UNIT-III

API, Application Programming Interface Introduction to API, What is API, Why we need API, API Types, API Endpoints, Paths and Parameters. API Authentication and Postman, JSON, Making GET Requests with the Node HTTPS Module Parse JSON, Using Express to Render a Website with Live API Data, Using Body Parser to Parse POST Requests to the Server, API Authentication and Types.

Introduction to API

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

API connects two devices or programs in order to facilitate the exchange of information between them. It is the interface that serves the other parts of the software.

The API specifications are the standards or documents designed to describe the creation of such connections. If a computer system meets these standards, then it is said to expose an API. The specification or implementation both are known as the API.

Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API.

Why we need API/ Advantage of API

Application Programming Interface (API) plays a crucial part in the economy. APIs are not easily noticed, but they are almost everywhere in our lives and affect us significantly. API is crucial in sharing data and aids communication between two individuals, apps or devices. Following are some of the common daily life functions of API.

Weather Monitoring

Weather monitoring is one of the most common daily life applications of API. The weather update can be seen on Google Search, smart home devices or Apple's Weather app. whenever you search for the weather of your city using any of these platforms, you'll see an instant result.

For example, although Google doesn't do Business in weather, you will get data immediately if you search on Google. It is because Google takes data from a third party using API. After getting this data, Google reformats it and presents it to the user. Many APIs have weather data functionality and are crucial in providing weather conditions and forecasts.

For Logging in

APIs are also helpful for logging into accounts such as Facebook, Google, Twitter, Github, etc. It helps in accessing the functionalities in these accounts after you log in.
API does not directly login into a user's account as it would pose a threat to security and privacy. Applications use the API of platforms to authenticate the user with each login.

Payments

The payment functionality is also built using APIs. Using APIs, it is ensured that the application is only exposed to the necessary data rather than the sensitive data. API prevents the apps from accessing the unintended data and helps it achieve the set objective.

The working of the payment system is similar to the login functionality. After the user sends the pay command to the application, he sends an order with the payment amount. Additionally, he also adds other relevant information. A pop-up will then confirm the payment by authenticating the user. Subsequently, the API sends payment confirmation to the application and hence the user.

Bots

Bots are another everyday application of the API. For example, there are many Twitter Bots which are Twitter accounts that carry out functions such as a tweet, follow and send messages directly with the order from the software. Some of the Bots are listed below;

- TinyCare Bot: It has the function of reminding the user to drink water, stretch, get fresh air, etc.
- Grammar Police: Rectifies the grammar mistakes of the users and makes corrections accordingly.
- Netflix Bot: Tweets content-related updates from Netflix.

API Types

The various types of web APIs are as follows;

Open APIs: It is also known as the external or public API and is the API that has minimal restrictions for the user. These sometimes require registration or API key or sometimes are completely open. These are meant for external users to access data. Therefore, the developers can access the data without the registrations.

Internal APIs: Internal APIs differ from open APIs as these are hidden from external users. These are useful for businesses and are used in a company to share resources within the company. It facilitates different sections and departments of the company to access each other's data. Internal APIs are useful in providing a standard interface for the connection of multiple services in a company.

Partner APIs: Partner APIs are very much similar to open APIs. The only difference is that these feature restricted areas that are controlled by an API gateway, a third-party gateway. These APIs are used for specific services, such as the paid service on any software.

Composite APIs: Composite APIs are unique as they facilitate the developer to access several endpoints in one call. It may have different endpoints for a single API or multiple data sources and services. Composite types of APIs are useful when there is the need to access data from multiple services for the completion of a single task.

The composite APIs have three major benefits, which are as follows;

- It reduces the server load
- Improves the performance of the application
- Fetches multiple service data in a single call

Paths and Parameters

Path parameters are request parameters attached to a URL that point to a specific REST API resource. The path parameter is separated from the URL by a \`/\`, and from the query parameter(s) by a question mark (\`?\`).

The path parameter defines the resource location, while the query parameter defines sort, pagination, or filter operations.

The user's input (the query) is passed as a variable in the query parameter, while each path parameter must be substituted with an actual value when the client makes an API call. The path parameter is contained within curly braces

Example: surfreport/beachId?days=3&units=metric&time=1400\`\`\`

What do we see in this GET request?

- \`/surfreport/{beachID}\` \`/surfreport\` is the endpoint, and \`beachID\` is the path parameter. The \`/surfreport/{beachID}\` path parameter takes a geographic beach code to look up the resource associated with that code.
- \`?\` is where our query string begins. This information is returned in a JSON file.

API Authentication

The API authentication process validates the identity of the client attempting to make a connection by using an authentication protocol.

The protocol sends the credentials from the remote client requesting the connection to the remote access server in either plain text or encrypted form. The server then knows whether it can grant access to that remote client or not.

1. HTTP basic authentication

If a simple form of HTTP authentication is all an app or service requires, HTTP basic authentication might be a good fit. It uses a locally derived username and password and relies on Base64 encoding.

Authentication uses the HTTP header, making it easy to integrate. Because this method uses shared credentials, however, it's important to rotate passwords on a regular basis.

2. API access tokens

API keys have unique identifiers for each user and for every time they attempt to authenticate. Access tokens are suitable for applications where many users require access. Access tokens are secure and easy to work with from an end-user perspective.

3. OAuth with OpenID

Despite having auth in the name, OAuth is not an authentication mechanism. Instead, it provides authorization services to determine which users have access to various corporate resources.

OAuth is used alongside OpenID, an authentication mechanism. Using OAuth and OpenID together provides authentication and authorization. With OAuth 2.0, OpenID can authenticate users and devices using a third-party authentication system. This combination is considered one of the more secure authentication/authorization options available today.

4. SAML federated identity

Security Assertion Markup Language (SAML) is another tokenlike authentication method often used in environments that have federated single sign-on (SSO) implemented. This XML-derived open standard framework helps seamlessly authenticate users through the organization's respective identity provider.

For larger organizations working to consolidate the number of authentication mechanisms within the company, the use of SAML and federated SSO is a great fit.

Postman

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration to make working with API easy. It also helps to build, test and modify APIs.

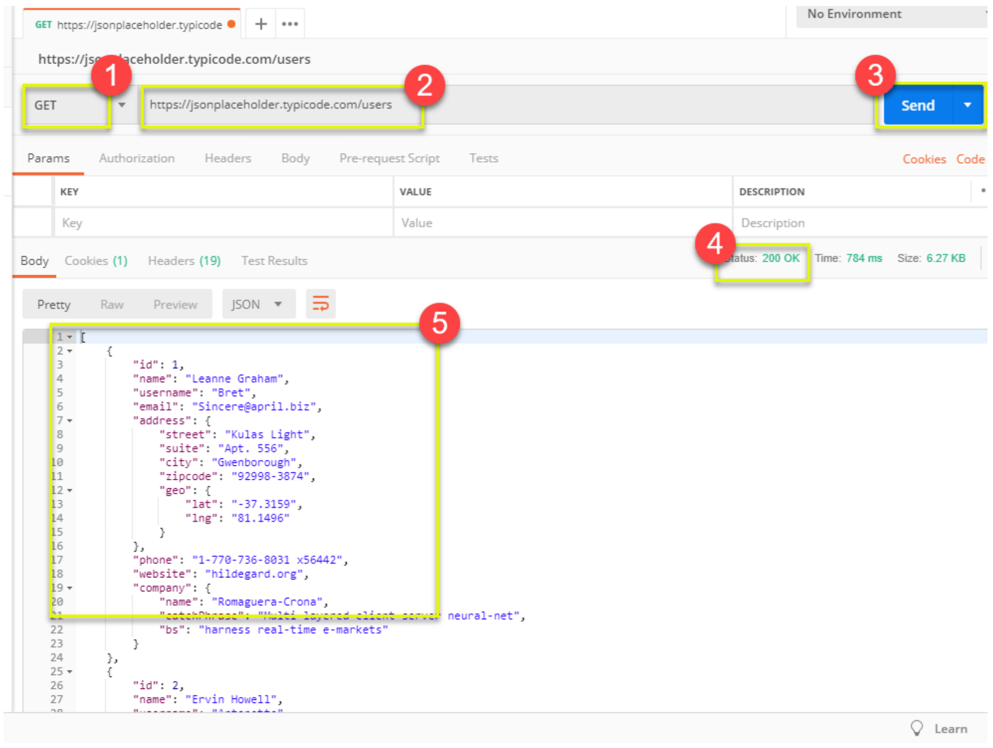
It has the ability to make various types of HTTP requests(GET, POST, PUT, PATCH), saving environments for later use, converting the API to code for various languages(like JavaScript, Python).

