1. API Integration in React

1.1 Introduction to API: what do we mean by API?

- **API (Application Program Interface)**: In general terms, API is a software/framework that allows your application (for example, your www.apple.com website) to communicate and exchange data with other applications (for example, with YouTube).
    - **Interface**: Interface is the Keyword which in pure English means a point where two systems meet and interact. An API is a point where an application interacts with another application. In short, API is an interface with a set of functions/methods/code that developers use to access data from another application and integrate it into their application.
- **API explained in simple example**: Assume you are taken back to the year 1650 when telephone was not invented. Let us say you were in your house and your mom was in her house few blocks away from yours. Assume you wanted to ask your mom for her "*doro wot*" recipe, but you were too lazy to run down the road yourself. So, you sent your daughter with a message request for your mom. Your daughter ran down there and brought back the recipe back to you. In this example, your daughter is the API, she sends your recipe request to your mom and brings you back the response recipe from your mom while you and your mom stayed where you were.
- **History of APIs**: APIs have been around almost as long as computing. For instance, before the Internet, APIs existed, but in local networks form (covering limited area).
    - **Modern APIs**: However, the history of modern APIs (web-based APIs), began taking shape in early 2000s when Salesforce launched its web API followed by Amazon and eBay. These companies took advantage of the new medium, the internet, to make products and services available to customers via a single website and ensure that partners and third-party resellers could extend the reach of their websites. As internet (websites) continued to be the default medium of doing business, business companies that collect data from their customers started to be overwhelmed by their data to the extent that they became unable to read/share customer's data or even tell who their customer was or what they want. Because of that, companies started to hire third-party companies called Customer Relationship Management (CRM) platforms that store their data in the cloud.

- **Customer Relationship Management (CRM) platforms**: In addition to storing a company's data in the cloud, the CRM platforms use software to analyze a company's data to improve the company's customer service, to assist in customer retention and to ultimately to drive the company's sales growth. Examples of renowned CRM companies/software are Salesforces, Zoho CRM and Bitrix24

- **Problem APIs were created to solve**: As companies increased their business online, they started to require their sales and human resource employees to make "data-driven" decisions. Because more and more of company data was being stored "in the cloud" by Customer Relationship Management (CRM) systems like Salesforce, data that could easily be used by any employee of a company became very hard to access from these third-party CRM systems. We all know that sales and HR employees do not have a higher level of software literacy to query their company's data from these third-party CRM platforms. Due to this, many companies were forced to pay for other web service like Tableau to fetch their own data from their CRM in order to use the data to make "data driven" decisions for their business. The fact that companies had to pay both for a CRM platform (to store their data) and to pay for other companies (to fetch their data from CRM platforms) made these companies to force their CRM platforms to simplify their complex way of transferring data. **CONCLUSION**: Therefore, web APIs were created to simplify the way organizations fetch their own data from their third-party CRM platforms so that these organizations use their own data to make business related decisions.

- **How and why are APIs used by software developers?** Assume you are building a website that sells different products and you want to include Amazon's products in your website. One way to get the list of all of Amazon's products is by contacting Amazon and requesting for a spreadsheet of all their products. But this means that you must find a way to import that spreadsheet into your application. Even if you find a way to import the spreadsheet data, that data will become outdated very quickly, for instance, the price might change, or the item might no longer be available. It will be better if Amazon provides you with a way to query their application/website to get the data (with its update) you want to use in your websites. **Below, you will see why APIs are chosen way of sharing data from one application to another**:

- **Integration**: API integration is a process of connecting two or more applications through their APIs. Meaning, API integration is the way by which a developer can include data from another application/website in their website. For instance, connecting your www.apple.com application/website with YouTube application to show Apple's videos in your website.

- **Efficiency**: Rather than developers spending excessive time coding their own interface, companies can take advantage of the easy integration provided by APIs. This allows development teams to focus their efforts on other priorities.

  - **Security**: APIs supplement an extra layer of protection between your data and server, for example by integrating a two-factor authentication or by creating a password to log in. However, developers can add more security to APIs by using transport layer security (TLS) encryption (that keeps the internet connection private and checks if the data sent between two applications is encrypted). In short, with a little bit of knowledge, companies can easily protect their APIs from attacks.

- **Difference between an API and a remote server:** Do not be confused, an API is not a database or a server. The main difference between a remote server (Google's server) and an API (example Google Maps API) is the format of the request and response involved.

  - **Remote server (revision)**: It is computer that is remotely located with a web server software and database software installed in it to handle remote requests sent by the users of a browser/website.

    - To render a picture request (from google.com), your browser expects a response in HTML format

    - When you make a request from google.com, it is your browser that is the client

  - **API**: It is the part of the server that works as a middleman to communicate directly with the server of another application.

    - When you make a request/call to Google Maps API, it would just return the data with JSON or XML format.

        - When you (your website's server) make a request/call to Google Maps API's server, it is your website's server that is the client.

- **Why do organizations provide their APIs to the public for free (open API)?** Investing in APIs unlocks digital transformation and increases a company's competitive position.

