TOPICS TO LEARN

Created By

Raazi Muahmmed
Linkedin - Raazi Muhammed

Contributors

Ajmal
Linkedin - Ajmal Jaleel

Hafsana
Linkedin - Hafsana S

Akarsh
Linkedin - Akarsh G Kumar

Rabeeh
Linkedin - Rabeeh pk

TABLE OF CONTENTS

HTML & CSS	2
JavaScript	3
Node.js, Express	7
MongoDB	10
React	15
Redux	19
React Native	21
DSA	22
Hosting	27
System Design	28
Git	29
SQL: Postgres	31
Microservice	34
Docker	37
Kubernetes	39
Message Broker	42
TypeScript	44
Next.js	45
NestJS	**47**
Clean Code	48
Python	**49**
Others	52

For detailed explanations

Since people have been marking comments as resolved, I have set the docs to view only. If you need to see the comment please request permission to edit

HTML & CSS

1. HTML

2. **Basics**
3. **Block element and inline element**
4. Element
    a. Void elements
    b. Container Element
5. Attributes
    a. boolean attributes
    b. lang attribute
6. Nesting
7. <!DOCTYPE html>
8. **head**
    a. **<meta>**
    b. <meta charset="utf-8">
    c. Adding an author and description

9. VS

10. h1 vs title in head
11. vs <i>
12. <b> vs

13. GOOD TO KNOW

14. Whitespace
15. entity references
    a. < <
    b. > >
    c. " "
16. Open Graph Data

17. CSS

18. Anatomy of CSS ruleset
19. Selecters
    a. Element
    b. Id, Class
    c. Attribute
    d. Pseudo
20. Box model

JavaScript

1. ## DOM
    a. querySelector
    b. textContent
    c. addEventListener
    d. Order of Parsing
2. **event Propagation**
    a. event Bubbling
    b. event Capturing/ Trickling
    c. how to add both on program
3. event.stopPropagation();
4. inst
    a. e.target
        i. id
        ii. tagName
        iii. pros and cons
5. ## Architecture
    a. Execution context
        i. variable environment (memory)
        ii. Thread of execution (code)
        iii. - global & local execution context
        iv. - phases
            1. Memory allocation
            2. Code execution
    b. Synchronous single threaded app
    c. Call stack
    d. Proxy
        i. Proxy traps
        ii. Reflect
        iii. proxy vs reflect
    e. **Event loop**
        i. Callback queue/ task queue
        ii. Microtask queue
            1. mutation observer
        iii. Starvation
        iv. Memory Heap
    f. Just In Time Compilation
    g. Interpreter vs Compiler
    h. Abstract Syntax Tree
    i. Concurrency model
6. ## Theory
7. Data types
    a. wrapper objects
    b. 0 vs new Number(0)
    c. **Numbers**
        i. 1_000_000
        ii. 1e9, 1e-6
        iii. Hex, binary and octal numbers
        iv. toString(base)
        v. Math.trunc
8. Operators
9. enum
    a. how to get enum in javascript
10. **Function**
    a. Function Statement
    b. Function Expression
    c. Function Declaration
    d. Anonymous function
    e. Named Function Expression
    f. Functional Programing
    g. **Higher order function**
    h. First class function
    i. **Decorator function**
        i. use
        ii. - count no of function call
        iii. - valid data of params
    j. **Pure function**
        i. pros and cons
        ii. rules
        iii. pure vs impure
    k. IIFE
        i. pros
11. Advantages and disadvantages of JS
12. **Set Map Flat**
    a. set

        i. add, delete, has, clear, kyes, values, entries
        ii. <setName>.size
    b. map
        i. get, set, has, delete, clear, keys, values, entries, forEach
        ii. iterating
    c. object vs map
    d. weekSet()
        i. features
    e. weekMap()
        i. features
        ii. key is private
    f. Week set and map summary
    g. falt()
    h. flatMap()
    i. reduceRight()
    j. copyWithin()
13. **Operators**
    a. Nullish coalescing operator
    b. Optional chaining
    c. || vs ??
    d. Ternary operator
    e. Type Operators
    f. **Unary operators**
        i. delete
        ii. typeof
        iii. !, ++, -, +
    g. **Bitwise Operators**
        i. bitwise OR
        ii. bitwise AND
        iii. uses
14. **Scope**
    a. Global scope
    b. Module scope
    c. Function scope
    d. Lexical scope
    e. Block scope
15. Shadowing & Illegal shadowing
16. **Prototype**
17. Types of error
    a. syntax, logic
18. **Closure**
    a. Disadvantage
    b. Uses
    c. lexical scope vs closure
    d. IIFE
19. **Garbage collection**
    a. How does it work?
    b. mark-and-sweep
    c. reachability
    d. **Optimizations**
        i. - Generational
        ii. collection
        iii. - Incremental collection
        iv. - Idle-time collection
20. **Hoisting**
    a. TDZ let, const vs var
    b. Function vs arrow function
21. **Call Apply Bind**
    a. function borrowing
    b. call vs apply vs bind
    c. polyfills
22. transpiler
    a. Babel.
    b. webpack
23. polyfills vs transpiler
24. This Keyword
25. **String Methods**
    a. Length, toUpperCase, LowerCase, Trim, Pad, charAt, Split, Concat, substring, indexOf, lastIndexOf, localeCompare
26. **Array Methods**
    a. Map, Filter, Reduce, Find, Sort, Foreach, Push, Pop, Shift, Unshift, Slice, Splice, concat, indexOf, lastIndexOf, forEach, split, join, reduceRight, iArray, fill, copy, flat
    b. spare array, jagged array, hols in array
    c. copy within
    d. typed arrays
27. **Object Methods**
    a. object constructor, literal

    b. deleting field
    c. Computed properties
    d. __proto__
    e. in
    f. Object.assign
    g. structuredClone
    h. _.cloneDeep(obj)
    i. methods
    j. this keyword
    k. Symbol type
28. Symbol
    a. properties
    b. use
    c. ongo
    d. global symbol registry
    e. for, keyFor, iterator, toPrimitive
29. **Loop**
    a. for
    b. do while vs while
    c. labelled statements
    d. - break
    e. - continue
    f. for...in
    g. for...of
30. **Callback**
    a. callback hell
    b. inversion of control
31. **Promises**
    a. Promise states
    b. Promise chaining
    c. Promise.all
    d. Promise.allSettled
    e. Promise.any
    f. Promise.race
    g. Promise.resolve
    h. Thenable
    i. Finally
    j. Catch
    k. immutable
    l. promisify
    m. pros and cons
32. **Async await**
    a. async always return a promise
    b. error handling in async await
33. **Debouncing & Throttling**
    a. both are used for optimising performance of a web app
    b. by limiting the rate of API calls
34. Spread and Rest Operator
35. DOM, BOM
36. Window Object
37. **ES6 and its features**
    a. Let, Var, Const
    b. Ternary operator
    c. Arrow function
    d. Template literals
    e. Default Parameters
    f. Classes
    g. Modules
    h. Iterators
    i. Object & Array Destructuring
38. **Primitive and non-primitive**
    a. Pass by value and pass by reference
39. Message queue
40. Life
41. Generator
42. **Prototype**
    a. Prototype chain
    b. Prototypal Inheritance
    c. uses?
    d. Circular reference
    e. Object.key
43. **Recursion**
    a. recursive call to function
    b. condition to exit
    c. pros and cons
    d. display the fibonacci sequence
    e. use
44. JavaScript is dynamically types
45. **Currying**
    a. function inside function
46. **Type Casting**
    a. Implicite (Coercion)
    b. Explicit (Conversion)
47. Microtask queue

48. **Shallow copy vs Deep copy**
    a. primitive vs structural
    b. how to make these copies
    c. pros and cons
    d. Mutable vs Immutable
    e. Object.freeze()
49. TCP/IP
50. DNS
51. **IIFE**
    a. pros and cons
52. **Composition vs Inheritance**
53. Function recursion
54. [Symbol.iterator]
55. Truthy and falsy value
56. Strict mode in JS
57. this substitution
58. **VS**
    a. label vs func
    b. == and ===
    c. Let, const, var
    d. Synchronous vs asynchronous
    e. While vs do while
    f. Foreach Vs Map
    g. Parameters, Arguments
    h. for in, for of
    i. Undefined, Null
    j. Keywords & Identifiers
    k. Type casting vs Type coercion
    l. textContent vs innerText
    m. identifiers vs variables
    n. defer vs afb sync
59. **Good to Know**
60. interpreted and compiled doe
61. Server-side vs client-side code
62. with in js