POSTMAN & Working with GET Requests

The following URL is used to make GET requests

https://jsonplaceholder.typicode.com/users

1. Set your HTTP request to GET.
2. In the request URL field, input link
3. Click Send
4. You will see 200 OK Message
5. There should be 10 user results in the body which indicates that your test has run successfully.



https://jsonplaceholder.typicode.com/users

Status: 200 OK Time: 784 ms Size: 6.27 KB

```
[
    {
        "id": 1,
        "name": "Leanne Graham",
        "username": "Bret",
        "email": "Sincere@april.biz",
        "address": {
            "street": "Kulas Light",
            "suite": "Apt. 556",
            "city": "Gwenborough",
            "zipcode": "92998-3874",
            "geo": {
                "lat": "-37.3159",
                "lng": "81.1496"
            }
        },
        "phone": "1-770-736-8031 x56442",
        "website": "hildegard.org",
        "company": {
            "name": "Romaguera-Crona",
            "catchPhrase": "Multi-layered client-server neural-net",
            "bs": "harness real-time e-markets"
        }
    },
    {
        "id": 2,
        "name": "Ervin Howell",
```

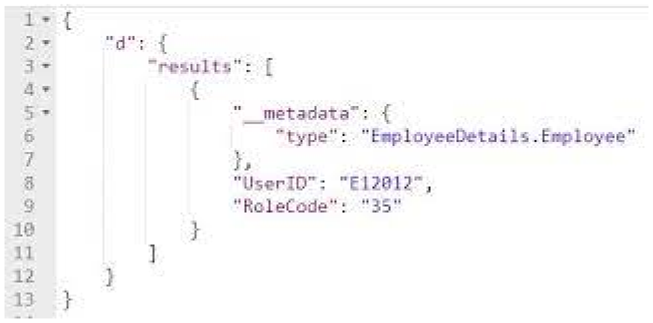
JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.



```
{
    "d": {
        "results": [
            {
                "__metadata": {
                    "type": "EmployeeDetails.Employee"
                },
                "UserID": "E12012",
                "RoleCode": "35"
            }
        ]
    }
}
```

Making GET Requests with the Node HTTPS Module Parse JSON

There are many approaches to create different kinds of network requests. These are 2 different approaches of them.

- Using AXIOS module
- Using HTTP module

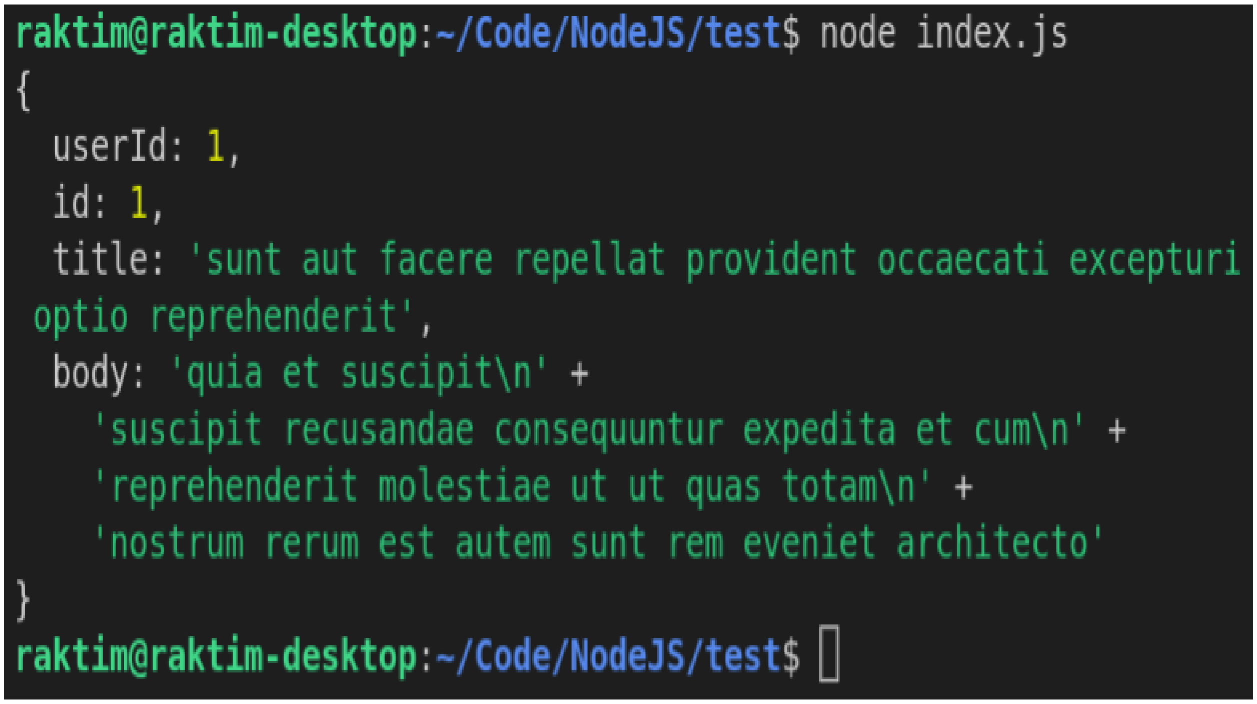
Using Axios:

Axios is a promise base HTTP client for NodeJS. You, can also use it in the browser. Using promise is a great advantage when dealing with asynchronous code like network request.

>npm i axios

```
const axios = require('axios')

// Make request
axios.get('https://jsonplaceholder.typicode.com/posts/1')

  // Show response data
  .then(res => console.log(res.data))
  .catch(err => console.log(err))
```

OUTPUT:



```
raktim@raktim-desktop:~/Code/NodeJS/test$ node index.js
{
  userId: 1,
  id: 1,
  title: 'sunt aut facere repellat provident occaecati excepturi
 optio reprehenderit',
  body: 'quia et suscipit\n' +
    'suscipit recusandae consequuntur expedita et cum\n' +
    'reprehenderit molestiae ut ut quas totam\n' +
    'nostrum rerum est autem sunt rem eveniet architecto'
}
raktim@raktim-desktop:~/Code/NodeJS/test$ 
```

HTTP Module

NodeJS have built in HTTP module to make network request. But the drawbacks is that, it is not too user friendly like the other solution. You, need to manually parse the data after received.

```
// Importing https module
const http = require('http');

// Setting the configuration for
// the request
const url = 'jsonplaceholder.typicode.com'


// Sending the request
const req = http.request(url, (res) => {
	let data = ''

	res.on('data', (chunk) => {
		data += chunk;
	});

	// Ending the response
	res.on('end', () => {
		console.log('Body:', JSON.parse(data))
	});
```