    - **Brand building**: When a company provides its API for the public for free, it will allow them to reach more people than they could ever do on their own.
    - **Advertising money**: Some API consumers integrate advertising in their apps, providing revenue for the API provider. In turn, the API provider shares some of that advertising revenue with API consumers.

- **Why companies provide APIs but never the direct access to their database?** If you are an owner of an application, and you are willing to open some of the data in your database to employees (like developers) or the public (like external developers), you do not want to provide the public with a direct access to your database. What you prefer to do is, to create an interface (API) that returns some pre-selected list of things as per the developer's request. **Note**: It would be the application owner's responsibility to write the interface code (API) that returns the requested data. So, why not provide direct access to your database?

    - **Security**: The data in a company's database contains various facts about the company, the company's customers and its employees that should be kept away from public.

        - **Example**: If a user has unrestricted SELECT on a table, they can select any record on that table, even those not belonging to them. A salary table would be a bad one. If any user has DELETE or UPDATE, they may forget the WHERE clause, and there goes your table.

        - **Note**: However, some group of employees, like data base administrators (DBAs), need direct access in order to do their job.

- **To provide only relevant data for API consumers**: People accessing the data (through API) should only have access to data that is relevant to their application or relevant for role (if data is to be used by an employee of the API provider).
- **To allow people with no SQL/programming knowledge simply access data**: To directly retrieve data from a database, the user must be able to write code to query from the database (example: SQL query). Very few employees possess such skills and would not be able to accurately retrieve the data that they need.

- **Types of API (Web API, Program API and Local API)**: We said that an API is a set of definitions and protocols used to integrate data from one application into another.
  - **First party API vs third party API**: To choose between first and third party APIs, developers need to look at what the API will do for the organization's customers. If an AAPI is going to be very specific for a client's need, then a first party API might be a better option. However, if the API's purpose is general, developers should save money by using third party APIs.
    - **First party API**: A First Party API is an API that was made in-house by the internal developers and is not requested by any third parties. Developing first party APIs is needed when there is a need to monitor the entire API lifecycle or when unique client cases or when there is a security need. However, developing and implementing an API in-house is expensive, takes time/ resources and requires constant upkeep by the API developers whenever your code or project changes.
    - **Third party API**: Third party APIs are APIs developed by a third party organization and not developed by in-house by the internal developers. A third-party API integration happens when a business uses a third-party's API to power its web service. For the most part, third party APIs are reliable in providing you with all the data you might need to run your business, without the hassle of developing it yourself. However, these APIs will have their own rules for use and might require extra effort to integrate them with your application. However, that is not a huge problem to overcome.

        - For instance, if your business uses/integrates Google's open weather API for your business' website.
    - **REST/RESTful API vs Web API vs SOAP API**: See below for more explanation.

- **REST/RESTful API vs Web API vs SOAP API**: APIs are tools we use to transfer data from one app to another and this transfer requires clear protocols and architectures. API protocols/architectures are the rules that govern an API's operation. Before explaining the rules, we should follow to create RESTful APIs, let us first understand the 2 key terms related to REST API client and resource:
    - **Client**: The client is the person or software who uses the API. For example, you, as a developer, you can call Facebook's API to read and write data from Facebook by reading/creating a new post. The client can also be a software, like, the web browser. When you browse Facebook's website, your browser is the client who calls Facebook's API and uses the returned data to render information on the browser.
    - **Resource**: By resource we mean any data (object) provided by an API. For example, Google's map API provides resources like city names, zip codes or other information.
    - **Web API**: Web API as the name suggests, is an API over the web/internet which can be accessed using HTTP protocol. A Web API can be developed using various technologies like Java and ASP.NET. These APIs can be RESTful or not. Most HTTP APIs we write are not RESTful. This framework implements HTTP protocol specification and hence you hear terms like URIs, request/response headers, caching, versioning, various content types(formats).

- **REST**: REST stands for **RE**presentational **S**tate **T**ransfer. REST is architectural style, or it is the rules and constraints one must follow when building APIs (or when writing code to request data from a server). In short, it is the rule one must follow when writing the code (API) to request data from server of an application/website.

- **REST/RESTful API**: A REST API is an API that is backed by the architectural style of REST. In A RESTful website/application, it is the representation of the state of the resource that is transferred to the requesting client mainly via HTTP. The representation of the resources can be delivered via HTTP in one of these formats: JSON (JavaScript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular file format to use because, despite its name, it is as readable by both humans and machines. Therefore, REST API is basically like using a website from your browser. In both website browsing and REST API calling, you make a call from a client/browser to a server of another application, and you will get data back as a response with HTTP protocol. However, in API call, instead of clicking buttons or filling out forms from the browser, you write code to explicitly request data from a server. In addition, unlike requesting a website from your browser, when you make an API request/call, you will receive the file in JSON/XML format, not in HTML format. **Example**: Let us request for GitHub's home page by typing the https://github.com/ URL into our browser. Here, you will get the response in HTML format. Now, let us request for GitHub's API by typing the https://api.github.com/ URL into the browser. Guess what, we just made an API request in our browser to GItHub's API. Notice here, the response is a JSON format that has resources like the URLs for the current GitHub user, URL for their followers, etc.

