Folder Structures

TABLE OF CONTENT

W03: JAVASCRIPT........................... 3
1. THEORY... 3
35. VS.. 4
W04: NODE.JS EXPRESS...............5
THEORY.. 5
18. HTTP..5
19. EXPRESS...6
31. Core Express................................ 6
32. EJS... 6
35. VS...6
W05: MONGODB..............................8
1. THEORY..8
42. VS..10
48. INTERVIEW QUESTION............ 11
56. GOOD TO KNOW......................11
FD 01: HTML & CSS.............................12
1. HTML...12
9. VS.. 12
13. GOOD TO KNOW..................... 12
17. CSS.. 12
FD 01: JAVASCRIPT............................ 13
1. DOM.. 13
9. THEORY.....................................13
56. VS...15
57. GOOD TO KNOW...................... 15
FD 01: NODE.JS EXPRESS..................16
THEORY..16
64. HTTP.. 16
65. EXPRESS..................................17
77. Core Express.............................17
78. EJS.. 18
81. VS...18
W07: GIT...19
1. THEORY......................................19
23. GOOD TO KNOW..................... 19
REACT... 20
1. Basics...20
8. Theory.. 20
22. Good to Know............................21
W12: HOSTING....................................22
1. Nginx.. 22
19. SSH.. 22
23. SSL...22
25. Linux...22
W12: GIT..23
27. THEORY...................................23
55. GOOD TO KNOW..................... 23
W13: DS & Algorithms........................ 24
58. Algorithms.................................24
64. Problems.................................. 24
65. Complexity Analysis.................. 24
66. Asymptotic Notations.................24
67. Memory.....................................24
72. Data Structures..........................24
78. OTHERS....................................25
W14: DS & Algorithms........................ 26
82. Algorithms.................................26
83. Data Structures..........................26
W15: DS & Algorithms........................ 27
89. Data Structures..........................27
96. Algorithms.................................28
106. Question................................ 28
General..28
W16: SQL: Postgres............................29
1. Theory.. 29
13. Postgres.................................. 29
19. Commands............................... 29
23. Functions..................................29
44. Other theory and functions......... 31
48. Pendings...................................31
W17 REACT...32
1. Set up... 32
8. Theory.. 32
39. Good to Know............................34
W18 REACT...35
59. OTHERS....................................35
W19 REDUX...36
60. Theory...................................... 36
83. Other...36

92. JWT.. 36
FD 02: JAVASCRIPT............................ 37
61. DOM.. 37
69. Architecture............................... 37
70. THEORY....................................37
119. VS... 39
120. GOOD TO KNOW................... 40
FD 02: NODE.JS EXPRESS..................41
THEORY..41
117. HTTP...................................... 42
121. EXPRESS................................42
132. Core Express...........................43
133. EJS..43
136. VS...43
FD 02: MONGODB............................... 44
72. THEORY....................................44
114. VS... 47
120. INTERVIEW QUESTION..........47
128. GOOD TO KNOW................... 47
FD 02: REACT...................................... 48
101. Set up..................................... 48
108. Theory.................................... 48
139. Good to Know..........................50
145. Week 2.................................... 50
160. OTHERS..................................50
FD 02: DS & Algorithms.......................51
111. Algorithms............................... 51
117. Problems..................................51
118. Complexity Analysis..................51
119. Asymptotic Notations................ 51
120. Memory....................................51
125. Data Structures.........................51
131. OTHERS..................................52
Week 2... 52
135. Algorithms................................52
136. Data Structures.........................52
Week 3.. 53
142. Data Structures.........................53
149. Algorithms................................54
159. Question.................................. 54
General..54
W21: MICROSERVICES.......................55
Concepts & Theory...........................55
W21: Docker..57
13. Commands................................ 57
W21: Kubernetes................................. 59
64. Security......................................60
69. Yaml...61
74. Commands k8s...........................61
79. Good to know............................ 61
W21: Message Broker.......................... 62
Kafka... 62
RabbitMQ.. 62
gRPC...63
W21: MICROSERVICES 2.....................64
Design Patterns...............................64
TYPESCRIPT..66
Git Repo.. 66
Theory... 66
NEXT.JS..67
17. Theory....................................... 67
CLEAN CODE.......................................69
7. SOLID Design Principles.............69
1. Things..69
15. Related Topics............................69
17. Rules... 69
19. Videos..70
OTHERS..71
1. SASS...71
4. REST API.................................... 71
17. CI CD (git)..................................71
18. JSDoc..71

