Learn with Rabbil

TERMINOLOGY

Mandatory Knowledge for
web development

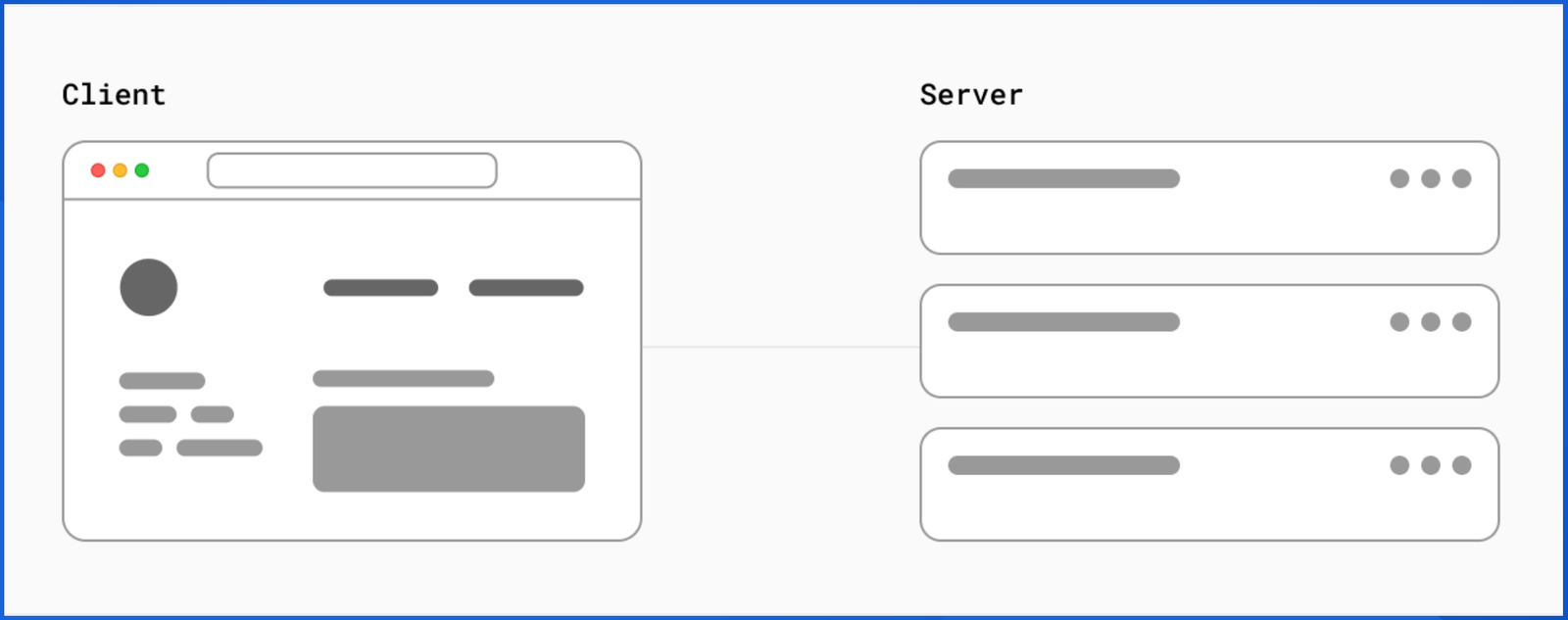
Client-Side

The client refers to the browser on a user's device that sends a request to a server for your application code.

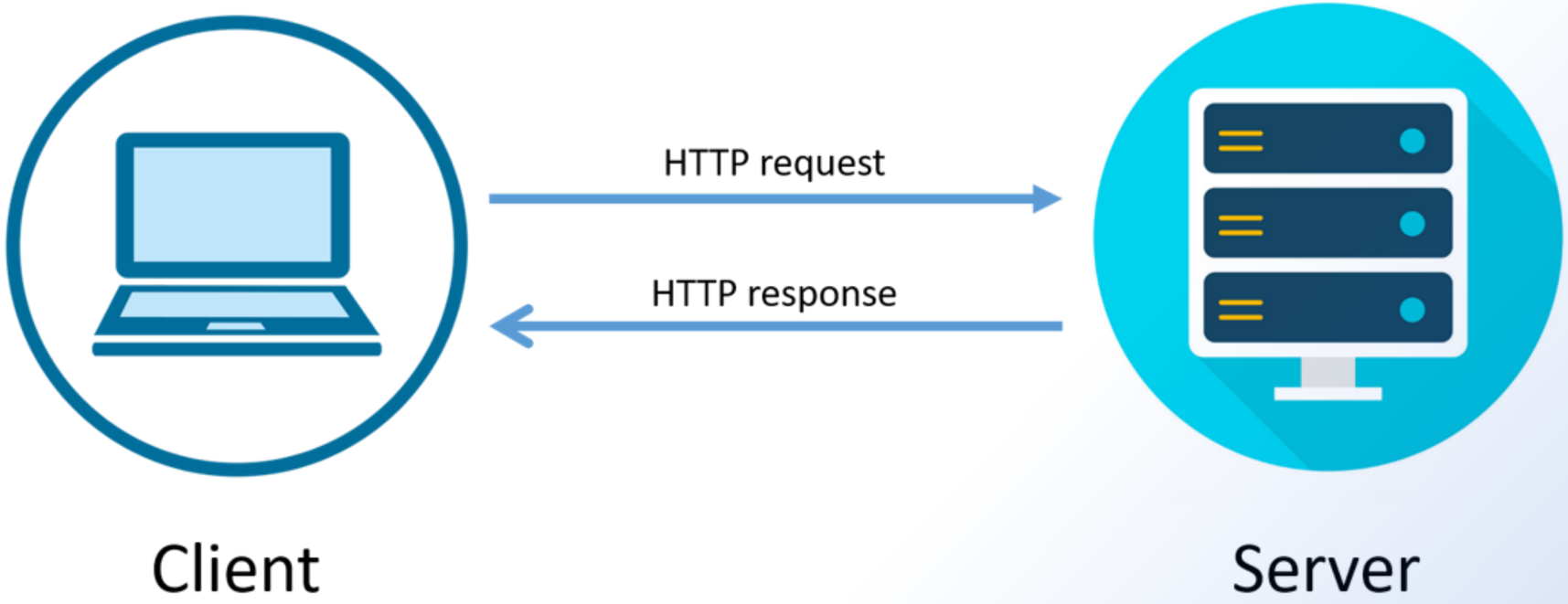
It then turns the response it receives from the server into an interface the user can interact with.