Node.js, Express

Theory

1. What is Node.js
2. why v8 Engine
3. Advantages & Disadvantages of Node.js
4. How node works
5. libuv
6. Node Module System
7. Concurrency vs parallelism
8. REPL , Cli
    a. _
9. NPX
10. Globals
    a. __dirname
    b. __filename
    c. **Module**
    d. Process
11. **Modules**
    a. **Core Modules.**
    b. local Modules.
    c. Third-party Modules.
    d. module.exports:{}
    e. require
    f. ESM
        i. import and export
12. **NPM**
    a. local and global
    b. npm init
13. npm install or i Nodemon
    a. scripts
        i. start
        ii. dev
    b. npm run dev
14. package.json
15. package-lock.json
16. Event loop
17. Event Queue
18. Events
    a. **Events emitter**
        i. on, emit
    b. Http module
19. **Streams**
    a. type of streams
        i. writable, readable, duplex, transform
    b. createReadStream()
    c. readFile vs readFileSync
    d. pipe()
    e. Buffers
    f. Transfer-Encoding: chunked
20. **Cron-job**
    a. * * * * *
    b. 1st* = second
    c. 2nd* = minute
    d. 3rd* = hour
    e. 4th* = day of month
    f. 5th* = month
    g. 6th* = day of week
    h. or, range selector
    i. time zone
    j. validation
21. **CORS**
    a. preflight request
        i. header
        ii. accept-control-allow-origin: *
        iii. accept-control-allow-methods: *
22. **Cluster**
23. Multithreading in node.js
    a. require('worker_theads')
    b. new Worker
24. thread pool
25. worker thread
    a. creating worker,
    b. parent port
26. cluster vs workerthread
27. child process
    a. methods
    b. - fork
    c. - exec
    d. - execFile

e. - spawn
f. spawn vs fork
g. child_procees.fork() vs cluster.fork()

28. HTTP

a. https
b. How does it work?
c. SSL certificate working
d. default port
e. request response cycle
f. Stateless protocol
    i. Local storage, Sessions and Cookies
g. Request
    i. General (start line)
        1. method/target/version
    ii. header
    iii. body
h. Response
    i. General (start line)
        1. version/statuscode/statustext
    ii. header
        1. content type
    iii. body
        1. requested resource
i. **HTTP Methods**
    i. GET
    ii. POST
    iii. PUT
    iv. PATCH
    v. DELETE
    vi. HEAD
    vii. CONNECT
    viii. OPTIONS
    ix. TRACE
j. Idempotent
k. Safe Methods
l. User-Agent
m. Headers
n. writeHead vs setHead
o. Status code
    i. 1xx: Informational
    ii. 2xx: Success
        1. 200 - Success
        2. 201 - Success and created
    iii. 3xx: Redirect
        1. 301: moved to new URL
        2. 304: not changed
    iv. 4xx: Client Error
        1. 401: Unauthorised
        2. 402: Payment Required
        3. 403: Forbidden
        4. 404: Page not found
    v. 5xx: Server Error
p. MIME type
q. HTTP v2
r. TCP and IP

29. XSS
30. CSRF
31. MMA
    a. referral header
32. SQL injection
    a. prepared statements

33. Express

34. npm install express –save
35. app = express()
    a. get()
        i. status()
        ii. send()
        iii. sendFile()
    b. post()
        i. express.urlencode()
        ii. Form vs JS
    c. put()
    d. patch()
    e. delete()
    f. all()
    g. use()
    h. listen()
36. Static files
    a. public

b. express.static()

37. **API**
    a. json()
38. **Params, Query String**
39. Route Parameter
40. Query string/url Parameter
41. Path params
42. **MIddleware**
    a. what is middleware
    b. used for what?
    c. req, res, next
    d. next()
    e. app.use in middleware
    f. passing two middleware
    g. **Types of Middleware**
        i. Application-level middleware
        ii. Third party middleware
            1. morgan
            2. multer
        iii. Router-level middleware
        iv. Built-in middleware
        v. Error-handling middleware
            1. err.statusCode
            2. err.message
43. **Routing**
    a. router
    b. express.Router()

44. Core Express

a. **Session**
    i. i express-session
    ii. secret
    iii. resave
    iv. saveUninitialized
    v. destroy()
b. **Cookies**
    i. i cookie-parser
c. Core middleware
d. Core routing
e. Build own API
f. Core views
g. database integration

Questions

45. How to send find as response
46. Transaction in node.js

47. EJS

a. i ejs
b. server side rendering
c. view engine
d. render()
e. <% %>, <%- %>, <%= %>
f. partials

48. **Rest API**
    a. RESTful
49. fragment identifier

50. VS

51. API vs HTTP
52. API vs SSR
53. HTTP vs HTTPS
54. URIs vs URLs vs URNs
55. Session vs Cookies
56. GET vs POST
57. PUT vs PATCH
58. SSL vs TLS
59. **Build-in Modules (only imp)**
    a. os
    b. path
        i. join()
        ii. basename()
        iii. resolve()
    c. fs
        i. fs sync
        ii. - readFileSync()
        iii. - writeFileSync()
        iv. - appendFileSync()
        v. - unlinkFileSync()
        vi. - statusSync()
        vii. - mkdirSync()
            1. recursive: true
        viii. **fs async**
        ix. - readFile( )
        x. - writeFile()
    d. http
        i. createServer()

MongoDB

1. **Theory**
2. SQL(relational) v s
3. NoSQL ()
4. What is MongoDB?
5. Run on JS Engine
6. How does mongoDB work?
7. Non-relational Document based
8. Advantage and Disadvantages
9. BSON
10. MongoDB Structure
11. MongoDB architecture
12. JSON vs BSON
13. MongoDB shell
14. CRUD Operations
15. Cursor, Iterate a Cursor
16. Time to Leave
17. Maximum Document Size : 16Mb
18. **Storage engines**
    a. **types**
        i. WiredTiger
        ii. ger engine
        iii. In-memory engine
        iv. MMAPv1
    b. GridFS
    c. Journal
19. **Data types in MongoDB (BSON)**
    a. ObjectId
        i. timestamp
        ii. random value
        iii. incrementing counter
    b. String
    c. Int, longInt, Double
    d. Array, Object
    e. Boolean
    f. Date
    g. Decimal128
    h. Regex
    i. Javascript
        i. with scope
        ii. without scope
    j. MinKey, MaxKey
    k. Binary data
20. Cursor
    a. cursor methods
    b. - toArray
    c. - forEach
    d. cursor.allowPartialResults()
21. **Collection**
    a. db
    b. db.createCollection(collectionName)
    c. show collections
    d. renaming Collection
22. **Documents**
    a. adding new Documents
    b. Nested Documents
        i. advantage
23. **Inserting Document**
24. Insert One and Many
25. what are the additional methods used for inserting
26. **Finding / Querying**
    a. find()+
        i. iterate (it)
        ii. pretty()
    b. findOne({ *filter* })
    c. finding In nested Array
        i. "*field.field*"
        ii. match
        iii. exact match
        iv. multiple match
    d. Array
        i. finding in specific order
        ii. without regard to order
        iii. query by array index
        iv. query by array length
    e. **Projection**
        i. explicitly include fields
    f. Null, $type: 10, $exists
27. **Filtering**
    a. find(*filter*)
    b. find(*{filter}, {fieldsToGet}*)
28. **Method Chaining**
    a. count()

    b. limit()
    c. sort(1 or -1)
    d. skip()
29. **Operators** (denoted by $)
    a. {$gt: number} $gte
    b. $lt, $lte
    c. $eq, $ne
    d. $or $and $not
    e. $in: [1,2,3], $nin: [1,2]
    f. $all
    g. $set, $unset
    h. $addToSet
    i. **$elemMatch**
    j. $slice
    k. $size
    l. $inc: 1, $inc: -1
    m. $pull, $push
    n. $each [1, 2]
    o. $eq, $ne
    p. $currentDate
    q. $exists
    r. **$expr**
    s. **$cond**
    t. $rename
    u. $min, $max
    v. $mul
    w. $ifNull
    x. $let
    y. **Array Operator**
        i. $push
        ii. $each
        iii. $pull
        iv. $pullAll
        v. $pop
        vi. $elemMatch
30. **Deleting**
    a. deleteOne({ *field:value* })
    b. deleteMany()
    c. remove()
    d. delete vs remove
31. **Updating**
    a. updateOne({*whichObject*} , {$set: {*field: value, field: value*} })
    b. **Operators**
        i. $set
        ii. $unset
        iii. $rename
    c. updateMany()
    d. replaceOne()
    e. incrementing & decrementing
    f. adding and remove from array
    g. upsert
    h. update() vs updateOne()
    i. updateOne vs replaceOne
32. **bulkWrite()**
    a. ordered: false
    b. ordered vs unordered
    c. advantages and disadvantages
33. **Commands**
    a. mongosh
    b. db
    c. show dbs
    d. db.stats
34. **Aggregation**
    a. How does it work
    b. advantages
    c. types of aggregation
    d. distinct
    e. **Aggregate stages**
        i. $addFields
        ii. $match
        iii. $group
            1. grouping by
            2. -nested field
            3. -multiple field
        iv. $sort
        v. $set
        vi. $count
        vii. - other ways to count
        viii. - client and server side counting
        ix. $limit, $skip
        x. $merge
        xi. $out
        xii. $project
        xiii. $lookup

xiv. $unwind
xv. $facet
xvi. $fill
xvii. $bucket
1. $bucketAuto
xviii. $densify
xix. $redact
xx. $search
xxi. allowDiskUse: true

f. “$name” vs ”name”
g. **Accumulator Operators**
i. $sum, $avg, $max, $min
h. **Unary Operators**
i. $type, $lt $gt $or $and $multiply
i. **Aggregation Pipeline**
i. How does aggregation pipeline work?
ii. memory limit : 100mb
1. spill to disk
j. Batch sizing
k. Iterator Size
l. Query routing
m. **Map Reduce**
i. for what is it used?
ii. find sum, avg

35. **Indexes**
a. pros and cons of Indexes
b. createIndex({ *filed: value* })
c. options when creating Index
i. background: true
ii. unique: true
iii. name: “<*indexName*>”
d. getIndex()
e. dropIndex(), dropIndexes
f. reIndex()
g. rename Index
h. hiding index
i. covered query
j. **Types of Indexes**
i. Single Field Index
ii. Compound Index
iii. Multikey Index
iv. Text Index
v. Geospatial, Hashed, Clustered Index
vi. Covered query
vii.