W03: JAVASCRIPT

1. **THEORY**
2. **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. **Pure Function**
    j. **Function composition**
    k. **IIFE**
3. Advantages and disadvantages of JS
4. Scope, Lexical scope
5. Prototype
6. **Closure**
    a. Disadvantage
    b. Uses
7. Garbage collection
8. **Hoisting**
    a. TDZ
    b. let, const vs var
    c. Function vs arrow function
9. Call Apply Bind
10. This Keyword
11. Temporal Dead Zone
12. **String Methods**
    a. Length
    b. toUpperCase, LowerCase
    c. Trim
    d. Pad
    e. charAt
    f. Split
    g. Concat
    h. substring
13. **Array Methods**
    a. Map
    b. Filter
    c. Reduce
    d. Find
    e. Sort
    f. Foreach
    g. Push
    h. Pop
    i. Shift
    j. Unshift
    k. Slice
    l. Splice
14. **Object Methods**
    a. freeze
15. Callback and callback hell
16. **Promise**
    a. Promise.all
    b. Promise.allSettled
    c. Promise.race
    d. Thenable
    e. Finally
    f. Catch
17. Async await
18. Spread and Rest Operator
19. DOM, BOM
20. Call stack
21. Event loop
22. **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
    j. SetInterval

23. **Primitive and non-primitive**
    a. Pass by value and pass by reference
24. Message queue
25. Life
26. Generator
27. **Prototype**
    a. Prototype chain
    b. Prototypal Inheritance

28. JavaScript is dynamically types
29. Currying
30. **Type Casting**
    a. Implicite (Coercion)
    b. Explicit (Conversion)
31. Microtask queue
32. Shallow copy
33. Deep copy
34. Immutable
35. **VS**
    a. == and ===
    b. Let, const, var
    c. Synchronous vs asynchronous
    d. While vs do while
    e. Foreach Vs Map
    f. Parameters, Arguments
    g. for in, for of
    h. Undefined, Null
    i. Keywords & Identifiers
    j. Type casting vs Type coercion

W04: NODE.JS EXPRESS

THEORY

1. What is Node.js
2. why v8 Engine
3. Advantages & Disadvantages of Node.js
4. How node works
5. Node Module System
6. REPL, Cli
7. NPX
8. Globals
    a. __dirname
    b. __filename
    c. **Module**
    d. Process
9. **Modules**
    a. **Core Modules.**
    b. local Modules.
    c. Third-party Modules.
    d. module.exports:{}
    e. require
    f. ESM
        i. import and export
10. **NPM**
    a. local and global
    b. npm init
    c. npm install or i
11. Nodemon
    a. scripts
        i. start
        ii. dev
    b. npm run dev
12. package.json
13. package-lock.json
14. Event loop
15. Event Queue
16. Events
    a. **Events emitter**
    b. Http module
17. Streams
    a. type of streams
        i. writable, readable, duplex, transform
    b. createReadStream()
    c. pipe()
18. **HTTP**
    a. https
    b. How does it work?
    c. request response cycle
    d. Stateless protocol
        i. Local storage, Sessions and Cookies
    e. Request
        i. General (start line)
            1. method/target/version
        ii. header
        iii. body
    f. Response
        i. General (start line)
            1. version/statuscode/statustext
        ii. header
            1. content type
        iii. body
            1. requested resource
    g. **HTTP Methods**
        i. GET
        ii. POST
        iii. PUT
        iv. DELETE
    h. Idempotent
    i. Headers
    j. 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: 402 Payment Required
            - 3. 403: Forbidden
            - 4. 404: page not found
        - v. 5xx: Server Error
    - k. MIME type
    - l. HTTP v2
    - m. TCP and IP

