Software - Interview Prep Document

At Cowlar, we believe our work should be a blend of hard work, innovation & excellence. This requires us to constantly strive to make ourselves better at what we do. This also makes it necessary for our hiring philosophy and process to be unique from what you're probably expecting.

Please read this entire document carefully. Please use the materials below to prepare for the interview.

Once you have completed your preparation and feel like you are ready, please email us with three preferred dates (s) of availability. We try to schedule the interview on your preferred dates[1]. The venue for the interview/ test is our office located (**here**) in Islamabad. Our requirements, interview process & hiring philosophy are unique. It's important that you review the information carefully and decide if you want to work here.

General Requirements

- Hardworking, decent & honest
- Intelligent
- Should have the ability to assimilate complex ideas
- MUST be able to document your work and develop clear and easy to understand documents
- MUST understand how to create loom videos
- Recently graduating candidates MUST have the ability to ***learn by doing*** [1] [2] using online learning methods/tutorials
- Experienced candidates MUST have excellent communication skills & understand how to train, manage & help less experienced colleagues grow
- Must have the ability to follow instructions[2]
- Must be a team player [1] [2]
- Understand good work ethic [1]
- Must be a good fit into our team culture
- Must value your time & our time [1]
- Should treat everyone [3] with respect

[1] Please provide three (03) dates that work for you in order of preference. We try to schedule on your first choice date unless it's already booked.

[2] Please follow instructions provided to you during the interview, test and training phase. We don't like asking twice. This is especially important for fresh graduates.

[3] Everyone means everyone, including our support & cleaning staff

- Must know how to use the internet [4]
- Must have clear core concepts
- Excellent Programming & algorithm development skills
- Must have the ability to learn independently (without being asked)
- Must be good at reading documentation
- Experienced candidates must be good at writing documentation
- Must be able to summarize documents/tutorials/videos & teach core concepts to other team members
- Must be passionate
- Must enjoy work
- Must be ambitious & motivated

Hiring Process / Philosophy

Instead of going over multiple rounds of interviews (analytical, technical, HR, etc) with a candidate over a period of time, we believe it's best to do one long test /interview where the candidate spends time in our office answering questions & solving problems. You're welcome to ask us any questions ahead of time. You can also ask as many questions as you want **after** the interview.

Based on how the interview goes we may make an offer right away or get back to you in a couple of days. Alternatively, some candidates are placed on a wait-list and called when there is a requirement generated from our team.

For fresh graduates, we have some initial training requirements that need to be met before the selected candidate can begin work at our startup. The orientation part is usually a couple of weeks. You will typically spend 75% of your time going through the training program (learning the important things that will allow you to start contributing towards the actual work we do). The remaining 25% time will be spent on implementing what you learned (by doing small tasks/projects). As time passes you'll spend more time doing and less time learning.

This means that learning is an essential part of everyone's job. Not learning is *not an option here*. Everyone at our company typically spends about 25% of their time learning new things and working on cutting edge (sometimes bleeding edge tech). Our employees are expected to always be comfortable with learning new things, solving complex problems, and working as part of a team so we can deliver true value to our customers.

We typically have a probation period of three to six months. We offer competitive salary packages and offer opportunities for upward mobility within the organization. Our salary packages are based

[4] We don't expect you to know the answer to every question but we do expect you to know where to look. You should know how to google solutions to problems and use stackoverflow / online user forums / online community resources / discussion groups to your advantage. We expect you to know how and where to search for solutions to problems that you don't understand. We expect that before you ask a colleague, you will have tried your best to search for a solution online (by yourself)

on employee skills, learning ability, contribution to successful projects and how well you fit into the team culture.

Reasons to work here

- Opportunity to Learn from an exceptionally talented core team
- Our management style, processes, policies & team skills are constantly in the process of improving. Joining us will help you become part of a great team
- Access to purpose-built in-house specialized training modules to constantly make you better
- Access to paid online courses to accelerate growth & inspire you to build great products
- Freedom to work with core-team to choose an area of specialization over time that you excel in & that is mutually beneficial to you & the team. We want our people to love at least 90% of their work & the tasks they do.
- Opportunity to work in a conducive and comfortable work environment (We take care of our people, treat them with respect & expect the same in return)
- Free Meals & Snacks (Breakfast, Lunch, Dinner & unlimited Snacks). We have a Full-time dedicated Chef with a Full Kitchen in our office that cooks and serves fresh, clean meals for the entire team. Free Tea, Coffee, Snacks are available completely free of cost.
- Multiple opportunities for rapid professional & personal growth
- A Competitive Salary Package
- Lots of opportunities of upward mobility within our startup by demonstrating you can execute
- Opportunities to get multiple salary increments per year based on clearly defined goals / performance / contributions to team's core goals.
- The founders of Cowlar strive to be good leaders & lead by example [1]

How to Prepare for the Test

1. Prepare from the materials in this document. There are many links to useful resources. It is not expected that you will know everything on day one [5]. We're just looking for people who can show the willingness to invest some time improving their skills before they come in for an interview. This is a win-win scenario for you (the candidate). Preparing for the interview/test can seem challenging at first. But if you dedicate some time and effort into learning and executing then it dramatically improves your chances of being hired asap. In the unlikely event you do not get an offer from us, you would still have learned some skills which can help you in future interviews and set you on a path of rapid career development. Our test is **not easy**. This is because we are looking for the best people. We're not looking for button pushers, clock watchers, or people who come to work as a compulsion / necessary 2.

[5] Please note. If you're an experienced developer, the threshold of what you should know will be different. We don't care about the number of years you've worked in the Pakistani software services industry, we want to see what happens when the rubber hits the road. Everyone says they're good. Please show us, don't tell us.

3. chore. We are looking for talented and passionate people who love their field of work and enjoy solving difficult and challenging problems.

4. **It is perfectly fine to reschedule your interview if you think you need more time to prepare.** There is no negative marking for delaying or rescheduling the interview date. You can reschedule as many times as you want [6] .That's perfectly ok from our point of view. It is NOT OK to show up unprepared and waste your & our time. People who show up unprepared are quickly offered a cup of tea & their interview ends in no time. We only spend time interviewing people if we think there is a chance we can hire them. We generally only interview candidates ONCE.

5. You can use the internet during the interview. There is no negative marking for using the internet during the interview. You can also access this document (or any bookmarks, code snippets, previous projects during your test for reference materials. You can bring as many cheat sheets with you as you want.

6. It is generally recommended that you bring your own laptop so it's installed with your preferred software/ code editors etc. If you cannot bring your own device, one will be provided to you.

7. As you watch the videos in the test materials section below, increase the playback speed to **1.25x. Then as you get comfortable increase speed to 1.5x and eventually to 2x speed**. Eventually, you'll be able to watch a 40 minute lecture in 20 minutes without a problem.

8. Look at the [1] level links for each section before diving in. Sometimes this will not be the case, you need to realise when you need to abort / close a link and move on to the next link.

9. During the interview and demo portion at our office. You can use two life lines.

   (A) **Can use the internet.** If you're experienced, you should have a decent hold of syntax (you should not have to look at a website in order to write a for loop)

   (B) **Phone a Friend/Person** (You are allowed to call anyone you want (you will have 5 minutes to have a conversation over the phone)

10. Please use online resources to revise and practice your core concepts. Here is a little secret. If you follow it, we will hire you. Worse case scenario is you become better skilled at what you do. The secret is to work hard and spend the necessary time learning. We are telling you in advance what questions we will ask and where/ how to prepare for them.

11. You need to get comfortable with learning online and offline. One of the benefits of working at Cowlar is having paid resources and training materials available to you. If we hire you we will have purchased more

[6] If you want to reschedule your interview on the same day it was supposed to take place, you'll need to give us four hour's advance notice. This helps us differentiate from people who want to reschedule & those who don't show up. **2nd**

than 100+ paid courses / online tutorials from sites like Udemy, VueMastery, VueSchool, SkillShare, Pluralsight etc. so our team has the best conducive environment for learning. We also arrange to provide webinars that might be useful for your professional growth.

Test Preparation Materials

Please note. As we receive hundreds of applications for each position, ultimately it might come down to who can answer/demo the most skills (practically) from the questions/topics below.

Please do not ask to be interviewed if you have not prepared from the materials below

TABLE OF CONTENTS

6	Misc
7	Flutter (Bonus)

Floor, Plot # 5-A EOBI Building, Crown Plaza, F-7 Markaz, Islamabad, Pakistan
info@cowlar.com +92 51 8317562

COWLAR

JavaScript

Take this JS crash course to revise.

Your interview questions will be from the following links

[0] [1] [2] [3] [4] [5] [6]

1. What is the Event Loop [1]
2. How would you make a copy of an array [1]
3. JSON, XML & AJAX [1] [2] [3] [4]
4. DOM [1] [2]
5. Push, Pop, Shift, Unshift
6. Closure in JS [1] [2]

Functions in JS can access variables outside the function body which comes within the scope. Since JS uses lexical scoping which allows visibility of variables and their positions at the time of declaration. Closures is that we want inner function to remember the state of inner variables so they can be effectively used afterwards. Following is the example:

```
function outer() {
  let outerVar = "I'm from the outer function";

  function inner() {
    console.log(outerVar);  // Inner function has access to outerVar
  }

  return inner;
}

const closureFunc = outer();  // outer() returns inner function
closureFunc();  // Output: "I'm from the outer function"
```

7. Array manipulation [1] [2] [3]
8. Higher order functions on arrays (map for JS) [1] [2]

Higher order functions on arrays are that take one or more functions as arguments or return a function. Methods like map or reduce are the best example. For example:

Array.map(number => number*2) it taking a function and returning array with function applied so we not have to call function manually on all the elements of array separately.

9. Higher order functions - Filter & reduce method [1] [2] [3]

The filter() method creates a new array with all elements that pass the test implemented by the provided function. filter is a higher order function.

The reduce() method applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value. reduce is a higher order function.

```
let numbers = [1, 2, 3, 4, 5];

let sum = numbers.reduce((sum, number, index, array) => {
console.log(number, 'is the', index, 'in the array:', array);
  sum += number;
  return sum;
}, 0);

console.log(sum);
```

10. Functional programming [1]
11. s
12. Parsing strings [1] [2] [3]
13. Loading JSON data from a URL (Async Callbacks) [1]

For loading JSON data, we simply create a callback function like this loadJSON(url, function(getData){}) so in this way we can get data from a http url.

The loadJSON() method is a function used in JavaScript, particularly in the **p5.js** library, to load a JSON file or URL asynchronously. It retrieves JSON data and makes it available for use in your application, allowing you to load data such as configurations, game levels, or any structured information.

```
loadJSON(path, [callback], [datatype])
```

14. Axios [1] [2]

We can limit number of results which we want from api by writing {params: {_limit: 3}} alongside with the url in the axios.

Axios.spread((name_of_fields_available_in_that_api)=>{showOutput(name)}) so we donot have to do this res[0] or res[1] to get specific results.

Intercepting requests and responses in a browser essentially allows developers to monitor, modify, or control HTTP requests and responses between the client (browser) and the server. This capability can be used for various purposes such as logging, altering requests before they are sent, handling errors, adding custom headers, retrying requests, etc.

```
axios.interceptors.request.use(
  function (config) {
    // Do something before sending the request
    // E.g., Attach an authorization token to every request
    config.headers['Authorization'] = 'Bearer token';
    console.log('Request:', config);
```

```
    return config;
  },
  function (error) {
    // Handle the error if the request fails to be sent
    return Promise.reject(error);
  }
);
```

We can also transform our responses before sending using

```
// TRANSFORMING REQUESTS & RESPONSES
function transformResponse() {
  const options = {
    method: 'post',
    url: 'https://jsonplaceholder.typicode.com/todos',
    data: {
      title: 'Hello World'
    },
    transformResponse: axios.defaults.transformResponse.concat(data => {
      data.title = data.title.toUpperCase()
    })
  }

  axios(options).then(res => showOutput(res))
}
```

And in case we want the default globals for all the urls which we sent through axios we can use this:

```
// AXIOS GLOBALS
axios.defaults.headers.common['X-Auth-Token'] =
  'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
  eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
  SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c';
```

If we want to cancel the request we can do it using this approach:

```
// CANCEL TOKEN
function cancelToken() {
  const source = axios.CancelToken.source();

  axios
    .get('https://jsonplaceholder.typicode.com/todos', {
      cancelToken: source.token
    })
    .then(res => showOutput(res))
    .catch(thrown => {
      if (axios.isCancel(thrown)) {
        console.log('Request canceled', thrown.message);
      }
    });

    if(true) {
      source.cancel();
    }
}
```

We can also create an axios instance using

Const axiosInstance = axios.create({
baseUrl: 'http://localhost:3000'
})
axiosInstance.get('/comments')

We can also do the timeout using {timeout: n-ms} so request does not take more than this time.

15. Sockets [1]

WebSockets allow two-way event-driven communication between web browser and the server. Firstly the handshaking is done that is done by initially sending and http request for hand-shaking and then it upgrades to WebSocket protocol. This is how to setup a basic socket server:

// server.js

const WebSocket = require('ws');

// Create a WebSocket server at port 8080
const wss = new WebSocket.Server({ port: 8080 });

wss.on('connection', (ws) => {
console.log('A new client connected!');

```
  // Send a welcome message to the client
  ws.send('Welcome to the WebSocket server!');

  // Handle messages received from the client
  ws.on('message', (message) => {
    console.log(`Received from client: ${message}`);
    ws.send(`Server echo: ${message}`);
  });

  // Handle client disconnects
  ws.on('close', () => {
    console.log('Client has disconnected');
  });
});

console.log('WebSocket server is running on ws://localhost:8080');
```

And this is the client side code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>WebSocket Demo</title>
</head>
<body>
  <h1>WebSocket Client</h1>
  <input type="text" id="messageInput" placeholder="Type a message..." />
  <button id="sendButton">Send Message</button>
  <div id="output"></div>

  <script>
    // Create a new WebSocket connection to the server
    const socket = new WebSocket('ws://localhost:8080');

    // Handle when the connection is established
```

```
  socket.onopen = function () {
    console.log('Connected to the server');
    document.getElementById('output').innerHTML += '<p>Connected to the server</p>';
  };

  // Handle incoming messages from the server
  socket.onmessage = function (event) {
    console.log('Received from server:', event.data);
    document.getElementById('output').innerHTML += `<p>Server: ${event.data}</p>`;
  };

  // Handle errors
  socket.onerror = function (error) {
    console.log('WebSocket error:', error);
  };

  // Handle when the connection is closed
  socket.onclose = function () {
    console.log('Connection closed');
    document.getElementById('output').innerHTML += '<p>Connection closed</p>';
  };

  // Send a message to the server when the button is clicked
  document.getElementById('sendButton').addEventListener('click', () => {
    const message = document.getElementById('messageInput').value;
    socket.send(message);
    document.getElementById('output').innerHTML += `<p>You: ${message}</p>`;
  });
 </script>
</body>
</html>
```

16. Can I use Javascript to detect if there is an internet connection?

Use navigator.online method to check if the current internet connection is available or not. Following is the code:

```
<script>
  function updateNetworkStatus() {
    const status = document.getElementById('status');
```

```
  if (navigator.onLine) {
    status.textContent = 'You are online!';
    status.style.color = 'green';
  } else {
    status.textContent = 'You are offline.';
    status.style.color = 'red';
  }
}

// Run on page load
updateNetworkStatus();

// Listen for online/offline events
window.addEventListener('online', updateNetworkStatus);
window.addEventListener('offline', updateNetworkStatus);
</script>
```

17. When a user is logged in, How would you save their authentication information?

By using either JWT stored in localStorage, sessions-storage or cookies or session-based authentication (traditional-creating session at backend and then sending session Id to frontend) or using OAuth or OpenID authentication methods (3rd party apis) but keep in mind to check for these protections, XSS protection, CSRF protection and token expiration.

18. What can sockets be used for?

Sockets are mainly used for realtime things like RT chat application, online gaming, RT notifications, Live data streaming, RT collaboration, File transfer, RT Dashboards, Remote Control management, Collaborative Environment and WebRTC(for video chatting applications.)

19. How to get the value of a certain key from a Javascript object? [1]

Use Object.values(array) and Object.keys(array). If you want to obtain individual, either use for loop, or you can use map of Map method like,

```
Const map = new Map()
Map.set(keys[i], values[i])
```

20. Decoding encoded bit data from a variable

In case data is encoded in simple binary, then we can use operators like &,|,<<,>> at our disposal.