36. **Schema**
a. pros and cons of using schema
b. optional schema
c. validation action

37. **Relationships**
a. embedding
b. referencing
c. one-to-one
d. one-to-many
e. one-to-squillions
f. many-to-many

38. **Replication**
a. replica set
b. advantage and disadvantages of replication
c. **Replication Architecture**
i. primary and secondary nodes
ii. arbiter
iii. process of election
iv. heartbeat
d. Process of Election
e. Replication lag
f. operation log (oplog)
g. **Types of replication**
i. Asynchronous Replication
ii. Synchronous Replication
iii. Majority Commit
iv. etc...

39. **Sharding**
a. advantages and disadvantages
b. **Sharding Architecture**
i. What is Mongos/Router
ii. Config Server
c. **Types of sharding**

    i. Hashed sharding
    ii. Ranged sharding
    iii. Zone Sharding
  d. **Shard key**
    i. shard hotspots
    ii. normal shard key
    iii. hashed shard key
  e. Vertical and horizontal scaling
  f. Zones
  g. mongos
  h. auto balancer
  i. scatter-gather
40. **Cluster**
  a. types of cluster
  b. config servers
41. **Data Modeling**
  a. embedded data model
  b. reference data model
  c. linking vs embedding
42. **Transactions**
  a. How to do transaction
    i. **Session**
    ii. startTransaction
    iii. abortTransaction
    iv. commitTransaction
  b. ACID Transaction
  c. A- Atomicity
  d. C- Consistency
  e. I - Isolation
  f. D - Durability
43. Create view in Mongodb
44. CAP Theorem
  a. theorem
  b. C- Consistency
  c. A - Availability
  d. P - Particle tolerance
45. **Isolation levels**
  a. Read Concerns
  b. - local
  c. - maojiry
  d. - available
  e. Write Concerns
  f. - w:1 (Acknowledged)
  g. - w:0 (Unacknowledged)
  h. - majority
  i. - all
  j. - journaled
46. **VS**
  a. $or vs $in
  b. $all vs $in
  c. $elemMatch vs $in
  d. drop() vs remove()
  e. findAndModify() vs findOneAndUpdate()
  f. Primary key vs secondary key
  g. join vs lookup
  h. dot notation vs nested form
  i. $currentDate vs $$NOW
  j. delete() vs remove()
  k. bulkWrite vs InsertMany
  l. replace vs update
  m. shard vs node vs cluster
  n. Aggregation Pipeline vs Map Reduce
  o. vertical scalability vs horizontal scalability
  p. load balancer vs sharding
  q. odm vs driver
  r. stage operator vs accumulator operator
  s. normal shard key vs hashed shard key
  t. aggregate([$count:"tota"]) vs find({}).count()
  u. replication vs replica set
  v. transaction vs query
  w. scaling up vs scaling down vs scaling out?
  x. config servers vs mongos
  y. load balancer vs auto balancer
  z. countdocument vs count
47. What is a MongoDB driver?
48. Capped collection and it's advantages
49. Profiler
50. Explain

51. Soft deleting

52. Interview Question

53. What to do when your quireing becomes slow?
54. What to do when your files are getting very big?
55. How to condense large volumes of data?
56. How to search for text in MongoDB?
57. How does MongoDB schema change?
58. How can we Backup and Restore in MongoDB?
59. What are the pros and cons of Normalising Data in MongoDB

60. Good to Know

61. Atomicity
62. Type Bracketing
63. Dot Notation
64. Cursor behaviour
65. Aggregation Pipeline
66. Retryable Writes and Reads
67. MongoDB CRUD Concepts
68. B-Tree
69. ACID compliance
70. Mongoose
71. Network Components
    a. load balancer
    b. firewall
72. **CAP Theorem**
    a. consistency
    b. availability
    c. partition tolerance
73. Firewall
74. **Mongo Utilities**
    a. mongoexport
    b. mongoimport
    c. mongodump
    d. mongorestore
    e. mongostat
    f. mongotop
    g. mongooplog
75. Clustered collections
76. WAL

React

1. **Set up**
2. npx create-react-app <appName >
3. components
    a. default is App
4. rafce, tsrafce
5. calling function on button click
    a. without parameter
    b. with parameter
6. Fragments
7. Children Prop
8. **Theory**
9. What is React
10. DOM
    a. DOM vs Virtual DOM
    b. Reconciliation
        i. working
    c. Diffing Algorithm
    d. React Fibre
        i. incremental rendering
    e. Shadow DOM
11. Dynamic rendering
12. props vs state
13. Server Side vs Client Side Rendering in React
14. Synthetic Events
    a. Event Pooling
15. Life Cycle
16. View Oriented
17. Memoization
18. Pure functions and components
19. Strict Mode
20. SPAs vs MPAs
21. CSR vs SSR
22. Static vs Dynamic rendering
    a. ISR, SPA
23. **Components**
    a. A React render tree
        i. top-level components
        ii. leaf components
    b. Props
        i. immutable
    c. Forwarding props
    d. children
    e. Importance of making them pure
    f. local mutation
24. **JSX**
    a. Rules of JSX
    b. Fragment
    c. JavaScript in JSX
    d. HTML VS JSX
25. Conditional rendering
26. Key
27. **UI as a tree**
    a. Render trees
    b. Module Dependency Tree
    c. Bundler
        i. eg: Webpack
        ii. Compiling
        iii. Loader
        iv. Code splitting
28. **Rendering steps**
    a. Triggering
    b. Rendering
    c. Committing
29. Rerendering
30. Batching updates
31. **State**
    a. Behaviour
    b. Queueing updates
    c. Updater function
    d. Updating object
    e. local var vs state var
    f. local mutation
    g. Lifting state
    h. Reducer
32. Declarative vs Imperative UI
33. **Event handlers**
    a. onClick, onSubmit etc...d
    b. Stopping propagation
    c. Preventing default
34. Lifecycle Methods
    a. What is Mounting, Unmounting
    b. **Phases**
    c. **- Mounting phase**

i. constructor
ii. render
iii. getDerivedStateFromProps
iv. componentDidMount

d. - Updating phase

i. shouldComponentUpdate
ii. componentWillUpdate
iii. componentDidUpdate
1. getSnapshotBeforeUpdate

e. - Unmounting phase

i. componentWillUnmount

f. - Error Handling

i. getDerivedStateFromError
ii. componentDidCatch

35. Hooks

a. useState
i. changeValue
ii. changeValueWithFunction
b. useRef
i. html
ii. useState vs useRef
iii. forwardRef
iv. useImperativeHandle
v. flushSync

c. useEffect

i. dependency
ii. return in useEffect
iii. useLayoutEffect

d. useMemo
i. sample
ii. recache
iii. pros and cons
iv. referential equality

e. useHistory
i. push
ii. pop
iii. replace
iv. Redirect

f. useNavigate
i. navigate()
1. route
2. -1, 1

g. useCallback
i. sample
ii. useMemo vs useCallback
iii. uses

h. useContext
i. sample

i. useReducer

j. Create custom hooks

i. useDebugValue

k. useTransition
l. useDeferredValue
m. useId
i. sample

n. useImperativeHandle

36. Props

a. default prop
b. PropDrilling
c. Children

37. Components

a. Creating Components
b. Controlled vs Uncontrolled Components
i. Inputs

c. Higher order components
d. Pure components

38. React Router

a. install

b. Hooks

i. useHistory
ii. useNavigate

c. use

d. Link

i. replace
ii. reloadDocument
iii. state={}
iv. - useLocation()

v. NavLink

1. -isActive
2. end

vi. Navigate

1. useNavigate

2. navigate(-1)

e. Types of Router

i. BrowserRouter
ii. HashRouter
iii. HistoryRouter
iv. MemoryRouter
v. StaticRouter
vi. NativeRouter

f. params (:id)
g. cont {<name>} = useParams()
h. useSearchParams

i. Nesting Routes

i. index
ii. location
iii. shared element with children
iv. outlet
v. - useOutletContext()
vi. Nesting in separate file
vii. useRoute

39. Good to Know

40. Immer
41. Object.entries(e)
42. Icons
43. Experimental Hooks
    a. useEffectEvent
    b. use
    c. useFormStatus
44. useOptimistic

45. Week 2

46. Render props
47. Higher order components
48. Custom hooks
49. Code splitting
    a. Route based
    b. Component based
    c. React.lazy
50. Higher order comps
51. **Lazy Loading**
    i. fallback ui
    ii. suspense
    iii. **Error boundaries**
    iv. componentDidCatch
    v. Fallback UI
    vi. Nested & Propagation
52. **useReducer**
    a. Dispatch
    b. useReducer vs useState
    c. useReducer vs redux
    d. payload
53. **PropTypes**
    a. types => name, string, any
    b. required, optional,
    c. node, element type
    d. oneof, shape
    e. PropTypes vs Typescript
54. **useMemo vs useCallback**
    a. React.Memo vs useMemo
    b. Object reference
    c. Pros and cons of memoization
55. **Context API**
    a. Provider
    b. Consumer
    c. useContext
    d. useReducer
56. **Webpack**
    a. Module Bundler
    b. Code Splitting
    c. Webpack Dev Server
    d. Hot Module Replacement (HMR)
    e. Tree Shaking
57. **Babel**
    a. Transpilation
    b. Plugins
    c. Runtime Polyfills
    d. Dynamic Import
58. useDeferedValue
59. dead code elimination
60. useTransition