- **Criteria to make an API a RESTful API**

    - **Requests must be managed through HTTP**: The client-server architecture has to be made up of clients, servers, and resources, with requests managed through HTTP

    - **Stateless client-server communication**: We said that API is basically how one application (Ex. website) transfers data to another application.

For an API to be RESTful, the communication between the communicating applications has to be stateless. Meaning, neither the requesting application nor the state/data providing application store data about each other whenever there is an API call from one to another.

- **Data must be transferred in a standard form**:
    - Resources requested must be identifiable and separate from the representations sent to the client
    - Representation of the resources sent to the client must be self-descriptive with enough information to describe how the client should process it.
    - The representation of resources sent to the client must be manipulatable by the receiving client because the representation contains enough information about the resource to do so.

- **CRUD and HTTP in REST APIs:** Both REST and CRUD describe manipulating data on a database. CRUD commands often play a role in REST APIs, where they map (not perfectly) to the HTTP methods GET, POST, DELETE, PUT, and PATCH.
    - **Most REST APIs use HTTP protocol to transfer data**: RESTful API calls involve HTTP methods like GET (reads the representation of the resource from a record in the database), POST (creates a new record in the database), PUT (changes a record's information in the database) and DELETE (removes a record from the database).
    - **CRUD**: CRUD is an acronym for the four fundamental database commands a user can perform on a set of data. These are CREATE (to create a new record in the database), READ (to read information from a record in the database), UPDATE (to change a record's information in the database) and DELETE (to remove a record from the database).
    - **CRUD commands correlate to the HTTP methods in REST API**: Many programming languages (like SQL) and protocols (like HTTP) have their own equivalent of CRUD, often with slight variations in what they do. For example, CRUD commands (Create, Read, Update, and Delete) map/correlate to the HTTP

protocol's POST, GET, PUT, and DELETE methods respectively. CRUD commands (CREATE, READ, UPDATE and DELETE) also correlate to the INSERT, SELECT, UPDATE and DELETE commands of SQL respectively. **Note**: REST APIs REST APIs are not limited to CRUD functions as long as they use the appropriate HTTP method. **Example**: REST API can still allow clients to reboot a server, a command that does not equate to any of the four CRUD commands. Look at the below table for more explanation:

CRUD operation/command	**Corresponding HTTP Methods**	**Corresponding SQL commands**
Create	PUT/POST	INSERT
Read	GET	SELECT
Update	PUT/PATCH	UPDATE
Delete	DELETE	DELETE

- **SOAP (Simple Object Access Protocol)**: SOAP is an established Web API protocol for exchanging structured information. It uses XML to Authenticate, Authorize, and Communicate processes running on operating systems. Since web protocols like HTTP run on most operating systems, SOAP allows clients to invoke web services and receive responses irrespective of language and platform.
    - **Building/developing an API**: This is basically the process exposing your data to allow others (or even you) access your data in many different places/ways or if you want to allow customers or partners limited or complete access to your data. APIs do not have to be publicly available at all. In fact, most companies that have an API only use them internally to allow different parts of their website to talk to each other. You probably do not need to build an API if you just need a landing page (such as a portfolio website) or you're your app will not grow much or you have no plan to expand your website to a mobile or desktop app. You will need to secure (require user to issue API key), have an authentication system (to check whom you want to give access to) prepare documentation (as instruction for developers on how to use your API) when building an API. **Blocks needed to build an a API**:
        - **An API needs a data base**: This can be a database like MySQL or MongoDB

- **An API needs a format for making requests**: When a user wants to use an API, they make a "request". This request usually includes request methods like GET, a URL path to the server the requested data is stored at and a payload (example a form or JSON data). Good APIs offer rules for making these requests in their documentation.
- **API needs to return a response**: Once the API processes the request and gets/saves data to the database, it should return a "response" which usually includes a status code like 404/Not Found, 200/Okay, or 500/Server Error and a payload (usually text or JSON data)

- **Building a simple REST API with NodeJS and Express**: Please remember the difference between serving a static page data and serving dynamic data. Serving static files is serving your HTML, CSS and JavaScript pages as they are. The reason they're called static files is because they are not changed by the server nor run, they're merely sent back as files for your browser.