Server-Side

Server refers to the computer in a data centre that stores your application code, Receives requests from a client, does some computation, and sends back an appropriate response.

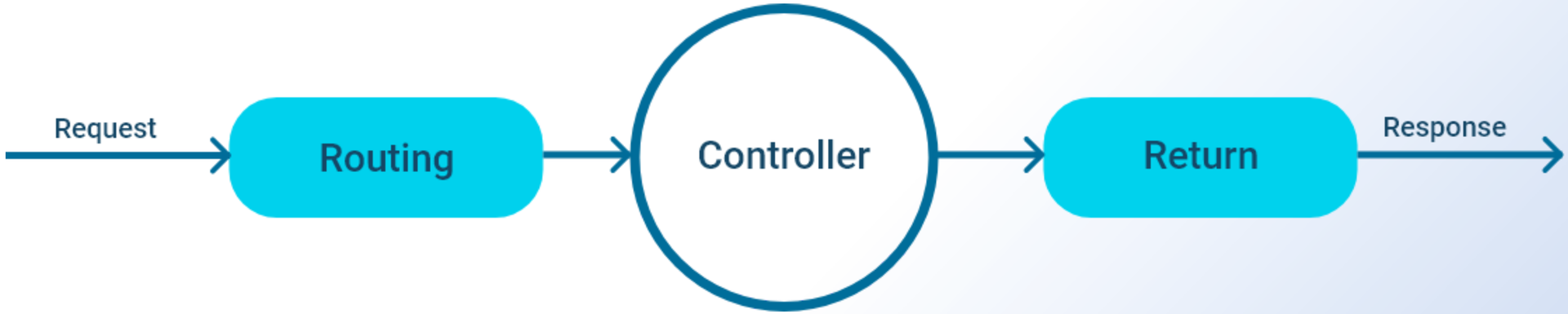


Request-Response Model



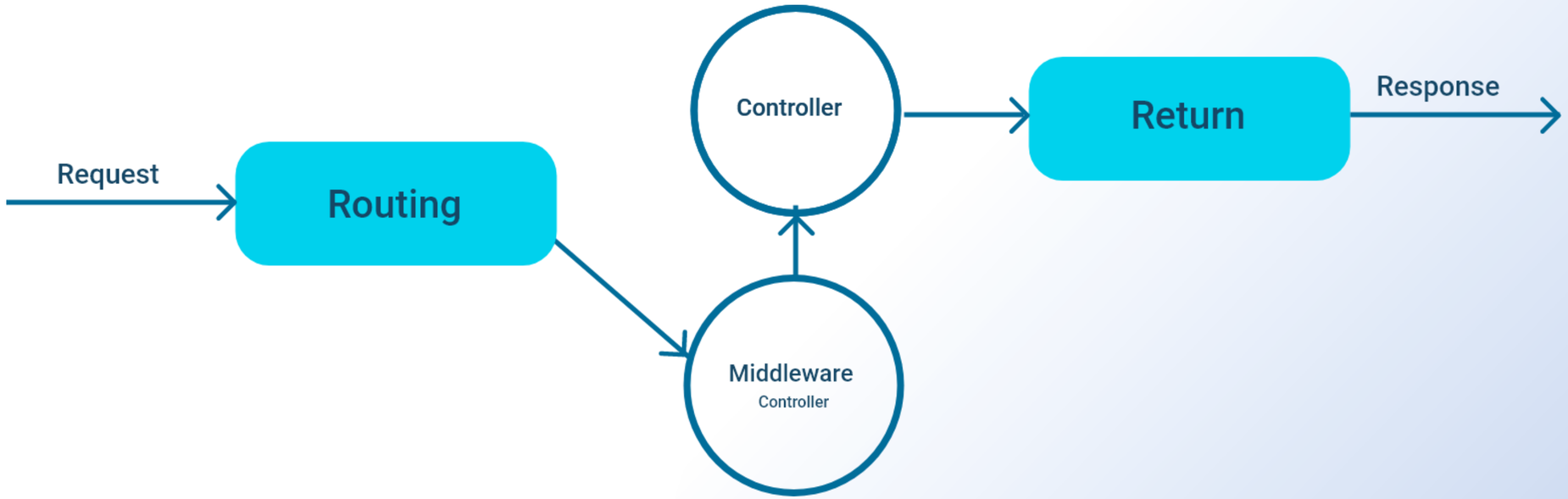
Request-Response Routing & Controller

- Routing is the URL/URI path to send request to server
- Controller is a function/class/code respond on request and process response
- Controller acts as a bridge between a request and a response.



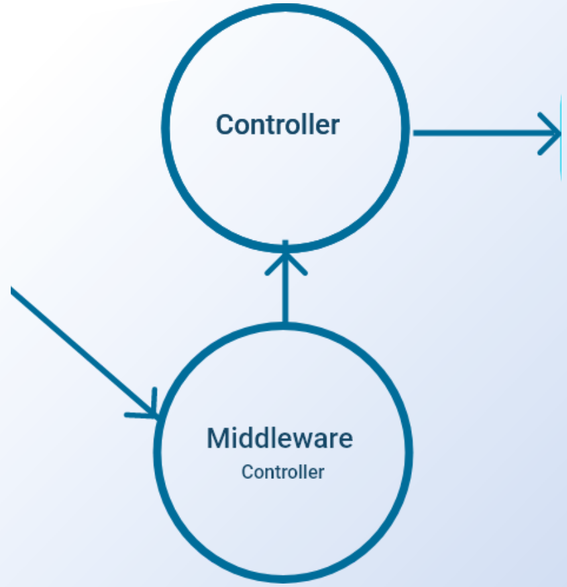
Middleware

- Middleware is a special types of controller executed after request but before in response.
- It is a type of filtering mechanism to ensure API securities and more.
- Middleware acts as a bridge between a request and a response.



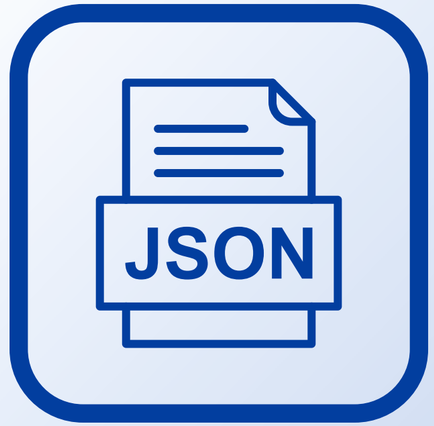
Uses of Middleware

- Use to implement API key, user-agent restriction, CSRF, XSRF security, token based API authentication.
- Use to implement API request rate limit.
- Logging of incoming HTTP requests.
- Redirecting the users based on requests.
- Middleware can inspect a request and decorate it, or reject it, based on what it finds.
- Middleware is most often considered separate from your application logic.
- Middleware gives you enough freedom to create your own security mechanism.