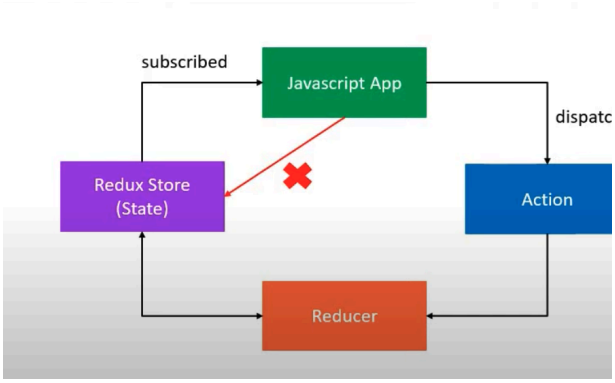
61. Others

a. forward ref
b. useDebugValue
c. useImperativeHandle
d. Axios interceptor
e. Concurrent Requests

i. axios.all(), axios.spread()
ii. cancel Token

Redux

62. ## Theory
63. Why, what
64. Redux
65. How redux stores data
66. Architecture
67. Store
68. pros and cons
69. Redux store
70. Middleware
71. Calling APIs
72. React reducer vs Redux
73. **Store**
    a. Dispatch
    b. subscribe
        i. unsubscribe
    c. getState
    d. replaceReducer
    e. Store enhancer
74. **Action**
    a. Action creator
75. **Reducer**
    a. rules
76. **Redux flow**



77. **Redux principles**
    a. Store
    b. Action
    c. Reducer
78. Selectors
    a. Memoized selector
79. **Middleware**
    a. - Logger, crash reporting
    b. - Perform async tasks
    c. applyMiddleware
    d. Redux Thunk
        i. Thunk vs saga
        ii. Payload creator
    e. Adding multiple middleware
80. **Slice**
    a. init state
    b. reducers
    c. extraReducers
81. **Redux toolkit**
    a. Nanoid
    b. Redux Query.
82. Normalising Data
    a. Normalised state
    b. createEntityAdapter
    c. shallowEqual, reference equality
83. Serializing
84. Hydrating
85. redux vs flux
86. saga vs thunk

87. ## Other

88. Immer and the working of Immer in redux.
89. Access store outside of redux components
90. Flux by fb
91. Log rocket
92. createAsyncThunk
93. createEntityAdapter
94. createSelector
95. createListenerMiddleware

96. ## JWT

97. What?
98. **Structure**
    a. Header
    b. Payload
        i. iat
        ii. exp/eat
    c. Signature
99. Authentication working
100. Pros and cons
101. Expiration Time
102. Bearer token

103. Revocation
104. refresh token
105. Authentication vs Authorization
106. Types of Claims
    a. public
    b. registered
    c. private

React Native

Components

1.

Components

1. Text
2. View
    a. default flexbox layout
3. TouchableOpacity
4. TouchableHighlight
5. TouchableWithoutFeedback
6. ActivityIndicator
7. Button
8. Flatlist
    a. optimises scroll performance
    b. Item separation
9. Flatlist vs Map
10. ScrollView
11. SafeAreaView
12. Image
    a. ImageBackground
    b. react-native-svg
13. Modal
14. Alert
15. Switch
16. StatusBar

Styling

17. style
18. StyleSheet Utility
19. NatvieWind

DSA

1. ## Algorithms
    - **Search**
    - Binary Search(recursive also)
    - Linear Search
2. Recursion
3. Iterative & recursive
4. Virtual memory
5. Amortised resizing
6. Dynamic programing
    - Memoize approach
    - Bottom up approach
7. ## Problems
    - Factorial, fibonacci, prime number (with and without recursion)
8. ## Complexity Analysis
    - Time complexity
    - Space complexity
9. ## Asymptotic Notations
    - Ranking
    - Big O notation
    - Omega Notation
    - Theta Notation
10. ## Memory
11. **Memory Allocation**
    - Bit vs byte
    - Memory address
    - Contiguous memory allocation
    - Non-contiguous memory allocation
    - **Stack**
        i. Primitive types are stored in stack
    - **Heap**
        i. Reference type are stored in heap
        ii. Eg: Arr, fun, obj
12. **Memory Leak**
    - Symptoms
    - **Garbage Collections**
        i. Process
    - Reasons for memory leak
    - How to debug
13. **Big O Notation**
    - Linear time complexity
    - Constant time complexity
    - Quadratic time complexity
    - Qubic
    - Logarithmic complexity
    - Exponential complexity
14. **Operations in normal array**
    - Init
    - Set
    - Get
    - Traverse
    - Insert
    - Delete
15. ## Data Structures
16. What is DS?
17. Advantages and Disadvantages
18. Examples
    - DOM
    - Undu & Redo
    - Os job scheduling
19. **Dynamic Array**
    - It's working and memory allocation?
    - Set
20. **Linked List**
    - Advantages and disadvantages
    - Applications
    - **Creating a linked list**
    - **Operation**
        i. Init
        ii. Set
        iii. Get
        iv. Traverse
        v. Insert
        vi. Delete
    - Singly Linked List
    - Double linked list

- Circular linked list
- Array vs linked list

21. # OTHERS

22. **Build in DS in JS**
    - **Array**
        i. Push, pop, shift, unshift, forEach, map, filter, reduce, concat, slice, splice ,sort()
        ii. some(), every(), find(), findIndex(), fill(), flat(), reverse(), sort()
    - **Objects**
        i. Insert, Remove, Access, Search,
        ii. Object.keys(), Object.values(), Object.entries()
    - **Sets**
        i. add, has, delete, size, clear
    - **Maps**
        i. set, get , has, delete, size, clear
    - Array vs Set
    - Object vs Map
    - **Strings**
        i. Primitive and object string
        ii. Escape char
        iii. ASCII
            1. 32 - Space
            2. 48-57 == (0-9)
            3. 65-90 == (A-Z)
            4. 97-122 == (a-z)
        iv. Unicode
        v. UTF-8

23. **Custom DS**
    - Stacks
    - Queue
    - Circular queues
    - Linked lists
    - Hash tables
    - Trees
    - Graphs

Intermediate

24. ## Algorithms
    - **Sorting**
    - Bubble sort
    - Insertion sort
    - Quick sort
        i. Divide and conquer
        ii. Partition method
        iii. Pivot selection
        iv. Last, first
        v. average/median
    - Heap sort
    - Merge sort
        i. Divide and conquer
    - Merge vs Quick sort

25. ## Data Structures

26. **Stacks**
    - LIFO
    - Push, pop
    - Stack underflow
    - Stack overflow
    - Use cases
    - **Types of Stack**
    - Linear Stack
    - Dynamic Stack
    - Array-based
    - Linked list based
    - Monotonic stack

27. **Queue**
    - FIFO
    - Enqueue
    - Dequeue
    - Peek
    - Priority queue
    - Circular queue
    - Uses
    - **Types of Queue**
    - - Linear Queue
    - - Circular Queue
    - - Priority Queue

- - DEqueue (Double ended queue)
    i. Input restricted
    ii. Output restricted
  - - Blocking Queue
  - - Concurrent Queue
  - - Delay Queue

28. **Hash Table**
  - Searching O(1)
  - Hash function
  - Collision
  - Dynamic restructuring
  - Uses
  - Load factor
  - **Operations**
  - Init
  - Insert
  - Search
  - Delete
  - Traverser
  - **Please Note**
  - Week set, week map
  - **Collisions Handling**
  - - Separate Chaining
  - - Open Addressing
    i. Linear Probing
    ii. Quadratic Probing
    iii. Double Hashing
    iv. Clustering
  - - Cuckoo hashing
  - - Robin Hood hashing
29. **SHA: Secure Hashing Algorithm**

Advanced

30. Linear, non-linear, hierarchical
31. ## Data Structures
32. **Tree**
  - Features
  - Uses
  - parent, child, root, leaf, sibling, ancestor, descendent, path, distance, degree, dept, height,edge,subtree
  - **Types of trees on nodes**
  - - Binary tree
  - - Ternary tree
  - - K-array tree
  - - Threaded binary tree
  - **Types of trees on structure**
  - - Complete tree
  - - Full tree
  - - Perfect tree
  - **- Degrenarted**
    i. Left-skew
    ii. Right-skew
33. **Binary Search Tree (BST)**
  - BST vs BT
  - Uses
  - Balanced vs unbalanced tree
  - Properties of BST
  - **Operations**
  - - Inserting
  - - Deletion
  - **- Traversal**
    i. DFS
    ii. - InOrder
    iii. - PreOrder
    iv. - PostOrder
    v. BFS
34. **Balanced Search Tree**
  - AVL tree
  - Red-black tree
  - Prefix tree
  - M-way search tree
  - - B Tree
  - - B+ Tree
  - Merkle Tree
  - Red-black tree vs AVL
35. **Heap**
  - Min Heap
    i. **To get value of**
    ii. - Left child
    iii. - Right child
    iv. - Parent
    v. Operations
    vi. - Init/ Heapify

vii. - Insert
viii. - Delete
- Max Heap
- Heapfity
  i. Bottom-up
  ii. Top-down
- DEPQ

36. **Trie**
- String vs Trie
- **Operations**
- - Init
- - Insertion
- - Delete
- - Search
- Prefix and Suffix tree
- - terminator char
- **Compressed Trie**
- - Radix Tree (Patricia Trie)