- **Steps to building a RESTful API using Express server**
  - **Step 1**: Create folder and open it with VSCode
  - **Step 2**: Open terminal/commandliine at this folder
    - We can assume that you have Node.js installed on your computer. To check, just type this on your terminal to see which version of Node you have: node –v
  - **Step 3**: Now, initialize a new app by typing this: npm init. This will create your package.json file and your entry point (index.js is the default file name). There are a series of questions you have to click enter for so that your package.json and your entry point files are clreated
  - **Step 4**: Install Express.js app from your terminal by typing this: npm install express –save
  - **Step 5**: Create your app file, example, app.js in your folder
  - **Step 6**: Create your express server

```
var express = require("express");
```

```
var app = express();
```

- **Step 7**: Set the app you created to listen to a specific port, example, port 3333.

```
app.listen(3333, () => {

console.log("Server listening at port 3333");

});
```

  - Your app will now be accessible using http://localhost:3333

- **Step 8**: Run your localhost by typing this on your browser: http://localhost:3333

  - Here, you will get "Cannot GET/" message on your browser because you do not have an API end point set yet (there is no URL route you set up using GET for your app) but express is responding with the error message

- **Step 9**: A server receives requests, processes them and returns a response. So, you need to use routes to handle this request. The requests can be a GET, POST, PUT or DELETE ones. For our purpose, go ahead and create a simple GET request that returns a list of users. Make your event handler function allow the express app to use the "/users" url to trigger the callback that follows it.

```
app.get("/female", (req, res) => {

    res.send({

    name: "Alem",

    age: 33,
```

```
});
```

- **Step 10**: Run your app using this command: node app.js
- **Step 11**: To view our data, open your localhost (browser) and enter http://localhost:3333/users. You will see a response with JSON format on your localhost/browser.

- *Congrats, you just built a simple useless REST API with NodeJS and Express* 🤣
- **Note (building an API without code)**: Building an API without writing lengthy code is getting easier with ready-made tools like Sheetsu, Airtable, WrapAPI and Bubble.

- **Making an API call:** We have seen above how we can build our own API using Node and Express. In the coming section, we will look at how we can make an API/Rest API call. However, let us see what JSON is and how we can use a JSON format data while making an API call from an application's server.

1.2 What is JSON? Why is JSON format preferred to transmit data in web applications?

- **What is JSON?** JSON stands for JavaScript Object Notation. Just like XML, JSON is also a format for storing and transporting data. JSON is the most widely used data format when exchanging data between two applications (or between 2 two servers the application is hosted at). The standard file type for storing a JSON document in the filesystem is .json.
- **Why is JSON preferred as a way of sending data between applications?** All modern programming languages like Java, JavaScript, Ruby, and Python provide excellent support for producing and consuming JSON data.
  - **Its simplicity**: JSON is a human readable as well as a machine-readable format. So, while applications/libraries can parse the JSON documents, humans can also look at the data and derive the meaning from it.
  - **JSON data is easy to manipulate at client side**: We cannot manipulate an XML data as easily on the client-side, especially in browsers. It ends up being a lot of extra work just to do normal data transfer.

- **JSON is very similar with how JavaScript structures objects:** Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

- **REST APIs accept JSON for request and send responses to JSON**: JSON is the standard for transferring data and REST APIs accept requests for JSON data and send a response in JSON format. **Note**: JavaScript has built-in methods to encode and decode JSON either through the Fetch API or another HTTP client. There are methods that we can use to make sure our REST API app responds with JSON (example, by setting Content-Type in the response header to "application/json").

- **What does a JSON document look like?** A valid JSON document can contain two structures:

  1. An Object surrounded by curly braces and containing multiple name/value pairs.

     Example:

```
 {
    "students": {
    "firstName": "Abebe",
    "lastName":"Bekele"
    }
    }
```

  2. An Array or ordered list of values surrounded by square brackets.

     Example:

```
{"students":
[
{"firstName":"Abebe", "lastName":"Bekele"},
{"firstName":"Almaz","lastName":"Debbie"},
{"firstName":"Challa", "lastName":"Kebede"}
]
```

```
}
```

- **Making API call**: An API call is the process by which a client application makes a request for data from an external server/application and the receiving server delivers the requested data back to the client. Traditionally, performance limitations dictated APIs to make synchronous requests which meant that users taking actions within an application, for example pulling specific info from a database had to await a backend response before continuing onward.
- **Almost all API calls are asynchronous**: Almost modern applications are designed to scale quickly with user activity and more user activity means more requests. Due to that, most APIs these days make asynchronous requests. to We know that asynchronous APIs are great in instances where user activity is heavy. Asynchronous API requests excel at executing background tasks without getting in each other's way.
- **General steps to make an API Call:**
    - **Find the URI of the external server or program**: To make an API call, the first thing you need to know is the Uniform Resource Identifier (URI) of the server or external program whose data you want. This is basically the digital equivalent of a home address. Without this, you will not know where to send your request. **Example**: We can use reqres' fake API and apply this URI as their server's address where we can send our request to: https://reqres.in/api/users
    - **Add an HTTP verb**: Once you have the URI, then you need to make a request by including either one of these verbs: GET, POST, PUT, DELETE. **Example**: Nationalize.io provides a public API whereby we can predict nationality of a person based on name we provide. If you make a request for the name "Mekdes" from Nationalize.io using its server's URI (https://api.nationalize.io/?name=mekdes) using the GET verb, the GET request tells their server to search its database for the name "Mekdes" and provide you with the requested data, i.e., the predicted name of the country the name "Mekdes" belongs to in a JSON format. Below, look at the output in JSON format.