JavaScript Object Notation (JSON)

- JSON is a lightweight data-interchange format that is completely language independent.
- It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data
- The official Internet media type for JSON is application/json.
- It was designed for human-readable data interchange.
- The filename extension is .json.



Uses of JSON

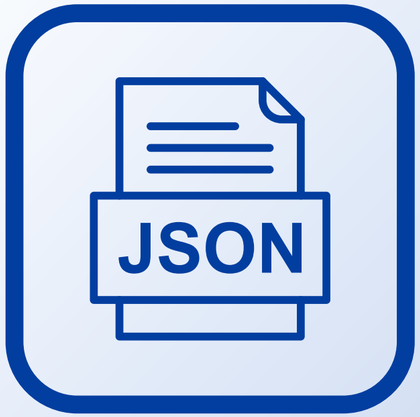
- It is used while writing JavaScript based applications that includes browser extensions and websites.
- JSON format is used for serialising and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications.
- Web services and APIs use JSON format to provide public data.

Characteristics of JSON

- JSON is easy to read and write.
- It is a lightweight text-based interchange format.
- JSON is language independent.

Understanding JSON Structure

```
{                                                    Object Starts
    “Title”: “The Cuckoo’s Calling”
    “Author”: “Robert Galbraith”,
    “Genre”: “classic crime novel”,
    “Detail”: {                                      Object Starts
        “Publisher”: “Little Brown”                  Value string
        “Publication_Year”: 2013,                    Value number
        “ISBN-13”: 9781408704004,
        “Language”: “English”,
        “Pages”: 494
    }                                                Object ends
    “Price”: [                                       Array starts
        {                                            Object Starts
            “type”: “Hardcover”,
            “price”: 16.65,
        }                                            Object ends
        {                                            Object Starts
            “type”: “Kindle Edition”,
            “price”: 7.03,
        }                                            Object ends
    ]                                                Array ends
}                                                    Object ends
```

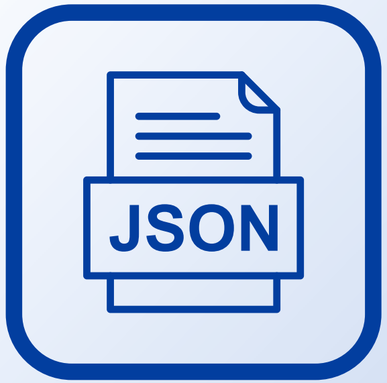


JSON - Data Types

- **Number:** Double- precision floating-point format in JavaScript
- **String:** Double-quoted Unicode with backslash escaping
- **Boolean:** True or False
- **Array:** An ordered sequence of values
- **Value:** It can be a string, a number, true or false, null etc
- **Object:** An unordered collection of key:value pairs
- **Whitespace:** Can be used between any pair of tokens
- **null:** Empty

Bad Special Characters Inside JSON & Solution

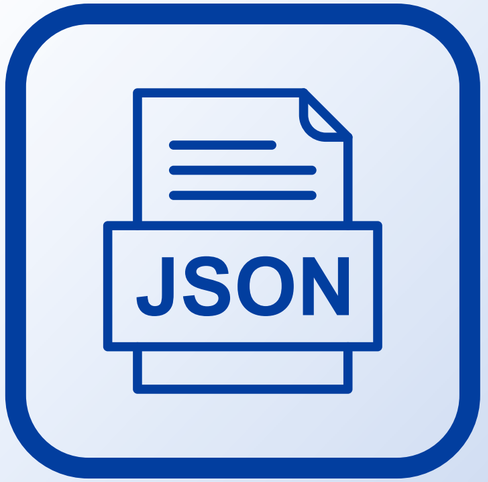
- **Backspace** : Replace with **\b**
- **Form feed :** Replace with **\f**
- **Newline :** Replace with **\n**
- **Carriage return :** Replace with **\r**
- **Tab :** Replace with **\t**
- **Double quote :** Replace with **\"**
- **Backslash :** Replace with ****



Best practices of JSON

- Always enclose the Key : Value pair within double quotes

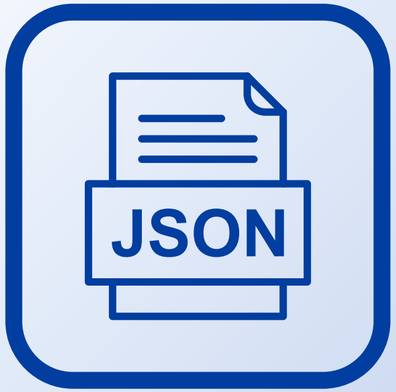
```
{'id': '1','name':File} is not right ✗

{"id": 1,"name":"File"} is okay  ✓

{"id": "1","name":"File"} is the best  ✓
```



Best practices of JSON

- Never use Hyphens in your Key fields

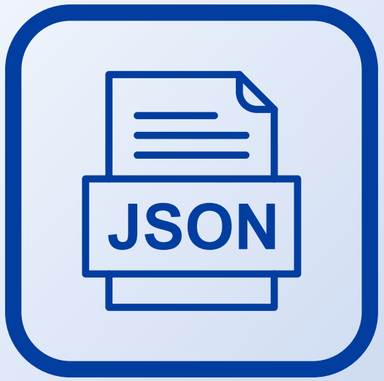
```
{"first-name":"Rachel","last-name":"Green"}  is not right. ✗

{"first_name":"Rachel","last_name":"Green"} is okay ✓

{"firstname":"Rachel","lastname":"Green"} is okay ✓

{"firstName":"Rachel","lastName":"Green"} is the best. ✓
```



Best practices of JSON

- Bad Special Characters And Solution

```
"<h2>About US <br>\r\n</h2>\r\n<p>It has been exactly 3 years since I wrote my first blog series
\r\nentitled “Flavorful Tuscany”, but starting it was definitely not easy. \r\nBack then, I didn’t know
much about blogging, let alone think that one \r\nday it could become <strong>my full-time job</strong>.
Even though I had\r\n many recipes and food-related stories to tell, it never crossed my mind\r\n that I
could be sharing them with the whole world.</p>\r\n<p>I am now a <strong>full-time blogger</strong> and
the curator of the <a data-cke-saved-href=\"https://ckeditor.com/ckeditor-4/#\"
href=\"https://ckeditor.com/ckeditor-4/#\">Simply delicious newsletter</a>, sharing stories about
traveling and cooking, as well as tips on how to run a successful blog.</p>\r\n<p>If you are tempted by
the idea of creating your own blog, please think about the following:</p>\r\n<ul><li>Your story (what do
you want to tell your audience)</li><li>Your audience (who do you write for)</li><li>Your blog name and
design<br>\r\n\t</li></ul>\r\n<p>After you get your head around these 3 essentials, all you have to do
is grab your keyboard and the rest will follow.</p>"
```



Best practices of JSON

- Always create a Root element.

