

Web Services Notes

Rest

- REST is an acronym name for **Representational State Transfer**
 - standardized way to provide data to other applications.
- it is used for building and communicating with web services.
- best way to transfer data across the applications
- mandates resources on the web are represented in
 - JSON,
 - HTML,
 - or XML.

API

- An API is an acronym for **Application Programming Interface**,
 - an interface that defines the interaction between different software components.
- Web APIs => what exactly request is made to the component.

IMP API METHODS

- GET
 - most common method for get some data from component.
 - returns some data from the API based on
 - the endpoint we hit
 - any parameter we pass.
- POST
 - creates the new records
 - updates the new created record in the database.
- PUT
 - takes the new records at the given URI.
 - If the record exists, update the record.
 - If record is not available, create a new record.
- PATCH
 - It takes one or more fields at the given URI.
 - used to update one or more data fields.
 - If the record exists, update the record.
 - If the record is not available, create a new record.
- DELETE
 - It deletes the records at the given URI.

REST FRAMEWORKS

- Rest frameworks, also known as RESTful frameworks or RESTful APIs,
 - tools and libraries that provide a set of conventions and functionalities to build and implement RESTful web services.
- REST stands for Representational State Transfer,
 - which is an architectural style for designing networked applications. It
 - is commonly used in web development to create APIs that
 - allow communication between client-side applications and server-side systems.

Key features of REST frameworks include:

- HTTP Methods:
 - RESTful APIs use standard HTTP methods like GET, POST, PUT, DELETE, etc., to perform operations on resources (data) exposed by the API.
- Resource-based URLs:
 - Resources are represented by URLs, making it easy for clients to access and manipulate them via HTTP methods.
- Statelessness:
 - RESTful APIs are stateless,
 - meaning that each request from a client to the server must contain all the information needed to understand and process the request.
 - The server does not store any client information between requests.
- Representations:
 - Resources can have multiple representations, such as JSON, XML, HTML, etc., allowing clients to choose the format they prefer.

Hypermedia as the Engine of Application State (HATEOAS): This principle suggests that the API responses should contain hyperlinks to related resources, allowing clients to navigate the API dynamically.

REST frameworks provide a structured way to implement these principles, handling HTTP request and response handling, routing, serialization, authentication, and other common tasks.

- Some popular REST frameworks include:
 - Django REST framework
 - Express.js
 - Spring Boot
 - ASP.NET Core

DJANGO REST FRAMEWORK

- Django Rest Framework (DRF) is a package built on the top of Django
 - create web APIs.
- provides the most extensive features of
 - Django,
 - Object Relational Mapper (ORM),
- which allows the interaction of databases in a Pythonic way.
- Hence the Python object can't be sent over the network, so we need to translate Django models into the other formats like JSON, XML, and vice-versa. This process is known as **serialization**, which the Django REST framework made super easy.

key components of the Django framework:

- Models:
 - handles the data layer.
 - developers to define the data models and relationships between them using Python classes.
 - These models are then translated into database tables, making it easy to work with the database without writing SQL queries directly.
- Views:
 - handle the business logic of the application.
 - receive requests from the user's browser, interact with the models to fetch data from the database, and then render templates to generate the HTML response to be sent back to the user.
- Templates:
 - responsible for generating the HTML presentation of the data.
 - They provide a way to separate the design from the business logic.
 - Django's template engine allows developers to embed Python code within the HTML templates, making it easy to dynamically generate content.

- URL Routing:
 - URL routing system that maps URLs to views.
 - The URLs are defined in a central URL configuration file, which makes it easy to organize and manage the different endpoints of the web application.
- Forms:
 - Django includes a powerful form handling system that simplifies the process of validating user input and processing form submissions.
 - This feature helps developers create and manage HTML forms in a more straightforward manner.
- Admin Interface:
 - automatic admin interface that allows developers to manage the application's data through a web-based interface without having to create a separate admin section manually.
- Middleware:
 - supports middleware, which allows developers to process requests and responses globally before they reach the view or after they leave the view.
 - This is useful for tasks like authentication, caching, and error handling.
- Security:
 - includes various built-in features to help developers protect against common web vulnerabilities, such as cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL injection.
- Internationalization (i18n) and Localization (l10n):
 - Django provides tools to build applications that can be easily translated into multiple languages and localized for different regions.

