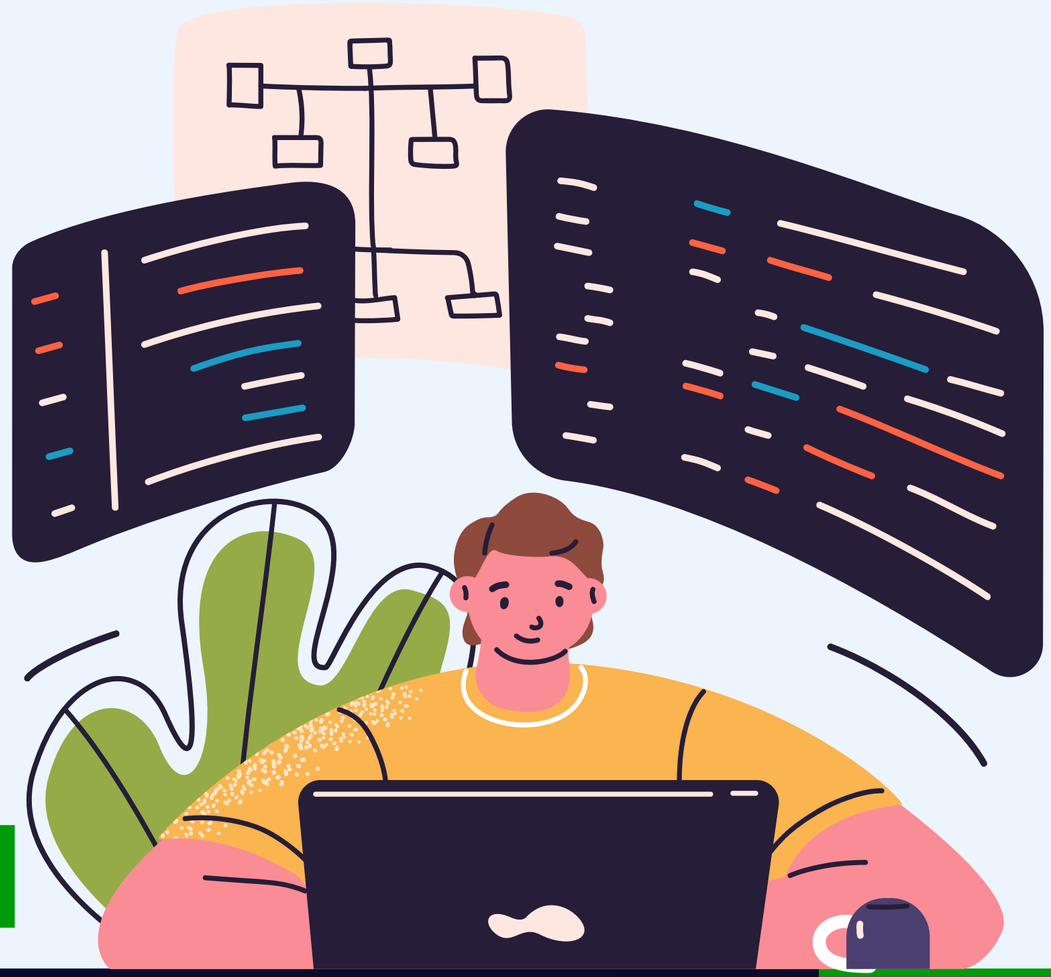
When to use?

- Session authentication for:
  - Apps needing login, temporary authentication
  - Tracking user activity during the session

Summary

- Session authentication: Unique session ID generated after login.
- ID stored both client and server-side.
- Enables server to remember user across requests.
- No need for repeated logins.