JSON with root element

```
{
"menu": [
  {
    "id": "1",
    "name":"File",
    "value": "F",
    "popup": {
      "menuitem": [
        {"name":"New", "value": "1N", "onclick": "newDoc()"},
        {"name":"Open", "value": "1O", "onclick": "openDoc()"},
        {"name":"Close", "value": "1C", "onclick": "closeDoc()"}
       ]
      }
  },
  {
    "id": "2",
    "name":"Edit",
    "value": "E",
    "popup": {
      "menuitem": [
        {"name":"Undo", "value": "2U", "onclick": "undo()"},
        {"name":"Copy", "value": "2C", "onclick": "copy()"},
        {"name":"Cut", "value": "2T", "onclick": "cut()"}
       ]
      }
  }
  ]
}
```

JSON without root element

```
[
  {
    "id": "1",
    "name":"File",
    "value": "F",
    "popup": {
      "menuitem": [
        {"name":"New", "value": "1N", "onclick": "newDoc()"},
        {"name":"Open", "value": "1O", "onclick": "openDoc()"},
        {"name":"Close", "value": "1C", "onclick": "closeDoc()"}
       ]
      }
  },
  {
    "id": "2",
    "name":"Edit",
    "value": "E",
    "popup": {
      "menuitem": [
        {"name":"Undo", "value": "2U", "onclick": "undo()"},
        {"name":"Copy", "value": "2C", "onclick": "copy()"},
        {"name":"Cut", "value": "2T", "onclick": "cut()"}
       ]
      }


  }
]
```

HTTP Client

HTTP Client Library	Language
Volly	Java
Retrofit	Java
Rest Sharp	C#
Axios	JavaScript
cURL	PHP

Application Level Client: Library used in client side application to generate request and receive response.

Browser Client: Browser is the primary HTTP Client responsible for load the web application.

POSTMAN Http Client

Postman is an HTTP Client application, used to test request-response communication.

Postman is widely used for API testing and generating documentation

- Quickly and easily send REST, SOAP, and GraphQL requests directly within Postman.
- Generate and publish beautiful, machine-readable API documentation.
- Checking performance and response times at scheduled intervals.
- Communicate the expected behavior of an API by simulating endpoints and their responses

Http Request Segments:

HTTP Request is the first step to initiate web request/response communication. Every request is a combination of request header, body and request URL.

Request Area	Standard Data Type
Body	Simple String, JSON, Download, Redirect, XML
Header	Key Pair Value
URL Parameter	String

Request Compare GET vs. POST

Key Points	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be	Never
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions

Request Compare GET vs. POST

Key Points	GET	POST
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL. Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

When use GET() When POST()

- **GET** is used to request something from server with less amount of data to pass.
- When nothing should change on the server because of your action.
- When request only retrieves data from a web server by specifying parameters
- Get method only carries request url & header not request body.

- **POST** should be used when the server state changes due to that action.
- When request needs its body, to pass large amount of data.
- When want to upload documents , images , video from client to server

Http Request Throttling

Throttle Request refers to a process in which a user is allowed to hit the application maximum time in per second or per minute. Throttling is also known as request rate limiting.

- Essential component of Internet security, as DoS attacks can tank a server with unlimited requests.
- Rate limiting also helps make your API scalable by avoid unexpected spikes in traffic, causing severe lag time.

Http Response Segments

Http response is the final step of request-response communication. Every response is a combination of response header, body and cookies.

Response Area	Standard Data Type
Body	Simple String, JSON, Download, Redirect, XML
Header	Key Pair Value
Cookies	Key Pair Value

HTTP Response status messages

Code	Meaning	Description
200	OK	The request is OK (this is the standard response for successful HTTP requests)
201	Created	The request has been fulfilled, and a new resource is created
202	Accepted	The request has been accepted for processing, but the processing has not been completed
203	Non-Authoritative Information	The request has been successfully processed, but is returning information that may be from another source
204	No Content	The request has been successfully processed, but is not returning any content
205	Reset Content	The request has been successfully processed, but is not returning any content, and requires that the requester reset the document view

HTTP Response status messages

Code	Meaning	Description
206	Partial Content	The server is delivering only part of the resource due to a range header sent by the client
400	Bad Request	The request cannot be fulfilled due to bad syntax
401	Unauthorized	The request was a legal request, but the server is refusing to respond to it.
403	Forbidden	The request was a legal request, but the server is refusing to respond to it
404	Not Found	The requested page could not be found but may be available again in the future
405	Method Not Allowed	A request was made of a page using a request method not supported by that page

HTTP Response status messages

Code	Meaning	Description
408	Request Timeout	Request Timeout
500	Internal Server Error	A generic error message, given when no more specific message is suitable
502	Bad Gateway	The server was acting as a gateway or proxy and received an invalid response from the upstream server
503	Service Unavailable	The server is currently unavailable (overloaded or down)

URI Naming Convention

URIs as resources as nouns:

- One of the most recognizable characteristics is the predominant use of nouns in URIs
- URIs should not indicate any kind of functionality.
- Instead, should allow you to manipulate a resource.

Example: `/users/{id}` instead of `/getUser`

URI Naming Convention

Forward slashes for hierarchy:

Forward slashes are conventionally used to show the hierarchy between individual resources and collections

> *Example:* `/users/{id}/address` clearly falls under the `/users/{id}` resource which falls under the `/users` collection.

URI Naming Convention

Punctuation for lists:

When there is no hierarchical relationship (such as in lists), punctuation marks such as the semicolon, or, more frequently, the comma should be used.

Example: `/users/{id1},{id2}` to access multiple user resources

URI Naming Convention

Query parameters where necessary:

In order to sort or filter a collection, a REST API should allow query parameters to be passed in the URI.

Example: `/users?location=USA` to find all users living in the United States

URI Naming Convention

Lowercase letters and dashes:

By convention, resource names should use exclusively lowercase letters. Similarly, dashes (-) are conventionally used in place of underscores (_).