- **Include an API key**: API key is a unique identifier made of up of a string of letters and numbers used to authenticate calls to an API by identifying the client app making the request. API key also tracks the number of requests made for usage and billing purposes.
- **Wait for the response**

- **Different ways of making API call in JavaScript**: JavaScript provides some built-in browser objects and some external open-source libraries to interact with the APIs. There are 5 ways of making an API call in JavaScript: XMLHttpRequest, jQuery,fetch and axios. We will only look at the last two, **the fetch and axios way of making API calls**.
- **Making an API request using fetch()**: fetch() API is a JavaScript method (built in the browser) for getting resources from a server or in short for consuming REST APIs from.
  - **Syntax**: fetch(resource, options)
    - **resource**: This is a mandatory parameter that defines the resource that you wish to fetch. This is basically the URL of the request
    - **options**: This is the second and the optional parameter of fetch() that lets you configure the request. The most useful options are:
      - **options.method**: This is HTTP method to perform the request (GET, POST, DELETE, PUT or UPDATE),
      - **options.body**: the body of the HTTP request. For example. If you are using a POST request, body will be where the data you are posting/inserting will go
      - **option.headers**: an object with the headers to attach to the request.
  - **Calling fetch() returns a promise**: fetch() starts a request and returns a promise. When the request completes, the promise resolves to the response object. This response object provides useful methods to extract data with formats. But to parse data from JSON, you need just one method the response.json() method.

- **Fetch()'s optional parameter**: You can also optionally pass in an init options object as the second argument. **Note**: The response we get from fetch() API method is just a regular HTTP response and not the actual JSON. In order to get the JSON body content from the response, we have to change the response to actual JSON using the json() method on the response. **Note**: Because fetch() method comes as inbuilt with the browser, we do not need to install it (if we were to use it in React) or add it as a CDN (in vanilla JavaScript)

- **Making a GET fetch() API request**: For this, let us use reqres' fake API and apply this URI as address of the server where we send our request to: https://reqres.in/api/users

  - **1. Create an HTML and JavaScript files**: To use fetch() to make a REST API call in from the browser (using vanilla JavaScript), create any JavaScript file to write your code. Make sure to link your HTML and JavaScript file to your HTML file.

  - **2. Fetch the data using the https://reqres.in/api/users URI address:**

    fetch("https://reqres.in/api/users")

  - **3. No need to add the GET HTTP verb for GET requests**: The Fetch API makes a GET request by default, therefore no need to add the GET HTTP request verb

  - **4. No need to use an API key**: The API from reqres is public and you will not need to obtain an API key from their website.

  - **5. Extract the JSON body content from the Response object**: The Response object does not directly contain the actual JSON response body, instead a representation of the entire HTTP response. So, we must extract the JSON body content from the Response object. To do that, we use the json() method, which returns a second promise that resolves with the result of parsing the response body text as JSON.

```
fetch("https://reqres.in/api/users")
.then((res) => res.json())
// code to handle the response data
```

```
  .then((data) => console.log(data))
  .catch((err) => {
  // code to handle request errors
  });
```

- **6. Explaining how catch() only handles fetch() related errors and not other errors, such as, status error**. **Note**: The above API shows only 6 users per page. If we want to get a specific user, like user 5, we can change our URL to fetch to this: https://reqres.in/api/users/5. However, let us say we made a request for the 23rd user by mistake like this: https://reqres.in/api/users/23. In such a case, we will see a 404 status error in our console, but the GET request (fetch) has happened without any problem. You might have thought that we can handle this error with catch() and we would see the "There is error" message in the console in the below code. However, this error cannot be caught with catch() as our GET request has happened and because fetch() always happens/succeeds as long as there is no failure with fetch itself (maybe due to network error or internet connection problem with the browser that causes the data not to be accessed/fetched).

```
fetch("https://reqres.in/api/users/23")
.then((res) => res.json())
.then((data) => console.log(data))
.catch((error) => console.log("There is error"));
```

- **7. Explaining how we can handle status related errors (such as 404 status errors)** : Right after receiving the data from fetch and before applying the json() method to it, check if the response is ok first like below.

```
fetch("https://reqres.in/api/users/22")
.then((res) => {
if (res.ok) {
console.log("This is success");
```

```
} else {
console.log("This is Error");
}
return res.json();
})
.then((data) => console.log(data));
```

- **Making a POST Request with fetch()**: With fetch() you want to do more than getting data from a server. You usually want to post a data to a server (example when you submit forms), update or delete data from the server. For this, you will use the option section of the fetch() method.
  - **When calling fetch() for a POST request, you need to use fetch()'s optional parameters such as**:
    - **Setting the HTTP method parameter as POST**: We said that fetch() does a GET request by default. Therefore, when making any request other than GET, we need to provide the HTTM request verb. Look at the example code
    - **Setting the body parameter with an object and stringifying the object**: You then write the "body" parameter and then set it with a JavaScript object that contains the data you want to post/send into the server. **Note**: When sending data to a web server, the data has to be a string. Therefore, you need to use the JSON.stringify() method to convert JavaScript object data into a string. **Example**: Let us see how the JSON.stringify() method works by converting the below JavaScript object into a string (JSON) format.

```
const myObject = { name: "Ty", age: 3 };
const myJSON = JSON.stringify(myObject);
console.log(myJSON); // prints {"name":"Ty","age":3
```

- **Setting the header parameter with Content-Type:** Most servers require you to indicate explicitly that you POST/PATCH/ PUT) your data in JSON format. In such a case, you must indicate the content type of data you are sending by setting the Content-Type header to application/json.