Diff between rest and other frameworks

The main difference between REST frameworks and other frameworks lies in their focus and purpose. Let's look at the distinctions:

Aspect	REST Frameworks	Other Frameworks
Purpose	Specialized for building RESTful APIs	General-purpose for web app development
Architecture	Follows REST architectural style	May follow MVC or other patterns
Focus	API-centric	Application-centric
HTTP Methods	Relies on standard HTTP methods (GET, POST, PUT, DELETE)	May or may not use standard HTTP methods
Resource-Based URLs	Emphasizes resource-based URLs	URLs may or may not be resource-based
Statelessness	Stateless design	May or may not be stateless
Uniform Interface	Provides a uniform and consistent interface for API interactions	May have varied interfaces for different functionalities

CRUD Operations	Maps CRUD operations to HTTP methods	CRUD operations may not be standardized
Hypermedia (HATEOAS)	Follows HATEOAS principles	May or may not implement HATEOAS
Simplicity	Generally simpler and more opinionated	May be more flexible and customizable
Learning Curve	Lower learning curve	May have a steeper learning curve
Use Cases	Ideal for building APIs to expose data and services	Suitable for building complete web apps
Examples	Django REST framework, Express.js	Ruby on Rails, Laravel, Angular, React
	Flask, Spring Boot, etc.	

Structure:

- refers to the organization and design of the components that make up the web service
- includes how the code, data, and functionalities are organized and divided to create a cohesive and maintainable web service.
- A well-structured web service follows a clear architecture (e.g., RESTful, SOAP),
 - separates concerns effectively
 - follows best practices for code organization.
 - is easier to understand, modify, and scale,
 - crucial for maintaining and evolving a successful web service over time.

Schema:

- a schema refers to the definition of the data format used to exchange information between the client and the server.
- defines the structure and organization of the data that can be sent and received by the web service.
- For example, in a RESTful API, the schema may define the JSON or XML format used for representing resources and data.
- In a SOAP-based web service, the schema may be defined using XML Schema (XSD) to specify the structure of the XML messages.
- The schema ensures that data exchanged between the client and the server is
 - well-defined,
 - consistent,
 - can be parsed correctly by both sides.

Endpoints:

- endpoints are specific URLs that the web service exposes to clients for accessing its functionalities.
- Each endpoint corresponds to a specific operation or resource in the web service.
- For example, in a RESTful API, endpoints might include URLs like "https://api.example.com/products" for accessing a list of products or "https://api.example.com/orders/123" to retrieve details about a specific order.
- Clients (e.g., web browsers, mobile apps) interact with the web service by making HTTP requests to these endpoints.
- Each HTTP method (GET, POST, PUT, DELETE, etc.) typically maps to a specific action on the resource represented by the endpoint.

Network Calls:

- network calls refer to the communication that occurs between the client and the server over a network.
- client needs to interact with a web service, it initiates a network call by making an HTTP request to the appropriate endpoint.
- The HTTP request includes the necessary information for the server to understand the client's intent (e.g., HTTP method, request headers, request body).
- The server processes the request
- executes the appropriate actions
- sends an HTTP response back to the client.
- The HTTP response contains the
 - requested data
 - the result of the operation.
- Network calls are at the core of how web services allow clients to interact with and consume their functionalities over the internet.

Protocols:

- set of rules and conventions
 - define how different components of a web service communicate and exchange data with each other.
- play a crucial role in enabling
 - interoperability
 - Standardization
 - allowing web services to work seamlessly across
 - different platforms, devices, and programming languages
- Key protocols include:
 - HTTP
 - Fundamental for web communication, uses methods like GET, POST, etc.
 - SOAP
 - Uses XML for structured information exchange.

structure - organization of the web service's code and components

schema - data format used for communication

endpoints - URLs that clients use to access specific functionalities

network calls - requests and responses that facilitate communication between the client and the server.

Protocols - sets of rules and conventions

Node express javascript

- fast, minimalistic, and flexible web application framework for Node.js.
- provides a set of robust features and tools to build web applications and APIs with ease.
- Key points about Node.js Express:
 - Web Application Framework
 - Express simplifies the process of building web applications and APIs by providing essential functionalities for handling routes, middleware, and HTTP requests and responses.
 - Lightweight
 - Express is designed to be lightweight and unopinionated, allowing developers the freedom to structure their applications as they see fit.
 - Routing
 - Express enables easy definition of routes to handle different HTTP methods (GET, POST, PUT, DELETE, etc.) and URLs. It makes it effortless to respond to various client requests.
 - Middleware
 - Middleware functions are a critical aspect of Express. They can intercept and modify HTTP requests and responses, enabling tasks like authentication, logging, error handling, and more.
 - Template Engines
 - While not included by default, Express allows you to integrate template engines like EJS or Pug to render dynamic HTML pages on the server-side.
 - Robust Ecosyste
 - Express has a massive community and a vast ecosystem of third-party middleware, extensions, and plugins, providing additional features and functionalities to enhance development productivity.

request.js

```
// Importing the express module
const express = require('express');
const bodyParser = require('body-parser');
const url = require('url');
const querystring = require('querystring');

let app = express();
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
```



```
// Sending the response for '/' path
app.get('/', (req,res)=>{
    let page = req.query.name;
    let limit = req.query.age;
    let data = {'name':page,"age":limit};
    // Sending the data json text
    res.json(data);
})

// Setting up the server at port 3000
app.listen(4000 , ()=>{
    console.log("server running");
});
```