19. **EXPRESS**
20. npm install express –save
21. 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()
22. Static files
    - a. public
    - b. express.static()
23. **API**
    - a. json()
24. Params, Query String
25. Route Parameter
26. Query string/url Parameter
27. **MIddleware**
    - **a.** what is middleware
    - b. used for what?
    - **c. 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
    - d. req, res, next
    - e. next()
    - f. app.use in middleware
    - g. passing two middleware
28. **Routing**
    - a. router
    - b. express.Router()
29. Cluster
30. Multithreading in node.js
    - a. require('worker_theads')
    - b. new Worker
31. **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
32. **EJS**
    - a. i ejs
    - b. server side rendering
    - c. view engine
    - d. render()
    - e. <% %>, <%- %>, <%= %>
    - f. partials
33. **Rest API**
    - a. RESTful
34. fragment identifier
35. **VS**
36. API vs HTTP
37. API vs SSR

38. HTTP vs HTTPS
39. URIs vs URLs vs URNs
40. Session vs Cookies
41. GET vs POST
42. PUT vs PATCH
43. SSL vs TLS
44. **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. **fs async**
        v. - readFile()
        vi. - writeFile()
    d. http
        i. createServer()
            1. url
            2. listen()
            3. write()
            4. writeHead()
            5. end()
    e. util
        i. util.promisify
        ii. util.callbackify
    f. events
        i. eventEmmitter
            1. .emit()
            2. .on()
        ii. on()
    g. net
    h. crypto
        i. password hashing
        ii. .crateHmac(sha256, secret).update('<valuetoIncript>').digest('hex')

W05: MONGODB

1. **THEORY**
2. SQL(relational) vs
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.
12. MongoDB architectureJSON vs BSON
13. MongoDB shell
14. CRUD Operations
15. Cursor, Iterate a Cursor
16. TTL - Time to Leave
    a. Partial TTL
17. Maximum Document Size : 16Mb
    a. GridFS
18. **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
19. Cursor
    a. cursor methods
    b. - toArray
    c. - forEach
20. **Collection**
    a. db
    b. db.createCollection(collection Name)
    c. show collections
    d. renaming Collection
21. **Documents**
    a. adding new Documents
    b. Nested Documents
        i. advantage
22. **Inserting Document**
23. Insert One and Many
24. what are the additional methods used for inserting
25. **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
26. **Filtering**
    a. find(*filter*)
    b. find(*{filter}, {fieldsToGet}*)
27. **Method Chaining**
    a. count()
    b. limit()
    c. sort(1 or -1)
    d. skip()
28. **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. $elemMatch
i. $slice
j. $size
k. $inc: 1, $inc: -1
l. $pull, $push
m. $each [1, 2]
n. $eq, $ne
o. $currentDate
p. $exists
q. $expr
r. $cond
s. $rename
t. $min, $max
u. $mul
v. Array Operator
   i. $push
   ii. $each
   iii. $pull
   iv. $pullAll
   v. $pop
   vi. $
   vii. $elemMatch

29. Deleting
a. deleteOne({ *field:value* })
b. deleteMany()
c. remove()

30. 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

31. bulkWrite()
a. ordered: false
b. ordered vs unordered
c. advantages and disadvantages

32. **Commands**
a. mongosh
b. db
c. show dbs
d. db.stats

33. Aggregation
a. How does it work
b. advantages
c. types of aggregation
d. distinct
e. Aggregate stages
   i. $match
   ii. $group
      1. grouping by
      2. -nested field
      3. -multiple field
   iii. $sort
   iv. $count
   v. - other ways to count
   vi. - client and server side counting
   vii. $limit, $skip
   viii. $out
   ix. $project
   x. $lookup
   xi. $unwind
   xii. allowDiskUse: true
f. “$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

34. **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. **Types of Indexes**
        i. Single Field Index
        ii. Compound Index
        iii. Multikey Index
        iv. Text Index
        v. Geospatial, Hashed, Clustered Index

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

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

37. **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...