- **Example of Making a POST request with fetch**: In the below example, let us create a new user called, "User 1" to the requires server using fetch and POST by applying the above information and create/send a new user, called user 1 to reqres server.

```
fetch("https://reqres.in/api/users", {
    method: "POST",
    body: JSON.stringify({
    name: "User 1",
    }),
    headers: {
    "content-type": "application/json",
    },
    })
    .then((res) => {
    return res.json();
    })
    .then((data) => {
    console.log(data);
    console.log(data.name); // returns User 1
    })
    .catch((error) => console.log(error));
```

- **Making an API request with Axios in plain JavaScript**: Axios is a JavaScript library used to make HTTP requests from node.js or from the browser. Axios makes it easy to send asynchronous HTTP requests to REST endpoints and perform CRUD operations. It can be used in plain JavaScript or with a library such as React. In our example, we are using it in our plain JavaScript file. Because Axios does not come as a built in JavaScript method or API, we have to manually import it in our project by installing it from npm (if we are trying to make API request from Node) or include its CDN (if we are trying to make API request from browser).
    - **1.Make sure to include the latest Axios CDN link into your html file**: Make sure to list the CDN on top of your JavaScript file.
    - **2**. **Make sure to set the different parameters axios() takes**: Axios takes three parameters. **Syntax**: axios(method, URL, dataToSend).
        - **Method**: This is basically the HTTP method the request must be sent in
        - **URL**: The URL address of the server the request must be sent to
        - **DataToSend:** This is in case of POST, PUT and PATCH requests that represent the data that you want to send/post to the server in the body of your request
    - **3. Sending a GET request with Axios**: Notice below that there was no need to use the json() method because in Axios responses are already returned as JSON by default, no need to parse, simply get response and access data.

```
axios.get("https://reqres.in/api/users")
.then((res) => {
console.log(res.data);
});
```

    - 3. **How does Axios handle status errors (errors not related to fetching)?** Unlike the fetch() method, axios can handle status errors with catch(). In the below example, the 404 status error is caught by the catch() method. That is why the console in the catch() method prints a message "Request failed with status code 404". Below, let us use a URL for reqres API to get the 22nd user (**Note**: the

reqres API only provides 6 users per page by default. Thus, the URL is incorrect).

```
axios.get("https://reqres.in/api/users/22")
.then((res) => {
console.log(res.data);
})
.catch((error) => {
console.log(error);
});
```

- **4. How does Axios handle network errors?** How does axios handle errors related to the fetching of the data (like incorrect URL, bad internet for browser)?

```
axios.get("https://reqress.in/api/users/2")
.then((res) => {
console.log(res.data);// Axios will provide "Network
  Error" message here
})
.catch((error) => {
console.log(error);//Request failed with status code 404
});
```

- **5 Sending a GET request with Axios**: When sending a POST request with Axios, you will not only need to set the HTTP request method and server's URL, but also need to set the third parameter/option that contains the data that will be sent/posted in the body of the request. There are two ways of making POST request with Axios.
    - **Method 1 of creating a POST request with Axios**:

```
axios.post("https://reqres.in/api/users",        {name:
"User 1",})
.then(({ data }) => {
console.log(data);//shows we posted/sent "User  1" to
server
console.log(data.name); // prints User 1
})
.catch((error) => {
console.log(error);
})
```

- **Method 2 of creating a POST request with Axios**: Note here that fetch() uses the "body" property for a post request, while Axios uses the "data" property

```
axios({
    method: "post",
    url: "https://reqres.in/api/users",
    data: {
    name: "User 1",
    },
    })
    .then(({ data }) => {
    console.log(data); // shows that you created the
  User 1  data in server
    })
    .catch((error) => {
    console.log(error);
```

```
        });
```