```
let encodedData = 0b11010101; // binary for 213 in decimal
```

```
// Extract specific bits using bitwise operations

// Get the last 4 bits (rightmost 4 bits)
let last4Bits = encodedData & 0b1111; // 0b0101 or 5 in decimal
console.log(last4Bits); // Output: 5

// Get the first 4 bits (leftmost 4 bits)
let first4Bits = (encodedData >> 4) & 0b1111; // 0b1101 or 13 in decimal
console.log(first4Bits); // Output: 13

// Check if the second bit (from right) is set
let secondBit = (encodedData >> 1) & 1; // 1 if set, 0 if not
console.log(secondBit); // Output: 0
```

Now in case, encoding is done in base64, then use methods **atob()** for base64 to original or **btoa()** for reverse.
And in case you are representing single bit flags, then use this method:

```
const READ = 0b0001;  // 1st bit
const WRITE = 0b0010; // 2nd bit
const EXECUTE = 0b0100; // 3rd bit
const DELETE = 0b1000;  // 4th bit

// Suppose the user has permissions encoded as:
let permissions = 0b0110; // User has WRITE and EXECUTE permissions

// Decode the permissions
let canRead = (permissions & READ) === READ;
let canWrite = (permissions & WRITE) === WRITE;
let canExecute = (permissions & EXECUTE) === EXECUTE;
let canDelete = (permissions & DELETE) === DELETE;

console.log(canRead);    // Output: false
console.log(canWrite);   // Output: true
console.log(canExecute); // Output: true
console.log(canDelete);  // Output: false
```

21. How would you remove duplicate values from an array [1] [2]

```
let a = [1, 2, 5, 2, 1, 8];

obj = {};

for(let i of a){
  obj[i] = true;
}

let b = Object.keys(obj);

console.log(b)
```

22. What is the ***'this'*** keyword in JavaScript?

The 'this' keyword in JavaScript refers to the context in which the current code is executing. Its value depends on how and where it's used. **call()**: Calls a function with a specified this context and arguments. **apply()**: Similar to call(), but arguments are passed as an array. **bind()**: Creates a new function where this is permanently set to the provided value.

23. What are exceptions? [1]

Simply remember **try, catch, throw and finally,** keywords.

24. How are exceptions handled in JavaScript? [1] [2] [3] [4]

In case of promises, it is better if the functions in which promises are dealt should not do anything rather they should return promises and then results should be dealt in another function so we can show errors to users as well.

```
throw new EvalError()
// Represents an error that occurs regarding the global function eval().

throw new InternalError()
// Represents an error that occurs when an internal error in the JavaScript engine is thrown. E.g. "too much recursion".

throw new RangeError()
// Represents an error that occurs when a numeric variable or parameter is outside of its valid range.

throw new ReferenceError()
// Represents an error that occurs when de-referencing an invalid reference.

throw new SyntaxError()
// Represents a syntax error that occurs while parsing code in eval().

throw new TypeError()
// Represents an error that occurs when a variable or parameter is not of a valid type.

throw new URIError()
// Represents an error that occurs when encodeURI() or decodeURI() are passed invalid parameters.
```

25. Which is faster if/else statement or switch case (and why ?)

Basically, some JS engines render the switch statement as "Jump table" or the "Lookup Table".

Type of Conditions:

- If the conditions involve complex evaluations (e.g., checking multiple variables or using complex expressions), if/else might be more appropriate.
- If you're simply comparing one variable to multiple constant values, switch is often better.

26. Must know how to debug JavaScript [1]

In simple words, we can use chrome developer tools and in their, using the 'Source' tab, and their you got three sections, first listing the files, second the code and third the other code execution related options. Now, if we want some click listeners related code to be found as your bug triggered on clicking, then from the third pane, select 'Event Listener Breakpoints' and in their you can select relevant and the code will stop at the appropriate point regarding that listener. At some point if you stopped execution at some line, then by using original console of dev tools, which will now get access to all the variables which have been executed till now. From console you can correct any type related or other sort of errors.

27. LinkedList [1]

In this manner, you can create a linked list data structure,

```
// Construct Single Node
class Node {
  constructor(data, next = null) {
    this.data = data;
    this.next = next;
  }
}

// Create/Get/Remove Nodes From Linked List
class LinkedList {
  constructor() {
    this.head = null;
    this.size = 0;
  }

  // Insert first node
  insertFirst(data) {
    this.head = new Node(data, this.head);
    this.size++;
  }

  // Insert last node
  insertLast(data) {
    let node = new Node(data);
    let current;

    // If empty, make head
    if (!this.head) {
      this.head = node;
    } else {
      current = this.head;

      while (current.next) {
        current = current.next;
      }
```

```
    current.next = node;
  }

  this.size++;
 }

 // Insert at index
 insertAt(data, index) {
  //  If index is out of range
  if (index > 0 && index > this.size) {
    return;
  }

  // If first index
  if (index === 0) {
    this.insertFirst(data);
    return;
  }

  const node = new Node(data);
  let current, previous;

  // Set current to first
  current = this.head;
  let count = 0;

  while (count < index) {
    previous = current; // Node before index
    count++;
    current = current.next; // Node after index
  }

  node.next = current;
  previous.next = node;

  this.size++;
 }

 // Get at index
 getAt(index) {
  let current = this.head;
  let count = 0;

  while (current) {
    if (count == index) {
      console.log(current.data);
    }
    count++;
    current = current.next;
  }
```

```
    return null;
  }

  // Remove at index
  removeAt(index) {
    if (index > 0 && index > this.size) {
      return;
    }

    let current = this.head;
    let previous;
    let count = 0;

    // Remove first
    if (index === 0) {
      this.head = current.next;
    } else {
      while (count < index) {
        count++;
        previous = current;
        current = current.next;
      }

      previous.next = current.next;
    }

    this.size--;
  }

  // Clear list
  clearList() {
    this.head = null;
    this.size = 0;
  }

  // Print list data
  printListData() {
    let current = this.head;

    while (current) {
      console.log(current.data);
      current = current.next;
    }
  }
}

const ll = new LinkedList();

ll.insertFirst(100);
```

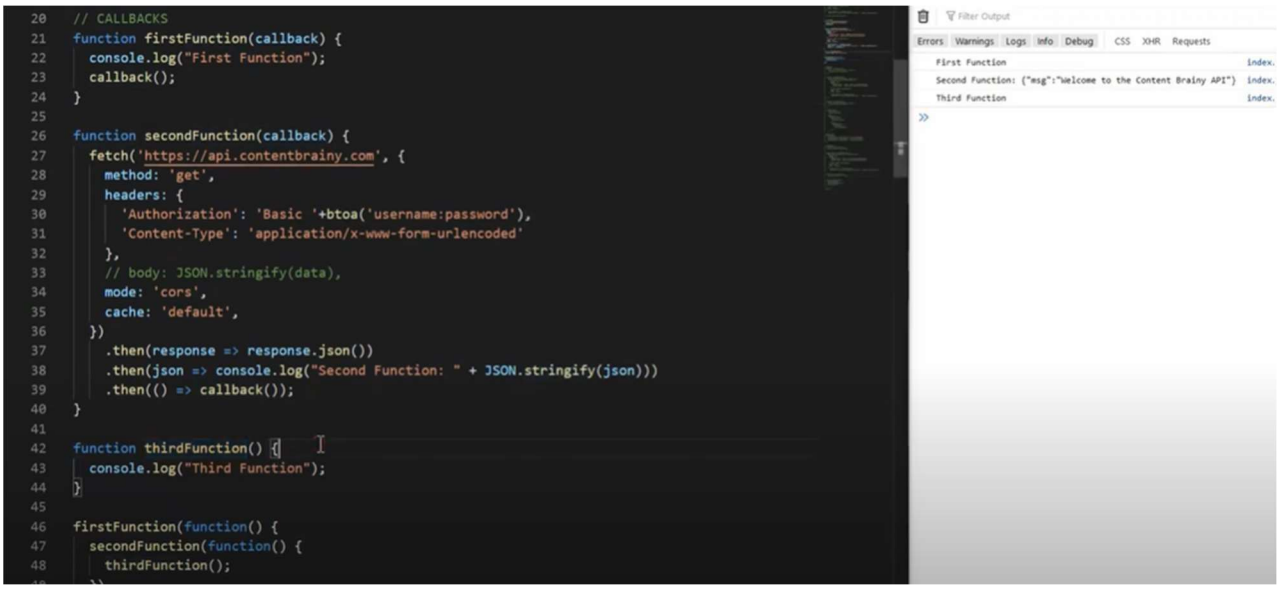
```
ll.insertFirst(200);
ll.insertFirst(300);
ll.insertLast(400);
ll.insertAt(500, 3);

// ll.clearList();
// ll.getAt(2);

ll.printListData();
```

28. Difference between Callbacks vs Promises vs Async/await? [1] [2]

JS is typically synchronous but can also be asynchronous with callbacks and promises and async/await mechanisms. In JS, functions are basically objects so functions can take functions as variables so in case we want to make the flow of program to be synchronous, we can make our fetch functions as functions taking callbacks as arguments which we ensure the laminar flow of the program.



But this approach is mostly error-prone. So we can use **promises at our disposal.**

```
// PROMISES
function firstFunction() {
  return new Promise((resolve, reject) => {
    resolve("First Function");
  })
}

function secondFunction() {
  return new Promise((resolve, reject) => {
    fetch('https://api.contentbrainy.com', {
      method: 'get',
      headers: {
        'Authorization': 'Basic '+btoa('username:password'),
        'Content-Type': 'application/x-www-form-urlencoded'
      },
      // body: JSON.stringify(data),
      mode: 'cors',
      cache: 'default',
    })
      .then(response => response.json())
      .then(json => resolve("Second Function: " + JSON.stringify(json)));
  });
}
```

```
firstFunction()
  .then((data) => {
    console.log(data)
  })
  .then(() => {
    secondFunction()
      .then((data) => {
        console.log(data);
      })
      .then(() => {
        thirdFunction()
          .then((data) => {
            console.log(data);
          })
      })
  })
```

But again the best option is to use **async/await methodology.**

```
async function main() {
  firstFunction();
  await secondFunction();
  thirdFunction();
}
```

Callback example: ***function fetchData(callback) { setTimeout(() => { callback('Data fetched with callback'); }, 1000); }***
fetchData((result) => { console.log(result); // Output: Data fetched with callback });

Promises Example:
```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Data fetched with promise');
    }, 1000);
  });
}

fetchData()
  .then((result) => {
    console.log(result);  // Output: Data fetched with promise
  })
  .catch((error) => {
    console.error(error);
  });
```

Async/Await Example:
```
function fetchData() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve('Data fetched with async/await');
    }, 1000);
  });
}

async function getData() {
  const result = await fetchData();
  console.log(result);  // Output: Data fetched with async/await
```