Server.js

```
// Importing the express module
const express = require('express');
const app = express();

// Initializing the data with the following json
const data = {
    portal: "TutorialsPoint",
    tagLine: "SIMPLY LEARNING",
    location: "Hyderabad"
}

// Sending the response for '/' path
app.get('/', (req,res)=>{

    // Sending the data json text
    res.json(data);
})

// Setting up the server at port 3000
app.listen(4000 , ()=>{
    console.log("server running");
});
```

https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm

<https://www.tutorialspoint.com/express-js-express-json-function>

GraphQL

- data query language and runtime
- developed by Facebook.
- provides a powerful and flexible approach to data fetching in web applications.
- In traditional RESTful APIs,
 - the server defines endpoints for specific resources,
 - clients can fetch data by making HTTP requests to these endpoints.
 - this can lead to overfetching, where clients receive more data than they need, or underfetching, where they need to make multiple requests to gather all the required data.
- With GraphQL, clients can specify exactly what data they need in their queries.
 - The client defines the structure of the response,
 - indicating the specific fields and relationships it wants.
 - allows clients to request a customized set of data
 - eliminating the problems of overfetching and underfetching
 - also reduces the number of requests required to fetch all the necessary data, as the server can respond with a single GraphQL query containing all the requested information.
- GraphQL can also be used for updating data on the server using mutations.
- Mutations allow clients to specify the changes they want to make to the data, and the server processes these mutations and returns a response.
- GraphQL has a strong typing system, which means that the data types of all the fields are explicitly defined.
 - This ensures that the responses are predictable and well-structured, making it easier for clients to understand and work with the data.

GraphQL is a query language and runtime for APIs that enables more efficient and flexible communication between clients (usually frontend applications) and servers (backend services). It was developed by Facebook and has gained popularity due to its ability to address many of the limitations of traditional REST APIs.

Here's an explanation of the key concepts in GraphQL: structure, schema, endpoints, and network calls.

Structure: GraphQL structures data in terms of types and fields. Each field on a type can be queried or mutated (for updates) directly by clients. The data structure is hierarchical and mirrors the way clients need the data. This contrasts with REST APIs where endpoints often return fixed data structures that might over-fetch or under-fetch data for specific use cases.

Schema: The schema is the contract between the client and the server, defining how data can be queried, what types of data can be retrieved, and how it can be modified. It is a crucial aspect of GraphQL. The schema is defined using the GraphQL Schema Definition Language (SDL), which outlines the types available, their fields, relationships, and the operations that can be performed.

Endpoints: Unlike REST APIs which often have multiple endpoints (URLs) for different resources, GraphQL typically has a single endpoint (URL) for all interactions. This simplifies the API surface and allows clients to request exactly the data they need in a single request, reducing over-fetching of data.

Network Calls: In GraphQL, clients make queries or mutations to the GraphQL server over HTTP. The client specifies the fields it needs, and the server responds with exactly those fields, avoiding over-fetching. The client constructs a query that resembles the structure of the data it wants. Network calls can be made using standard HTTP methods like POST or GET. The server processes the query, validates it against the schema, and returns the requested data in the shape requested.

To sum up the typical flow:

The client sends a GraphQL query to the server over HTTP.

The server parses and validates the query against the defined schema.

The server resolves the query, fetching the necessary data from the backend.

The server responds to the client with the requested data in the same shape as the query.

In addition to queries for fetching data, GraphQL also supports mutations for making modifications to the data on the server. This includes creating, updating, and deleting data.

Overall, GraphQL offers a more efficient and flexible approach to building APIs compared to traditional REST APIs. It allows clients to request just the data they need, avoids multiple network requests, and provides a clear and predictable structure for API interactions.

GraphQL query mutation

GraphQL queries and mutations are two essential components of the GraphQL language, allowing clients to request and modify data on a server.

GraphQL Query:

- A GraphQL query is used to request data from the server.
- Clients can specify the exact data they need in the query,
- and the server responds with the requested data in the same shape as the query.
- The basic structure of a GraphQL query looks like this:

```
query {  
  Field1  
  Field2  
  ...  
}
```

- In a query, clients can request specific fields and nested fields to retrieve the data they require. For example:

```
query {  
  user(id: "123") {  
    Name  
    email  
    posts {  
      Title  
      content  
    }  
  }  
}
```

In this example, the client is requesting the name and email fields of a user with ID "123" and also fetching the title and content fields of the user's posts.

GraphQL Mutation:

- A GraphQL mutation is used to modify data on the server.
- it allows clients to specify what changes they want to make,
 - Creating
 - Updating
 - deleting data.