Using Express to Render a Website with Live API Data

—------> OPEN Weather or OMBD Website Program <—---------

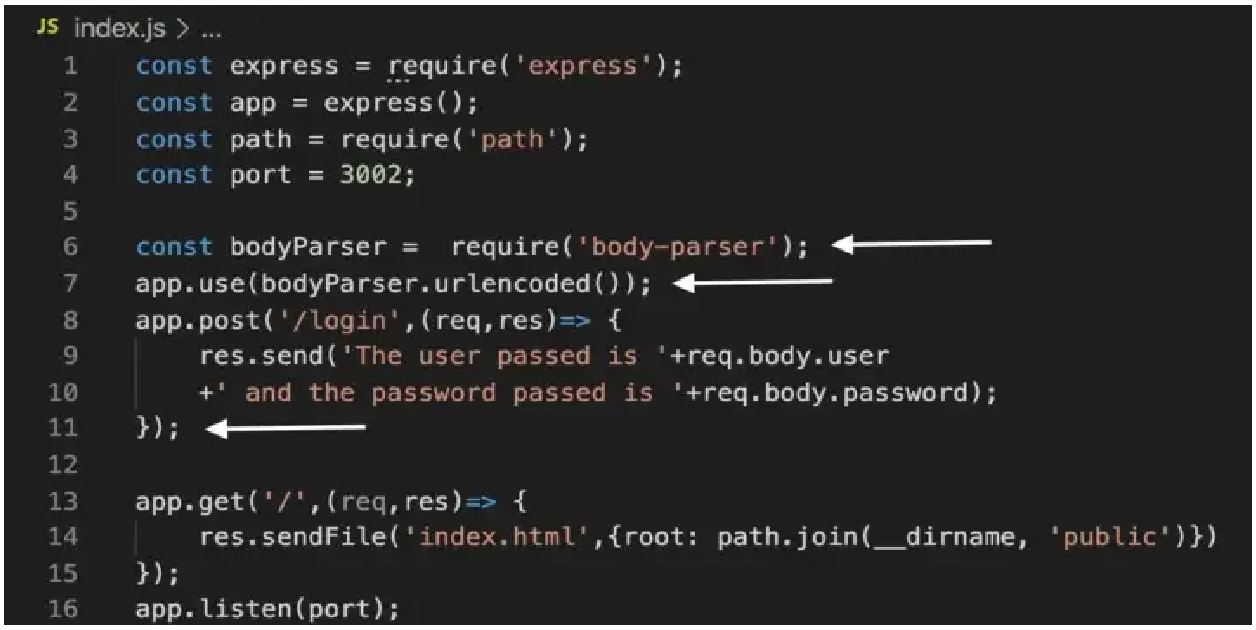
Using Body Parser to Parse POST Requests to the Server

What Is Body-parser?

- Body-parser parses is an HTTP request body that usually helps when you need to know more than just the URL being hit.
- Specifically in the context of a POST, PATCH, or PUT HTTP request where the information you want is contained in the body.
- Using body-parser allows you to access req.body from within routes and use that data.

  For example: To create a user in a database.

The URL-encoded form body parser

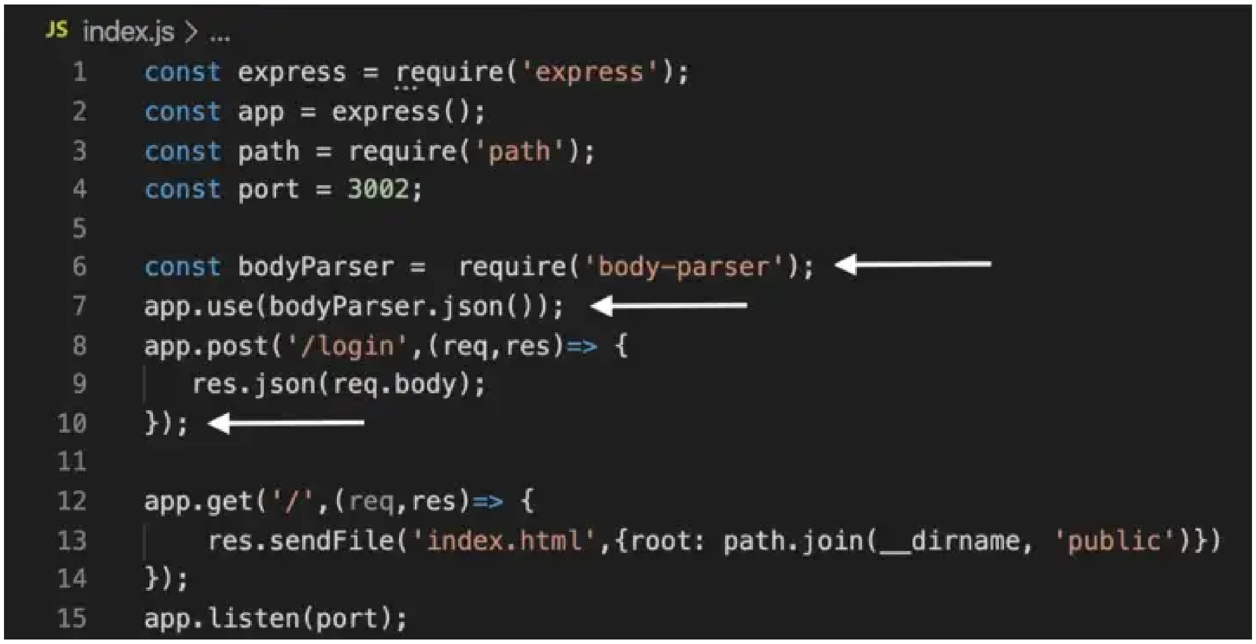


```javascript
const express = require('express');
const app = express();
const path = require('path');
const port = 3002;

const bodyParser =  require('body-parser');
app.use(bodyParser.urlencoded());
app.post('/login',(req,res)=> {
    res.send('The user passed is '+req.body.user
    +' and the password passed is '+req.body.password);
});

app.get('/',(req,res)=> {
    res.sendFile('index.html',{root: path.join(__dirname, 'public')})
});
app.listen(port);
```

`index.html`

```html
<html>
<body>
<form action="/login" method="POST">
Username : <input type="text" name="user"><br>
Password : <input type="password" name="password"><br>
<input type="submit">
</form>
</body>
</html>
```

JSON Body Parser:

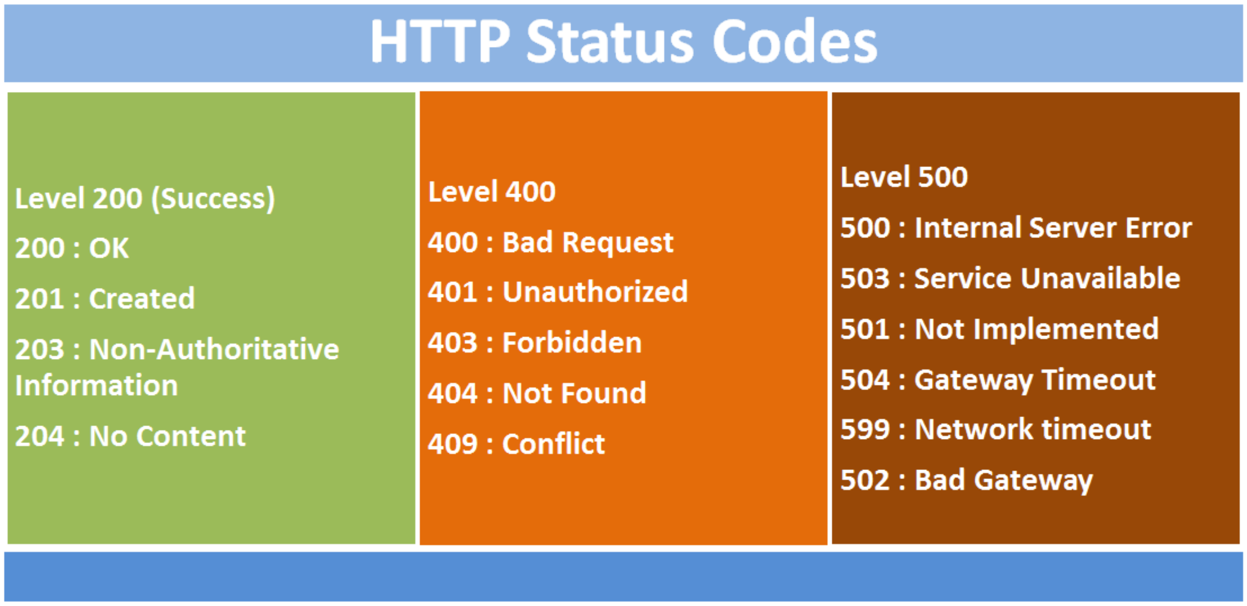

JS index.js > ...

```javascript
const express = require('express');
const app = express();
const path = require('path');
const port = 3002;

const bodyParser =  require('body-parser');
app.use(bodyParser.json());
app.post('/login',(req,res)=> {
    res.json(req.body);
});

app.get('/',(req,res)=> {
    res.sendFile('index.html',{root: path.join(__dirname, 'public')})
});
app.listen(port);
```

HTTP METHODS & Status Codes

The primary or most commonly-used HTTP methods are POST, GET, PUT, PATCH, and DELETE. These methods correspond to create, read, update, and delete (or CRUD) operations, respectively.

HTTP Method	CRUD operation	Entire Collection (e.g. /users)	Specific Item (e.g. /users/{id})
GET	Read	200 (OK), list of entities. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single entity. 404 (Not Found), if ID not found or invalid.
POST	Create	201 (Created), Response contains response similar to **GET** /user/{id} containing new ID.	**not applicable**
PATCH	Update	Batch API	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	204 (No Content). 400(Bad Request) if no filter is specified.	204 (No Content). 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	**not implemented**	**not implemented**



HTTP Status Codes

Level 200 (Success)	Level 400	Level 500
200 : OK	400 : Bad Request	500 : Internal Server Error
201 : Created	401 : Unauthorized	503 : Service Unavailable
203 : Non-Authoritative Information	403 : Forbidden	501 : Not Implemented
204 : No Content	404 : Not Found	504 : Gateway Timeout
	409 : Conflict	599 : Network timeout
		502 : Bad Gateway

No.	SOAP	REST
1)	SOAP is a **protocol**.	REST is an **architectural style**.
2)	SOAP stands for **Simple Object Access Protocol**.	REST stands for **REpresentational State Transfer**.
3)	SOAP **can't use REST** because it is a protocol.	REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP.
4)	SOAP **uses services interfaces to expose the business logic**.	REST **uses URI to expose business logic**.
5)	**JAX-WS** is the java API for SOAP web services.	**JAX-RS** is the java API for RESTful web services.
6)	SOAP **defines standards** to be strictly followed.	REST does not define too much standards like SOAP.
7)	SOAP **requires more bandwidth** and resource than REST.	REST **requires less bandwidth** and resource than SOAP.
8)	SOAP **defines its own security**.	RESTful web services **inherits security measures** from the underlying transport.
9)	SOAP **permits XML** data format only.	REST **permits different** data format such as Plain text, HTML, XML, JSON etc.
10)	SOAP is **less preferred** than REST.	REST **more preferred** than SOAP.

Model Questions

1. What is an API? Explain in detail. Why do we need them?
2. Explain about various advantages in using API in detail.
3. What is an API? Explain about various types of API used in the communication
4. Explain about PATHS, Parameters and Query in an API.
5. Explain various Authentication methods used in API communication.
6. Explain about OAUTH in detail.

Unit 2

Fundamentals of Node JS & Express:
What is NodeJS, Install Node.js on Mac/ Windows, The Node REPL (Read Evaluation Print Loops), The NPM Package Manager and Installing External Node Modules
Node with Express:

What is Express?, Creating Our First Server with Express, Handling Requests and Responses: the GET Request, Responding to Requests with HTML Files, Processing Post Requests with Body Parser.

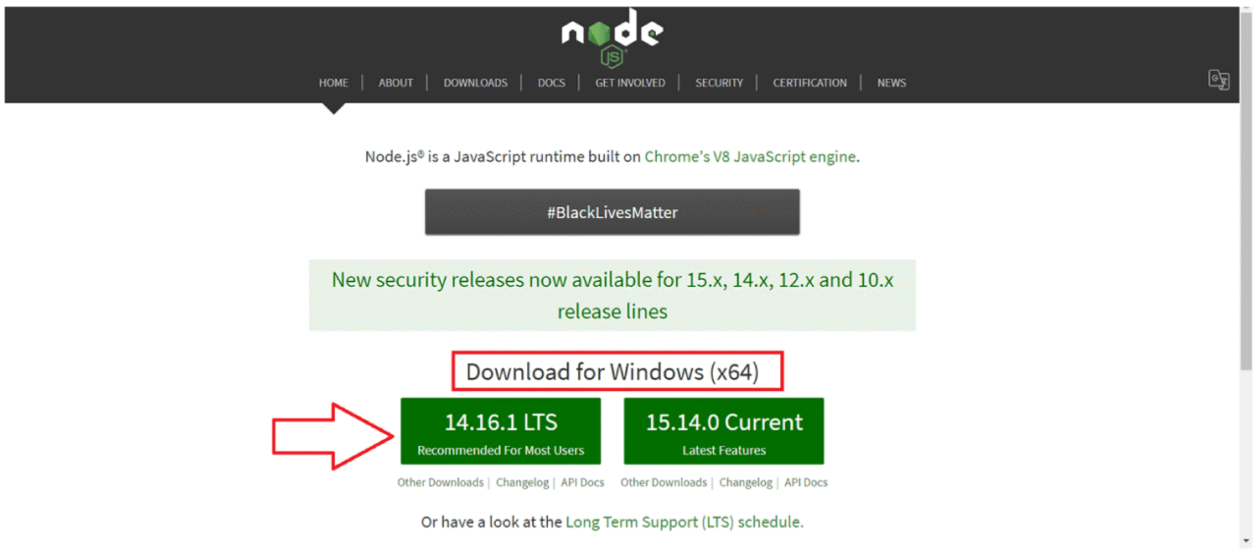
What is NodeJS, Install Node.js on Mac/ Windows

NodeJs is a cross-platform environment that runs and executes JavaScript codes outside the browser.

It's a version of Chrome's V8 JavaScript runtime engine, allowing you to develop server-side JavaScript applications. It comes up with everything which is required to run a program written in JavaScript.

How to Install Node.js and NPM on Windows?

Step 1: Download the Installer Download the Windows Installer from NodeJs official website. Make sure you have downloaded the latest version of NodeJs. It includes the NPM package manager. Here, we are choosing the 64-bit version of the Node.js installer.


HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS
Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.
#BlackLivesMatter
New security releases now available for 15.x, 14.x, 12.x and 10.x release lines
Download for Windows (x64)
14.16.1 LTS
Recommended For Most Users
15.14.0 Current
Latest Features
Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs
Or have a look at the Long Term Support (LTS) schedule.

Running the Node.js installer.