```
}

getData();
```

29. What are the drawbacks of Callbacks? [1] [2]

The main drawbacks are callback hells. It simply means that we got a promise inside another promise and this chain continues indefinitely which in terms of callbacks functions like setTimeout, the code become extravagant and unnecessary.

30. Promises

31. What is a drawback of Promises

A **Promise** in JavaScript is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value. It acts as a placeholder for data that is not available yet but will be resolved (fulfilled) or rejected in the future. But following are its drawbacks:

- **Chaining Can Be Hard to Read (Promise Hell)**.
- Error Handling is Still Tricky
- Multiple Promises Lead to Complexity
- Lack of synchronous like flow
- Promises can be eager.
- No easily cancelling promises.

32. Cache

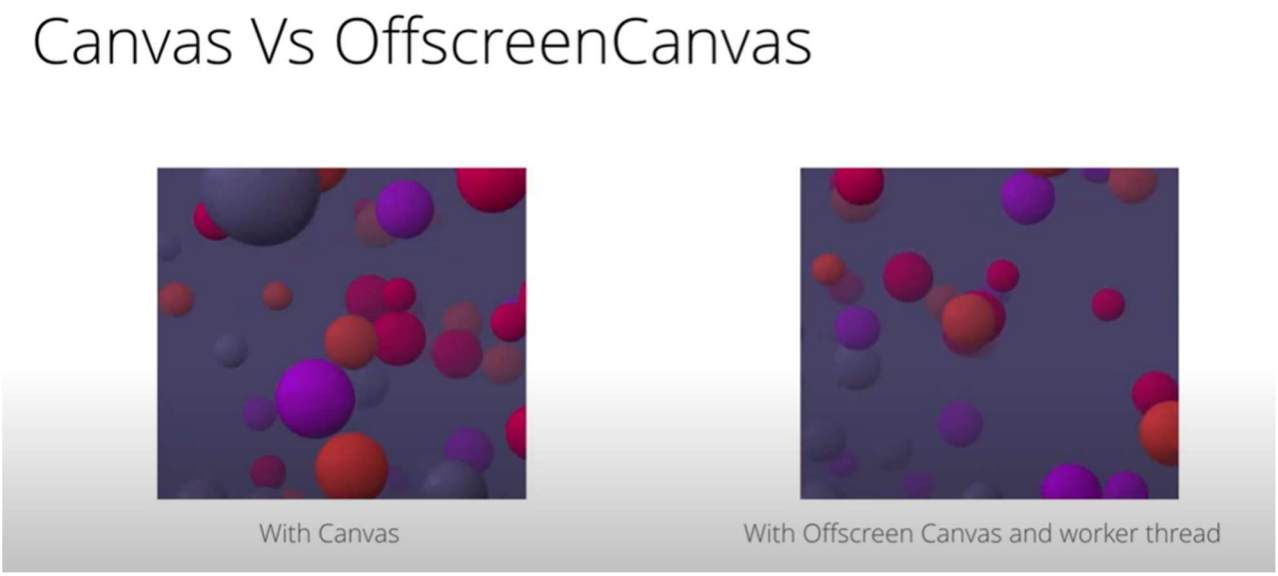
33. AMP [1]

AMP (Accelerated Mobile Pages) is an open-source framework developed by Google with the goal of improving mobile web performance. It allows web pages to load quickly on mobile devices by enforcing strict standards on HTML, JavaScript, and CSS.

And we can enforce amp-html tags and use their required properties by using <amp-html/> type tags.

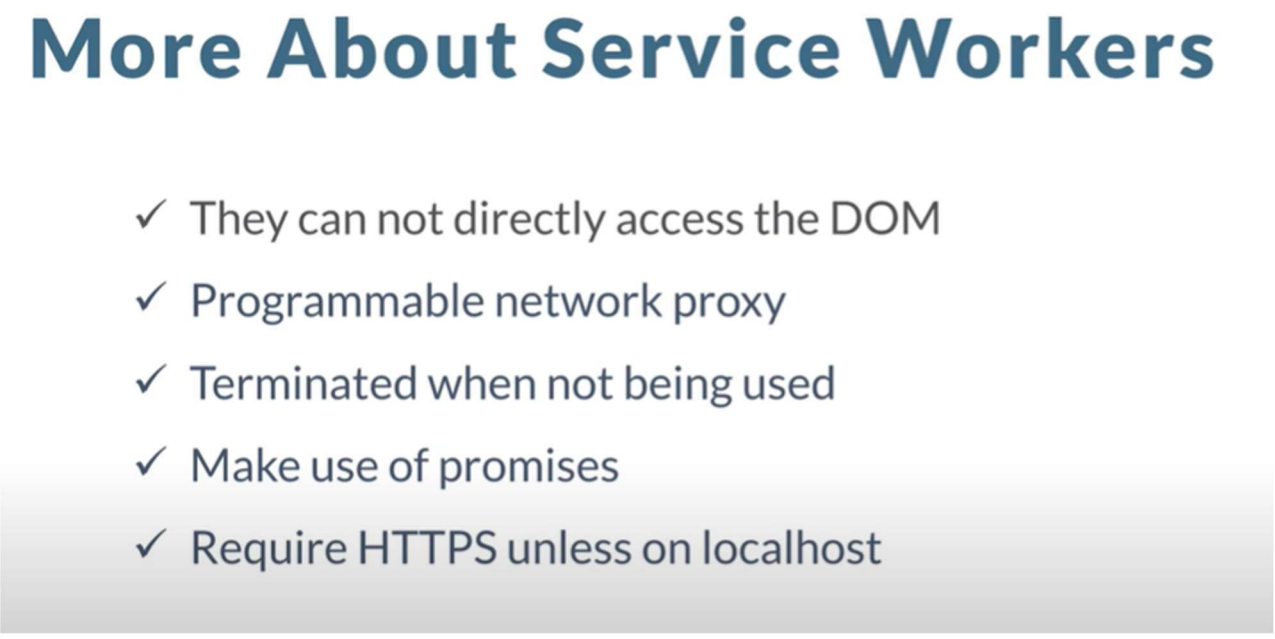
34. Offline Canvas in Chrome [1]

Canvas is simply a drawing API. Its only main drawback that it runs on main thread and not on the separate thread. So slow page performance. So in order to improve performance, we cna use **offscreen canvas** that we move logic of canvas to web-workers API.

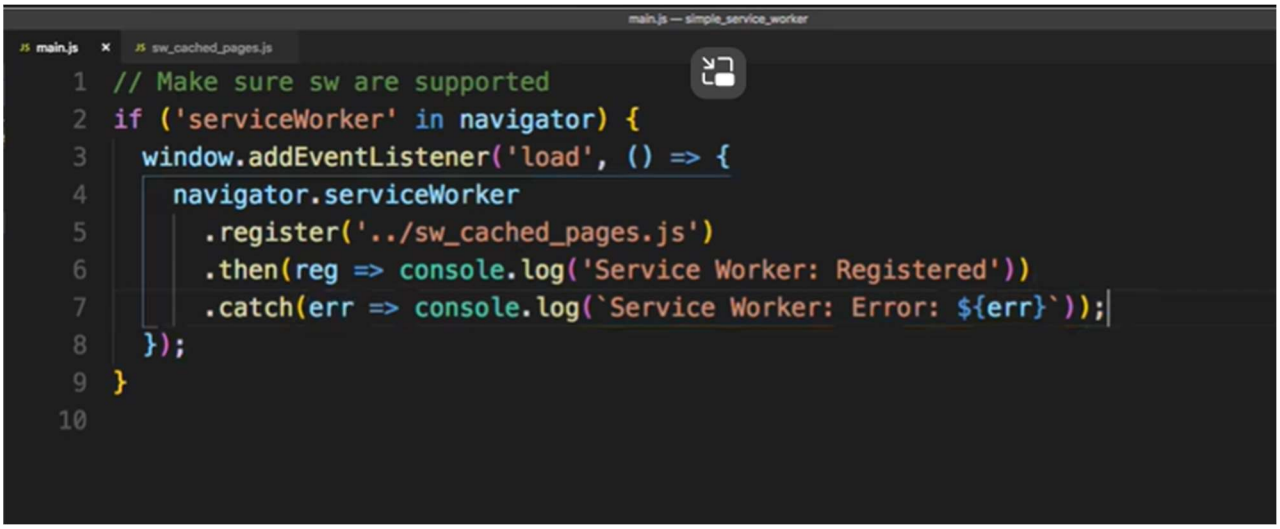


35. Service workers & Caching [1] [2] [3]

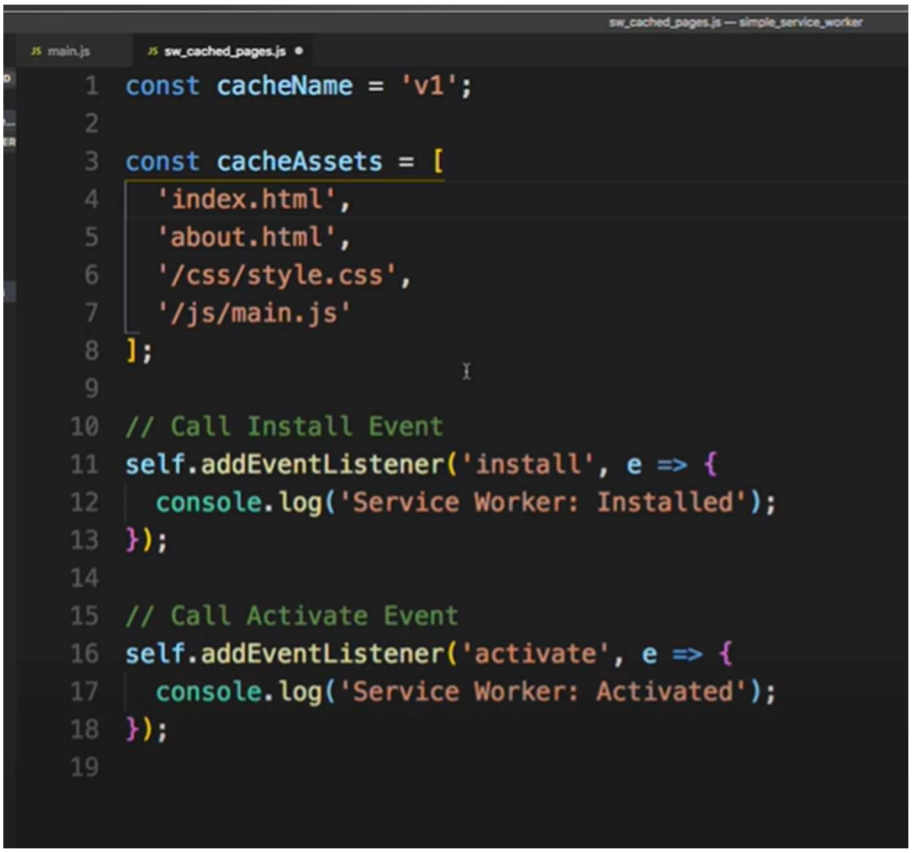
Service Worker is simply a single js script file that registers with the browser and stays registered even offline till next reload. It can load content with internet connection. So after it, the flow is like from browser to worker then to remote server.



In main.js,



And then in sw_cached_pages.js,



For declaring separate assets,

```
// Call Install Event
self.addEventListener('install', e => {
  console.log('Service Worker: Installed');

  e.waitUntil(
    caches
      .open(cacheName)
      .then(cache => {
        console.log('Service Worker: Caching Files');
        cache.addAll(cacheAssets);
      })
      .then(() => self.skipWaiting())
  );
});
```

```
// Call Activate Event
self.addEventListener('activate', e => {
  console.log('Service Worker: Activated');
  // Remove unwanted caches
  e.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
        cacheNames.map(cache => {
          if(cache !== cacheName) {
            console.log('Service Worker: Clearing Old Cache');
            return caches.delete(cache);
          }
        })
      )
    })
  );
});
```

Now in order to fetch all the cached pages we can use fetch method of service workers api,

```
// Call Fetch Event
self.addEventListener('fetch', e => {
  console.log('Service Worker: Fetching');
  e.respondWith(fetch(e.request).catch(() => caches.match(e.request)));
});
```

36. The Service Worker Lifecycle [1]

The lifecycle of a **Service Worker** in JavaScript involves several stages that allow it to manage resources and control network requests for a web application. Here's a simple breakdown:

1. **Registration**

When the web page first loads, the service worker script is registered by calling `navigator.serviceWorker.register()`. This step tells the browser where to find the service worker script so it can begin controlling the site.

```js
navigator.serviceWorker.register('/service-worker.js')
  .then(function(reg) {
    console.log('Service worker registered!', reg);
  });
```

2. **Installation (Install Event)**

Once registered, the browser tries to install the service worker. During this phase, the service worker caches the necessary resources (like images, styles, and scripts) that will be used offline. The install event is triggered here, and you can define what to cache.

```js
self.addEventListener('install', (event) => {
  console.log('Service Worker installing...');
  // Add resources to cache here
});
```

- If the installation is successful, the service worker moves to the next phase.
- If something goes wrong, the installation is aborted, and the service worker won't activate.

3. **Activation (Activate Event)**

After installation, the service worker enters the activation phase. This step allows the service worker to clean up old caches and prepare to take control of the pages. At this point, it can start controlling pages.

```js
```

```
self.addEventListener('activate', (event) => {
  console.log('Service Worker activating...');
  // Perform clean-up or cache management here
});
```

4. **Fetch Event (Working Mode)**
Once activated, the service worker enters its "working mode." It can now intercept network requests made by the application (such as fetching images, scripts, etc.), and decide whether to serve them from the cache or the network, improving performance and enabling offline use.

```js
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request)
      .then((response) => {
        // Return cached response or fetch from the network
        return response || fetch(event.request);
      })
  );
});
```

5. **Idle and Re-activation**
After serving the fetch events, the service worker goes idle until the next request or event. However, when new versions of the service worker script are uploaded, the new worker won't take control immediately unless all tabs using the previous version are closed. Once they are, the new service worker will **activate** and take over control of the page.

6. **Update**
Service workers can check for updates periodically. If the code changes, the browser will download the new version and attempt to install it. This process leads to a new install and activation phase.

Summary of Lifecycle Stages:
1. **Registration**: Service worker is registered by the app.
2. **Install**: Caching assets for offline use.

3. **Activate**: Clearing old caches, preparing to take control.
4. **Fetch**: Managing network requests, returning cached or online resources.
5. **Idle & Re-activation**: The service worker sleeps until the next event or update.

By understanding this lifecycle, you can create powerful offline capabilities for web apps, manage caching, and improve the user experience.

37. Debugging Service workers using DevTools [1]
38. How can we compress data in Javascript [1]

Install dependency named "**compression**" in js. This is how you can use the compression library in nodejs application:

```
app.use(
  compression({
    level: 6,
    threshold: 100 * 1000,
    filter: (req, res) => {
      if (req.headers['x-no-compression']) {
        return false
      }
      return compression.filter(req, res)
    },
  })
)
```

39. Create an interactive JavaScript 3d Model [1]
40. Build a 360-view image slider with JavaScript [1]
41. What is the difference between a linter and a formatter?

A **linter** analyzes your code to **find errors, potential bugs, and style issues** based on specific rules and guidelines. It enforces coding standards and best practices, ensuring the code adheres to a particular style or pattern. A **formatter** automatically reformats your code to conform to a specific style guide. It focuses on the **aesthetic consistency** of your code, such as indentation, line breaks, spacing, and punctuation (e.g., quotes, semicolons).

42. What formatting tools are available [1]

Prettier is only a code formatter to ensure code consistent styles across the working teams. While Rome is an all-in-one toolchain for JavaScript and TypeScript development. It handles tasks such as linting, formatting, testing, bundling, and more.

NodeJS

Please look at this **[Link]**

43. What is Node **[1]**
44. What is NPM ? **[1] [2] [3]**
45. Callbacks & Promises in NodeJS **[1] [2] [3]**
46. Faster Async functions and promises **[1]**
47. Sequelize.JS (ORM) **[1] [2] [3] [4] [5] [6]**
48. TypeORM **[1]**

To using typeORM, use **createConnection** to create a new connection. Then **Entity** for the new entity, and **BaseEntity** for applying all the base class operations and **Column** for adding new attribute to the Schema. **PrimaryGeneratedColumn** for declaring a primary key and **CreateDateColumn** and **UpdateDateColumn** for created_at and updated_at attributes.

49. What are migrations **[1]**

It simply means instead of using GUI of these databases to create and maintain tables and schemas, we simply can translate each thing to code. We can do this using **knex module** of npm alongwith any sql language like mysql, postgresql etc.

Then use **npx knex init**

Then use **npx knex migrate:make cars_table**

Then in that file, write code like this:

Then use **npx knex migrate:latest**

We can also use **npx knex migrate:rollback**

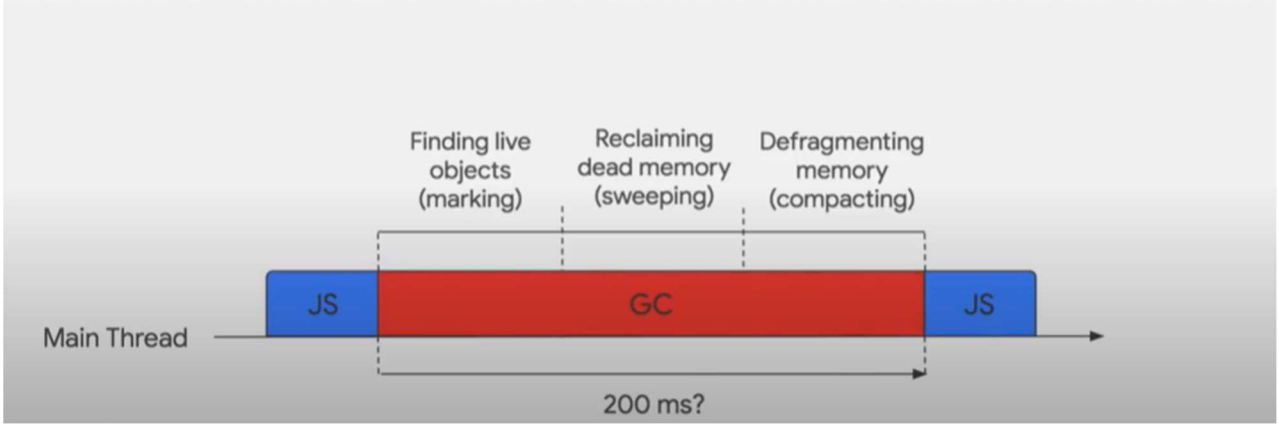
50. What are seeds [1] [2]

Seeds simply mean that we can create and push test data into our database before pushing our real data to confirm all the settings of database we had done.

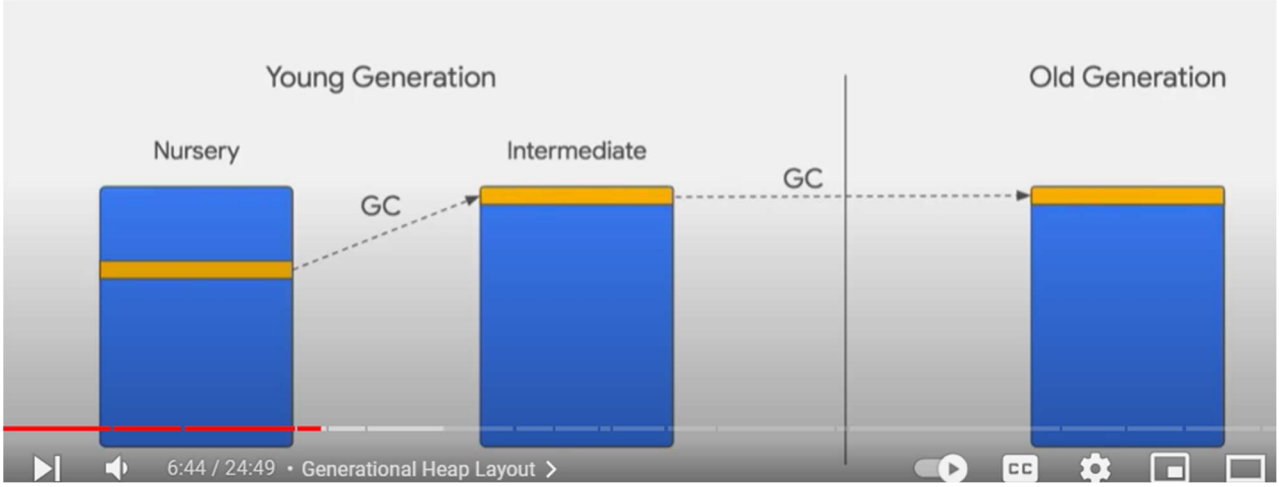
51. Building a RESTFUL API [1] [2]
52. Auto Generating API docs with Node & Swagger [1] [2] [3] [4]
53. Garbage collection [1]

You should know that GC works after the main thread stops execution but it takes its own some time to collect all the garbage variables and functions to bin.

Reality of Garbage Collection



Generational Heap Layout



Minor GC simply means any random processes which survive the first GC will moved (compact in real meanings) to top(From-space) and placed mark upon them and any new will be placed after them and then if they survive another then they are placed on old gen(To-space).

Minor GC (Scavenge)

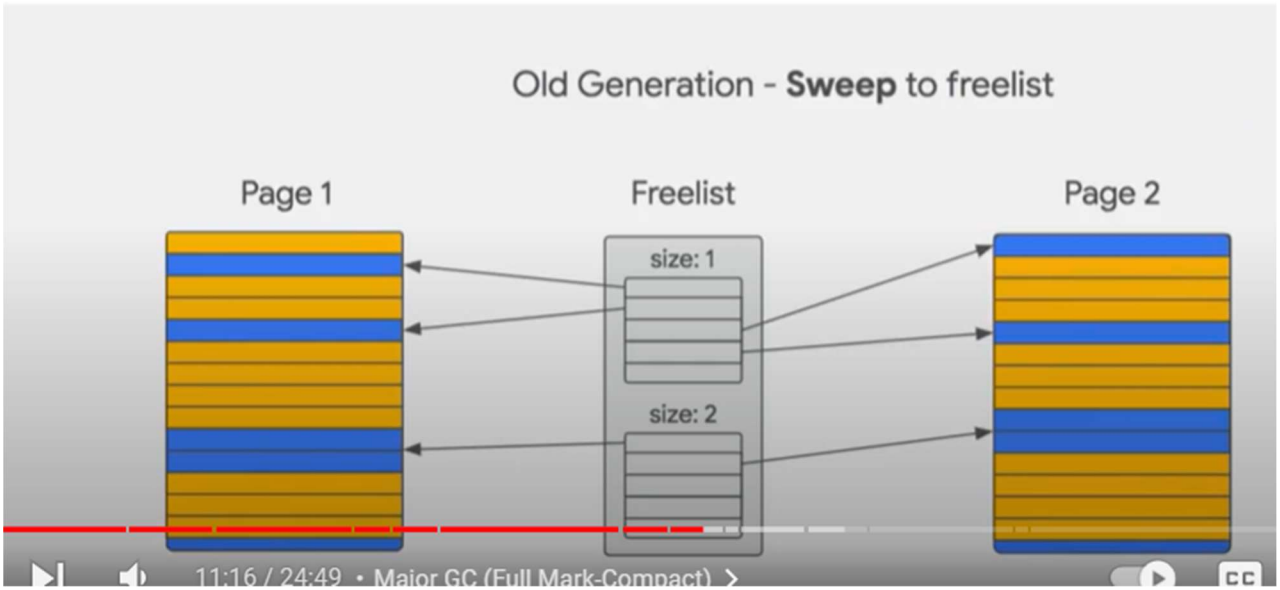
After one GC, an object becomes Intermediate

After the second GC, it moves to Old Space

New Space is always half-empty (or half-full?) to allow for evacuation

Don't forget to update those references!

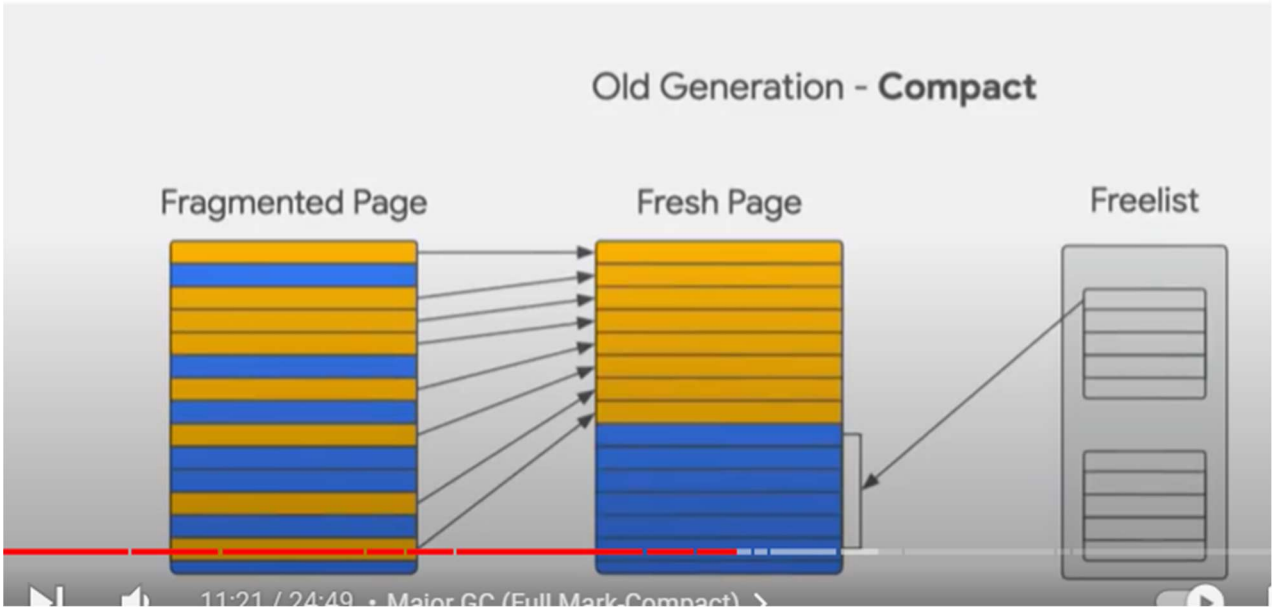
Major GC (Full Mark-Compact)



2nd Floor, Plot # 5-A EOBI Building, Crown Plaza, F-7 Markaz, Islamabad, Pakistan
info@cowlar.com +92 51 8317562

COWLAR

Major GC (Full Mark-Compact)



The Orinoco effort was to transfer a purely sequential garbage collector into a mostly *concurrent* and *parallel* garbage collector (with *incremental* backup)

2nd Floor, Plot # 5-A EOBI Building, Crown Plaza, F-7 Markaz, Islamabad, Pakistan
info@cowlar.com +92 51 8317562

Parallel

Main thread + helpers do equal work at the same time

Still 'stop-the-world' - no JS running

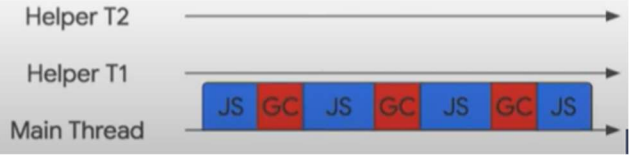
Reduces the atomic pause by a factor N

Incremental

Main thread intermittently does a little work

The JS heap will change between GC work

Better main thread latency



Concurrent

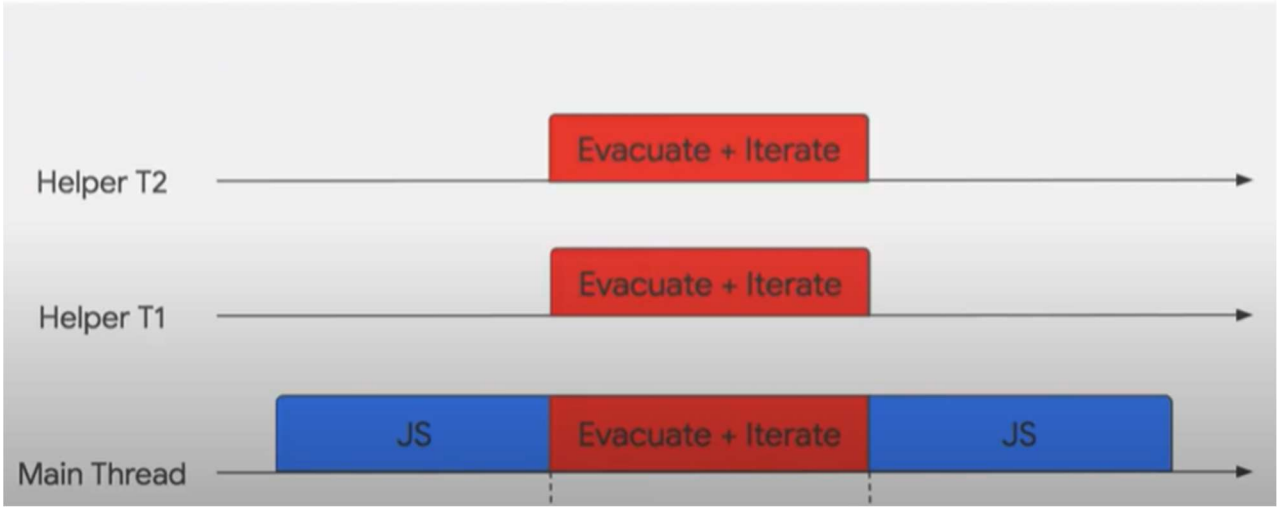
Main thread runs JS constantly

Helpers race constantly on every read/write

Main thread totally free (minus some synchronization required with helpers)

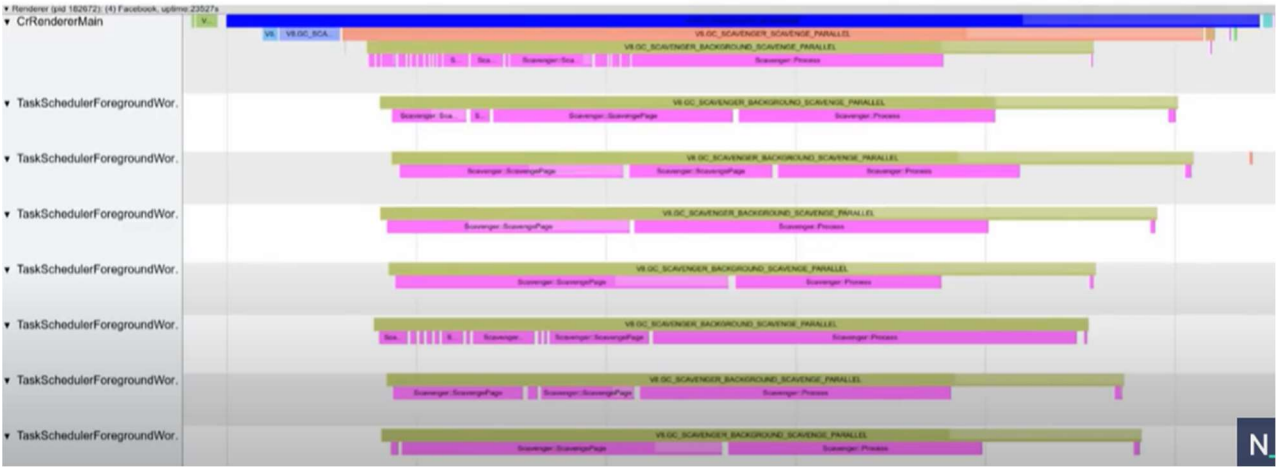
But the approach we are currently following is the,

Parallel Scavenging



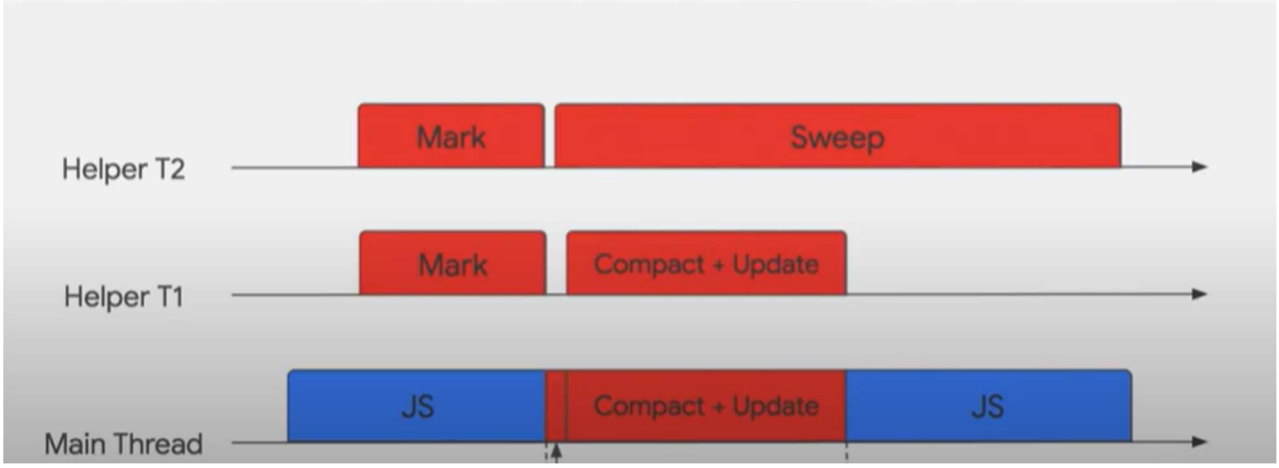
chrome://tracing – Loading facebook.com

8-thread parallel scavenge in 2.725 ms



2nd Floor, Plot # 5-A EOBI Building, Crown Plaza, F-7 Markaz, Islamabad, Pakistan
info@cowlar.com +92 51 8317562

Concurrent Marking + Sweeping

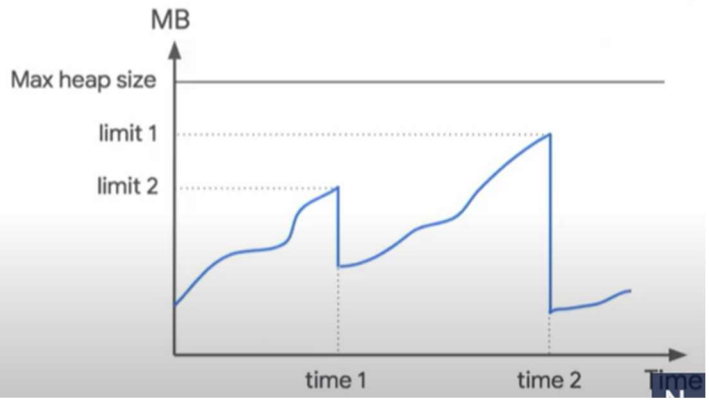


This take upto 10 ms of time.

GC Triggers

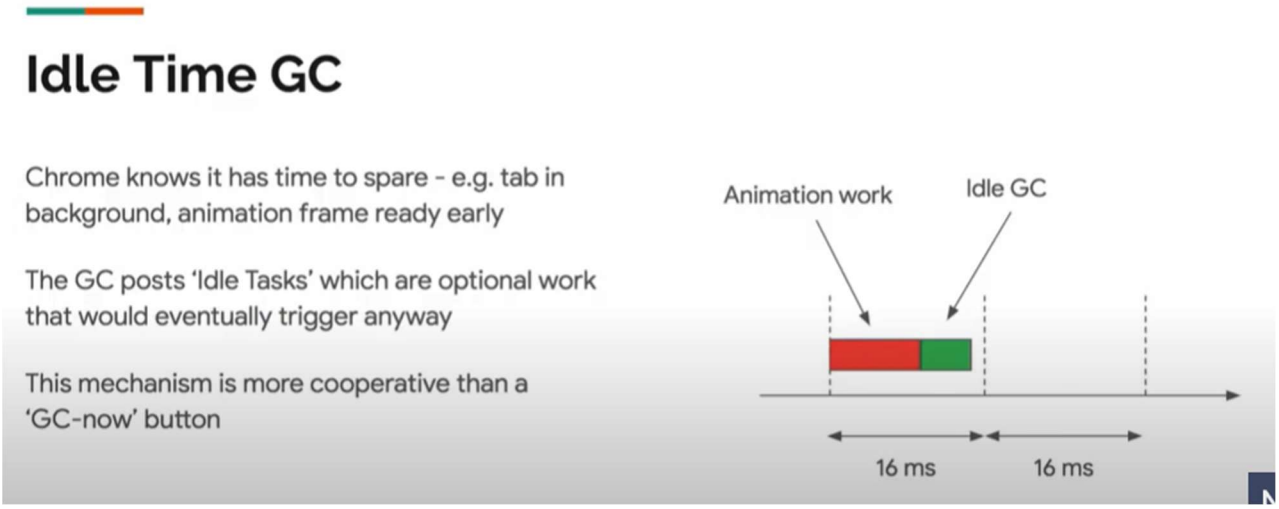
The heap starts small. We grow the total size and try to estimate the sweet spot.

We look at the allocation rate, survival rate and dynamically compute a limit for the next GC



GC triggers work in such a way that if you suddenly start taking too much memory and then suddenly drop, then your limit for current work is decreased accordingly.

We donot have anything like GC switches to trigger a collection. But we have alternative way to do this using Idle Tasks.



54. How can we use migrations [1]
55. Express.js Framework For Node.js [1] [2] [3] [4]
56. Fastiyfy [1]
57. What is a framework [1]
58. Fastify vs Express [1]

Here are the major differences between **Express** and **Fastify**:

1. **Performance**

- **Express**: Known for being a reliable and flexible framework, but its performance is not as optimized as newer alternatives.
- **Fastify**: Focuses on high performance and low overhead. It's much faster than Express due to its asynchronous architecture and optimization techniques.

2. **Middleware System**

- **Express**: Uses a middleware-based architecture where middleware functions are executed sequentially. It's easy to understand and use but can cause performance bottlenecks if not managed well.
- **Fastify**: Utilizes a plugin-based system rather than middleware. This architecture allows for better encapsulation and reusability, improving performance by reducing overhead.

3. **Routing**

- **Express**: Straightforward and flexible routing system. Routes can be created using `app.get()`, `app.post()`, etc., and it's widely familiar to developers.
- **Fastify**: Uses a more structured and efficient routing system. Fastify also has built-in schema validation and automatic generation of 404 responses for unmatched routes.

4. **Validation**

- **Express**: Does not provide built-in validation or schema support. You need external libraries like `Joi` or `express-validator` for request validation.
- **Fastify**: Comes with native JSON Schema validation, which makes it more efficient for data validation and serialization, improving performance and reducing bugs.

5. **Error Handling**

- **Express**: Has simple error-handling with middleware. Errors can be passed to the next function in the stack for handling, but the developer needs to manage it explicitly.
- **Fastify**: Fastify has built-in error-handling and validation systems, which make it easier to handle errors consistently. It also provides structured error responses out of the box.

6. **Plugins**

- **Express**: Has a rich ecosystem of middleware that can be plugged into an application, though these are not encapsulated, and dependencies between middleware may arise.
- **Fastify**: Fastify uses a plugin-based architecture where each plugin is isolated and encapsulated, allowing for better modularity and easier debugging. It has a large collection of plugins as well.

7. **Learning Curve**

- **Express**: Easier to learn and get started with, especially for beginners. The API is minimal and less opinionated, making it flexible but requiring more manual work for complex features.
- **Fastify**: Slightly steeper learning curve due to its plugin-based approach and built-in features like validation, serialization, and error handling, but it rewards with better performance and scalability.

8. **Extensibility**

- **Express**: Extremely extensible but with a higher potential for custom or external tools.
- **Fastify**: Highly extensible via its plugin system, where everything can be wrapped in a plugin, ensuring performance isn't affected.

9. **Ecosystem**

- **Express**: Has been around for a long time, so it enjoys a large and mature ecosystem with many third-party middleware options and community support.
- **Fastify**: The ecosystem is growing quickly, but it's still newer compared to Express. However, Fastify's core provides many built-in features that reduce the need for external libraries.

10. **Request and Response Objects**

- **Express**: The `req` and `res` objects in Express are quite unstructured, with no built-in support for validation, and you have to manipulate them manually.
- **Fastify**: These objects are more structured, and Fastify provides built-in support for validation and serialization, which results in better performance.

11. **TypeScript Support**
- **Express**: Can be used with TypeScript, but it requires third-party types and manual configuration.
- **Fastify**: Provides first-class TypeScript support with types included directly in the framework, making it easier to work with in TypeScript projects.

12. **Community and Maturity**
- **Express**: Established and widely adopted by the community. It is the most popular Node.js framework and is known for its stability.
- **Fastify**: Still growing but quickly gaining popularity for those needing performance-focused solutions. It's newer but has strong community backing.

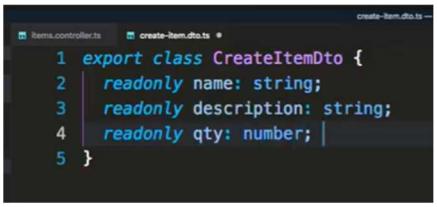
13. **Scalability**
- **Express**: Good for smaller to medium-sized applications but can become slower with heavy traffic due to its middleware architecture.
- **Fastify**: Designed with scalability in mind. The plugin-based architecture and efficient request/response lifecycle make it more suitable for high-performance and large-scale applications.

Summary:
- **Express** is well-suited for general-purpose applications with minimal setup and an easy learning curve.
- **Fastify** is ideal when performance and scalability are the primary concerns, providing better speed, built-in validation, and plugin encapsulation.

Would you like more details on how to set up one of these frameworks for a project?

59. What is NestJS [1]

NestJS is simply a nodejs based framework which provides more flexibility than the express JS. To generate a controller, use command **nest g controller <name>**. For an API endpoint, like GET, import **GET declarator** from the nestjs/common folder and then use that in the class.