TOKEN BASED AUTHENTICATION



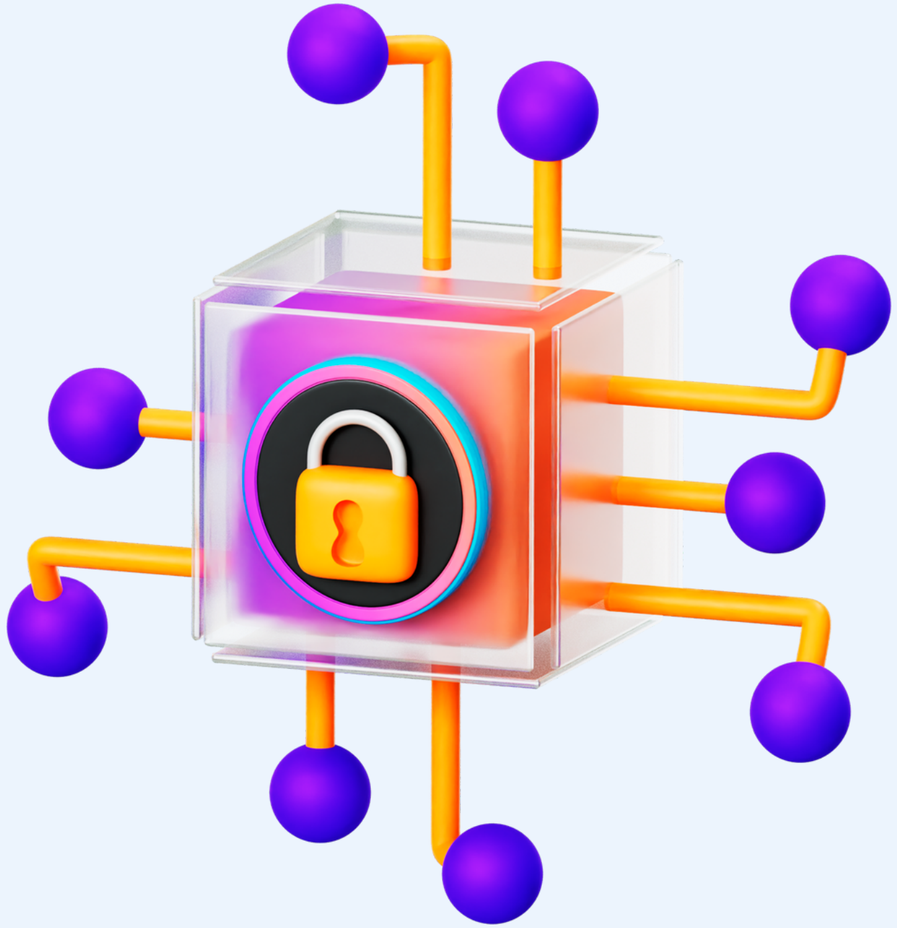
express JS

AUTHENTICATION

MASYNCTECH SCHOOL

MASYNCTECH
SCHOOL

JSON WEB TOKEN



express JS

AUTHENTICATION

WHAT IS JSON WEB TOKENS (JWT)

MASYNCTECH SCHOOL

High-Level Explanation

- JSON Web Tokens (JWT): Compact, secure identity confirmation
- Like a secure online note confirming your identity
- Used for representing claims between parties

Analogue

- Analogy: JWT as a club wristband
- Show ID, get wristband
- Wristband = JWT
- Proves identity and access, no need for repeated login

Important Rules

- Best practices for JWT use:
  1. HTTPS for token security.
  2. Short token lifetimes for reduced risk.
  3. Encrypt sensitive data in JWT.
  4. Standard claim names for clarity.
  5. Avoid storing sensitive info directly in JWT unless encrypted.

Deep Dive

- JWT structure: Header, payload, signature
- Header: Token type, signing algorithm
- Payload: Claims about an entity (e.g., a user)
- Signature: Ensures message integrity
- Encoded and concatenated with periods (.)
- Stateless, self-contained validation
- Scalable for multiple servers, no need for session storage

When to use?