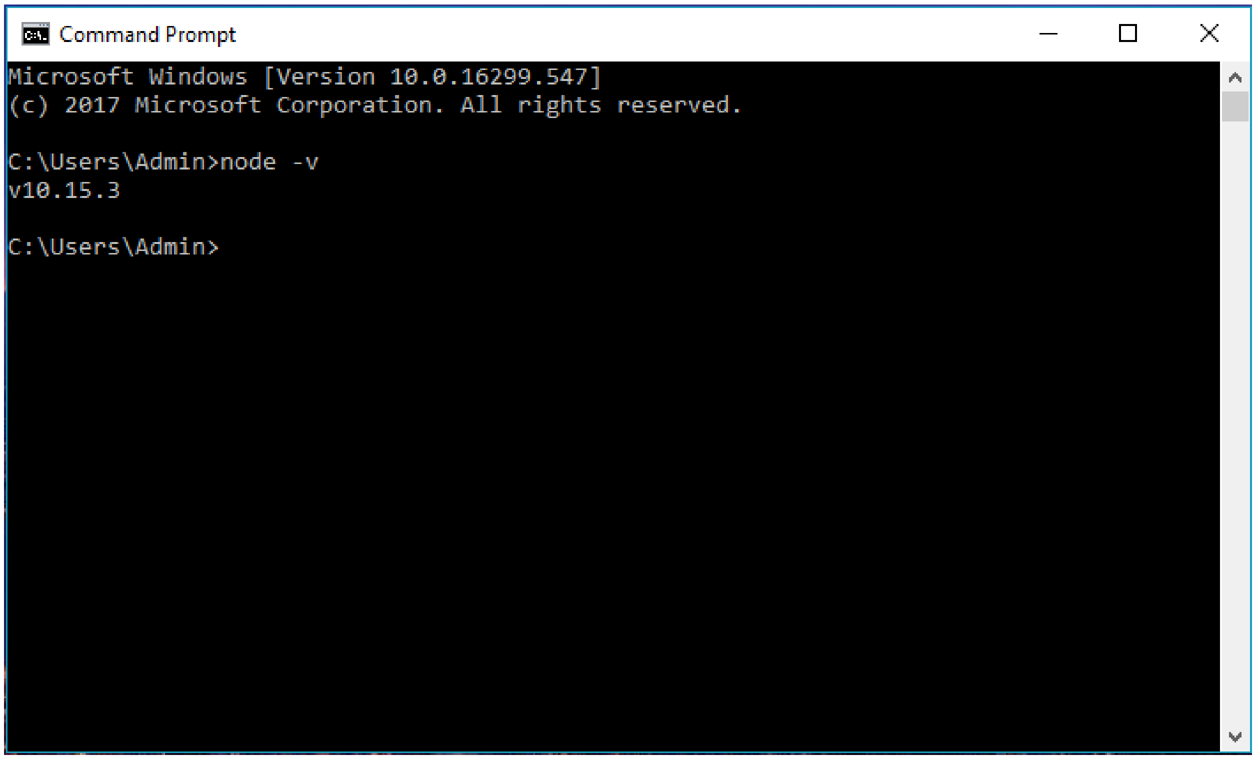
Now you need to install the node.js installer on your PC. You need to follow the following steps for the Node.js to be installed:-

Double click on the .msi installer.
The Node.js Setup wizard will open. Install it on desired directory.

Verify that Node.js was properly installed or not.

To check that node.js was completely installed on your system or not, you can run the following command in your command prompt or Windows Powershell and test it:-

C:\Users\Admin> node -v



```
Microsoft Windows [Version 10.0.16299.547]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Admin>node -v
v10.15.3

C:\Users\Admin>
```

UBUNTU Environment

Installing Node.js with Apt from the Default Repositories

use the apt package manager. Refresh your local package index first by typing:

sudo apt update

Then install Node.js:

```
sudo apt install nodejs
```

Check that the install was successful by querying node for its version number:

```
node -v
```

Output

```
V10.19.0
```

The Node REPL (Read Evaluation Print Loops)

A Read-Eval-Print Loop (REPL) is a simple, interactive computer programming environment. The term 'REPL' is usually used to refer to a LISP interactive environment but can be applied to command line shells and similar environments for programming languages like Python, Ruby etc.

In a REPL environment, an user can enter one and more expressions, which are then evaluated (bypassing the compile stage), and the result displayed. There are 4 components to a REPL (comes from the names of the Lisp primitive functions) :

- A read function, which accepts an expression from the user and parses it into a data structure in memory .
- An eval function, which takes the data structure and evaluates.
- A print function, which prints the result.
- A loop function, which runs the above three commands until termination

Node.js ships with a REPL. If you start the Node.js binary without any arguments, you will see the REPL command prompt, the > character.

The node.js REPL works exactly the same as Chrome's REPL, so from this prompt, you can type any Javascript command you wish. It is extremely useful for experimenting with node.js and debugging JavaScript code.

The NPM Package Manager

NPM (Node Package Manager) is the default package manager for Node.js and is written entirely in Javascript**.**

NPM manages all the packages and modules for Node.js and consists of command line client npm. It gets installed into the system with installation of Node.js. The required packages and modules in Node project are installed using NPM.

NPM can install all the dependencies of a project through the package.json file. It can also update and uninstall packages. In the package.json file, each dependency can specify a range of valid versions using the semantic versioning scheme, allowing developers to auto-update their packages while at the same time avoiding unwanted breaking changes

Installing NPM:
To install NPM, it is required to install Node.js as NPM gets installed with Node.js automatically. Install Node.js.

Checking and updating npm version:
Version of npm installed on system can be checked using following syntax:

Syntax: npm -v

Installing Packages:
After creating the project, the next step is to incorporate the packages and modules to be used in the Node Project. To install packages and modules in the project use the following syntax:

Syntax:
npm install package_name

Example: Installing the express package into the project. Express is the web development framework used by the Node.
Syntax:

npm install express

To use express in the Node, follow the below syntax:
Syntax:

var express = require('express');

Example: **To install a package globally** (accessible by all projects in system), add an extra -g tag in syntax used to install the package.

Installing nodemon package globally.

npm install nodemon -g

Controlling where the package gets installed:

To install a package and simultaneously save it in the package.json file (in case using Node.js), add –save flag. The –save flag is default in the npm install command so it is equal to npm install package_name command.

Example:

```
npm install express --save
```

By –save flag one can control where the packages are to be installed.
–save-prod : Using this packages will appear in Dependencies which is also by default.
–save-dev : Using this packages will get appear in devDependencies and will only be used in the development mode.

Example: `npm install node-color –save-dev`

What is Express?, Creating Our First Server with Express,

Express JS is a Node.js framework designed to build API's web applications cross-platform mobile apps quickly

First Server Program:

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Handling Requests and Responses: the GET Request

GET and POST is two common HTTP Requests used for building REST APIs. Both of these calls are meant for some special purpose.

GET request

Handling GET requests in Express is a pretty straightforward approach. You have to create instances of express and router. Here is a small snippet to achieve the same.

```
var express = require('express');
var app = express();
var PORT = 3000;

app.get('/', (req, res) => {
  res.send("GET Request Called")
})

app.listen(PORT, function(err){
    if (err)
            console.log(err);
            console.log("Server listening on PORT", PORT);
});
```

POST Request

Express requires an additional middleware module to extract incoming data of a POST request. This middleware is called ‘body-parser. We need to install it and configure it with Express instance.

```
var express = require('express');
var app = express();
var PORT = 3000;

app.post('/', (req, res) => {
  res.send("POST Request Called")
})

app.listen(PORT, function(err){
    if (err) console.log(err);
    console.log("Server listening on PORT", PORT);
});
```

Responding to Requests with HTML Files, Processing Post Requests with Body Parser.

To render an HTML file into the server using Express.js, we use **res.sendFile()**. This reads and renders the data included in one's HTML files

Syntax: res.sendFile(path [, options] [, fn])

To process content, You can install body-parser inorder to post the content to the server.

sudo npm install --save body-parser

Body-parser has a few modes such as:

- bodyParser.text - pass all the requests into text.
- bodyParser.json - parse data into JSON format.
- bodyParser.urlencoded - commonly used when getting data posted from HTML forms.