Example: `/users/{id}/pending-orders` instead of `/users/{id}/Pending_Orders`

URI Naming Convention

No file extensions

Leave file extensions (such as .xml) out of your URIs. We're sorry to say it, but they're ugly and add length to URIs. If you need to specify the format of the body, instead use the Content-Type header

Example: `/users/{id}/pending-orders` instead of `/users/{id}/pending-orders.xml`

URI Naming Convention

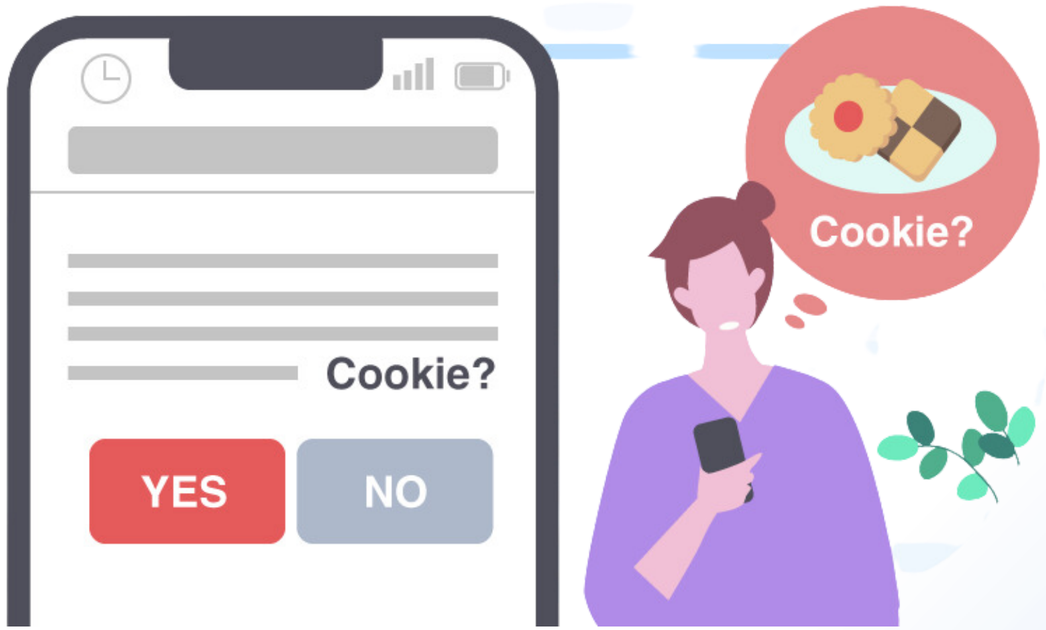
No trailing forward slash

Similarly, in the interests of keeping URIs clean, do not add a trailing forward slash to the end of URIs.

Example: `/users/{id}/pending-orders` instead of `/users/{id}/pending-orders/`

Cookies

- HTTP cookies are small blocks of data created by a web server.
- while a user is browsing a website and placed on the user's web browser.



What Are Cookies Used For

Session management

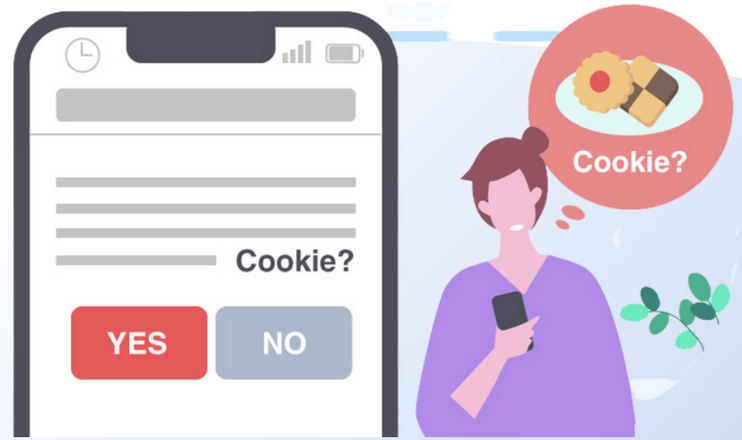
The primary use of the cookie is to manage user logins, shopping cart details, game scores, or anything else the server should remember when the user logs in next time.

Tracking

Tracking involves recording the user activity and analyzing their behaviours to personalize the content for them. It records and analyzes their habits and interests and finds the pages that they visit.

Personalization

Cookies help in personalizing user preferences, themes, and other settings. Most of the time, these settings are synchronized with a central database to personalize the things for the users when the user logs in for the first time.



Best Practices

Response Body

- Avoid providing response status, code, message via response body
- Use JSON best practices for JSON response body.
- For single result, can use String, Boolean directly.
- Provide proper JSON encode-decode before writing JSON Body.
- Follow discussion on JSON described before.

Response Cookies:

- A Restful API may send cookies just like a regular Web Application that serves HTML
- Avoid using response cookies as it is violate stateless principle.
- If required use cookie encryption, decryption and other policies

Best Practices

Response Header

- Provide proper http response status code.
- Provide proper content type, file type if any.
- Provide cache status if any.
- Authentication token should provide via response header.
- Only string data is allowed for response header.
- Provide content length if any.
- Provide response date and time.
- Follow request-response model described before.

Best Practices

Request Body

- Request body should be structured in JSON Array/ Object pattern
- Request body hold multipart/ form-data like images, audio, video etc
- Request body should not hold any auth related information.
- Request body should associated with specific request data model, setter getter can used for this

Request Header

- Request header should carry all security related information, like token, auth etc.
- Only string Key:Pair value is allowed for header .
- Request header should provide user agent information of client application.
- If necessary CSRF/XSRF should provide via header.
- Request header should associated with middleware controller, where necessary