37. **Graph**
- Vertex, Edge
- Can be stored as
- - Adjacency list
  i. as linked list
  ii. time O(V)
  iii. space O(V+E)
- - Adjacency matrix
  i. As array
  ii. time O(1)
  iii. space O(v^2)
- Spanning tree
  i. min spanning tree
- Graph indexing
  i. Vertex-centric indexes
  ii. Edge-centric indexes
- **Types**
- - Unidirectional (Direct graph)
- - Bidirectional (Un DIrected graph)
- - Cyclic
- - Disconnected
- - Weighted Graph
- - Unweighted Graph
- - Bipartite Graph
- **Traversal**
  i. BFS
  ii. DFS
- River size problem

38. **Algorithms**

39. Greedy method
40. Kruskal's Algorithm
41. Prim's Algorithm
42. Dijkstra's Algorithm
43. Bellman-Ford Algorithm
44. Topological Sorting
45. Floyd-Warshall Algorithm
46. Bipartite Graph Checking
47. Max Flow (Ford-Fulkerson Algorithm)

48. **Question**

49. Graph vs Tree
50. Forest (in Tree)
51. Forest > Graph > Tree > Linked list
52. Operators
    - Binary operators
    - Priority
    - Infix
    - Prefix (Polish notation)
    - Postfix (Reverse Polish notation)

General

1. How does Logarithms work
2. File structure vs Data Structure
3. Where is the DS used?
4. Void vs null
5. Dynamic data structure
   a. Uses
   b. Example
6. Dynamic memory management/ allocations
7. Heap be used over a stack
8. Data abstraction
9. Post fix expression
10. Signed number
11. Pointers in DS
    a. Uses
12. Huffman’s algorithm working
13. What is recursive algorithm

    a. Divide and conquer on recursion
14. Which is the fastest sorting algorithm available?
15. Multi linked
16. Sparse matrices
17. Disadvantages of implementing queues using arrays
18. Void pointer
19. Lexical analysis
    a. Lexeme
    b. Pattern

Hosting

1. ## Nginx

2. **Commands**
    a. systemctl nginx status
    b. restart and reload
3. Contex
    a. Eg: http, events, server
    b. Worker process and connection
    c. Directive & block
    d. Location block
        i. root, alias, try_files
4. Master Process
5. Worker Process
6. Firewall
7. DDOS protection
8. K8s IC
9. Sidecar proxy
10. Virtual host
11. Brute force
12. WAF
13. UFW
14. TCP vs UDP
15. TCP vs TCL
16. **Load Balancing**
    a. Round robin
    b. Least connection
    c. IP hash
17. Caching
18. **Proxy**
    a. Proxy server
    b. Reverse proxy
    c. Forward proxy
    d. Load balancer vs reverse proxy
19. Nginx vs Apache
20. Working of HTTPS

21. ## SSH

22. How does it work??
23. Private key
24. Public key

25. ## SSL

26. How does it work??

27. ## Linux

28. apt
29. rm
30. mkdir
31. touch
32. mv
33. nano
34. more, less
35. head, tail
36. >, <
37. /
    a. bin
    b. boot
    c. dev
    d. etc
    e. home
    f. root
    g. lib
    h. var

System Design

1. Dark scale distributed system
2. Scaling
    a. Vertical
    b. Horizontal
    c. Auto
        i. pros and cons
3. Asynchronous system
    a. Queue system
4. Scaling database and server
5. Rate limiter vs Limiter
6. **API Rate Limiting**
    a. Token Bucket
        i. pros and cons
    b. Leaky Bucket
7. Concereny controller
8. Handling Failure

Theory

9. **Components of System Design**
    a. Logical
        i. Data
        ii. Database
        iii. Users
        iv. Applications
        v. Cache
        vi. Communication protocol
        vii. Infra
        viii. Message queues
        ix. Presentation layer
    b. Tangible
        i. data - Text, image
        ii. database - SQL, NoSQL
        iii. App - Java, node
        iv. Cache - Reddist, mem-cache
        v. Infra - AWS, GCP
        vi. Comm - API, RPC, Message
        vii. Queues - Kafka, RabbitMQ
10. **Client Server Arch**
    a. Thick and Thin client
    b. 2-Tier, 3-Tier, N-Tier Client
11. **Fault and failure**
    a. Fail safe
    b. Fault tolerant
    c. graceful fail
    d. Testings
    e. Transient vs Permanent fault

Other

12. Critical and non critical tasks

Git

53. **THEORY**

54. **Centralised** Version control system vs **Distributed** Version control system
55. Config
56. Working directory
57. Staging area
58. git init
59. git clone
60. git status
61. git log
62. **Creating Version**
    - git add *file*
        i. git add - - all
        ii. git add .
    - **git commit**
        i. -m "<*message*>"
        ii. Commit without staging
    - commit id
        i. check sum
        ii. content
            1. author details
            2. preview details
            3. date
            4. etc..
        iii. sha-1 hash
    - label
    - **branch**
63. touch
64. **git log**
    - git log
    - git log - - all
    - git log –p -1
    - git log graph
65. git diff
66. git diff –staged
67. **Restore**
    - git restore
    - git restore –staged
68. **Branching**
    - git branch <*branchName*>
    - git branch
    - git branch —all
    - Creating branch
    - Deleting branch
    - git checkout vs git switch
    - switching b/w branches
    - commit id
    - branch name
69. **Stashing**
    - git stash
    - git stash apply
    - git stash drop
    - git stash list
70. **Merging**
71. git merge <*branchName*>
72. **Types of merging**
    - fast-forward merge
    - **recursive merge**
        i. conflict
73. **Git server**
    - git remote add <*name*> <*url*>
        i. git remote
        ii. git remote -v
    - git push <*remoteName*> <*branchName*>
    - git push set upstream
    - **Cloning**
    - git clone <*url*>
    - git pull
    - pull vs pull request?
    - pull vs fetch
74. **Tags**
    - Simplified
    - Annotated
    - git tag
    - Should Pushing tags
75. **Forking**
76. git rebase
77. vim .gitignore
78. gist
79. **ci cd**
80. git projects

81. ## GOOD TO KNOW

82. rebase
83. tree

SQL: Postgres

1. **Theory**
2. SQL vs NoSQL (Relational vs non-relational)
3. Web-scaled
4. When to use SQL and NoSQL
5. Expression, Statement, Operators
6. **Data types SQL**
    a. null, bit
    b. int, real / float
    c. char, varchar, text
    d. boolean
    e. date, datetime, timestamp
    f. xml/json
    g. – char vs varchar vs text
    h. – datetime vs timestamp
    i. – JSON vs JSONB
7. **Operators**
    a. Arithmetic, Logical, Comparison, Bitwise
8. Primitives: Integer, Numeric, String, Boolean
9. Structured: Date/Time, Array, Range / Multirange, UUID
10. Document: JSON/JSONB, XML, Key-value (Hstore)
11. Geometry: Point, Line, Circle, Polygon
12. Customizations: Composite, Custom Types
13. **Postgres**
14. Forks
15. client/server model
16. **Data types Unique to Postgres**
    a. interval
    b. point
    c. bigserial
    d. etc...
17. Database cluster
18. **Constraints**
    a. UNIQUE
    b. NOT NULL
    c. PRIMARY KEY
        i. as UUID
    d. FOREIGN KEY
    e. CHECK (<condition>)
    f. - Adding & removing constraints after creating table
19. **Commands**
    a. list db
    b. to connect
    c. list tables
    d. Move to super
    e. list specific table
    f. List current table
20. Creating
    a. Database
    b. Table
21. Drop
    a. Drop DB
    b. Drop Table
    c. Drop constraints
22. Commands
        i. – or /* */
    b. Database migration
        i. Add, Delete, Migration
        ii. Up migration
        iii. Dow migration
23. **Functions**
    a. SELECT
        i. LIMIT
        ii. FETCH
        iii. OFFSET
        iv. AS
        v. DISTINCT
        vi. GROUP BY
            1. HAVING
            2. GROUPING SETS
            3. ROLLUP
            4. CUBE
        vii. Having vs Where
        viii. Limit vs Fetch

b. FROM
c. WHERE
    i. AND, OR
    ii. LIKE, ILIKE
    iii. BETWEEN
    iv. IN
    v. IS NULL, IS NOT NULL
d. ORDER BY
    i. DESC, ASC
e. DELETE
f. DELETING FOREIGN KEY
    i. CASCADE
g. UPDATE
    i. SET
h. RENAME COLUMN
i. **JOIN**
    i. INNER JOIN
        1. ON
    ii. LEFT JOIN
    iii. RIGHT JOIN
    iv. FULL JOIN (FULL OUTER JOIN)
    v. SELF JOIN
    vi. CROSS JOIN
    vii. NATURAL JOIN
j. VIEWS
    i. Pros and Cons
    ii. CREATE VIEW
    iii. Materialized View
        1. Write amplification
k. UNION
l. COALESCE
m. NULLIF
n. Index
    i. multi index