Program:

```
const express = require('express');
const bodyParser = require('body-parser');
const https = require('https');
const app = express();
app.use(express.static(__dirname + '/public'));

app.use(bodyParser.urlencoded({ extended: true }));

app.get("/", (req, res) => {
  res.sendFile(__dirname + "/index.html");
});

app.post("/", (req, res) => {
  const query = req.body.movieName;
  const url = "https://www.omdbapi.com/?t=" + query + "&apikey=785d4dec";

  https.get(url, (response) => {
    let chunks = " ";
    response.on("data", (data) => {
      chunks += data;

    });
    response.on("end", function() {
      var result = JSON.parse(chunks);
      console.log(result)
      var imdbResult = result.imdbRating;
      res.render(“The rating is “ +imdbResult)

    });
  });
});

app.listen(3000, function(req, res) {
  console.log("server started");
});
```

Model Questions

1. What is NodeJS? How to Install Node.js on various platforms like Mac/ Windows?
2. Explain in detail about NODE REPL?
3. What is a Node Package Manager? Explain its significance in NodeJS in detail?
4. What are External modules in NodeJS? Differentiate between External and Internal Modules in detail
5. Explain about the process of Handling Requests and Responses with the help of Node and Express
6. What is a Body Parser? Explain it in detail?
7. Explain the procedure of how to respond to Requests with HTML Files
8. What is body Parse? Explain how it is useful in post processing of data on the server?
9. Write a program to capture live weather data from a server?

Unit - 1

CSS Position, Rounded Corner, CSS Text Effects, CSS Gradients, Pagination, Shadows, CSS Flexbox, CSS Animation

Introduction to Responsive Web Design: Viewport, Introduction to Media Queries, Responsive Web Page Design using Media Queries, Introduction to CSS Grid, Layout, Elements, Grid Rows, Grid Columns, Grid Gaps, Grid Rows, Columns, Grid Template

CSS Position:

The position CSS property sets how an element is positioned in a document. The top, right, bottom, and left properties determine the final location of positioned elements. The element is positioned according to the normal flow of the document. The top, right, bottom, left, and z-index properties have no effect.

There are five types of CSS position properties are there.

1. Static Position: This is the default CSS position applicable to all HTML elements. This position places the HTML elements based on normal document flow. Using this position, top/right/bottom/left properties can not be applicable to elements (ie., static position elements don’t obey top/right/bottom/left properties).

Note: Page scrolling does affect this position.

Example:

```
.p1 {
    width:50px;
    height:50px;
    background-color:cornflowerblue;
    position:static;
    }
```

2. Relative Position:

This position places the element relative to its normal position. This position is relative to normal document flow. Here top/right/bottom/left properties can be applied to elements.

Note:

Page scrolling does affect this position.

If position only applied without top/right/bottom/left properties, it will act like Static position.

Example:

```
.p2 {
   width:50px;
   height:50px;
   background-color:lightgreen;
   position:relative;
   left:60px;
   top:20px;
   }
```

3. Absolute Position:

This position places the element at the exact position specified.

This position is relative to

- its parent element position when parent element position is relative/absolute.
- document body (browser viewport) when parent element position is static.
- document body (browser viewport) when there is no parent element.

Note:

Page scrolling does affect this position.
This position element is completely removed from normal document flow.

Example:
```
.p3 {
   width:50px;
   height:50px;
   background-color:lightgreen;
   position:absolute;
   left:60px;
   top:20px;
   }
```

4. Fixed Position

This position places the element at a fixed place relative to the viewport. Page scrolling does not affect this position.

Note: This position element is completely removed from normal document flow.

Example:
```
.p1 {
   width:50px;
   height:50px;
   background-color:lightgreen;
   position:fixed;
   left:40px;
   top:40px;
   }
```

5. Sticky Position:

Sticky position is a combination of relative and fixed. It is relative until it crosses a specified threshold, at which point it is treated as static position. The elements gets stick to the edges of the viewport.

CSS Text Effects

These are the properties that can be used for adding the effects on text. Using CSS, we can style the web documents and affect the text. The properties of the text effect help us to make the text attractive and clear.

There are some text effect properties in CSS that are listed below

- word-break
- text-overflow
- word-wrap
- Writing-mode

Word-break:

It specifies how words should break at the end of the line. It defines the line break rules.

Syntax: word-break: normal |keep-all | break-all ;

The default value of this property is normal.

- keep-all: It breaks the word in the default style.
- break-all: It inserts the word break between the characters in order to prevent the word overflow.

word-wrap
CSS word-wrap property is used to break the long words and wrap onto the next line. This property is used to prevent overflow when an unbreakable string is too long to fit in the containing box.

word-wrap: normal| break-word

- normal: This property is used to break words only at allowed break points.
- break-word: It is used to break unbreakable words.

Text-overflow

It specifies the representation of overflowed text, which is not visible to the user. It signals the user about the content that is not visible. This property helps us to decide whether the text should be clipped or show some dots (ellipsis).

This property does not work on its own. We have to use white-space: nowrap; and overflow: hidden; with this property.

text-overflow: clip | ellipsis;

- clip: It is the default value that clips the overflowed text.
- ellipsis: This value displays an ellipsis (…) or three dots to show the clipped text. It is displayed within the area, decreasing the amount of text.

Writing-mode

It specifies whether the text will be written in the horizontal or vertical direction. If the writing direction is vertical, then it can be from left to right (vertical-lr) or from right to left (vertical-rl).

writing-mode: horizontal-tb | vertical-lr | vertical-rl |

- horizontal-tb: It is the default value of this property in which the text is in the horizontal direction and read from left to right and top to bottom.
- vertical-rl: It displays the text in the vertical direction, and the text is read from right to left and top to bottom.
- vertical-lr: It is similar to vertical-rl, but the text is read from left to right.

CSS Gradients

CSS gradient is used to display smooth transitions within two or more specified colors.

CSS Gradient Advantages

- The download time and bandwidth usage can also be reduced.
- It provides a better look to the element when zoomed, because the gradient is generated by the browser.

There are 3 types:

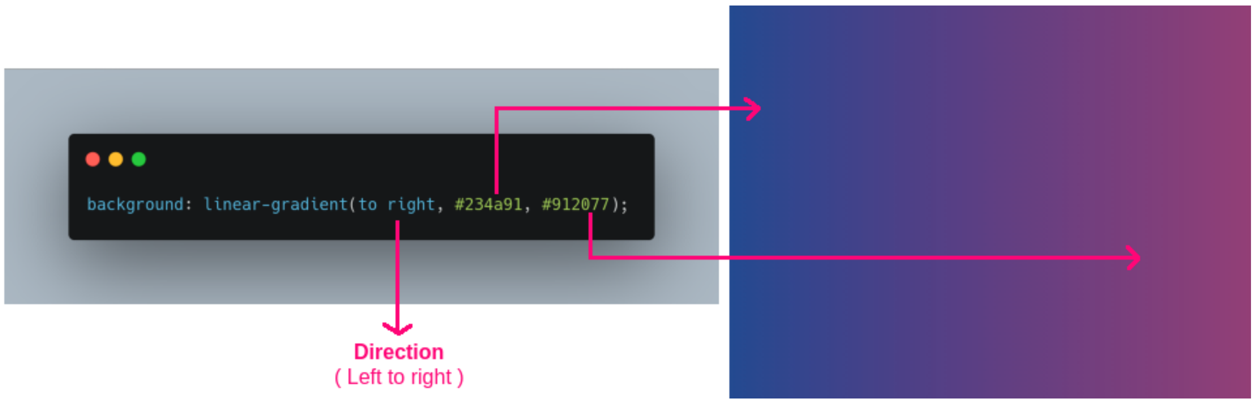
1.Linear Gradients:

The most common and useful type of gradient is the linear-gradient(). It creates an image consisting of a progressive transition between two or more colors along a straight line.