Best Practices

Controller Best Practices

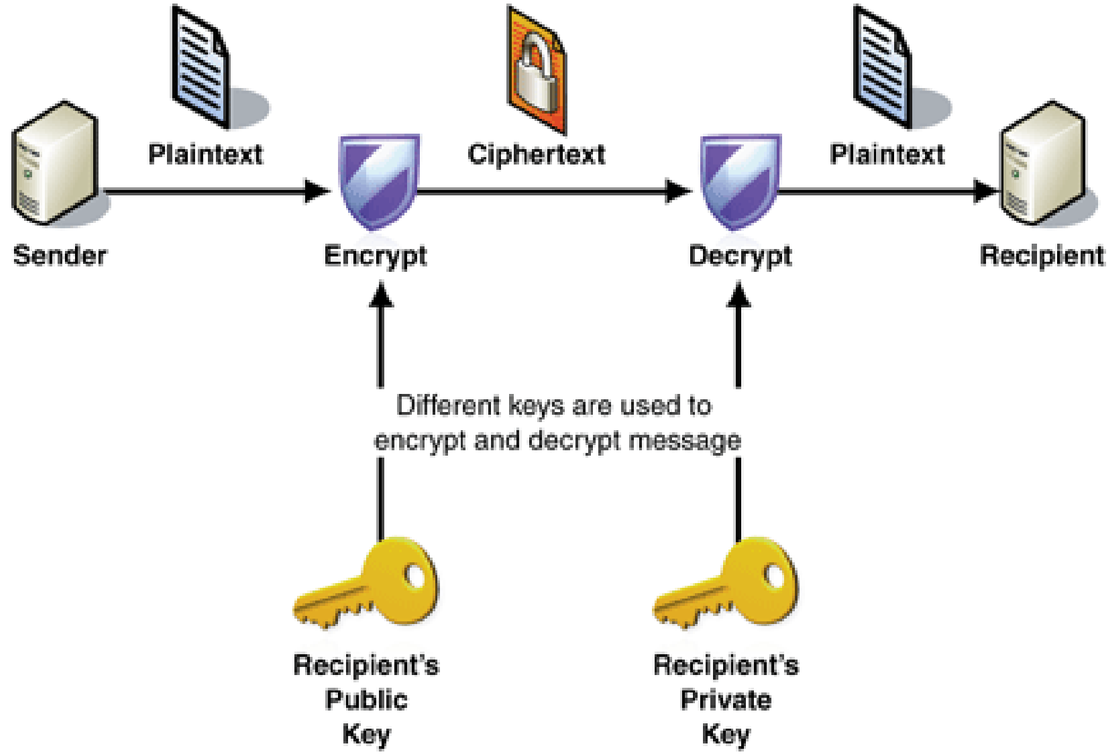
- The controllers should always be as clean as possible. We shouldn’t place any business logic inside it.
- Controllers should be responsible for accepting http request
- Consider API versioning
- Use async/await if at all possible.
- Follow solid principles to manage controller classes.
- Mention which method is responsible for GET() and which for POST().
- Controller should be only responsible for calling model, return response , redirect to action etc.

Best Practices

Middleware Controller Best Practices

- Use to implement API key, user-agent restriction, CSRF, XSRF security, token based API authentication.
- Use to implement API request rate limit.
- Logging of incoming HTTP requests.
- Redirecting the users based on requests.
- Middleware can inspect a request and decorate it, or reject it, based on what it finds.
- Middleware is most often considered separate from your application logic.
- Middleware gives you enough freedom to create your own security mechanism.

Encryption and Decryption



Common Encryption Algorithms

- HS384
- HS512
- RS256
- RS384
- RS512
- ES256
- ES384
- ES512
- PS256
- PS384
- PS512

JSON WEB TOKEN

- Compact and self-contained way for securely transmitting information between parties as a JSON object.
- Information can be verified and trusted because it is digitally signed.

JWT USES:

- **Authorization:** Allowing the user to access routes, services, and resources
- **Information Exchange:** Way of securely transmitting information between parties.

JSON WEB TOKEN STRUCTURE

Header

- Type of the token
- Signing algorithm

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

PAYLOAD

- iss (issuer)
- iat (Issued At)
- exp (expiration time)
- sub (subject)
- aud (audience)