- Mutations are similar in structure to queries but are defined using the mutation keyword. The basic structure of a GraphQL mutation looks like this:

```
mutation {  
  createEntity(input: { field1: value1, field2: value2, ... }) {  
    field  
    Field2  
    ...  
  }  
}
```

In a mutation, clients can specify the input data required to perform the operation. For example:


```
mutation {  
  createPost(input: { title: "New Post", content: "This is a new post." }) {  
    id  
    title  
    content  
  }  
}
```

In this example, the client is creating a new post by providing the title and content, and the server responds with the id, title, and content of the newly created post.

GraphQL queries and mutations provide a flexible and efficient way for clients to request and modify data on the server, making it a powerful language for building APIs that cater to specific data needs of modern web applications.

Diff between rest and graphql

Aspect	REST	GraphQL
Architectural Style	REST (Representational State Transfer)	GraphQL (Query Language for APIs)
Communication Protocol	Uses standard HTTP methods (GET, POST, PUT, DELETE, etc.)	Uses HTTP POST method with a single endpoint
Endpoint Structure	Typically uses multiple endpoints for different resources	Uses a single flexible endpoint for all queries
Data Fetching	Fetches fixed data structures (predefined by the server)	Allows clients to request specific data structures as needed
Overfetching/Underfetching	Can suffer from overfetching (getting more data than needed) or underfetching (not getting enough data in a single request)	Solves overfetching/underfetching issues by allowing clients to request precisely the data they need

 Reg

Response Format	Typically returns JSON or XML	Returns JSON by default, but the client specifies the format needed
Versioning	May use URL versioning or custom headers	Versioning is not necessary as clients specify the needed fields
Caching	Uses standard HTTP caching mechanisms	Requires custom caching strategies
Server Complexity	The server determines the structure of the response data	The client specifies the structure of the response data
Learning Curve	Generally well-known and easy to understand	May have a steeper learning curve due to the unique query language
Flexibility	May lack flexibility in data fetching	Highly flexible, allowing clients to define data requirements
Ecosystem	Wide range of tools and libraries available	Growing ecosystem with fewer tools and libraries compared to REST

REST Framework:

A REST framework is designed to build APIs following the principles of the REST architectural style.

It simplifies the creation of RESTful APIs using standard HTTP methods and resource-based URLs.

Popular REST frameworks include Django REST framework, Express.js, and Flask.

Diff between REST and Other Frameworks:

REST frameworks are API-centric, while other frameworks focus on building complete web applications.

REST frameworks adhere to REST principles, while other frameworks may use different architectural patterns.

RESTful APIs have statelessness and a uniform interface, whereas other frameworks may not follow these constraints.

GraphQL Query Mutation:

GraphQL is a query language for APIs, allowing clients to request precise data from the server.

Queries are used to fetch data, and clients define the structure of the response they need.

Mutations are used to modify data, allowing clients to specify changes they want to make on the server.

Structure:

Structure refers to the organization and arrangement of code, data, and components in an application.

A well-structured application is easier to understand, maintain, and scale.

Schema:

In the context of databases, a schema defines the structure and organization of data within the database.

It specifies tables, fields, data types, and constraints to ensure consistent data storage and retrieval.

Endpoints:

Endpoints are specific URLs in a web API used by clients to access different functionalities or resources.

Each endpoint corresponds to a specific operation or action in the API.

Network Calls:

Network calls refer to data exchange over a network, usually using protocols like HTTP. In web services, clients initiate network calls to interact with the server and receive responses.

Node Express JavaScript:

Node.js Express is a fast, flexible, and minimalist web application framework for Node.js.

It simplifies building web apps by handling routes, middleware, and HTTP requests and responses.

Diff between REST and GraphQL:

REST follows a fixed data structure, while GraphQL allows clients to request custom data structures.

REST uses multiple endpoints, while GraphQL uses a single flexible endpoint for all queries.

GraphQL reduces overfetching and underfetching issues seen in RESTful APIs.

Protocols:

Protocols are sets of rules and conventions defining how components in web services communicate.

Key protocols include HTTP (for RESTful APIs), SOAP, and GraphQL (for query language).

GraphQL

GraphQL is a query language for APIs, developed by Facebook in 2012.

It allows clients to request exactly the data they need from the server, reducing overfetching and underfetching issues.

Clients define the structure of the response they want, improving the efficiency of data retrieval.

GraphQL uses a single flexible endpoint, which accepts all queries and mutations.

It has a strongly typed schema, making the API self-documented and improving communication between frontend and backend teams.

GraphQL supports real-time subscriptions, enabling real-time updates to data.

Advantages include reduced data transfer, efficient batching of requests, versionless APIs, and a rich ecosystem of tools and libraries.