Date: 02/03-06-2021

Why do we need an API?

Api stands for Application Programming Interface. APIs are needed to bring applications together in order to perform a designed function built around sharing data and executing pre-defined processes. They work as the middleman, allowing developers to build new programmatic interactions between the various applications people and businesses use on a daily basis.

Rest API: REpresentational State Transfer

It is an architectural style as well as an approach for communication purposes that is often used in various web services development. It uses less bandwidth so it is often referred to as “language of the Internet”. In the rest api, when the client sends the request server responds with the state that’s why it is called state transfer API.

REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses less bandwidth, simple and flexible making it more suitable for internet usage. It’s used to fetch or give some information from web services. All communication done via REST API used only HTTP requests.

Principles of REST API

- Stateless
- Client Server
- Uniform Interface
- Cacheable
- Layered System
- Code on Demand

Methods/Working in REST API

A request is sent from client to server in the form of a web URL as HTTP GET or POST or PUT or DELETE. The response from the server can be anything like HTML, XML, IMage or JSON. But now JSON is the most popular format being used in web services.

- **Create:** POST method
- **Read:** GET method
- **Update:** PUT method
- **Delete:** DELETE method
- **GET:** The HTTP GET method is used to **read** a representation of a resource. In the safe path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).
- **POST:** The POST verb is most-often utilized to **create** new resources. In particular, it's used to create subordinate resources. That is, subordinate to some other resource. On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.
- **PUT:** It is used for **updating** the capabilities. However, PUT can also be used to **create** a resource in the case where the resource ID is chosen by the client instead of by the server. In other words, if the PUT is to a URI that contains the value of a non-existent resource ID. On successful update, return 200 (or 204 if not returning any content in the body) from a PUT. If using PUT for create, return HTTP status 201 on successful creation. PUT is not a safe operation but it's idempotent.
- **PATCH:** It is used for **modifying** capabilities. The PATCH request only needs to contain the changes to the resource, not the complete resource. This resembles PUT, but the body contains a set of instructions describing how a resource currently residing on the server should be modified to produce a new version. This means that the PATCH body should not just be a modified part of the resource, but in some kind of patch language like JSON Patch or XML Patch. PATCH is neither safe nor idempotent.

- **DELETE:** It is used to **delete** a resource identified by a URI. On successful deletion, return HTTP status 200 (OK) along with a response body.

Different Ways to Implement REST API in frontend

(1) Axios: Axios is a popular, promise-based HTTP client that sports an easy-to-use API and can be used in both the browser and Node.js. Making HTTP requests to fetch or save data is one of the most common tasks a client-side JavaScript application will need to do.

-This is how Http requests are made in axios.

```
axios({
        method: 'post',
        url : '/login',
        data: {
                user: 'kruti',
                lastname: 'panchal'
        }
});
```

(2) Fetch: The Fetch API is a simple interface for fetching resources. Fetch makes it easier to make web requests and handle responses than with the older XMLHttpRequest, which often requires additional logic.

-The fetch() method takes the path to a resource as input. The method returns a promise that resolves to the Response of that request.

Example:

```
fetch('examples/example.json')
.then(function(response) {
  // Do stuff with the response
})
.catch(function(error) {
```

```
  console.log('Looks like there was a problem: \n', error);
});
```

GraphQL API

GraphQL is a query language and server-side runtime for application programming interfaces (APIs) that prioritizes giving clients exactly the data they request and no more.

GraphQL is designed to make APIs fast, flexible, and developer-friendly. It can even be deployed within an integrated development environment (IDE) known as GraphiQL. As an alternative to REST, GraphQL lets developers construct requests that pull data from multiple data sources in a single API call.

Example of GraphQL query:

```
(i)  {
        city {
                state
        }
    }
```

It will return:

```
{
        "city": {
                "State" :  "Gujarat"
        }
 }
```

Different Ways to Implement GraphQL API in frontend

Apollo Client is a comprehensive state management library for JavaScript that enables you to manage both local and remote data with GraphQL. Use it to fetch, cache, and modify application data, all while automatically updating your UI.

(1)Apollo Client

With Apollo's declarative approach to data fetching, all of the logic for retrieving your data, tracking loading and error states, and updating your UI is encapsulated by the useQuery Hook. This encapsulation makes integrating query results into your presentational components a breeze!

-We wrap our components in ApllоProvider to use apollo client api.

Read and Write Operation in Apollo Client:

-Queries: To fetch the data we use the useQuery() hook and we pass the GraphQL query string that needs to be executed in it. useQuery() returns the object from Apollo Client that contains loading, error and data properties to render in the UI.

- To create a Query in GraphQL:

```
import {gql, useQuery} from '@apollo/client';
const GET_USERS = ggql`
        query GetUsers {
                users{
                        Id
                        Name
                }
        }
`;
```

- To get the data in component,

  const {loading, error, data} = useQuery(GET_USERS);

(2) UrQl

- UrQl library can be used to connect our app with GraphQl APIs.
- This library has almost similar functionality and syntax as Apollo but is a little bit smaller in size and requires less setup code. It gives caching capabilities if we choose, but it doesn't have an integrated state management library like apollo.
- **To use UrQl, we install:** npm install UrQl graphql
- Just like Apollo, It uses the useQuery() hook from the urlQl package, but we can skip using gql function as in UrQl we can just use a template literal to write the query.

If we need to manage small task or just have to make one call for graphQL, then we can use the following libraries to make an api call for the GraphQL

(I) GraphQl Request: it uses QueryClient, QueryClientProvider from 'react-query'. It also supports useQuery hook and gql function to write Query.

(II) Axios: axios can be used in useQuery hook to fetch the data from the graphQL.

(III) Fetch: fetch can also be used in useQuery hook to fetch the data from the graphQL.