```
items.controller.ts • create-item.dto.ts
1  import { Controller, Get, Post, Put, Delete, Body } from '@nestjs/common';
2  import { CreateItemDto } from './dto/create-item.dto';
3
4  @Controller('items')
5  export class ItemsController {
6    @Get()
7    findAll(): string {
8      return 'Get all items';
9    }
10
11   @Post()
12   create(@Body() createItemDto: CreateItemDto): string {
13     return 'Create item';
14   }
15 }
16
```

To generate a service, use command **nest g service <name>**.

Use this link to further revide nestJS concepts, GitHub - bradtraversy/nestjs_rest_api: Rest api built with Nest and MongoDB

60. What are Race Conditions and how to handle it [1]

A race condition is a type of *programming error* that can occur when multiple processes or threads are accessing the same shared resource, for instance, a file on a file system or a record in a database, and at least one of them is trying to modify the resource. This is the simplest example,

```
async function main () {
  const transaction1 = sellGrapes() // NOTE: no `await`
  const transaction2 = sellOlives() // NOTE: no `await`
  await transaction1 // NOTE: awaiting here does not stop `transaction2`
                     // from being scheduled before transaction 1 is completed
  await transaction2
  const balance = await loadBalance()
  console.log(`Final balance: ${balance}`)
}
```

If we execute this code we might end up with different results. In one case we might get the correct outcome:

sellOlives - balance loaded: 0
sellOlives - balance updated: 50
sellGrapes - balance loaded: 50
sellGrapes - balance updated: 100
Final balance: 100

But in other cases we might end up in a bad state:

sellGrapes - balance loaded: 0
sellOlives - balance loaded: 0
sellGrapes - balance updated: 50
sellOlives - balance updated: 50
Final balance: 50

But if we use this approach, then we will always get consistent results:

```
async function main () {
  await sellGrapes() // <- schedule the first transaction and wait for completion
  await sellOlives() // <- when it's completed, we start the second transaction
                     //    and wait for completion
  const balance = await loadBalance()
  console.log(`Final balance: ${balance}`)
}
```

This implementation will consistently produce the following output:
sellGrapes - balance loaded: 0
sellGrapes - balance updated: 50
sellOlives - balance loaded: 50
sellOlives - balance updated: 100
Final balance: 100

Now we can also use **async-mutex locks** to handle the transactions so it will only try to acquire once the allocation is free of any lock. In case we donot want to import any extra module, we can simply make our own mutex lock.

The idea is to inizialize our global mutex as an instance of a resolved promise:

```
let mutex = Promise.resolve()
```

Then in our critical path we can do something like this:

```
async function doingSomethingCritical() {
  mutex = mutex.then(() => {
    // ... do stuff on the critical path
  })
  .catch(() => {
    // ... manage errors on the critical path
  })
  return mutex
}
```

61. Push Notifications using NodeJS + web-push [1]

Use this link to see code for push notifications, in short, you initially have to register a service worker, then register a push notification then send a push notification to the user document only after subscribing user for push notification.

GitHub - bradtraversy/node_push_notifications: Example using Node.js and service workers to send and show push notifications

62. Unit testing (Jest, Mocha, Jasmine)
63. Bundlers, Task runners, Build / Dev tooling **[Grunt, Gulp, NPM, Webpack]**

Grunt

Grunt is a task runner for automating repetitive tasks in web development, such as minification, compilation, unit testing, and linting. It relies on configuration files (`Gruntfile.js`), where tasks are predefined using plugins.

When to use Grunt:

- You need a simple and predefined way to automate tasks.
- Your project has an existing setup with Grunt.
- You prefer a configuration-based approach.

Gulp

Gulp is a task runner like Grunt but more focused on using streams and code-based workflows rather than configuration. It handles tasks such as file transformations and automation (e.g., minification, compiling Sass/LESS, image optimization).

When to use Gulp:

- You prefer a code-based approach over configuration files.
- You want faster task execution, as Gulp uses streams to handle tasks in memory rather than temporary files.
- You need a more flexible and efficient task runner for custom workflows.

NPM (Node Package Manager)

NPM is a package manager for JavaScript that comes with Node.js. While its primary function is to manage dependencies, NPM scripts can also be used to automate tasks (e.g., `npm run build`, `npm run test`).

When to use NPM:

- You don't need a complex task runner and can handle automation with basic scripting.
- You're working on a project where installing additional tools (like Gulp or Grunt) is unnecessary.
- You want to manage and run tasks using package-specific scripts without adding a task runner.

Webpack

Webpack is a module bundler that processes JavaScript files and their dependencies, turning them into a single or multiple bundled files for the browser. It can also handle CSS, images, fonts, and other assets through loaders.

When to use Webpack:

- You need to bundle multiple JavaScript files/modules and assets into optimized bundles.
- You're working with modern JavaScript frameworks/libraries like React, Vue, or Angular.
- You need advanced features like code splitting, tree shaking, and module hot reloading.

When to Use Each:

- **Grunt**: Simple task automation for legacy projects.
- **Gulp**: Efficient task automation when you prefer code-based workflows.
- **NPM**: Lightweight task automation and dependency management, ideal for simple needs.
- **Webpack**: Bundling and optimizing assets and modules, mainly for larger JavaScript applications.

Each tool has different strengths depending on the complexity of your project and your specific needs. Webpack is more common for modern development, while NPM scripts or Gulp are simpler alternatives for task automation.

64. Google Sheets for Node JS developers **[1]**

Simple is that, by leveraging google cloud functions, we can simply automate our tasks and using googleapi module to call specific google sheets url to fill data in it automatically.

[50 Interview questions & answers for NodeJS]

VueJS

Don't know VueJs? Click here

[Program with Erik]

[Why do FrontEnd with VueJS]

[Vue over react]

[Vue.js 2 in 60 minutes]

[Learn Vue.js in three hours]

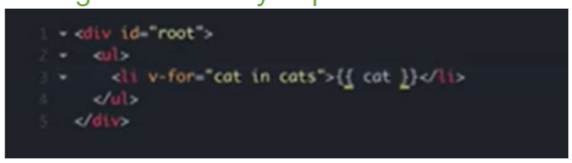
[10 Tips to become a better VueJS developer]

[Intro to Vue - Build a to-do app in 20 minutes]

Vuejs got the component-based architecture.

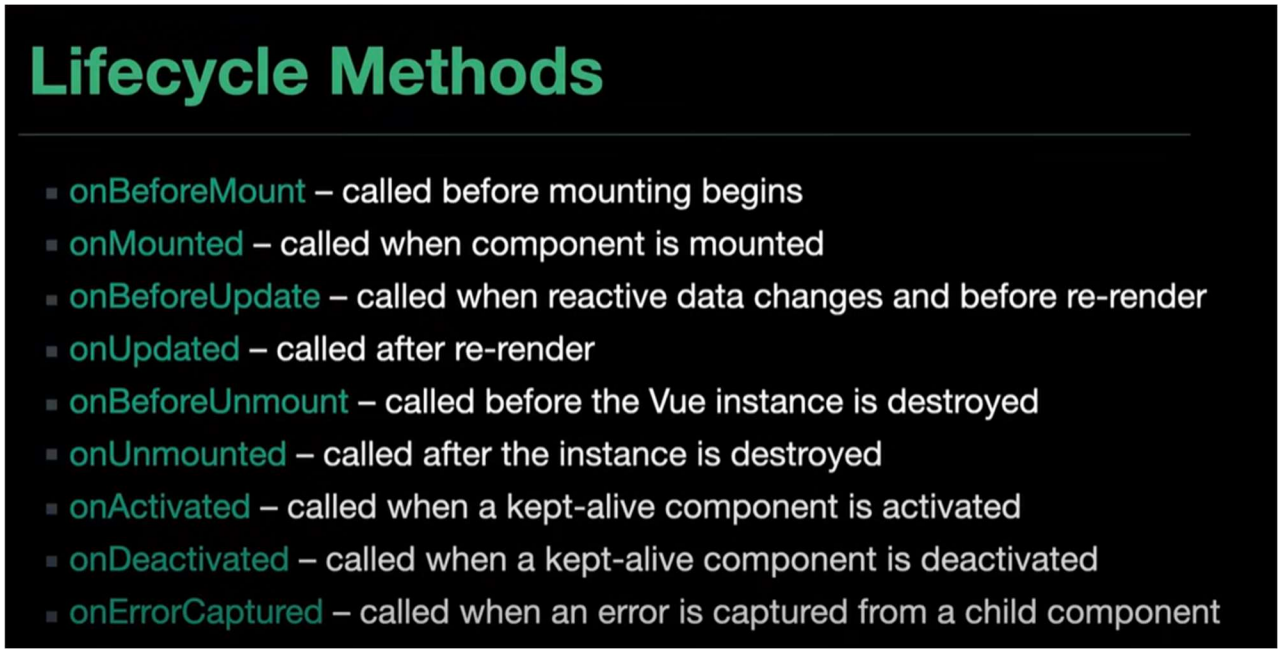
In Vuejs, we got **directives** with the html elements like for input, we got <input v-model=”<data.variable>”> which applies specific rules on the html tag.

- We also got a directive called **v-if** which can only trigger if condition stated inside it is true. Similarly, we also got **v-else-if** and **v-else.**
- We also got a directive called **v-bind** which takes up vanilla html attribute like disabled and then allow us to make condition to help us like button v-bind:diabled=”email.length>8” and **v-once** for allowing only once change at a time and then never no change
- If we got some array to present then we can use **v-for** like this:

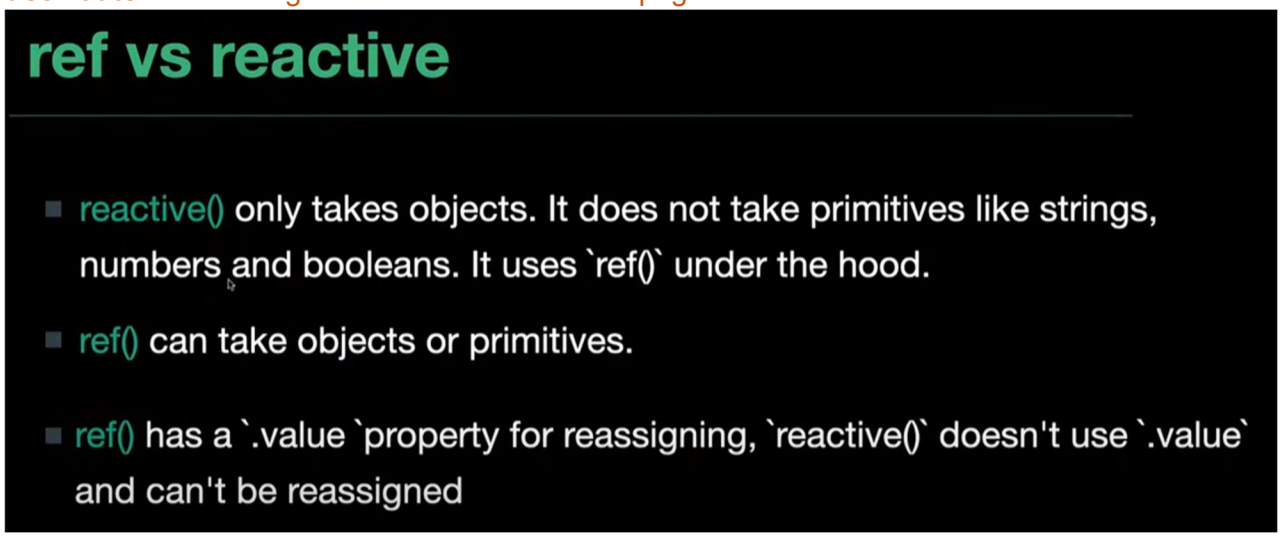


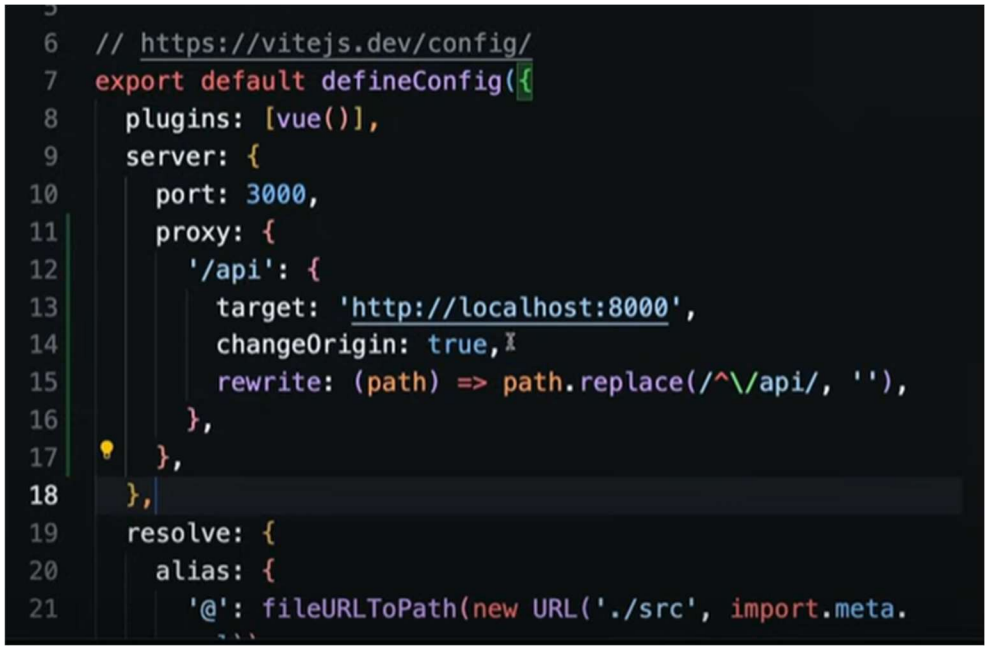
- In case, we want to add a new object, we can use this: also for input we got @keyup.enter to specify this method.

- We can also apply filters and then access in our html and also **computed** allow us to specify that variable or function which fullfils certain conditions.
- By saying **Vue.component(),** we can create new custom components.
- We also got **ref()** method to make components reactive in nature and they are not accessed by this.name but now by name.value.
- We got the **v-model** directive



- We also got **defineProps** method from the vue which enables to get the props in a component.
- We got the **computed** function which returns value based on other values.
- We can introduce routing in vue using **vue-router.**
- We got **RouterLink** like the Link of react so spa rules are not violated and also the **useRoute** method to get the current route of the page.





```
// https://vitejs.dev/config/
export default defineConfig({
  plugins: [vue()],
  server: {
    port: 3000,
    proxy: {
      '/api': {
        target: 'http://localhost:8000',
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/api/, ''),
      },
    },
  },
  resolve: {
    alias: {
      '@': fileURLToPath(new URL('./src', import.meta.
```

65. Why Choose VueJS or React? [1] [2] [3]
66. What are VueJS components

Vue components are the building blocks of a Vue.js application. Each component encapsulates its own functionality and can be reused throughout the application. Here's what Vue components typically consist of:

1. **Template**

- The HTML structure that defines how the component will be rendered. It can include directives, bindings, and dynamic content.

```html
<template>
  <div class="my-component">
    <h1>{{ title }}</h1>
    <button @click="increment">Click me</button>
  </div>
</template>
```

2. **Script**

- The JavaScript logic of the component, including data properties, methods, lifecycle hooks, and computed properties.

```javascript
<script>
export default {
  data() {
    return {
      title: 'Hello, Vue!',
      count: 0
    };
  },
  methods: {
    increment() {
      this.count++;
    }
  }
};
</script>
```

3. **Style**
- The CSS styles specific to the component, which can be scoped to avoid affecting other components.

```css
<style scoped>
.my-component {
  color: blue;
}
</style>
```

4. **Props**
- Custom attributes that allow data to be passed from a parent component to a child component.

```javascript
props: {
  initialCount: {
    type: Number,
```

```
      default: 0
    }
  }
```

5. **Emits**
  - Events that a component can emit to communicate with parent components.

  ```javascript
  emits: ['incremented'],
  ```

6. **Slots**
  - Placeholder elements that allow content to be passed from the parent component into the child component, enabling more flexible layouts.

  ```html
  <slot></slot>
  ```

Example Component Structure

Here's a complete example of a Vue component:

```html
<template>
  <div class="counter">
    <h1>{{ title }}</h1>
    <p>Count: {{ count }}</p>
    <button @click="increment">Increment</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      title: 'Counter Component',
```

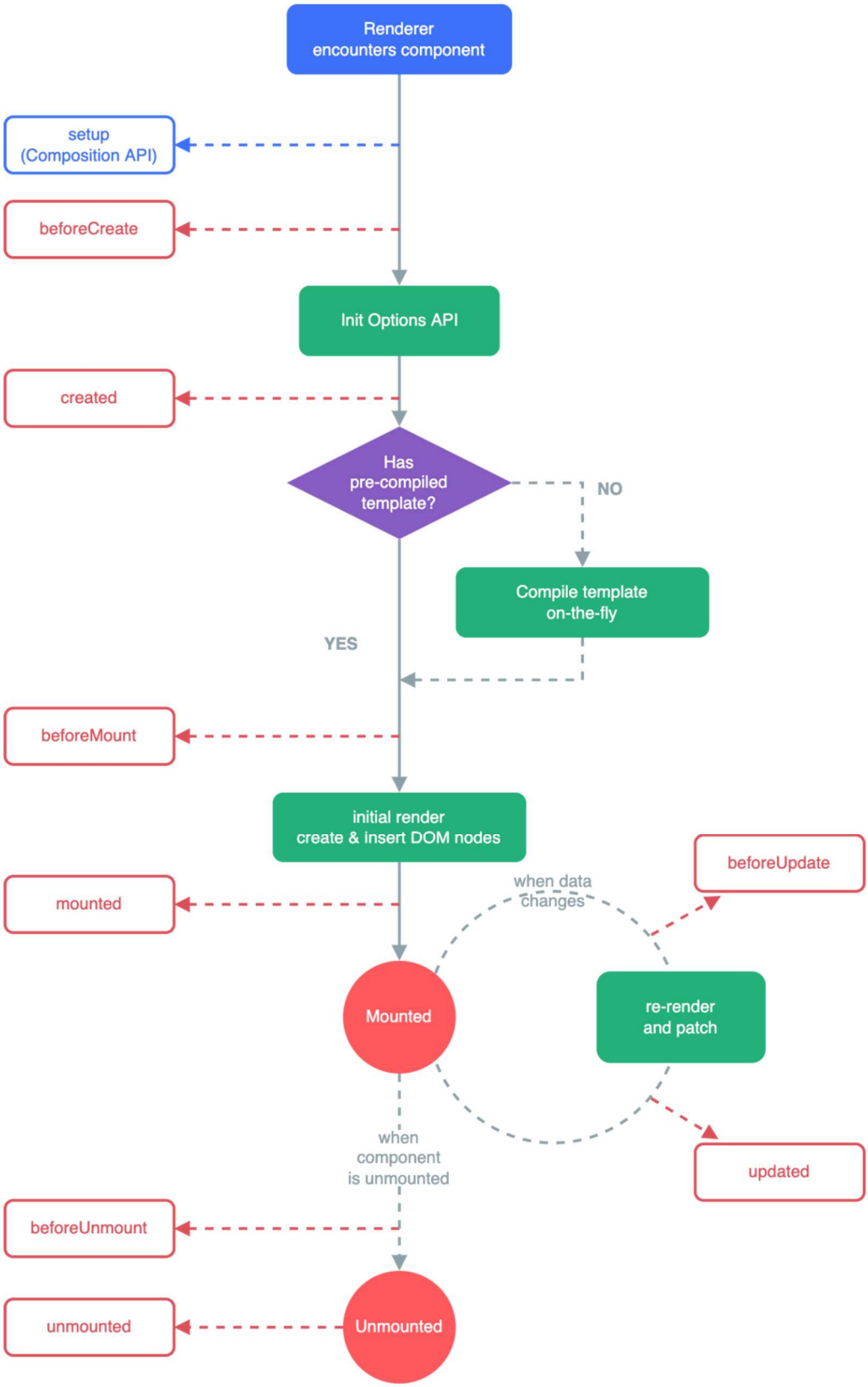
```
      count: 0
    };
  },
  methods: {
    increment() {
      this.count++;
      this.$emit('incremented', this.count);
    }
  }
};
</script>