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

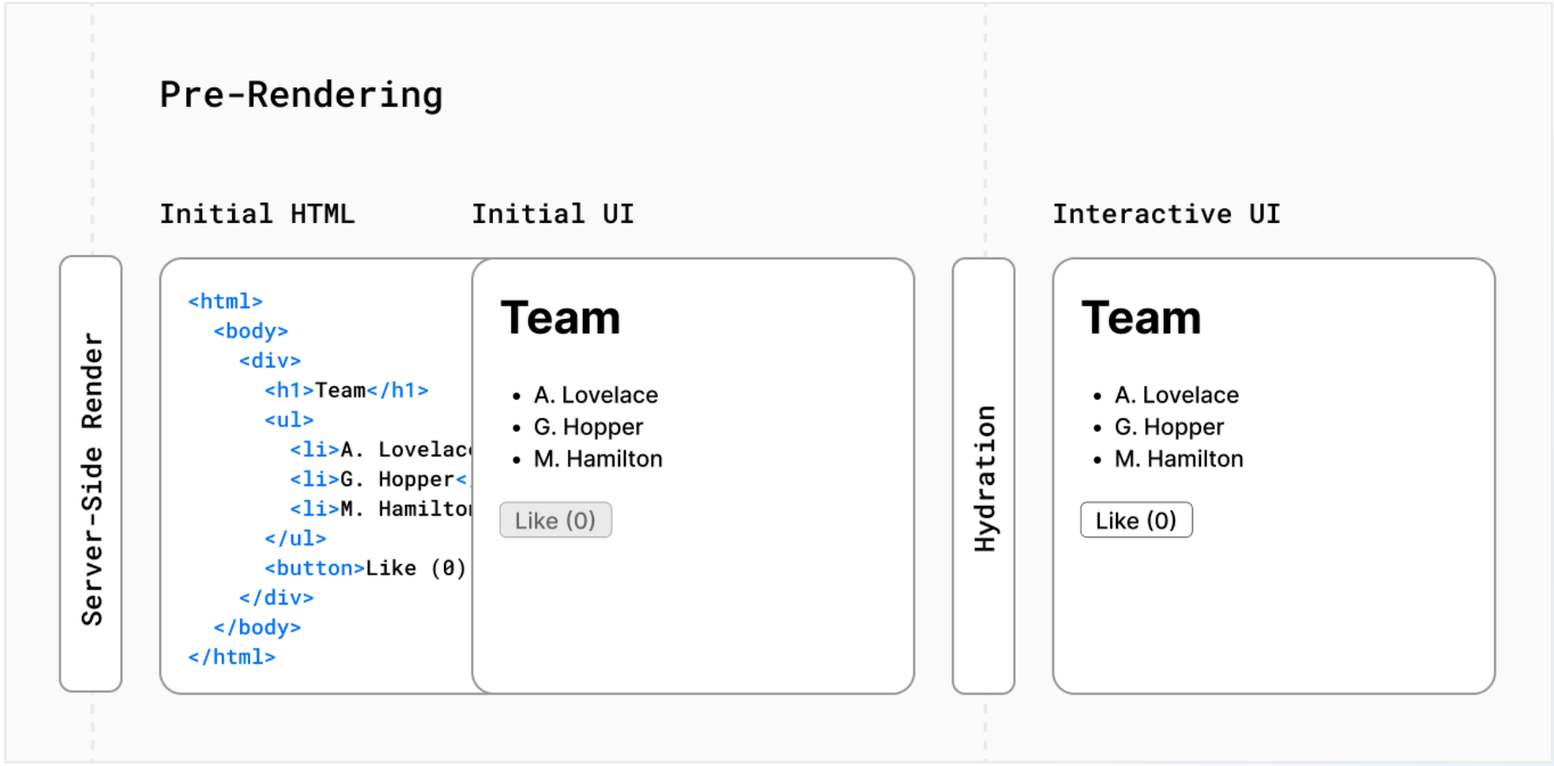
SIGNATURE

- Public - Private Key

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key in SPKI, PKCS #1, X.509 Certificate, or JWK string format.
  ,
  Private Key in PKCS #8, PKCS #1, or JWK string format. The key never leaves your browser.
)
```

Server-Side Rendering

- With server-side rendering, the HTML of the page is generated on a server for each request.
- The generated HTML, JSON data, and JavaScript instructions to make the page interactive are then sent to the client.



BENEFITS OF SSR

Better SEO

Server-side rendering allows search engine bots to crawl and index the content of your website easily, leading to better search engine rankings.

Faster initial loading

Server-side rendering provides faster initial loading of web pages by sending HTML content to the browser before the JavaScript is executed. This helps reduce page load time and improves user experience.

Improved accessibility

Server-side rendering can improve website accessibility by ensuring that content is available to all users, including those with slow or unreliable internet connections, or those using assistive technologies.

Improved security:

Server-side rendering can improve website security by reducing the risk of cross-site scripting (XSS) attacks that are common with client-side rendering.

Better performance on low-powered devices:

Server-side rendering can improve the performance of web pages on low-powered devices such as mobile phones and tablets by reducing the processing requirements on the client-side.

Learn with Rabbil

OPPOSITE OF SSR

Higher server load

Server-side rendering requires more server resources to generate and serve pages, which can increase server load and affect website performance.

Limited interactivity

Server-side rendering does not allow for complex interactive features that require heavy client-side processing, such as real-time updates or animations.

Limited flexibility

Server-side rendering may limit the ability to customize the user interface or provide personalized experiences, as the HTML is generated on the server.

Higher development costs

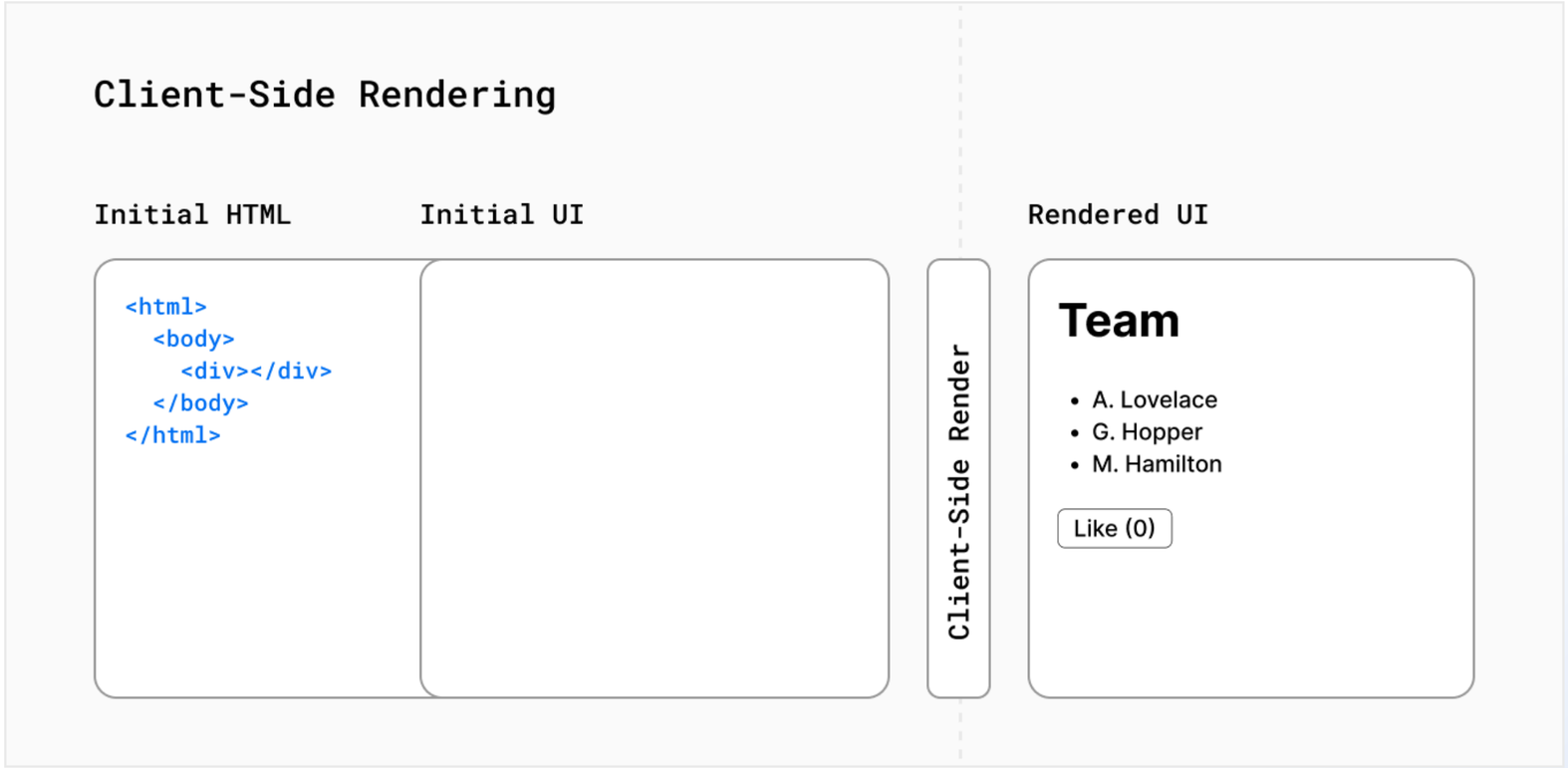
Server-side rendering may require additional development resources and expertise, leading to higher development costs.

Complexity

Implementing server-side rendering can be complex and require specific technologies and configurations, which can be difficult for some developers to manage.

Client-Side Rendering

- With client-side rendering, the HTML of the page is generated inside the web browser.
- This approach allows for faster and more dynamic interactions with the web page.as the browser can update specific parts of the page without having to reload the entire page from the server



Learn with Rabbil

BENEFITS OF CSR

Improved Performance

Client-side rendering reduces server load and bandwidth consumption, initial page load requires a smaller amount of data to be transferred from the server to the client rendering process is performed on the client-side, which reduces the server's workload and frees up resources.

Better User Experience

Client-side rendering provides a more interactive and dynamic user experience. The content can be updated dynamically, and page navigation is faster, as only the necessary data is fetched from the server.

Easier Development

Client-side rendering frameworks, such as React, Angular, and Vue.js, provide developers with tools and libraries that simplify the development process.

Faster Iteration

With client-side rendering, developers can iterate quickly and test changes in real-time without having to wait for server-side rendering. This speeds up the development process and reduces the time it takes to release new features and updates.

OPPOSITE OF CSR

SEO Challenges

While modern search engines are better at indexing client-side rendered pages, there can still be challenges with search engine optimization (SEO). It can be difficult to ensure that the content is fully visible and crawlable by search engines, which can negatively impact search rankings.