38. **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

39. **Cluster**
    a. types of cluster
    b. config servers

40. **Data Modeling**
    a. embedded data model
    b. reference data model
    c. linking vs embedding

41. **Transactions**
    a. ACID Transaction
    b. A- Atomicity
    c. C- Consistency
    d. I - Isolation
    e. D - Durability

42. **VS**
    a. $or vs $in
    b. $all vs $in

c. drop() vs remove()
d. findAndModify() vs findOneAndUpdate()
e. Primary key vs secondary key
f. join vs lookup
g. dot notation vs nested form
h. $currentDate vs $$NOW
i. delete() vs remove()
j. bulkWrite vs InsertMany
k. replace vs update
l. shard vs node vs cluster
m. Aggregation Pipeline vs Map Reduce
n. vertical scalability vs horizontal scalability
o. load balancer vs sharding
p. odm vs driver
q. stage operator vs accumulator operator
r. normal shard key vs hashed shard key
s. aggregate([$count:"tota"]) vs find({}).count()
t. replication vs replica set
u. transaction vs query
v. scaling up vs scaling down vs scaling out?
w. config servers vs mongos
x. load balancer vs auto balancer
y. countdocument vs count

43. What is a MongoDB driver?
44. Capped collection and it's advantages
45. Profiler
46. Explain
47. Soft deleting
48. **INTERVIEW QUESTION**
49. What to do when your quireing becomes slow?
50. What to do when your files are getting very big?
51. How to condense large volumes of data?
52. How to search for text in MongoDB?
53. How does MongoDB schema change?
54. How can we Backup and Restore in MongoDB?
55. What are the pros and cons of Normalising Data in MongoDB
56. **GOOD TO KNOW**
57. Atomicity
58. Type Bracketing
59. Dot Notation
60. Cursor behaviour
61. Aggregation Pipeline
62. Retryable Writes and Reads
63. MongoDB CRUD Concepts
64. B-Tree
65. ACID compliance
66. Mongoose
67. Network Components
    a. load balancer
    b. firewall
68. **CAP Theorem**
    a. consistency
    b. availability
    c. partition tolerance
69. Firewall
70. **Mongo Utilities**
    a. mongoexport
    b. mongoimport
    c. mongodump
    d. mongorestore
    e. mongostat
    f. mongotop
    g. mongooplog
71. $bucket

FD 01: 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. <em> vs <i>
12. <b> vs <strong>
13. **GOOD TO KNOW**
14. Whitespace
15. entity references
    a. <   &lt;
    b. >   &gt;
    c. "   &quot;
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

FD 01: JAVASCRIPT

1. **DOM**
2. querySelector
3. textContent
4. addEventListener
5. Order of Parsing
6. **event Propagation**
    a. event Bubbling
    b. event Capturing/ Trickling
    c. how to add both on program
7. event.stopPropagation();
8. event Delegation
    a. e.target
        i. id
        ii. tagName
        iii. pros and cons
9. **THEORY**
10. Data types
11. Operators
12. enum
    a. how to get enum in javascript
13. **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
14. Advantages and disadvantages of JS
15. **Set Map Flat**
    a. set
        i. add()
        ii. has()
        iii. delete()
    b. map
        i. get ()
        ii. set ()
        iii. <mapName>.size
        iv. iterating
    c. object vs map
    d. weekSet()
        i. features
    e. weekMap()
        i. features
        ii. key is private
    f. falt()
    g. flatMap()
    h. reduceRight()
    i. copyWithin()
16. **Operators**
    a. Nullish operator
    b. Optional chaining
    c. Ternary operator
    d. Type Operators
    e. **Unary operators**
        i. delete
        ii. typeof
        iii. !, ++, -, +
    f. **Bitwise Operators**
        i. bitwise OR
        ii. bitwise AND
        iii. uses
17. **Scope**
    a. Global scope
    b. Module scope
    c. Function scope
    d. Lexical scope
    e. Block scope
18. **Prototype**
19. Types of error
    a. syntax, logic
20. **Closure**
    a. Disadvantage
    b. Uses
    c. lexical scope vs closure