24. AUTO_INCREMENT
25. ON CONFLICT
    a. DO NOTHING
    b. Upserting
    c. - DO UPDATE
        i. EXCLUDED
26. Date functions
    a. INTERVAL vs AGE
27. Aggregate functions
    a. AVG, MIN, MAX, SUM, ROUND, COUNT, CONCAT
28. Scalar Functions
    a. LCASE, CASE, LEN, MID, ROUND, NOW, FORMAT ,
    b. INITCAP , LEFT , RIGHT , CONCAT , ABS , CEIL , FLOOR,
    c. UPPER AND LOWER in psql.
29. Aggregate vs Scalar
30. Window function
    a. OVER
    b. - PARTITION BY, RANK, LEAD, LAG
    c. CASE
31. SQL Commands
    a. DDL
        i. CREATE, ALTER, DROP, TRUNCATE
        ii. DROP vs TRUNCATE
    b. DML
        i. INSERT, SELECT, UPDATE, DELETE
    c. **DCL**
        GRANT, REVOKE
    d. TCL
        i. COMMIT
        ii. ROLLBACK
        iii. SAVE POINT
    e. DQL
        i. SELECT
32. 3-Schema architecture
    a. Internal level
    b. Conceptual level
    c. External level
33. BIGINT VS BIGSERIAL
34. Combining queries
    a. UNION, UNION ALL
    b. INTERSECT, INTERSECT ALL
    c. EXCEPT, EXCEPT ALL
35. Normalisation
    a. Levels
        i. 1NF, 2NF, 3NF etc..
        ii. BCNF
    b. Anomalies

c. - Insertion anomalies
    i. Data redundancy
    ii. Missing data
d. - Deletion anomalies
    i. Losing data
e. - Updation anomalies
    i. inconsistency
    ii. Updating values on so many records unnecessarily

36. **Relationship**
    a. one to one
    b. one to many
    c. many to may
37. **Transaction & ACID**
38. **- Transaction**
    a. COMMIT
    b. ROLLBACK
    c. SAVE POINT
        i. RELEASE SAVEPOINT
    d. LOCK
        i. Exclusive Locks (X-Locks)
        ii. Shared Locks (S-Locks)
        iii. Update Locks (U-Locks)
        iv. Intent Locks
        v. Read and Write Locks
39. **- ACID**
    a. - Atomicity
    b. - Consistency
        i. Consistency in data
        ii. Consistency in reads
    c. - Isolation
        i. Read phenomena
        ii. - Dirty reads
        iii. - Non-repeatable reads
        iv. - Phantom reads
            1. Serialotions
        v. - (Lost updates)
        vi. Isolation level
        vii. - Read uncommitted
        viii. - Read committed
        ix. - Repeatable Reads
        x. - Transactions are Serialized
    d. - Durability
    e. How to implement ACID properties
40. EXPLAIN
41. Heap Scan
42. Parallel Scan
43. Planner

44. Other theory and functions

45. COPY
46. OLTP
47. MUCC

48. Pendings

49. Delete vs truncate
50. candidate key vs super key
51. stored procedure
52. ER diagram.
53. Practice nested queries.

Microservice

Concepts & Theory

20. What is a service?
21. Monolithic arch
    a. pros and cons
22. Microservice arch
    a. pros and cons
23. **Monolithic vs Microservice**
    a. deployment, scalability, reliability, development, flexibility, debugging
24. Security
25. **Cloud computing**
    a. Public IP address
    b. On-premises
    c. Iaas, Cass, Pass, Faas (Server less computer), Saas
    d. Private could
    e. Hybridge cloud
26. Scaling
27. Blue Green Deployment
28. Cloud Native vs Cloud Ready
29. Event-Driven Architecture
    a. Event producer
    b. Event broker
    c. consumer
    d. pub/sub
    e. eventual consistency
    f. cache layer
    g. idempotent
30. 12 Factor App
    a. Codebase
    b. Dependencies
    c. Config
    d. Backing services
    e. Build, release, run
    f. Processes
    g. Port binding
    h. Concurrency
    i. Disposability
    j. Dev/prod parity
    k. Logs
    l. Admin processes
31. Load balancing
    a. Round robin
    b. Least connection
    c. IP hash
32. Service Registry
33. Failed fast
34. Service Discovery
35. **Tools**
    a. os
    b. language
    c. api management
        i. postman
    d. messaging
        i. kafka
        ii. rabbitMQ
    e. toolkits
        i. fabric8
        ii. seneca
    f. orchestration
        i. kubernetes
        ii. Istio
    g. monitoring
        i. prometheus
        ii. logstash
    h. serverless tools
        i. claudia
        ii. AWS lambda
36. **Principles behind microservices**
    a. Independent and autonomous service
    b. Scalability
    c. Decentralisation
    d. Resilient services
    e. Real time load balancing
    f. Availability
    g. CICD
    h. Continuous monitoring
    i. Seamless API integration
    j. Isolation from failures
    k. Auto provisioning
37. **Security**
    a. Defence in depth mechanism

    b. Token and API gateway
    c. Distributed tracing
    d. First session
    e. Mutual SSL
    f. OAuth
38. API gateway
    a. client performance
    b. security
    c. rate limiting
    d. monitoring logging
    e. BFF
39. SOA vs Microservices
40. **Communication**
    a. Types
        i. synchronous blocking communication
        ii. asynchronous non blocking communication
    b. Request response
        i. REST over HTTP
        ii. RPC
    c. Event driven
        i. kafka

Design Patterns

1. need?
2. Aggregator
3. **API gateway**
4. Chained or chain of responsibility
5. Asynchronous messaging
6. Orchestration vs Choreography
7. **Database pattern**
    a. Database Per Service
    b. Shared Database
8. Event sourcing
9. Branch
10. Multi-tenant
    a. pros and cons
11. **CQRS**
12. **Circuit breaker**
13. SAGA
    a. Choreography
    b. Orchestration
14. Decomposition
    a. Vine or Strangle
15. **Database**
    a. Decentralised Data Management
        i. pros and cons
    b. **Data Consistency in microservice**
        i. Saga Pattern
        ii. Event-Driven Architecture
        iii. CQRS
        iv. Idempotent Operations
        v. Consistency Models
    c. Database per Microservice
    d. Shared Database
    e. Data Virtualization
    f. Distributed Data Mesh
16. **CI/CD**
    a. Github actions
    b. pros and cons
    c. running in parallel
    d. **Testing**
        i. unit tests, integration tests, and end-to-end tests.
    e. Artefact Repository
        i. JFrog
17. **Github actions**
    a. Workflows
    b. Events
    c. Jobs
    d. Actions
    e. Runners
    f. Using variables in your workflows
    g. Sharing data between jobs
        i. artefacts
            1. actions/download-artifact
    h. Literals
    i. Contexts
        i. uses
        ii. Context availability
        iii. github context

iv. env context
v. var context
vi. job context

j. Polyglot Persistence

18. **- commands**
    a. name
    b. on
        i. push
            1. branches
    c. jobs
        i. needs
        ii. steps
        iii. uses
        iv. with
        v. run
        vi. if
        vii. matrix
        viii. outputs
19. **Transactions in microservice**
    a. Two-phase commit
        i. voting phase
        ii. commit phase
        iii. pros and cons
    b. SAGA
        i. backward recovery
        ii. forward recovery
    c. correlation id
    d. imp of logging and monitoring

Docker

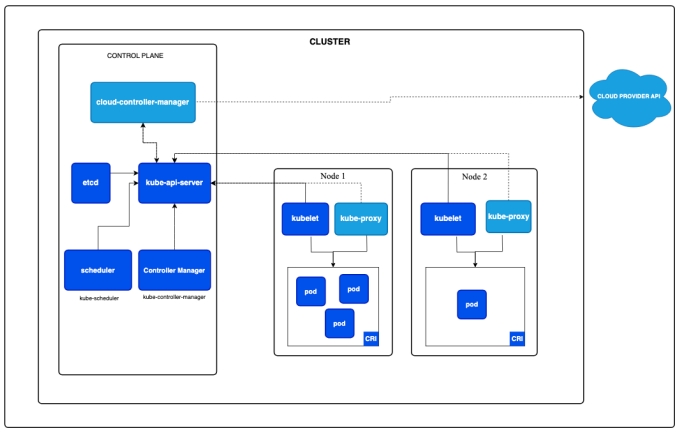
1. What, Why, When
2. Architecture
    a. client and server
    b. - server => docker engine
3. Container
    a. kernel namespaces
    b. C groups
    c. Container vs Virtual machine
4. Images & Container
    a. image vs container
    b. Isolated process
5. **Images**
    a. Image layers
    b. - base image layer
    c. - instruction layers
    d. - writable container layer
    e. Layer caching
6. docker run <ubuntu> vs docker pull <ubuntu>
7. Port mapping
8. Data persistence
9. DB Migration
10. Bind mounts.
11. run, start, rm
12. -t, -p

13. Commands

14. docker init
15. docker tag
16. docker build
    a. -t
    b. buildx
17. docker run
    a. --name
    b. -it
    c. -e
    d. -d
    e. -p
        i. port mapping
    f. --net
    g. --rm
18. docker container
    a. ls
    b. stop
        i. -t
    c. prune
    d. rm
        i. -f
19. docker logs <container>
    a. --follow/ -f
20. docker image
    a. ls
    b. history
        i. --no-trunc
21. docker network
    a. ls
    b. create <name>
        i. -d
        ii. --subnet
        iii. --gateway
22. **Manage containers**
    a. Docker container ls || docker ps
    b. Docker container ls -a || docker ps -a
    c. * Start
    d. * Stop
    e. * Restart
    f. * rm
    g. Docker system prune -a
23. **Network commands**
    a. Docker network ls
    b. Docker inspect bridge
24. **Volume**
    a. types
    b. - bind mounts.
    c. - volume mounts/ named volumes
    d. bind vs named mounts
    e. scratch space
    f. Volume claim
    g. docker volume
        i. create
        ii. inspect
    h. docker rm -f
25. dockerignore