1.3 API key: definition and steps to creating a YouTube API key

- In this and the coming section, we will be covering the class demo on how to use API to incorporate Apple's YouTube videos in our Apple.com website. In order to do that, we will need understand concepts like API key and array.map() method.
- **API Key**: It is a code (like a username and password) used to identify and authenticate an application or user making the API request. Most API providers require you to get an API key as a precaution to prevent abuse or malicious use. Meaning, API keys are used to lock anonymous traffic or to limit the number of calls made to an API to govern API consumption.
- **Steps to creating a YouTube API key**: Please note that the steps to creating a YouTube API key change from time to time as Google changes its website features every now and then. Therefore, you might need to research online for current information. However, here is a general step on how to create a YouTube API key:
    - If you do not have a Gmail account, create one and stay signed in
    - Click here, the Google Developer Console link and create a project called "Apple API Project". Wait until Google finishes creating your project
    - Select the "Apple API Project" project you created; you will see all the information about the project you created
    - Now, find "API & Services" and click on it. Then, click on "Library"
    - You will see Google's API library page with a search bar to look for the various API services provided by Google. Just type "YouTube" in the search bar
    - Click on "YouTube Data API v3" and click on the "Enable" button. Wait until Google finishes enabling it
    - You are now taken to the "YouTube Data API v3" page. Find "credentials"
    - Find "Create credentials" and click on it
    - Select the "YouTube Data API v3" option

- Select “public data” to answer the question about what data you will need to access
- Click “Next”. Excellent!!! You will now see your API key created
- If you copy and paste your API key at the end of the link provided below (right after the equal sign), you will see the YouTube videos' data in a JSON form
    - https://www.googleapis.com/youtube/v3/search?key=

1.4 How do we call/request a JSON data from YouTube Using API?

- **Please refer to the link below to get YouTube’s API documentation/references**:
    - https://developers.google.com/youtube/v3/docs?hl=en
- **This link is the main link you use to communicate with YouTube APIs**:
    - https://www.googleapis.com/youtube/v3
- **How to make an API request from YouTube for any YouTube channel**:
    - https://www.googleapis.com/youtube/v3/search?key=addyourapikeyhere&channelId=addChannelIdHere
- **How to make an API request from YouTube for Apple’s channel:** If you look at YouTube’s API documentation, the way to make any request is using this link: https://www.googleapis.com/youtube/v3/channels
    - **1. Provide your API key**: However, just typing the above link would not make a request. You will need to provide a valid API key.
        - https://www.googleapis.com/youtube/v3/channels?key=yourAPIkeyhere
    - **2. Select a filter in your search to show you are looking for apple channel**: Even if you provide your API key in the above URL link and make a request, you will not get any data back as you have not selected a filter specifying which channel you want from YouTube. Meaning, you still need to provide in your search that you want Apple’s channel on YouTube. For that, you will need to provide the “forUsername” filter with “Apple”. **Note**: In most channels, you can find the channel ID from the URL of any of the channel’s video. Apple does not show its channel ID on its YouTube video links, rather it has “Apple” as its

channel name. To see that, just go to Apple's official channel on YouTube and select one of its videos and look at the URL's part that says "ab_channel=Apple". Because Apple does not show its channel ID in the URL of its videos, we need to make a request to find the channel ID using this channel name. The below request is basically to get the Apple's channel ID:

- **3. Get the channel ID**: Just copy and paste the above link (after you inserted your API key), you will get a JSON file in response to your request that contains Apple's channel ID. Apple's channel ID is the text in red below.


```
{
    "kind": "youtube#channelListResponse",
    "etag": "mz69kTl4eTWQ_DNpITJReTTsCDY",
    "pageInfo": {
    "totalResults": 1,
    "resultsPerPage": 5
    },
    "items": [
    {
    "kind": "youtube#channel",
    "etag": "YBIB3kIgzFGk1Uu42kfTJsfeA4w",
    "id": "UCE_M8A5yxnLfW0KghEeajjw"
    }
    ]
    }
```


- **4. Set the channel ID**: Now that we have Apple channel's id, let us use the "channelId" parameter in our request

        - https://www.googleapis.com/youtube/v3/search?key=yourAPIkeyhere&channelId=UCE_M8A5yxnLfW0KghEeajjw
        - **Note**: The above request by default gives us 5 results (videos) per page with every information needed. However, in projects, you usually want to limit your search. Below is the step where we can limit our search using different parameters.
    - **5. Use the "part", "order" and "results" parameters to ensure that API responses only include the data that your application uses**: For our Apple website, let us only show 12 YouTube videos (that means, set the result parameter to 12), use only the snippet information (meaning, you need to set the part parameter to snippet) and finally order the videos by date (that means set the order parameter to date).
        - https://www.googleapis.com/youtube/v3/search?key=yourAPIkeyhere&channelId=UCE_M8A5yxnLfW0KghEeajjw&part=snippet,id&order=date&maxResults=12
        - Cust paste the above URL on your browser to test
- **In real life, what can we do with the data we fetched?** In real life project, we will not just be fetching the data and display it. In most cases, once the data is fetched, we want to manipulate/to do something to each single data and display it. For instance, when we fetched Apple's videos from YouTube, we went through each fetched video and displayed only the video's thumbnail, title and the snippet. In order to do something on each fetched item, we will use the Array.map() method/function.
- **JavaScript Array.map() method/function**: The map() is a method that is defined inside the JavaScript Array class. This method takes one array and apply a manipulation on each element of the array and then creates another array containing the new values.
    - **Syntax**: myArray.map(callBackFunc(currentVal,index,currentArr))
    - **The Array. map() method takes a mandatory callback function as its parameter**: The map() method allows us to iterate over an array and modify its elements using a callback function. The callback function will then be executed on each of the array's elements. Once this is done, the map() method finally creates a new array containing of the new values the function has returned.