- JWT advantages:
  - Stateless, scalable APIs
  - Mobile authentication ease
  - Simplified cross-domain/CORS
  - Server-to-server authorization

Summary

- JWTs: Secure, efficient claim representation
- Self-contained, carry verification and payload
- Scalable, stateless, like digital wristbands
- Access resources without repeated identity proof

MASYNCTECH
SCHOOL

HOW JWT WORKS

express JS

AUTHENTICATION

HOW JWT AUTHENTICATION WORKS

HOW JWT AUTHENTICATION WORKS

MASYNCTECH SCHOOL

High-Level Explanation

- JWT Authentication: VIP pass for a concert
- Log in, get JWT "VIP pass"
- Grants access to specific areas/routes
- No need to repeatedly prove identity

Analogue

- Analogy: JWT as a festival wristband
- Show ticket, get wristband (JWT)
- Wristband = access proof, no repeated proving

Deep Dive

- JWT Authentication process:
  - User submits credentials (username, password)
  - Server verifies credentials
  - Valid: Server generates JWT, sends to client
  - Client stores JWT
  - Subsequent requests to protected routes: Client sends JWT in Authorization header
  - Server verifies token, processes or denies request

Summary

- JWT Authentication: Server confirms identity via token
- Token issued after valid login
- Simplifies user verification for future interactions
- Ideal for stateless, distributed systems

MASYNCTECH
SCHOOL

JWT STRUCTURE

express JS

AUTHENTICATION

STRUCTURE OF A JWT

STRUCTURE OF A JWT

MASYNCTECH SCHOOL

High-Level Explanation

- JWT like a sandwich with 3 layers: Header, Payload, Signature
- Layers joined with dots (.) to create JWT

Deep Dive

- JWT components:
  - **Header**: Token type, signing algorithm
  - **Payload**: User claims and subject statements
  - **Signature**: Ensures token integrity

Encoded and joined with periods (.) as: `[Header].[Payload].[Signature]`.

Analogue

- Analogy: JWT as a sandwich
- Header = top bread (sandwich type)
- Payload = filling (details)
- Signature = bottom bread (seals, legitimacy)

Summary

- JWT structure: Header, Payload, Signature
- Encoded and concatenated with periods
- Widely used for statelessness, scalability
- Suitable for various applications

MASYNCTECH
SCHOOL

MVC

DESIGNPATTERN

express JS

DESIGN ARCHITECTURE

MASYNCTECH
SCHOOL

DESIGN PATTERN

express JS

DESIGN ARCHITECTURE

WHAT ARE DESIGN PATTERNS?

MASYNCTECH SCHOOL

High-Level Explanation

- Design patterns: Templates for common software problems
- Best practices for efficient, maintainable code

Deep Dive

- Software design patterns from architecture
- Fundamental in modern programming
- Well-proven solutions for recurring challenges
- Apply to object creation, component communication, interface composition

Analogue

- Analogy: Design patterns as recipes in coding
- Proven formulas for common coding problems
- Avoid starting from scratch, work more efficiently

When to use?

- Design patterns ideal for complex problems
- Multiple solutions, pick effective, maintainable one
- Vital in team settings, ensure consistency

Important Rules

- Choose design patterns purposefully, not blindly.
- Ensure team awareness for consistency.
- Document or comment pattern use for future reference.

Summary

- Design patterns: Proven solutions for recurring issues
- Customizable templates for specific needs
- Valuable for efficient, maintainable, scalable code

MASYNCTECH
SCHOOL

MVC

DESIGNPATTERN

express JS

DESIGN ARCHITECTURE

OVERVIEW OF MVC

MASYNCTECH SCHOOL

High-Level Explanation

- MVC: Model-View-Controller
- Organizes web app code
- Model manages data
- View displays data
- Controller handles Model-View interaction

Analogue

- Analogy: Online shopping app as MVC
- Model: Warehouse with items for sale
- View: Shop display, cart for item selection
- Controller: Shop staff helping with item selection, cart management, payment

Important Rules

- **Single Responsibility:** Focus on one aspect (data, display, or control).
- **Loose Coupling**: Independent Models, Views, Controllers.
- **Strong Cohesion:** Components contain needed resources.
- **DRY (Don't Repeat Yourself)**: Reuse code.

Deep Dive

- MVC: Separates app into three components
  - **Model**: Data, business rules, DB communication
  - **View**: UI, displays Model data, sends commands
  - **Controller**: Interface, processes input, updates Model, returns output to View

When to use?

