Notes

Understanding REST APIs

REST (Representational State Transfer) is an architectural style for designing networked applications. It relies on stateless, client-server communication over HTTP using standard methods and status codes. RESTful APIs are designed around resources, which can be anything from users and products to documents.

Key Concepts:

1. **Resources:** Everything that can be accessed via a RESTful API is considered a "resource." Each resource has a unique identifier (URI).
2. **Representations:** Resources are transferred in some representation like JSON or XML.
3. **Stateless Communication:** Each request from client to server contains all needed information; the server does not store any state about the client session.
4. **HTTP Methods:** REST APIs use standard HTTP methods to perform actions on resources.
5. **HTTP Status Codes:** Servers use HTTP status codes to indicate the outcome of a client's request.

HTTP Methods

HTTP methods define the action you want to perform on a resource. Here are the most common methods:

Method	Description	Idempotent?
`GET`	Retrieves a resource or a list of resources. Should not modify data on the server.	Yes
`POST`	Creates a new resource. The request body contains the data for the new resource.	No
`PUT`	Updates a resource by replacing it with new data. Requires complete new representation in the request body.	Yes

`PATCH`	Updates a resource by partially modifying it. Requires only the modified fields in the request body.	Yes
`DELETE`	Deletes a resource.	Yes

Important Notes on Methods:

- **Idempotence:** An idempotent method produces the same result if called once or multiple times with the same request (e.g., `GET`, `PUT`, `PATCH`, `DELETE`). `POST` is generally not considered idempotent.
- **Safe methods:** safe method should not modify server data (`GET`, `HEAD`, `OPTIONS`).

HTTP Status Codes

Status codes are three-digit numbers the server uses to indicate the outcome of a client's request. They are categorized into five classes:

Status Code Range	Meaning
1xx (Informational)	The request was received, continuing process
2xx (Success)	The request was successfully received, understood, and accepted.
3xx (Redirection)	Further action needs to be taken by the client to fulfill the request.
4xx (Client Error)	The client sent a request with invalid syntax or which could not be fulfilled.
5xx (Server Error)	The server failed to fulfill an apparently valid request. These errors typically indicate problems with the server or backend services.

Common Status Codes:

Code	Category	Description	Use Case
`200`	Success	OK: The request was successful.	A successful `GET` request.
`201`	Success	Created: A new resource has been created.	A successful `POST` request.
`204`	Success	No Content: The request was successful, but there's no content to return.	`DELETE` request when the resource is removed successfully.

`301`	Redirection	Moved Permanently: The resource has moved to a new URL.	Redirect old URL to new one
`302`	Redirection	Found: The resource has been found at a different URL (temporary redirect).	Temporary redirect to different URL
`304`	Redirection	Not Modified: The resource hasn't changed since the last request.	Conditional `GET` request to prevent unnecessary transfer of the resource.
`400`	Client Error	Bad Request: The request was malformed or invalid.	Invalid request body or missing parameters in a request.
`401`	Client Error	Unauthorized: User is not authenticated to access the resource.	Accessing a resource that requires authentication (e.g., an API key).
`403`	Client Error	Forbidden: User is authenticated but does not have permission to access the resource.	User does not have the required permission to access the resource.
`404`	Client Error	Not Found: The resource could not be found at the specified URL.	Invalid URL or resource does not exist.
`405`	Client Error	Method Not Allowed: The HTTP method is not supported for the given resource.	Using a `POST` request on a read-only endpoint.
`409`	Client Error	Conflict: The request could not be completed due to a conflict with the state of the resource.	When trying to delete an entity that has relationship with another entities
`422`	Client Error	Unprocessable Entity: Server understands request but can't process because of semantic errors	Valid syntax, but cannot process given data (e.g., trying to create a user with a duplicated email).
`500`	Server Error	Internal Server Error: Something went wrong on the server.	A generic server-side error.

`501`	Server Error	Not Implemented: The server does not support the functionality required to fulfill the request.	When the server does not support the requested operation.
`503`	Server Error	Service Unavailable: The server is currently unavailable.	Server is overloaded or down for maintenance.

RESTful API Design Best Practices:

- Use nouns to represent resources (e.g., `/users`, `/products`).
- Use plural nouns for collections (e.g., `/users`).
- Use HTTP methods according to their semantics (`GET` for read, `POST` for create, etc.).
- Use status codes appropriately to convey the outcome of the request.
- Keep your APIs consistent and predictable.
- Design stateless APIs that do not rely on session storage on the server.

Interview Questions and Answers:

Q1: What is a REST API?

A: REST (Representational State Transfer) is an architectural style for designing networked applications. It relies on stateless client-server communication using standard HTTP methods and status codes. RESTful APIs are designed around resources (like data), which can be accessed using methods like `GET`, `POST`, `PUT`, `PATCH`, `DELETE`.

Q2: What are the common HTTP methods used in REST APIs, and what do they mean?

A:

- `GET`: Retrieve a resource or a list of resources.
- `POST`: Create a new resource.
- `PUT`: Update a resource by replacing it with new data.
- `PATCH`: Update a resource by partially modifying it.
- `DELETE`: Delete a resource.

Q3: Explain the difference between `PUT` and `PATCH`.

A:

- `PUT`: Used to completely replace a resource. It expects the entire resource representation in the request body.
- `PATCH`: Used to partially modify a resource. It only requires the fields that need to be updated in the request body.

Q4: What are HTTP status codes, and why are they important in REST APIs?

A: HTTP status codes are three-digit numbers the server sends to the client to indicate the outcome of a request. They are important because they allow clients to understand whether a request was successful, failed, or needs more action. They also help in troubleshooting API issues.

Q5: Give some examples of common HTTP status codes and when they are used?

A:

- `200 OK`: The request was successful (e.g., retrieving a resource).
- `201 Created`: A new resource was successfully created (e.g., `POST` request).
- `400 Bad Request`: The request was invalid.
- `401 Unauthorized`: The client needs authentication (API key, credentials).
- `403 Forbidden`: The client is authenticated but does not have permission to access the resource.
- `404 Not Found`: The resource does not exist.
- `500 Internal Server Error`: A server-side error occurred.

Q6: What does it mean for a method to be idempotent? Which HTTP methods are idempotent?

A: An idempotent method produces the same result regardless of how many times it's called with the same request. `GET`, `PUT`, `PATCH`, and `DELETE` are idempotent methods. `POST` is usually not.

Q7: What does it mean for a method to be safe method? which HTTP methods are safe?A: A safe method should not modify any data on the server. `GET`, `HEAD`, and `OPTIONS` methods are safe method.

Q8: If an API operation fails, should the API return error status code?

A: Yes. API should always return proper error status code with a descriptive message. For ex.

- Use `400 Bad Request` for invalid request data.
- Use `404 Not Found` if resource is not available.
- Use `500 Internal Server Error` when a unexpected error happens in server.

Q9: What are some best practices for designing RESTful APIs?

- Use nouns to represent resources and pluralize collections.
- Use HTTP methods according to their semantics.
- Use status codes appropriately to convey the outcome of the request.
- Keep your APIs consistent and predictable.
- Design stateless APIs.

Q10: How do you handle errors in RESTful APIs?A:

- Use proper HTTP status codes to indicate the type of error.
- Include a descriptive error message in the response body (e.g., in JSON format).
- Log errors on the server side for debugging purposes.