    d. IIFE
21. Garbage collection
22. **Hoisting**
    a. TDZ
    b. let, const vs var
    c. Function vs arrow function
23. **Call Apply Bind**
    a. function borrowing
    b. call vs apply vs bind
    c. polyfills
24. This Keyword
25. **String Methods**
    a. Length
    b. toUpperCase, LowerCase
    c. Trim
    d. Pad
    e. charAt
    f. Split
    g. Concat
    h. substring
26. **Array Methods**
    a. Map
    b. Filter
    c. Reduce
    d. Find
    e. Sort
    f. Foreach
    g. Push
    h. Pop
    i. Shift
    j. Unshift
    k. Slice
    l. Splice
27. **Object Methods**
    a. freeze
28. **Loop**
    a. for
    b. do while vs while
    c. labelled statements
    d. - break
    e. - continue
    f. for...in
    g. for...of
29. **Callback**
    a. callback hell
    b. inversion of control
30. **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
31. **Async await**
    a. async always return a promise
    b. error handling in async await
32. **Debouncing & Throttling**
    a. both are used for optimising performance of a web app
    b. by limiting the rate of API calls
33. Spread and Rest Operator
34. DOM, BOM
35. Call stack
36. Event loop
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 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. VS

a. == and ===
b. Let, const, var
c. Synchronous vs asynchronous
d. While vs do while
e. Foreach Vs Map
f. Parameters, Arguments
g. for in, for of
h. Undefined, Null
i. Keywords & Identifiers
j. Type casting vs Type coercion
k. textContent vs innerText
l. identifiers vs variables
m. defer vs async

57. GOOD TO KNOW

58. interpreted and compiled doe
59. Server-side vs client-side code
60.

FD 01: NODE.JS EXPRESS

THEORY

45. What is Node.js
46. why v8 Engine
47. Advantages & Disadvantages of Node.js
48. How node works
49. Node Module System
50. REPL, Cli
51. NPX
52. Globals
    a. __dirname
    b. __filename
    c. **Module**
    d. Process
53. **Modules**
    a. **Core Modules.**
    b. local Modules.
    c. Third-party Modules.
    d. module.exports:{}
    e. require
    f. ESM
        i. import and export
54. **NPM**
    a. local and global
    b. npm init
    c. npm install or i
55. Nodemon
    a. scripts
        i. start
        ii. dev
    b. npm run dev
56. package.json
57. package-lock.json
58. Event loop
59. Event Queue
60. Events
    a. **Events emitter**
    b. Http module
61. **Streams**
    a. type of streams
        i. writable, readable, duplex, transform
    b. createReadStream()
    c. pipe()
    d. Buffers
62. **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
63. **CORS**
    a. preflight request
        i. header
        ii. accept-control-allow-origin: *
        iii. accept-control-allow-methods: *
        iv.

64. HTTP

a. https
b. How does it work?
c. request response cycle
d. Stateless protocol
    i. Local storage, Sessions and Cookies
e. Request
    i. General (start line)
        1. method/target/version
    ii. header
    iii. body
f. Response
    i. General (start line)
        1. version/statuscode/statustext
    ii. header

1. content type

iii. body

1. requested resource

g. HTTP Methods

i. GET
ii. POST
iii. PUT
iv. PATCH
v. DELETE
vi. HEAD
vii. CONNECT
viii. OPTIONS
ix. TRACE

h. Idempotent
i. Headers
j. writeHead vs setHead
k. 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

l. MIME type
m. HTTP v2
n. TCP and IP

65.EXPRESS

66. npm install express –save
67. 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()

68. Static files

a. public
b. express.static()

69. API

a. json()

70. Params, Query String

71. Route Parameter
72. Query string/url Parameter

73. 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

74. Routing

a. router
b. express.Router()