<style scoped>
.counter {
  text-align: center;
}
</style>
```

Summary

A Vue component consists of a template, script, and style, along with props, emits, and slots for enhanced functionality and communication between components. This modular design promotes reusability and maintainability in Vue applications.

67. How are VueJS components created
68. VueJS Components LifeCycle



2nd Floor, Plot # 5-A EOBI Building, Crown Plaza, F-7 Markaz, Islamabad, Pakistan
info@cowlar.com +92 51 8317562

```
<template>
  <div>
    <h1>{{ message }}</h1>
  </div>
</template>

<script>
export default {
  data() {
    return {
      message: 'Hello, Vue!'
    };
  },
  beforeCreate() {
    console.log('Component is about to be created.');
  },
  created() {
    console.log('Component has been created.');
  },
  beforeMount() {
    console.log('Component is about to be mounted.');
  },
  mounted() {
    console.log('Component has been mounted.');
  },
  beforeUpdate() {
    console.log('Component is about to be updated.');
  },
  updated() {
    console.log('Component has been updated.');
  },
  beforeDestroy() {
    console.log('Component is about to be destroyed.');
  },
  destroyed() {
    console.log('Component has been destroyed.');
  }
};
```

</script>

69. Routing in VueJS
70. What is the difference between **v-if** and **v-show**

v-if for conditionally rendering element in JS and not using css while v-show to always render and then styling is controlled using css. So latter, element is still rendered in DOM even if not seen on screen.

71. What is the ***updated*** function (what are its pros and cons)

The `updated` lifecycle hook happens right after the DOM tree has updated.

If we modify a property or do something else in the `updated` hook that triggers a new render, the `updated` hook will be called again after that new render, and we have most likely created an infinite loop.

To avoid an infinite loop we should always consider to use the `beforeUpdate` lifecycle hook instead of the `updated` lifecycle hook.

72. Filters in VueJS

A **Filter** is a simple JavaScript function which is used to change the output of a data to the browser. Filters in **Vue.JS** don't change the data directly wherever we store them, it only applies formatting to our data. The data remains the same only the output of a data to a browser is changed. Vue.JS doesn't give these filters by default, so we have to make these filters. With Vue.JS , we can use filters in two different ways i.e. **Global filter** and **Local filter**.

73. Fragments in Vue **[1]**
74. Plugins
75. Mixins [1]

An array of option objects to be mixed into the current component.

```
interface ComponentOptions {
  mixins?: ComponentOptions[]
}
```

In Vue.js, **mixins** are a flexible way to distribute reusable functionalities across components. They allow you to encapsulate a piece of code that can be shared among multiple components, making it easier to maintain and organize your codebase.

```
// myMixin.js
export const myMixin = {
  data() {
```

```
    return {
      mixinData: 'Hello from Mixin!'
    };
  },
  methods: {
    greet() {
      console.log(this.mixinData);
    }
  },
  created() {
    console.log('Mixin created hook called.');
  }
};


<template>
  <div>
    <p>{{ mixinData }}</p>
    <button @click="greet">Greet</button>
  </div>
</template>

<script>
import { myMixin } from './myMixin';

export default {
  mixins: [myMixin],
  created() {
    console.log('Component created hook called.');
  }
};
</script>
```

76. What is single Page Applications in Vue

https://clouddevs.com/vue/single-page-application/#:~:text=Single%2Dpage%20applications%20(SPAs),%2C%20flexibility%2C%20and%20high%20performance.

77. Declarative rendering

Declarative rendering in Vue enables us to render data to the DOM using straightforward template syntax. Double curly braces are used as placeholders to interpolate the required data in the DOM.

```
const parent = new Vue({
    el : '#parent',
    data : {

        // The data that will be
        // interpolated in the DOM
        priority1: "vue.js",
        priority2: "React.js",
        priority3: "Angular.js"
    }
})
```

```
<li>
          <strong>{{priority3}}</strong>
          is good too
        </li>
```

78. Class and style binding

A common need for data binding is manipulating an element's class list and inline styles. Since class and style are both attributes, we can use v-bind to assign them a string value dynamically, much like with other attributes. However, trying to generate those values using string concatenation can be annoying and error-prone. For this reason, Vue provides special enhancements when v-bind is used with class and style. In addition to strings, the expressions can also evaluate to objects or arrays.

79. Event handling

We can listen to events using **v-on** method. This method is a function which can perform another function in it.

```
<!-- the click event's propagation will be stopped -->
<a @click.stop="doThis"></a>

<!-- the submit event will no longer reload the page -->
<form @submit.prevent="onSubmit"></form>
```

```
<!-- modifiers can be chained -->
<a @click.stop.prevent="doThat"></a>

<!-- just the modifier -->
<form @submit.prevent></form>

<!-- only trigger handler if event.target is the element itself -->
<!-- i.e. not from a child element -->
<div @click.self="doThat">...</div>
```

80. How to pass data between components
81. Props
82. Type of loops in VueJS

By using v-for directive we can make loop of components so we can also do conditional rendering using v-if for items in the list.

83. How many ways can props be declared

You can pass dynamic props using v-bind and for defining props, you can use defineProps hook or simply export props from script to use in main template.

In Vue.js, there are several ways to declare props. Here are the primary approaches:

1. **Array Syntax**

This is the simplest form, where you just declare the prop names in an array. All props are assumed to be of any type.

```js
export default {
  props: ['title', 'content']
}
```

2. **Object Syntax with Type Declaration**

In this approach, you declare props as an object where the keys are the prop names and the values define the type of the prop.

```js
export default {
  props: {
    title: String,
    content: Number
```

```
  }
}
```

3. **Object Syntax with More Detailed Options**
You can provide more detailed configuration for each prop, such as type, required status, default values, custom validation, etc.

```js
export default {
  props: {
    title: {
      type: String,
      required: true
    },
    content: {
      type: Number,
      default: 0
    },
    isPublished: {
      type: Boolean,
      default: false
    }
  }
}
```

4. **Using the `withDefaults` Utility (Composition API)**
In Vue 3, with the Composition API, you can declare props and their defaults using the `withDefaults` utility.

```js
import { defineComponent, withDefaults } from 'vue'

export default defineComponent({
  props: withDefaults(defineProps<{
    title: string
    content?: number
  }>(), {
```

```
    content: 0
  })
})
```

5. **Using TypeScript (Vue 3)**
In Vue 3, you can define props using TypeScript to get type checking and other benefits.