The gradients "axis" can go from left-to-right, top-to-bottom, or at any angle you chose.

Declarations: These are various ways to declare linear gradients

- linear-gradient(45deg, blue, red); //A gradient tilted 45 degrees, starting blue and finishing red
- linear-gradient(to left top, blue, red);
- linear-gradient(0deg, blue, green 40%, red); //A gradient going from the bottom to top,starting blue, turning green at 40% of its length, and finishing red */



```
background: linear-gradient(to right, #234a91, #912077);
```

2. Radial gradient

The radial-gradient() function sets a radial gradient as the background image. A radial gradient is defined by its center.

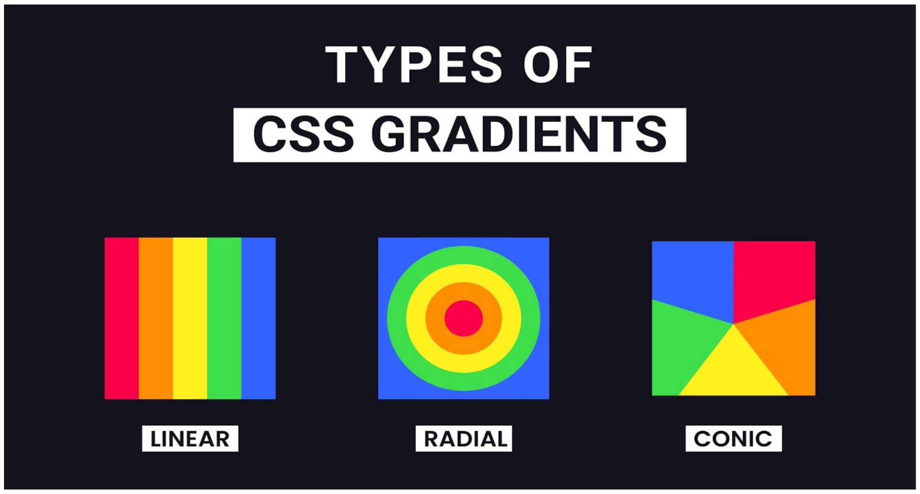
To create a radial gradient you must define at least two color stops.

Syntax:
radial-gradient(circle at center, red, blue, green 100   // A gradient at the center of its container, starting red, changing to blue, and finishing green

Conic Gradient:

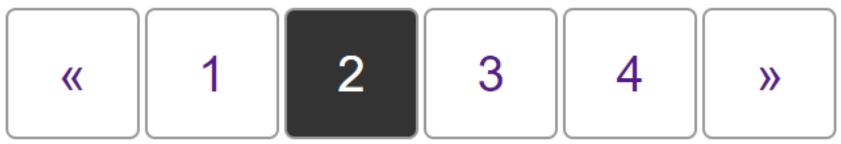
A conic gradient is similar to a radial gradient. Both are circular and use the center of the element as the source point for color stops. However, where the color stops of a radial gradient emerge from the center of the circle, a conic gradient places them around the circle.

Syntax: .conic-gradient {
  background: conic-gradient(#fff, #000);
}



CSS Pagination

This cookbook pattern demonstrates the navigation pattern used to display pagination, where the user can move between pages of content



Example:

```
<head>
<style>
.pagination {
  display: inline-block;
}

.pagination a {
  color: black;
  float: left;
  padding: 8px 16px;
  text-decoration: none;
}
</style>
</head>
<body>

<h2>Simple Pagination</h2>

<div class="pagination">
  <a href="#">&laquo;</a>
  <a href="#">1</a>
  <a href="#">2</a>
  <a href="#">3</a>
  <a href="#">4</a>
  <a href="#">5</a>
  <a href="#">6</a>
  <a href="#">&raquo;</a>
</div>

</body>
</html>
```

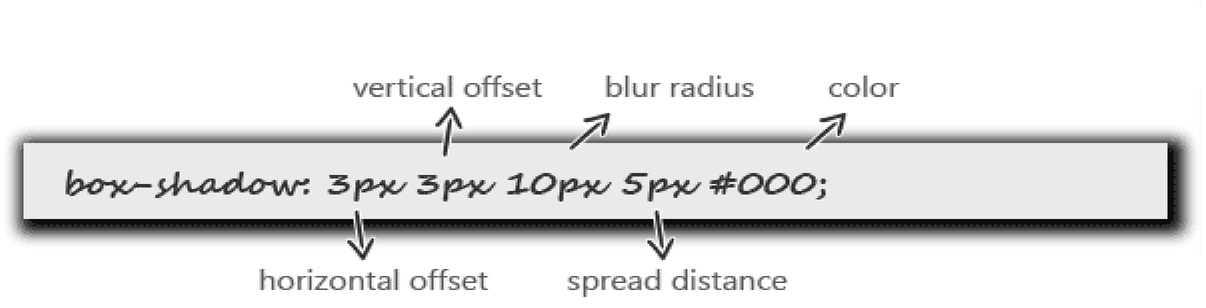
CSS Shadows:

They are two types of shadows that can be applied through CSS.

- Text-Shadow
- Box-Shadow

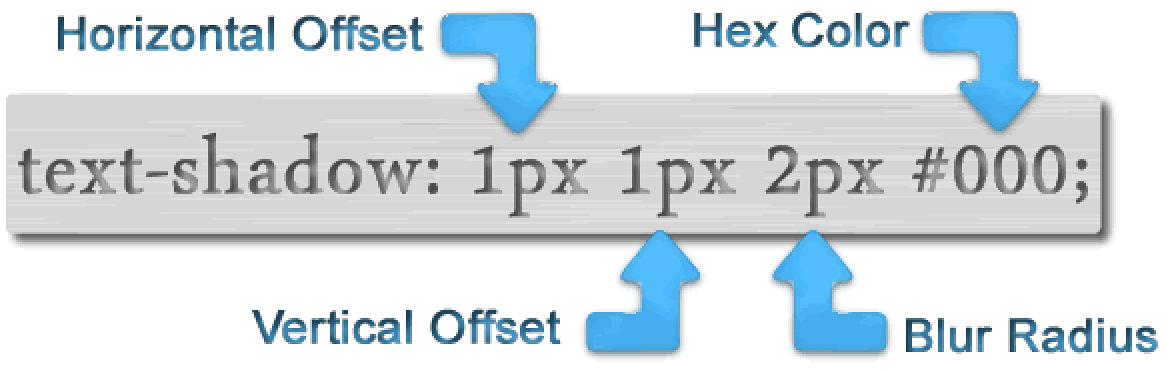
Box-Shadow:

The box-shadow property attaches one or more shadows to an element.



Text-Shadow:

The text-shadow property adds shadow to text.



CSS Flexbox

The main idea behind the flex box layout is to give the container the ability to alter its items' width/height (and order) to best fill the available space (mostly to accommodate to all kind of display devices and screen sizes). A flex container expands items to fill available free space or shrinks them to prevent overflow.

Syntax:

```
.container {
  display: flex; /* or inline-flex */
}
```

Flex box Properties:

Flex box allows users to design a flexible layout with the help of these following properties.

1. Flex Direction: This establishes the main-axis, thus defining the direction flex items are placed in the flex container.