26. **Docker hub**
    a. docker
        i. pull
        ii. push
        iii. rmi
27. **Docker compose**
    a. docker compose
        i. up
        ii. down
        iii. watch
        iv. ps
    b. services
        i. image
        ii. ports
        iii. environment
        iv. restart
            1. always
            2. on-failure
            3. unless-stopped
        v. depends_on
        vi. resources
            1. limits
            2. reservations
        vii. volume mapping
            1. read only, write only
    c. networks
    d. secrets
    e. volumes
        i. driver
28. **Dockerfile**
    a. FROM
    b. COPY
    c. WORKDIR
    d. RUN
    e. CMD
    f. EXPOSE
    g. ENTRYPOINT
    h. ENV
    i. ARG
    j. USER
    k. LABEL
    l. RUN VS CMD
29. **Docker network**
    a. Bridge
    b. Host
    c. None
    d. overlay
    e. macvlan
    f. IPvlan
30. Docker daemon

Kubernetes



31. aka k8s
32. pros
    a. other pros from doc
33. imperative vs declarative
34. self heading/ auto-heal
35. scaling, auto-scale
    a. HorizontalPodAutoscaler
36. cluster
37. context
38. namespaces
39. annotation
40. namespaces vs annotation vs labels
41. Finalizers
42. Node
    a. master node
    b. worker node
    c. node pool
    d. Node status
    e. Node heartbeats
    f. Node controller
        i. what it does
        ii. CIDR block
    g. Node topology
    h. Graceful node shutdown
        i. grace period
        ii. non-graceful shutdown
43. **Pod**
    a. communicate via
    b. ephemeral
    c. atomic
    d. scaling
    e. **Pods life cycle**
        i. when creating
        ii. when deleting
            1. grace period
    f. **Pod state**
        i. pending
        ii. running
        iii. succeeded
        iv. failed
        v. unknow
        vi. CrashLoopBackOff
    g. init container
    h. **Multi container pods**
        i. sidecar pattern
        ii. ambassador pattern
        iii. adaptor pattern
44. Container
    a. Images
    b. - Serial and parallel image pulls
    c. - image pull policy
    d. Container Environment
    e. Container Lifecycle Hooks
        i. PostStart
        ii. PreStop
45. Kubelet
46. Selectors
    a. metadata > labels
    b. spec > selector
47. **Workloads**
    a. pod
    b. replicaSet
        i. self-heading
        ii. template
    c. deployment
        i. replicas
        ii. revisionHistoryLimit
        iii. **Strategy**
            1. **RollingUpdate**
            2. - maxSurge
            3. - maxUnavailable
            4. - default
            5. - rollback

6. - rollout
7. **Recreate**

d. daemonSet
   i. daemon controller
   ii. uses
   iii. spec > toleration
e. statefulSet
   i. persistent identifier
   ii. creation & deletion
   iii. uses
   iv. headless service
f. job, cron job
g. replicaSet vs deployment
h. pods vs deployment

48. Volumes
    a. persistent volume
       i. claim
       ii. HostPath
       iii. drawback
       iv. reclaim policies
           1. delete (default)
           2. retain
       v. access modes
           1. ReadWriteMany
           2. ReadOnlyMany
           3. ReadWriteOnce
       vi. states
           1. available
           2. bound
           3. released
           4. failed
    b. storage class
    c. static and dynamic
49. Objects
50. ConfigMap
    a. static
    b. solve static with volume
51. Secret
    a. type
52. **Service**
    a. clusterIP
       i. port
       ii. targetPort
    b. nodePort
    c. load balancer
       i. L4
       ii. round robin
    d. ingress
       i. L7
53. NodePort
54. **k8s Cluster arch**
    a. **Node**
       i. container runtime
           1. containerized
           2. CRI-O
       ii. kubelet
       iii. kube proxy
    b. **Control Plane / Master node**
       i. kube-api server
       ii. kube-scheduler
           1. factor when scheduling
       iii. Kube controller manager
           1. built-in controllers
           2. Node controller
           3. job controller
           4. endpointSlice controller
           5. serviceAccount controller
       iv. Cloud controller manager
       v. ETCD
       vi. Addons
       vii. - DNS
       viii. - WEBUI (dashboard)
       ix. - cluster level logging
       x. - container resource monitoring
55. Cluster > Node > pod > container
56. CRI
57. Garbage Collection
58. Mixed Version Proxy
59. KubeCTL
60. Minikube
    a. rollout
61. Open Service Broker.

62. Ingress
63. Docker Swarm vs Kubernetes

64. Security

65. Image

a. Untrusted registries
b. Vulnerabilities in tools of OS or libraries

66. Authentication & Authorization
67. practices
    a. use linear images
    b. image scanning
    c. don't use root user
    d. manage user and permission
        i. RBAC
68. statefulSet
    a. master
    b. slave

69. Yaml

70. apiVersion
71. kind
72. metdat
    a. name
    b. label
    c. namespace
73. spec
    a. containers

74. Commands k8s

a. alias k=kubernetes
b. k get
    i. pods
    ii. svc
    iii. deploy
c. k delete -f <deployment.yaml> -f <service.yaml>
d. k exec <pod> – nslookup <svc>

75. k config
    a. current-context
    b. get-contexts
    c. use-context <name>
    d. delete-context <name>
76. namespace
    a. k get ns or namespace
    b. k create ns <name>
    c. k delete ns <name>
    d. k config set-context --current --ns=<namespace>
    e. k get pods -n <namespace>
77. node
    a. k get nodes
    b. k describe node
78. Probes
    a. startup
    b. readiness
    c. liveness

79. Good to know

80. grep
81. docker compose watch - https://www.youtube.com/live/I-htDVxmFGM?si=5Um3NCnMi0BeAgCz
82. chroot
83. Service Mesh

Message Broker

Kafka

1. used as key value but stored as binary in kafka
2. default port
3. serialisation and deserialization
4. pros and cons
5. Kafka cluster
    a. Fault Tolerance
    b. Scalability
    c. Distributed Processing
6. **Kafka Broker**
    a. topics
        i. compacted topics
    b. partitions
        i. leader
        ii. follower
        iii. replication
            1. replication factor
            2. key
    c. segments
7. **Producer**
    a. record
        i. header
        ii. key
        iii. value
        iv. timestamp
    b. retention period
    c. ack /nack
        i. no acks
        ii. leader acks
        iii. all acks
8. **Consumer**
    a. Queue vs Pub Sub
    b. Consumer group
9. Offset
10. Connectors
11. At most once
12. At least once
13. Exactly once
14. Exactly-Once Semantics
    a. Idempotent
    b. Two-Phase Commit
    c. alt
15. Persistent storage
16. Steam processing
17. Distributed system
    a. leader
    b. follower
    c. zoo keeper
        i. Metadata Management
        ii. Leader Election
        iii. Synchronisation
        iv. Heartbeats and Timeouts
        v. Monitoring
        vi. default port
        vii. gossip
18. long polling
19. Kafka Connect

RabbitMQ

84. TCP
85. HTTPv2
86. AMQP
87. RabbitMQ server
    a. default port
    b. Exchange Queues
88. Heartbeats
89. Connection pool
90. Channels
    a. Multiplexing
    b. Concurrency
91. Message TTL
92. Message Acknowledgment
    a. **Strategies**
    b. Automatic Acknowledgment (Ack)
    c. Positive Acknowledgment
    d. Negative Acknowledgment (Nack)

    e. Rejection with Requeue
    f. Rejection without Requeue
93. **Exchanges**
    a. Fanout exchange
        i. pros and cons
        ii. uses
    b. Direct exchange
        i. pros and cons
        ii. uses
    c. Header exchange
        i. pros and cons
        ii. uses
    d. Topics exchange
        i. pros and cons
        ii. uses
    e. Dead Letter Exchanges and Queues
94. Polyglot persistence
95. Durability
    a. Durable Queues
    b. Persistence message
    c. Combined Durability
    d. rabbitMQ
96. Routing Key
97. Request response
    a. architecture
    b. breaks
    c. pros and cons
98. Publish subscribe (pub/sub) model
    a. Queue/Channels/Topics
    b. Publisher/producer
    c. Consumer
    d. pros and cons
99. Multiplexing
100. Channel
101. Push model

gRPC

102. why?
103. http
104. protobuffer
105. Unary gRPC
106. Server streaming
107. Client streaming
108. Bidirectional

TypeScript

Git Repo

Fore more info click here

Theory

1. What is typescript
2. Disadvantages
3. Statically typed language
4. **Compiling project**
    a. tcs index.ts
5. setting type
    a. let age: number 20
6. Types
    a. implicit types an explicit types
    b. any type
    c. You will lose type case (It's not recommend to use any)
    d. unknown
    e. never
    f. enum
    g. Tuple
7. Objects
    a. Readyone
    b. Method
    c. Specitif valus
    d. Return type
8. Type alias
9. Union type
10. Type intersection
11. Literal types
12. Nullalbe type
13. Optione property, element, call
14. Interface
    a. Reopening interface
    b. Inheritance
15. Class
    a. Modifiers
    b. Getters and setters
    c. Abstand class
    d. Overrifdienr
    e. Diff b/w class and abstand class
16. Generics