```ts
export default defineComponent({
  props: {
    title: {
      type: String as PropType<string>,
      required: true
    },
    content: {
      type: Number as PropType<number>,
      default: 0
    }
  }
})
```

Summary
- **Array syntax** for simple props (no type enforcement).
- **Object syntax** for specifying types and additional options.
- **With detailed object syntax** to specify things like required, default, custom validation.
- **WithDefaults utility** when using the Composition API in Vue 3.
- **TypeScript** support in Vue 3 for strong type checking.

These approaches can be chosen based on the complexity of your props and whether you're using the Options API, Composition API, or TypeScript.

84. What are the pros and cons of using those different ways

Syntax	Pros	Cons
Array	- Simple to use - Quick for small apps	- No type validation - No flexibility
Object (Type Declaration)	- Type validation - Better error handling	- Limited flexibility (just types)

Object (Detailed Options)	- Full control (types, defaults, etc.) - Robust validation	- More verbose - May over-complicate simple use cases
withDefaults (Composition API)	- Works with TypeScript - Flexible - Easier default values	- Vue 3 only - Slightly harder to learn
TypeScript	- Strong typing - IDE support - Scales well	- Requires TypeScript - More verbose

85. Attributes

We got three built in attributes for vuejs:

- The **key** special attribute is primarily used as a hint for Vue's virtual DOM algorithm to identify vnodes when diffing the new list of nodes against the old list.
- **ref** is used to register a reference to an element or a child component.
- **is** used for binding dynamic components.

Also we got other types of attributes like html attributes like href etc, custom attributes like custom components props and dynamic attributes using v-bind upon them.

86. How are attributes used
87. Vue Apollo (GraphQL with Vue) [1]

Basically this is graphql integration with the vuejs. We can either use appolo object in script or we cna directly use AppolloQuery object inside template which itself contains template having **v-slot** in it.The ApolloQuery contains query attribute in which you can pass the query file in it where you had written graphql query. We can also pass the variables in it. We can also not only have the query but also the **mutations**. Mutation is for creating something new.

Vue Apollo is an integration between **Vue.js** and **Apollo Client**, which allows you to easily use GraphQL queries and mutations in your Vue components. GraphQL is a query language for APIs that enables declarative data fetching, and Apollo Client is a powerful library that helps manage and interact with GraphQL APIs on the client side. Vue Apollo provides a simple and seamless way to integrate Apollo Client into Vue applications, making it easy to interact with GraphQL APIs.

88. Vuex [1] [2]
89. Vue-DevTools [1]
90. Debugging VueJS [1]
91. VueJS Dropdown with Keyboard Accessibility [1]
92. VueJS search with vue-fuse [1]
93. How are HTTP requests sent via Vue
94. Vue Testing with Jest [1]
95. Unit testing [1]
96. State management in Vue

97. Lifecycle Hooks
98. Static site generation for Vue (Gridsome) [1] [2] [3]
99. Using Gridsome with Wordpress [1]
100. Server-side rendering (SSR) with VueJs
101. Faster Web applications with Vue 3 [1] [2]
102. Vue 3 Composition API [1]
103. Vite

HTML5 /CSS /BootStrap /Jquery

[Beginners HTML5]

104. Demonstrate excellent proficiency in HTML5, CSS3 [1] [2] [3]
105. What is HTML canvas [1] [2] [3]
106. Speed up HTML with Emmet [1] [2]
107. Bootstrap 4 [1]
108. What is an offset in Bootstrap [1] [2] [3] [4]
109. Media queries in Bootstrap [1] [2] [3] [4]
110. What is z-index (CSS) [1] [2]
111. Standard media queries [1]
112. CSS transitions [1]
113. Drawing + Animations on Canvas
114. What is Webkit in CSS [1] [2] [3]
115. CSS Grid [1] [2] [3] [4] [5]
116. Flexbox (CSS) [1] [2] [3] [4]
117. Responsive design (viewport & media queries) [1] [2] [3]
118. CSS3 Viewport [1] [2] [3]
119. I have 10 rows in a table, how can I change the color of the 4th row without using a class
120. Jquery [1] [2] [3]
121. Why do we get $undefined errors [1]
122. How can you play a video using HTML5 [1]
123. What are iframes [1] [2] [3] [4]
124. How can you add 360 degree images on HTML page [1]
125. Tricky CSS properties (Box Model,Font-relative, viewport etc) [1]
126. Must understand and know CSS tricks [1] [2]
127. Knowledge of CSS Preprocessors (Ideally Sass) [1] [2]
128. What are CSS image sprites [1] [2] [3] [4] [5] [6]
129. 3d models in HTML documents [1] [2]
130. Can I build a 360 view of an object using HTML
131. Basic understanding of Google Developer Tools [1]

132. How to improve load performance using Chrome DevTools [1]
133. Supercharging Page load [2]
134. Designing for speed & hacking user perception [1]
135. Advanced Chrome page load speed [1]
136. Charting Libraries D3, HighChart [1]

Git

137. Git & Github Crash Course [1]
138. Github [1]
139. What is GitFlow [1]
140. Resolving conflicts during Git merge [1]
141. Git Graph [1]
142. Git rebase [1][2]
143. Git Submodule
144. Git squash

Database

145. MySql
146. PostGres
147. mongoDB
148. Sqlite
149. Indexing

Misc Section

150. Linux basic command line [1] [2] [3]
151. Bash [1]

Following are all the commands:

1. Ls
2. Pwd
3. Cd

4.
5. File
6. Locate
7. Which
8. History
9. Whatis
10. Apropos
11. Man
12. Mkdir
13. Touch
14. Cp
15. Mv
16. Rm
17. Rmdir
18. Cat
19. More
20. Less
21. Nano
22. |
23. Sudo updated (-s for temporary root user privileges)
24. Exit
25. Su - <name of other user profiles>
26. Users
27. Id
28. Chmod
29. Watch
30. Killall
31.

152. SSH [1] [2]

It simply means secure shell. It is simply a protocol like http and ftp. Literally for doing anything on other computer. Traffic is encrypted.

153. VIM [1]

154. HTTP Crash course [1]

155. Google Chrome Dev tools crash course [1]

156. **What is a REST API?** [1]

In REST (Representational State Transfer), the term "representational state" refers to the way resources are represented and how their state can be manipulated over a network. Here's a breakdown of the concept:

Key Concepts

1. **Resources**:
   - In REST, everything is considered a resource, identified by a unique URI (Uniform Resource Identifier). Resources can be anything—documents, images, services, etc.

2. **Representation**:
   - Resources can have multiple representations (like JSON, XML, HTML). When a client requests a resource, the server sends back a representation of that resource's current state.

3. **State**:
   - The "state" of a resource is the information it holds at any given time. For example, the state of a user resource might include their username, email, and profile picture.

4. **Statelessness**:
   - RESTful services are stateless, meaning each request from a client contains all the information needed for the server to fulfill that request. The server does not store any state about the client session.

5. **Transition**:
   - Clients can change the state of a resource by performing actions (like creating, reading, updating, or deleting) through standard HTTP methods (GET, POST, PUT, DELETE). Each action can result in a new representation of the resource.

Example Workflow

1. **Client Requests Resource**:
   - A client sends a GET request to a URI (e.g., `/users/1`).

2. **Server Responds with Representation**:
   - The server responds with a representation of the user resource, typically in JSON format.

3. **Client Modifies State**:
   - If the client wants to update the user information, it sends a PUT request with the new data to the same URI.

4. **Server Updates State and Returns New Representation**:
   - The server updates the resource and returns the updated representation.

Conclusion

Representational state in REST emphasizes the importance of resources and their representations. By following the principles of REST, applications can achieve a scalable and stateless architecture, which enhances their performance and reliability.

157. **REST vs GraphQL** [1]
158. REST vs RPC vs GraphQL [1]
159. **GraphQL** [1]

A powerfull query language to communicate between client and server. GraphQL is named because we are going to jump ant then traverse through the graph, matching the nodes which is the required attribute of that object.

160. **MQTT** Protocol [0] [1] [2] [3] [4] [5]
161. **MQTT** Mosquitto broker [1]
162. **MQTT** Publishers / Subscribers [1]
163. tRPC [1]
164. Sending Curl
165. Nginx [1]
166. WebSockets [1]

HTTP 1.0: Open TCP connection, make client request, done server response, close TCP connection.

HTTP 1.1: Open TCP connection, make client request, get server response, leave this TCP connection open.

WebSocket: Uses HTTP 1.1 technology for persistent connection

167. What are Progressive Web Apps (PWAs)?
168. What is Webpack? [1] [2]
169. What's the difference between Webpack / Gulp / Grunt ?
170. What is tree shaking?
171. Must understand MVPs [1]
172. What is docker [1]

173. Must understand the difference between prototyping and building for production
174. Basic understanding of DevOps
175. Basic understanding of Kubernetes / Spinnaker / Pipelines
176. Will prefer someone with an understanding of software testing
177. Working knowledge of Git (Github / bitbucket etc)
178. Working knowledge of using Google Docs
179. Working knowledge of using Slack
180. Be able to create short videos/screengrabs using Nimbus Screen recorder or any alternative software. Must be able to quickly (and asynchronously communicate with the team via Slack / Whatsapp / short videos / well-written code & Good documentation.
181. Must be able to demonstrate ability to work as a team player. This means you should have good coding practices and should document/comment on your code properly. Being a good team member also means the ability to communicate effectively.

Here are some free resources that can help you. For fresh candidates, it's okay if you don't know everything. For experienced candidates, we expect more.

[Net Ninja]
[Dylan Israel]
[Coding Garden with CJ]
[Coding train]
[Functional JS with Fun Fun Function]

Flutter (Optional)

Here is a free Flutter **playlist** that is very good to learn and revise from. If we hire you we have multiple paid courses that you'll be able to access.

[How Flutter works in 8 min]

[Flutter - Widget Tree & Element Tree in 8 min]

[Flutter Cookbook]

182. Why Flutter, tell 5 pros and cons of flutter?
183. Why does Flutter use Dart ? **[1]**
184. Basic Dart programming and tools.**[1] [2] [3]**
185. Abstraction, Inheritance, polymorphism and encapsulation concepts in dart.
186. What is Pubspec.yaml file
187. Difference between Stateful and Stateless Widgets?
188. Asynchronous programming in flutter?and why do we need that?
189. What is Await in flutter?
190. Difference between then() and whenComplete() function?
191. Use of Native widgets for IOS and Android.
192. Should be able where and how to add Dart packages in flutter app.
193. How to pass data/variables across widgets in flutter app. 194. Use of Futures,Async, Await and timeout in flutter.**[1]** 195. Difference Between async and async* ?
196. State Management in flutter via Getx, Provider and other methods.
197. What is Isolates ?
198. JSON Manipulation (parsing, encoding) in flutter.
199. Knowledge about Web APIs (REST, SOAP) and consuming API calls in flutter.
200. Knowledge about GET,POST, PUT,CURL requests.
201. Should have knowledge about SQlite (or SQflite plugin in flutter) and all the basic database concepts and operations(CRUD). 202. Material App and how to use it in flutter.
203. Understanding of Android Studio project build, xml and other files, where to add permissions and dependencies for android and IOs project.
204. Callbacks in flutter.
205. Deep Understanding about Lists and Maps in dart.
206. Basic knowledge about firebase or any other commonly used cloud service to integrate with flutter app for the data exchange between app and cloud, or for push notifications etc.
207. Secrets in Flutter When to Use Keys - Flutter Widgets 101 Ep. 4 How To Store API Keys For Staging & Production Environments
208. Navigation to different screens in flutter (routes concept). **[1] [2]**

209. Local Push Notifications [1] [2]
210. How can you communicate Between Dart and Native code in flutter?
211. What is Pump and PumpSettle in flutter Testing?
212. Flutter Dev tools [1]
213. What is flutter for Web [1]

MUST PREPARE FOR THE INTERVIEW

Must have clear concepts!

1. GIT
2. Unit Test and Integration testing
3. Docker
4. API Integration
5. .env variables
6. ORMS
    a. Sequelize
    b. Prisma
    c. Mongoose
7. **ADDITION POINTS**
    a. MQTT
    b. Influx

c. GitLab - CI/CD

d. Github Actions

DO NOT LEAVE THIS SECTION

2nd Floor, Plot # 5-A EOBI Building, Crown Plaza, F-7 Markaz, Islamabad, Pakistan

info@cowlar.com +92 51 8317562

e.

Demo

Demonstrate your skills (optional)

Any projects you have done, bring them with you and showcase your code / etc on your laptop. This can include your professional projects, FYP, your python work, github contributions. Anything that showcases your ability. We are particularly interested in seeing how cleaning / well the code is written (do you write comments in your code etc)

Interview Tips

Here are some resources to use to reduce interview nerves [1] [2] [3] [4]. We highly recommend watching all these videos.

When to schedule an interview

This is a question we often get asked. This will depend on the candidate after they have had a chance to go over the test preparation materials. Generally speaking ...

For **fresh graduates**, it is highly recommended to invest some time in learning the material. Take a look at the topics, watch the videos / read the articles and **learn by doing**. You will remember what you learn *by doing* . When you feel like you have sufficient ideas and are 2x or 5x better than you were before you started your preparation, please contact us by email to set up a date. Please send us three dates that work for you. Do not think that there is a deadline for this position. Don't schedule the interview fearing that someone else will take this position. Schedule it when you're ready because we usually only interview candidates **ONCE** [7]. If you're good and promising, there is a very good chance we will hire you and put you through our training program. In the unlikely event where you don't get an immediate call, you might be placed on a short-list where we tap into first when looking to add new fresh graduates.

In the unlikely event that you don't hear back from us at all, (the worst case scenario), you will always be in a win-win position. This is because you will

- Still have gained some important & useful skills
- These skills will help you throughout your professional career

[7] We do make exceptions to this rule in exceptional circumstances. The chances of you getting that exception are about 2%

- Dramatically increase your chances of being hired by a good company in the future. - You'll learn how to learn online. That is an important asset.

For **experienced professionals**, please review the materials carefully. You might know a lot of these things. Our test / questions get increasingly difficult based on the duration of your industry experience and based on the reputation of the (good) companies you worked for in the past. You might learn something new while you revise and get better at what you do. There is really no reason to not learn.

From our point of view, the good thing about hiring experienced professionals is that they have already learned a lot and can begin contributing to our team in a short period of time. Sometimes, the bad thing about hiring experienced professionals is that they have learned some bad habits [8] in the past. Please leave all those bad habits behind when you come in for the interview. This can be difficult to do. Bad habits are difficult to break. But one can always try. It takes self-realization, will power & determination. The greatest minds are those that *can be* changed. We don't hire people with bad habits. The reason is that we cannot allow anyone to ruin our culture, no matter how brilliant they are. There is no & never will be any place for *brilliant jerks* at our startup.

[8] The bad habits can be programming or development related. They can be related to poor development methodologies or practices. They can also be related to not being a team player. They can be related to not acting responsibly. They can be related to not treating others with respect. They can be related to gossiping or participating in office politics. If you think writing good maintainable code, commenting, documentation, Testing, Software Quality Assurance is for losers, please schedule an interview for Feb 30, 2050

COWLAR

Interview FAQs

1. **What's the biggest mistake people make during the interview process?**

They don't invest enough time in learning from the test materials or in preparation for the interview. Sometimes, they just ask to be interviewed too early. Showing up with not enough adequate preparation reduces your chances of being offered a position exponentially.

2. **What time does the interview usually begin?**

The typical time for starting an interview is at 11.30 am. You pick the date, we pick the time (unless you're traveling from outside the city. In this case, please inform us beforehand in writing so we can accommodate you)

3. **How long does the interview/test last?**

The duration can be anywhere between 5 minutes to 5 hours. Please plan on being here for 5 hours in case the interview goes that long.

4. **What to expect on interview day?**

- You arrive between 11.15 am to 11.25 am. You will be judged for being late. Please inform us ahead of time in case you're running late. Please plan to arrive on time and consider traffic / your commute to our office.
- Your first question is water/coffee/tea or green tea? Please pick one of the choices. Nothing is one of the choices. This is a clear sign that you read this document carefully and are capable of following instructions.
- You are then given a piece of paper with some analytical questions and a pen + calculator. Please answer the questions. You can answer any 5 questions of your choice. If you want to use the internet, show us the questions so we can keep a score of how many without internet and how many with the internet.
- You're then asked a few simple questions about yourself and regarding the things mentioned in the CV.
- You will be given 3 to 5 minutes to present your case (pitch us why we should hire you and how your preparation went)
- We will then start asking questions. The questions and how you chose to answer them will determine how many people from our team join/leave the interview.
- The questions tend to increase in difficulty level over time. We might ask you to write some code.
- Please keep your answers short, concise, and to the point. We do appreciate it when someone explains a related concept with great clarity, but please don't go around in circles for every answer.
- Please try to answer all questions honestly and don't hesitate if you don't know the answer. Take a couple of seconds to think or ask us to repeat the question if you had difficulty understanding the question.

2nd Floor, Plot # 5-A EOBI Building, Crown Plaza, F-7 Markaz, Islamabad, Pakistan

info@cowlar.com +92 51 8317562

- Please don't think there are any bonus points for not using the internet. You can actually use the internet every time you're asked a question. In this case, we are judging (& hoping) that your answer is **correct** and taking note of the duration of how long it took you to answer. It's better to be late and correct.
- If we end up spending more than one hour, you can ask us how your interview went at the end (we only provide feedback optionally / if you ask us). We are also looking for feedback about the interview process and how you think we can improve as a startup.
- At any time, you can choose to decide that you don't want to work here and leave.
- Please feel free to use the toilet (in case you need to)
- Please feel free to pause the interview in case you need to pray. You can pray in the designated area in the office.
- Lunch, snacks, and dinner are offered free of cost. Please accept food /drinks that are offered and don't do 'takulf'.
- If you smoke, no smoke breaks are allowed during the interview. You are welcome to go outside and smoke as much as you want before or after the interview.
- Please treat our people with respect. We try very hard to do the same.
- Please keep all comments/feedback to yourself till the end of the interview (in case the interview lasts more than one hour).

5. **Why can the interview duration last 5 hours?**

As we spend more time interviewing a candidate, the difficulty level of the questions increases. Please understand that we only interview/test people for longer because they are overcoming the obstacles and increasing their chances of getting an offer on the spot (or very soon).

6. **Do you provide any remuneration for candidates traveling from outside Islamabad / Rawalpindi?** No

7. **Why is this document so long?**

We want to know if you can read, listen, understand & execute. We also want to know if you actually take the time out to read this entire document. We will be testing this during the interview. The length of this document is also meant to discourage people from asking to be interviewed in-person (esp who are not interested in working hard / reading documentation / learning new things). Your job as a software engineer requires you to know how to read well and how to write well too (although that is a much harder skill to master). Reading this document carefully is part of the test / interview :)

8. **Will you answer my questions about the YC experience or provide feedback about my startup idea?**

No

9. **What are you looking for in fresh candidates?**

In fresh candidates, we are looking for people who want to become polyglot software engineers /developers. During your initial training period, you will be trained in many facets of software development & software engineering. You must be okay with developing a decent understanding/working knowledge of various aspects of technology development (basically you need to know what other people are doing in the company so you can talk to them if you're in need of their expertise for your project/task.

10. How do you decide on Salary?

Our salary packages are based on a combination of employee skills, learning ability, ability to execute without mistakes, their contribution to successful team projects & how well you fit into the team culture. Simply put, the more you know and the faster you can learn is directly related to how fast you can execute. If we can ship good, reliable software faster, we can make more money. We can then share that (extra) money with you.

11. How do you decide which roles to assign to new team members once you hire them?

Generally speaking, over time, you (the employee) & us (the company) will mutually agree on your areas of specialization based on your preferences, skills, your proficiency, and the company's needs after the orientation and training period.

12. Best piece of advice to prepare for the interview/test?

Please try to relax. Try to avoid being anxious. We have made a special effort not to make the interview tense. This is because we want to see how you perform in a real job-like condition. Just prepare as much as you can, try your best and leave the results to Allah. Here are some resources to use to reduce interview nerves [1] [2] [3] [4]. We highly recommend watching all these videos.

This is the training section. Ignore this for now.

2nd Floor, Plot # 5-A EOBI Building, Crown Plaza, F-7 Markaz, Islamabad, Pakistan

info@cowlar.com +92 51 8317562

Cowlar Software (Web Dev track) training Document

HOW TO APPLY FOR THE JOB

Before applying, just know that our process is highly competitive,but for those who have prepared well will sail through the process and won't even notice it, but those who are applying blindly will have a tough time.

We really appreciate people who value their time, as we value yours and ours time.

How to apply?

It is quite simple, just go the work.cowlar.com,
Find your relevant job under the, "Open Job," section and apply!

Follow the simple steps as guided in the application form and you should hear back from us, if you are shortlisted. Next steps will be guided by the Talent Acquisition team via email or call.

In case, you have an issue with the job application, or there is an error, or you have any queries, you can write to us at, careers@cowlar.com, but please make sure you add relevant subject, and craft your email properly, a well written email can get you much faster response.

Best of luck for your application.

2nd Floor, Plot # 5-A EOBI Building, Crown Plaza, F-7 Markaz, Islamabad, Pakistan
info@cowlar.com +92 51 8317562