- **The currentVal parameter of the function**: This parameter is mandatory and returns the value of the current element.
- **The index parameter of the function**: This parameter is optional and specifies the index of the current array element
- **The currentArr parameter of the function**: This parameter is optional and returns the array of the current element.

- **Example showing how to use map() method**: Let us say we have a parent array with name Abebe that has numbers as its elements. If say for example, we want another array with the triple values of each of the Abebe array elements. The map method takes a function as a parameter that is going to multiply each of Abebe's elements by 3 and returns/creates a new array containing the new values the function has returned.

```
var Abebe = [2, 5, 3, 8];
var myNewArray = Abebe.map(function (currentVal) {
return currentVal * 3;
});
console.log(myNewArray); // returns [6, 15, 9, 24]
console.log(myNewArray[0]); // returns 6
```

- **Example of an API request using fetch() in React**: In the previous sections, we have seen how we can make an API call using fetch() and Axios in vanilla JavaScript.
  - **Create a react app**: Revise your React classes for this
  - **Create your component**: In your React app, under the "src" folder, create a functional component. Note: You can also use a class component
  - **If you are using functional component,**
    - **import useState and useEffect** *from React*

        - **Initialize your state and set your state updater function using useState()**: Create a variable to hold your state as well as the stateUpdater function via useState
        - Within your useEffect() function, call a callback function that fetches the data you want from the API's server. To do this, you will need to use the fetch() methods and the URL address of the resource you want to fetch
    - **If you are using class component, import Component from React**:
        - Call a constuctor and initialize your state
        - Within your componentDidMount(), call fetch() to fetch the data you want from the API's server

- **Why fetch() is a side effect in React functional components? (Revision of useEffect()hook)**: Components are used primarily to compute and render outputs whenever a prop/state of the component changes. However, there are times when a component makes computations that do not target the change in state/prop value. These calculations are called side-effects and we use useEffect() hook that we use to handl side-effects independent from renderings. Fetching data is a good example of side effects as the main purpose of any component is not to just fetch the data but to render output wherever there is a state change.
- **Simple example showing how to make an API request using fetch() in React**: Below, you will see how we can call an API using functional and class components in React. We will be using a free API from Makeup API, therefore, you will not need to create an API to access their data.
- **Making an API request in functional components**:

```
function MakeupAPI() {
    let [allMakeups, setMakeups] = useState([]);
    useEffect(() => {
```

```
fetch("https://makeup-api.herokuapp.com/api/v1/products.json          brand=maybelline")
      .then((response) => response.json())
      .then((mydata) => {
      setMakeups(mydata);
      console.log(mydata); //checks if data is fetched
      });
      }, []);
      return (
      <div>
      {allMakeups.map((singleMakeup) => {
      let makeupName = singleMakeup.name;
      let makeupPrice = singleMakeup.price;
      let makeupTodisplay = (
      <div>
      <div>{makeupName}</div>
      <div>{makeupPrice}</div>
      </div>
      );
      return makeupTodisplay;
      })}
      </div>
      );
      }
```

```
function MakeupAPI() {
    let [allMakeups, setMakeups] = useState([]);
    useEffect(() => {

fetch("https://makeup-api.herokuapp.com/api/v1/products.js
on    brand=maybelline")
    .then((response) => response.json())
    .then((mydata) => {
    setMakeups(mydata);
    console.log(mydata); //checks if data is fetched
    });
    }, []);
    return (
    <div>
    {allMakeups.map((singleMakeup) => {
    let makeupName = singleMakeup.name;
    let makeupPrice = singleMakeup.price;
    let makeupTodisplay = (
    <div>
    <div>{makeupName}</div>
    <div>{makeupPrice}</div>
    </div>
    );
    return makeupTodisplay;
    })}
```

```
      </div>
      );
      }
```

- **Making an API request in class components**:

```
class MakeupAPI extends Component {
      constructor() {
      super();
      this.state = { allMakeups: [] };
      }
       componentDidMount() {
                  fetch("https://makeup-api.herokuapp.com/api/v1/
products.json?brand=maybelline")
      .then((response) => response.json())
      .then((mydata) => {
      this.setState((state) => {
      return { allMakeups: mydata };
      });
      console.log(mydata); // check if data is fetched
      });
      }
      render() {
      return (
      <div>
      {this.state.allMakeups.map((singleMakeup) => {
```

```
let makeupTodisplay = (
<div>
<div>{singleMakeup.name}</div>
 <div>{singleMakeup.price}</div>
</div>
);
return makeupTodisplay;
})}
     {this.makeupTodisplay}
</div>
);
}
}
```

1.5 Demo on how to use the JSON data we obtained from YouTube API in our React component

- **Watch the class demo video for this section**:
  - https://www.youtube.com/watch?v=RL_dIA8HoOg&ab_channel=EvangadiTech