Next.js

17. **Theory**

18. Prerendering
    a. SSG (Static site generation)
    b. SSR (Server side rendering)
    c. Suspense SSR Arch
        i. HTML streaming
        ii. Selective hydration
    d. ISR (Incremental site generation)
    e. RSC (React server components)
    f. Pros and cons
19. **Routing**
    a. file based
    b. app based
    c. how to route
    d. dynamic route
    e. Catch all segments [...<slug>]
        i. optional catch all [[...]]
    f. Navigation
        i. Link component
            1. replace
        ii. usePathname
            1. startWith
        iii. useRouter
            1. push()
            2. replace()
            3. back()
            4. forward()
    g. Parallel Routes
        i. slots (@)
        ii. pros and cons
        iii. default.tsx
    h. Conditional Routes
    i. Intercepting Routes
        i. (.)<route>
        ii. (..)<route>
        iii. (..)(..)<route>
        iv. (...)<route>
20. **Routing metadata**
    a. why?
    b. static vs dynamic metadata
    c. priority
    d. layout vs page metadata
    e. title metadata
        i. absolute
        ii. default
        iii. template
21. **Pages**
    a. not-found.tsx & notFound()
    b. loading.tsx
    c. error.tsx
        i. Error boundary
        ii. error object
        iii. reset
        iv. error bubbling
    d. File colocation
    e. private folder
        i. _
        ii. advantages
        iii. %5F
    f. Route groups
22. **Layout**
    a. nested layout
    b. route group layout
23. **Templates**
    a. why?
    b. templates vs layout
    c. using both
24. **Component hierarchy**
    a. Layout > Template > Error Boundary > Suspense > Erro Boudy (not found) > Page
25. Route Handlers
26. RSC (React server component)
27. API routes
28. Rending
    a. client side
    b. server side
29. Date fetching
30. STyling
31. Optimization

32. Layouting
33. Loading state
34. Error bordering

35. SEO
    a. Metadata
36. Fetching data
    a. Using server comp
    b. In parallel
    c. Fetch data where It’s used
    d. Streaming and suspense
37. Deduplication
38. Caching
    a. ISR (Incremental site generation)
    b. {cache: force-cache}
    c. {cache: no-store}
    d. {next: {revalidate: 60}}
39. Dynamic params

NestJS

1. What
2. Why use NestJS
3. Module
    a. Global modules
    b. Third party modules
4. Dependency injections
5. **Decorators**
    a. Decorator factor
    b. TS vs JS
    c. Call, apply, bind
    d. This context in different functions
6. Bootstrap
7. Nest CLI
    a. create module
8. Controllers
9. Providers
10. Injectables
11. Nest validation
12. Pipes
    a. types of pipes
13. Class validators
    a. pipes vs class validators
14. Accessing code expires req
15. Strategy
16. Guards
    a. global
    b. local

Blockchain

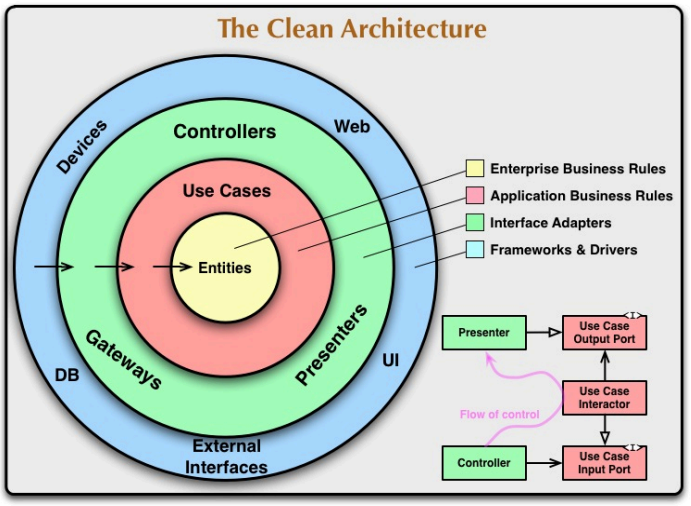
1. Centralised vs Decentralised
2. Blockchain
3. Ledger
4. Block
    a. data,
    b. hash,
    c. prev hash,
    d. Genesis block
5. Security
    a. Proof of work
    b. Consensus Rule
        i. Copy of chain in network
    c. Cryptographic proof
    d. Algo
        i. SHA256
        ii. RSA
        iii. MD5
6. Crypto
    a. no conversion
    b. centralised
    c. use case
        i. patient records
        ii. real estate - lottery
7. Smart contracts
8. Ethereum
9. Solidity
10. Mining

Clean Code

1. You are not done when it work
2. Invest the time to spend to write the program to make the program clean
3. Clean code what is expect when to read the code
4. Function should be verb (not noun)
5. **Function**
    a. Every things in the function should have the same abstraction
    b. Functions should be small
    c. Function should not have more than 3 params
    d. Don't pass boolean to a function
    e. Avoid switch statement
    f. The should not any side effect
    g. If a function return void, it should have side effects
    h. if a function returns a value, it should not have side effects
6. File should be <100 lines

7. SOLID Design Principles

8. - Single responsibility
9. - Open-closed
10. - Liskov substitution
11. - Interface segregation
12. - Dependency inversion



Clean Architecture

1. Things

2. Dependency Inversion Principle
3. Interface adapters
4.
5. Entities
    a. They have no dependency
6. Use cases
    a. they only depend on entities
    b. Interactor
    c. Interface
7. Controllers
8. Gateway
9. Presenter
10. Devices
11. Web
12. Database
13. UI
14. External Interface

15. Related Topics

16. Dependency Injection

17. Rules

18. Data flow from outside to inside

19. Videos

20. Using Clean Architecture for ...

Python

1. Basic

- Syntax
- Variables and Data Types
    1. Integers
    2. Floats
    3. Strings
    4. Booleans
    5. Lists
    6. Tuples
    7. Dictionaries
    8. Sets
- Type casting
- Slicing
- Scope of variables
- Operators
    1. Arithmetic Operators
    2. Comparison Operators
    3. Logical Operators
    4. Assignment Operators
    5. Bitwise Operators

2. Control Flow

- Conditional Statements
    - If
    - elif
    - else
- Loops
    - for loop
    - while loop
    - break, continue, pass
- Comprehensions
    - List Comprehensions
    - Dictionary Compre...
    - Set Compre...
- Exception Handling
    - Try
    - except
    - finally
    - Else
    - raise
- Exception Handling

3. Function

- Function
    - docstring
    - return
- Lambda Functions
- Types of function arguments
    - Defualt argument
    - Keyword argument
    - Positional Arguments
    - Arbitrary Keyword Arguments
- call by sharing
- Genarator
- Decorators
- Recursion
- Map
- Filter
- Reduce
- eval

4. Modules and Packages

- Importing Modules
- Creating Modules
- Using Packages
- init.py
- Standard Library Modules

5. File Handling

- Reading Files
- Writing Files

- Working with File Paths
- Context Managers

6. Other topics

- Programming Paradigms in Python.
- Method Resolution Order (MRO).
- Memory Allocation.
- Memory Management Mechanisms.
- Memory Leak in Python.

7. Object-Oriented Programming (OOP)

- Classes and Objects
  - constructor
  - Deconstructor
  - self parameter
- Instance Variables
- Methods
  - Static method
  - Instance method
  - Class method
- Inheritance
  - super()
  - Single Inheritance
  - Multiple Inheritance
- Polymorphism
- Encapsulation
- Abstraction
- Magic Methods
  - init, str, repr, etc.
-

8. QA

- role of init.py in package?
- Python is Intrepreter language ?
- Deffrence between oops and pops ?
- Mutable vs immutable?
- Shaloow coppy, deep coppy ?
- What is .pyc file ?
- Generate random number between 1 and 100 using lambda function?
- Deffrent between “is” and “==”?
- What is itrable ?
- What is primitive and non primitive ?

staticmethod vs classmethod
__name__
Regex
Iterator
Garbage collection
Pickling
Multy thrud language
Meta class
Multy processing
Memmary management
Referance conting
split

Others

1. SASS

2. @import "../node_modules/bootstrap/scss/bootstrap";
3. @use & @forward

4. REST API

5. it's about communication
6. RESTful
7. pros
    a. simple & standardised
    b. scalable & stateless
    c. high performance due to cachings
8. **Request**
    a. General (start line)
        i. method/target/version
    b. operation: get, post, put, delete
    c. endpoint
    d. header
        i. API key
        ii. authentication data
    e. body/ parameter
9. **Response**
    a. General (start line)
        i. version/statuscode/statustext
    b. header
        i. content type
    c. body
        i. requested resource
10. **HTTP Methods**
    a. GET
    b. POST
    c. PUT
    d. DELETE
11. Idempotent
12. Headers
13. Status code
    a. 1xx: Informational
    b. 2xx: Success
        i. 200 - Success
        ii. 201 - Success and created
    c. 3xx: Redirect
        i. 301: moved to new URL
        ii. 304: not changed
    d. 4xx: Client Error
        i. 401: Unauthorised
        ii. 402: 402 Payment Required
        iii. 403: Forbidden
        iv. 404: page not found
    e. 5xx: Server Error
14. MIME type
15. HTTP v2
16. TCP and IP

17. CI CD (git)

18. JSDoc

19. /**
    * function description
    * @param {string} description
    */
20. Params
21. Returns

22. Sequelize

23. Testin

24. Swagger