- MVC beneficial for separating concerns
- Effective in complex business logic
- Enhances collaboration among developers
- Improves maintainability and flexibility

Summary

- MVC: Model, View, Controller
- Organized, scalable, maintainable web apps

MASYNCTECH
SCHOOL

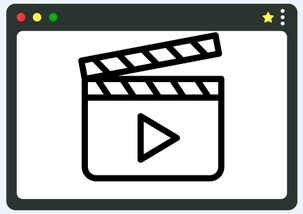
HOW MVC INTERACT

express JS

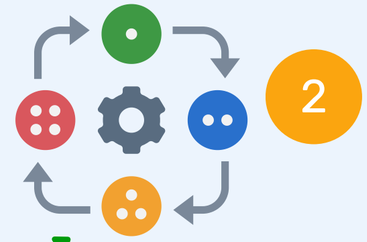
DESIGN ARCHITECTURE

HOW MVC INTERACTS

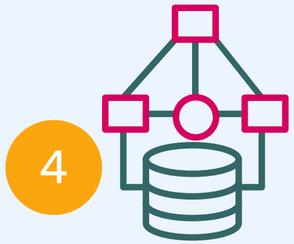
HOW MVC INTERACTS



V - View



C -Controller



M -Model

Database

HOW MVC INTERACTS

High-Level Explanation

- MVC: Model, View, Controller collaboration
- Controller manages user actions, updates Model
- Model updates View, displaying data to the user.

Analogue

- Analogy: MVC like a video game
- User (you) presses button (View)
- Game logic (Controller) decides outcome
- Model updates game (e.g., character jumps)
- View displays the result on the screen

Deep Dive

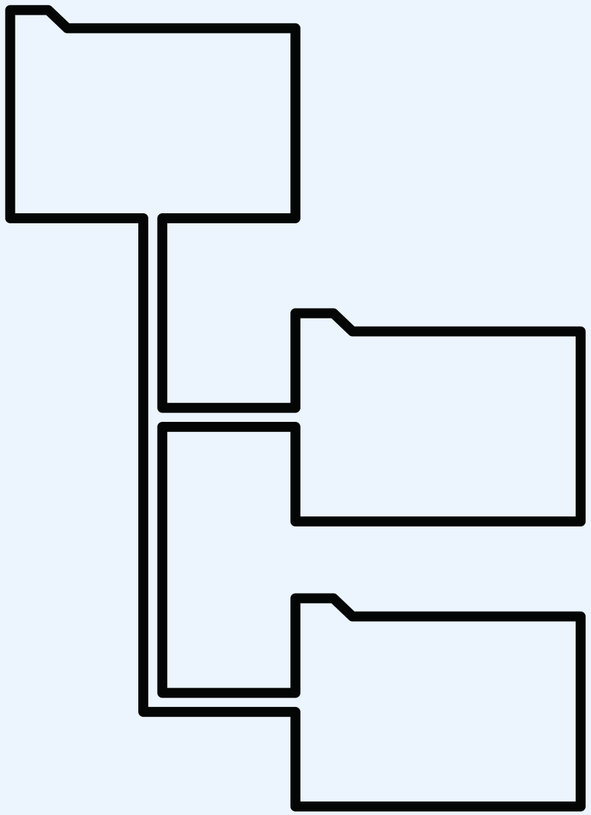
- MVC interaction:
  - User action in View, e.g., button click
  - Controller captures event, performs logic
  - Consults Model for data
  - Model executes logic, informs Controller
  - Controller updates View to reflect changes

Summary

- MVC: Controller mediates Model and View
- View actions → Controller functions
- Controller updates Model
- Model changes → View updates

MASYNCTECH
SCHOOL

MVC FOLDER STRUCTURE



express JS

DESIGN ARCHITECTURE

MVC FOLDER STRUCTURE

MASYNCTECH
SCHOOL

```
Project Root
├── models/
├── views/
├── controllers/
├── routes/
└── public/
```

MVC FOLDER STRUCTURE

High-Level Explanation

- MVC project structure:
  - `Project Root`: Main project folder
  - `models/`: Data, business logic
  - `views/`: UI templates
  - `controllers/`: Intermediary logic
  - `routes/`: URL path mapping
  - `public/`: Static files (CSS, images)