Syntax:
```
.container {
  flex-direction: row | row-reverse | column | column-reverse;
}
```

- row (default): left to right in ltr; right to left in rtl
- row-reverse: right to left in ltr; left to right in rtl
- column: same as row but top to bottom
- column-reverse: same as row-reverse but bottom to top

2. Flex-Wrap: Flex items will all try to fit onto one line. You can change that and allow the items to wrap as needed with this property.

```
.container {
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```

- nowrap (default): all flex items will be on one line
- wrap: flex items will wrap onto multiple lines, from top to bottom.
- wrap-reverse: flex items will wrap onto multiple lines from bottom to top.

3. Justify-Content:

It helps distribute extra free space leftover when either all the flex items on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

```
.container {
  justify-content: flex-start | flex-end | center | space-between | space-around |
space-evenly | start | end | left | right .}
```

- flex-start (default): items are packed toward the start of the flex-direction.
- flex-end: items are packed toward the end of the flex-direction.
- start: items are packed toward the start of the writing-mode direction.
- end: items are packed toward the end of the writing-mode direction.
- left: items are packed toward left edge of the container, unless that doesn't make sense with the flex-direction, then it behaves like start.
- right: items are packed toward right edge of the container, unless that doesn't make sense with the flex-direction, then it behaves like start.
- center: items are centered along the line
- space-between: items are evenly distributed in the line; first item is on the start line, last item on the end line
- space-around: items are evenly distributed in the line with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies.
- space-evenly: items are distributed so that the spacing between any two items (and the space to the edges) is equal.

4. Align-items

This defines the default behavior for how flex items are laid out along the cross axis on the current line.

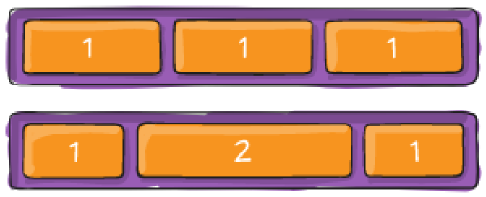
```
.container {
  align-items: stretch | flex-start | flex-end | center |
}
```

- stretch (default): stretch to fill the container (still respect min-width/max-width)

- flex-start / start / self-start: items are placed at the start of the cross axis. The difference between these is subtle, and is about respecting the flex-direction rules or the writing-mode rules.
- flex-end / end / self-end: items are placed at the end of the cross axis. The difference again is subtle and is about respecting flex-direction rules vs. writing-mode rules.
- center: items are centered in the cross-axis

5. Flex Grow: This defines the ability for a flex item to grow if necessary. It dictates what amount of the available space inside the flex container the item should take up.

```
.item { flex-grow: 4; /* default 0 */ }
```



6. Flex-Shrink: This defines the ability for a flex item to shrink if necessary.

```
.item {
  flex-shrink: 3; /* default 1 */
}
```

CSS Animation

CSS allows animation of HTML elements without using JavaScript or Flash!An animation lets an element gradually change from one style to another. You can change as many CSS properties you want, as many times as you want.

To use CSS animation, you must first specify some keyframes for the animation.

Keyframes hold what styles the element will have at certain times.

When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.

To get an animation to work, you must bind the animation to an element.

```
@keyframes example {
  from {background-color: red;}
  to {background-color: yellow;}
}

div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}
```

The **animation-duration** property defines how long an animation should take to complete. If the animation-duration property is not specified, no animation will occur, because the default value is 0s

The following example will change the background-color of the <div> element when the animation is 25% complete, 50% complete, and again when the animation is 100% complete:

```
@keyframes example {
  0%   {background-color: red;}
  25%  {background-color: yellow;}
  50%  {background-color: blue;}
  100% {background-color: green;}
}
div {
  background-color: red;
  animation-name: example;
  animation-duration: 4s;}
```

Introduction to Responsive Web Design:

Responsive web design makes your web page look good on all devices.

Viewport

The viewport is the user's visible area of a web page.The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.

Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size.

Introduction to Media Queries

Media queries allow you to customize the presentation of your web pages for a specific range of devices like mobile phones, tablets, desktops, etc. without any change in markups.

A media query consists of a media type and zero or more expressions that match the type and conditions of a particular media features such as device width or screen resolution.

```
@media screen and (min-width: 320px) and (max-width: 480px){
    /* styles */
}
/* Smartphones (portrait) ---------- */
@media screen and (max-width: 320px){
    /* styles */
}
/* Smartphones (landscape) ---------- */
@media screen and (min-width: 321px){
```

Example:

```
@media only screen and (max-width: 480px) {
  /* CSS rules to apply /*
}
```

For example: This media query will look for screens ("only screen" as written) with a max-width of 480px. If it finds one, the conditions will be executed, and the changes will be made to the HTML code.

Responsive Web Page Design using Media Queries

In the parenthesis, we add the conditions for the browser windows such as what is the minimum/maximum size you want the browser window to be for the conditions in the "body" to appear.

Example: Consider the following code:

```
<!DOCTYPE html>
<html>

<head>
        <style>
                body {
                        background-color: blue;
                }

                @media only screen and (min-width: 500px)
                                and (max-width: 700px) {
                        body {
                                background-color: red;
                        }
                }
        </style>
</head>

<body>
```

```
<p>
            On devices with minimum width of 500px
            and maximum width of 700px, the background
            color will be black
      </p>



<p>
            On the other hand, devices with less than the
            minimum width of 500px will have the
            body be displayed in blue
      </p>

</body>

</html>
```

Media queries are very useful when you want a specific code to only execute if the conditions regarding the window's size are met. For example, if you would like the background color of the page to be black on larger devices but blue on smaller devices, then media queries are the best way to handle such cases.

Introduction to CSS Grid

CSS Grid Layout is a two-dimensional layout system for the web.

A grid is a collection of horizontal and vertical lines creating a pattern against which we can line up our design elements. They help us to create layouts in which our elements won't jump around or change width.

To define a grid we use the grid value of the display property.

```
.container { display: grid; }
```

To see something that looks more grid-like, we'll need to add some columns to the grid. Let's add three 200-pixel columns.

```
container { display: grid; grid-template-columns: 200px 200px 200px; }
```

Flexible grids with the fr unit

In addition to creating grids using lengths and percentages, we can use the fr unit to flexibly size grid rows and columns. This unit represents one fraction of the available space in the grid container.

```
.container {
display: grid;
grid-template-columns: 1fr 1fr 1fr; }
```

Gaps between tracks

To create gaps between tracks we use the properties column-gap for gaps between columns, row-gap for gaps between rows, and gap as a shorthand for both.

```
.container {
 display: grid;
 grid-template-columns: 2fr 1fr 1fr;
 gap: 20px;
 }
```

Repeating track listings

You can repeat all or merely a section of your track listing using the CSS repeat() function. Change your track listing to the following:

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 20px;
}
```

Grid Templates

The grid-template property is a shorthand property for the following properties:

- grid-template-rows
- grid-template-columns
- Grid-template-areas

Syntax:

```
.item1 {
  grid-area: myArea;
}
.grid-container {
  display: grid;
  grid-template:
      'myArea myArea . . .'
      'myArea myArea . . .';
}
```

Grid-template rows:

The grid-template-rows property specifies the number (and the heights) of the rows in a grid layout. The values are a space-separated list, where each value specifies the height of the respective row.

```
.grid-container {
  display: grid;
  grid-template-rows: 100px 300px;
}
```

Grid-template Columns

The grid-template-columns property specifies the number (and the widths) of columns in a grid layout.

The values are a space separated list, where each value specifies the size of the respective column**.**

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto;}
```

Grid template Area

The grid-template-areas property specifies areas within the grid layout.

```
grid-template-areas: none|itemnames;
```

Model Questions

1. Explain various CSS Positions used in order to place an element in an HTML document
2. Differentiate between Absolute and Relative Positions in CSS
3. Explain about a) CSS Rounded Corners b) CSS Gradients c) CSS Shadows in detail
4. What are different text effects that can be applied in CSS? Explain
5. Explain about CSS Flex Box in detail
6. Explain about Keyframes in CSS Animation in detail
7. What is responsive web design? Explain in detail about viewport
8. What are media queries? Explain their significance in CSS in detail
9. Explain about CSS Grid in detail.
10. Difference between CSS Grid vs CSS Flex BOX