75. Cluster
76. Multithreading in node.js

a. require('worker_theads')
b. new Worker

77.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

78. **EJS**
    a. i ejs
    b. server side rendering
    c. view engine
    d. render()
    e. <% %>, <%- %>, <%= %>
    f. partials
79. **Rest API**
    a. RESTful
80. fragment identifier
81. **VS**
82. API vs HTTP
83. API vs SSR
84. HTTP vs HTTPS
85. URIs vs URLs vs URNs
86. Session vs Cookies
87. GET vs POST
88. PUT vs PATCH
89. SSL vs TLS
90. **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. **fs async**
        v. - readFile()
        vi. - writeFile()
    d. http
        i. createServer()
            1. url
            2. listen()
            3. write()
            4. writeHead()
            5. end()
    e. util
        i. util.promisify
        ii. util.callbackify
    f. events
        i. eventEmitter
            1. .emit()
            2. .on()
        ii. on()
    g. net
    h. crypto
        i. password hashing
        ii. .createHmac(sha256, secret).update('<valuetoIncript>').digest('hex')

W07: GIT

1. **THEORY**
2. Config
3. git init
4. git clone
5. git status
6. **Creating Version**
    - git add *file*
        1. git add - - all
        2. git add .
    - git commit
        1. -m "<*message*>"
    - commit id
        1. check sum
        2. **content**
            1. author details
            2. preview details
            3. date
            4. etc..
        3. sha-1 hash
    - label
    - branch
7. touch
8. git log
9. git diff
10. git stash
11. **git checkout**
    - commit id
    - branch name
12. git log - - all
13. **Branching**
    - git branch <*branchName*>
    - git branch
14. **Merging**
15. git merge <*branchName*>
16. **types of merging**
    - fast-forward merge
    - **recursive merge**
        1. conflict
17. **Git server**
    - git remote add <*name*> <*url*>
        1. git remote
        2. git remote -v
    - git push <*remoteName*> <*branchName*>
    - **Cloning**
    - git clone <*url*>
    - git pull
    - pull vs pull request?
18. **Forking**
19. vim .gitignore
20. gist
21. **ci cd**
22. git projects
23. **GOOD TO KNOW**
24. rebase
25. tree
26. brew install tree

REACT

More topics (Topics to Learn)

1. **Basics**
2. npx create-react-app <appName >
3. components
    a. default is App
4. rafce
5. calling function on button click
    a. without parameter
    b. with parameter
6. Fragments
7. Children Prop
8. **Theory**
9. What is React
10. Virtual DOM
    a. Reconciliation
    b. Diffing Algorithm
11. props vs state
12. Server Side vs Client Side Rendering in React
13. React Fibre
14. Synthetic Events
15. Life Cycle
16. VIew Oriented
17. **Hooks**
    a. useState
        i. changeValue
        ii. changeValueWithFunct ion
    b. useRef
        i. html
        ii. useState vs useRef
    c. useEffect
        i. dependency
        ii. return in useEffect
        iii. useLayoutEffect
    d. useMemo
        i. sample
        ii. recache
        iii. pros and cons
        iv. referential equality
    e. useCallback
        i. sample
        ii. useMemo vs useCallback
        iii. uses
    f. useContext
        i. sample
    g. useReducer
    h. Create custom hooks
        i. useDebugValue
    i. useTransition
    j. useDeferredValue
    k. useId
        i. sample
    l. useImperativeHandle
18. map
19. **Props**
    a. default prop
    b. PropDrilling
    c. Children
20. **Components**
    a. Creating Components
    b. Controlled Components
        i. Inputs
    c. Higher order components
    d. Pure components
21. **React Router**
    a. install
    b. Hooks
        i. useHistory
    c. use
    d. Link
        i. replace
        ii. reloadDocument
        iii. state={}
        iv. - useLocation()
        v. NavLink
            1. style={}
            2. -isActive
            3. 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

22. Good to Know

23. Object.entries(e)
24. Icons
25. Experimental Hooks
    a. useEffectEvent
    b. use
    c. useFormStatus
    d. useOptimistic

W12: 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. **Load Balancing**
    a. Round robin
    b. Least connection
    c. IP hash
16. Caching
17. **Proxy**
    a. Proxy server
    b. Reverse proxy
    c. Forward proxy
    d. Load balancer vs reverse proxy
18. Nginx vs Apache
19. **SSH**
20. How does it work??
21. Private key
22. Public key
23. **SSL**
24. How does it work??
25. **Linux**
26. apt
27. rm
28. mkdir
29. touch
30. mv
31. nano
32. more, less
33. head, tail
34. >, <
35. /
    a. bin
    b. boot
    c. dev
    d. etc
    e. home
    f. root
    g. lib
    h. var