Initial Load Time:

The initial load time of a client-side rendered web page can be slower than that of a server-side rendered page because the web browser needs to download the JavaScript code and execute it before displaying the content. This can result in slower page load times, especially on slower devices or slower internet connections.

Browser Compatibility:

Different web browsers may interpret and execute JavaScript differently, which can lead to compatibility issues. This can require additional testing and development work to ensure that the web page functions correctly across all major web browsers.

Security Concerns:

Client-side rendering can increase the risk of security vulnerabilities, such as cross-site scripting (XSS) attacks. This is because the client-side JavaScript code can be modified by malicious actors, which can lead to data theft or other security breaches.

Maintenance Complexity:

Client-side rendering frameworks and libraries can be complex and require a high level of technical expertise to maintain and update. This can make it challenging for smaller development teams to keep up with the latest updates and ensure that the web page remains secure and functional.

Learn with Rabbil

THE OPTMIZED SOLUTION

- Use Server and Client Components.
- We can build a good combination of SSR, CSR using modern JavaScript .
- Use SSR,CSR Where ncesseassary.

Express Rest API Packages

Express JS
REST API DEVELOPMENT

Essential Packages:

Express: The core backbone

Body-parser: This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.

Cookie-parser: Used to parse Cookie header and populate req.cookies with an object keyed by the cookie names.

Multer: This is a node.js middleware for handling multipart/form-data.

Jsonwebtoken: Securely transmitting information between parties as a JSON object

MySQL Driver: To access a MySQL database with Node.js

Mongo dB Driver: To access a Mongo database with Node.js

Express Rest API Packages

Express JS
REST API DEVELOPMENT

Essential Packages:

Dotenv: Dotenv is a zero-dependency module that loads environment variables

Cors: CORS is a node.js package for providing a Connect/Express middleware that can be used to enable CORS with various options.

Express-mongo-sanitize: Sanitizes user-supplied data to prevent MongoDB Operator Injection.

Express-rate-limit: Rate-limiting middleware for Express. Use to limit repeated requests to public APIs and/or endpoints.

Helmet: Helmet helps you secure your Express apps by setting various HTTP headers.

HPP: Express middleware to protect against HTTP Parameter Pollution attacks

Validator: A library of string validators and sanitizers.

Xss-clean: Connect middleware to sanitize user input coming from POST body, GET queries, and url params.

Express Rest API Structure

Express JS
REST API DEVELOPMENT

File Folder Structure

- For Monolithic Application MVC (Model, View, Controller)
- For Rest API MC (Model, Controller)

Express Rest API Structure

Express JS
REST API DEVELOPMENT

File Folder Structure

Index.js : Responsible for connecting the MongoDB and starting the server.

App.js : Configure everything that has to do with Express application.

Config.env: for Enivrement Variables.

Routes -> Routes.js: The goal of the route is to guide the request to the correct handler function which will be in one of the controllers.

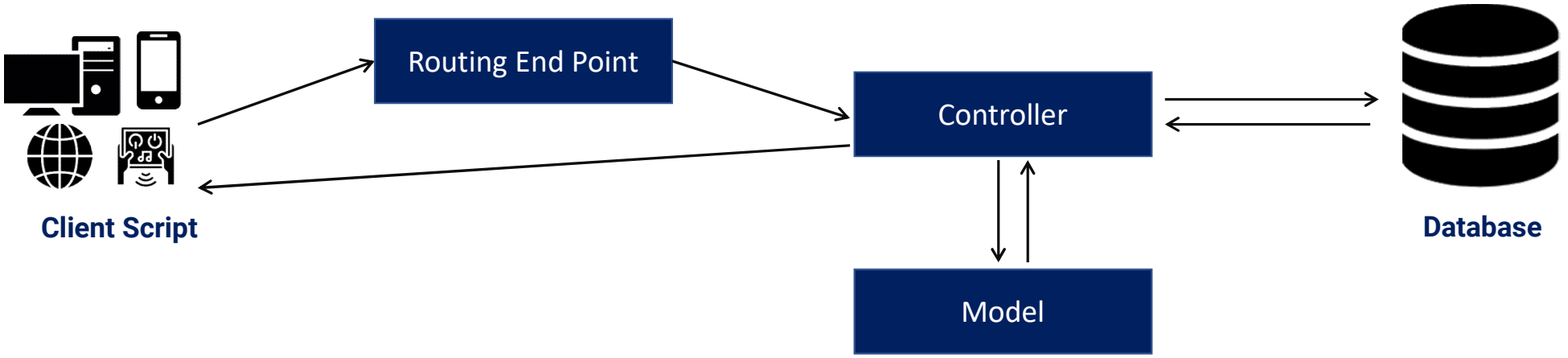
Controllers -> Controller.js: Handle the application request, interact with models and send back the response to the client.

Models -> Model.js: If we use Mongoose this will be schema definition for creating and reading documents from the underlying MongoDB database.

Express Rest API Structure

Express JS
REST API DEVELOPMENT

File Folder Structure:



Create My Express Rest API Project

Express JS
REST API DEVELOPMENT

1. Create a project folder
2. Create package.json
3. Install express
4. Install node packages
5. Create index.js @ root
6. Create app.js @root
7. Create config.env @root
8. Create src directory
9. Create src->controllers folder
10. Create src->models folder
11. Create src->routes folder