W12: GIT

27. **THEORY**
28. **Centralised** Version control system vs **Distributed** Version control system
29. Config
30. Working directory
31. Staging area
32. git init
33. git clone
34. git status
35. git log
36. **Creating Version**
    - git add *file*
        1. git add - - all
        2. git add .
    - **git commit**
        1. -m "*<message>*"
        2. Commit without staging
    - commit id
        1. check sum
        2. **content**
            1. author details
            2. preview details
            3. date
            4. etc..
        3. sha-1 hash
    - label
    - **branch**
37. touch
38. **git log**
    - git log
    - git log - - all
    - git log –p -1
    - git log graph
39. git diff
40. git diff –staged
41. **Restore**
    - git restore
    - git restore –staged
42. **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
43. **Stashing**
    - git stash
    - git stash apply
    - git stash drop
    - git stash list
44. **Merging**
45. git merge *<branchName>*
46. **Types of merging**
    - fast-forward merge
    - **recursive merge**
        1. conflict
47. **Git server**
    - git remote add *<name> <url>*
        1. git remote
        2. git remote -v
    - git push *<remoteName> <branchName>*
    - git push set upstream
    - **Cloning**
    - git clone *<url>*
    - git pull
    - pull vs pull request?
    - pull vs fetch
48. **Tags**
    - Simplified
    - Annotated
    - git tag
    - Should Pushing tags
49. **Forking**
50. git rebase
51. vim .gitignore
52. gist
53. **ci cd**
54. git projects
55. **GOOD TO KNOW**
56. rebase
57. tree

brew install tree

W13: DS & Algorithms

58. **Algorithms**
    - **Search**
    - Binary Search(recursive also)
    - Linear Search
59. Recursion
60. Iterative & recursive
61. Virtual memory
62. Amortised residing
63. Dynamic programing
    - Memoize approach
    - Bottom up approach
64. **Problems**
    - Factorial, fibonacci, prime number (with and without recursion)
65. **Complexity Analysis**
    - Time complexity
    - Space complexity
66. **Asymptotic Notations**
    - Ranking
    - Big O notation
    - Omega Notation
    - Theta Notation
67. **Memory**
68. **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
69. **Memory Leak**
    - Symptoms
    - **Garbage Collections**
        i. Process
    - Reasons for memory leak
    - How to debug
70. **Big O Notation**
    - Linear time complexity
    - Constant time complexity
    - Quadratic time complexity
    - Qubic
    - Logarithmic complexity
    - Exponential complexity
71. **Operations in normal array**
    - Init
    - Set
    - Get
    - Traverse
    - Insert
    - D45elete
72. **Data Structures**
73. What is DS?
74. Advantages and Disadvantages
75. Examples
    - DOM
    - Undu & Redo
    - Os job scheduling
76. **Dynamic Array**
    - It's working and memory allocation?
    - Set
77. **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

78. OTHERS

79. Build in DS in JS

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

80. Custom DS

- Stacks
- Queue
- Circular queues
- Linked lists
- Hash tables
- Trees
- Graphs

81. Trees

- **Binary tree**
    i. Complete binary tree
    ii. Full binary tree
    iii. Perfect binary tree
- **Heap**
    i. Features
    ii. Min Heap
        1. Creating Heap
        2. Insrt
        3. Dt
    iii. Max Heap

W14: DS & Algorithms

82. 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

83. Data Structures

84. Stacks

- LIFO
- Push, pop
- Stack underflow
- Stack overflow
- Use cases
- **Types of Stack**
- Linear Stack
- Dynamic Stack
- Array-based
- Linked list based

85. 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

86. Hash Table

- Searching O(1)
- Hash function
- Collision
- Dynamic restructuring
- Uses
- **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

87. SHA: Secure Hashing Algorithm

W15: DS & Algorithms

88. Linear, non-linear, hierarchical
89. **Data Structures**
90. **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 tre
    - **- Degrenarted**
        i. **Left-skew**
        ii. Right-skew
91. **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**
92. **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
93. **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
94. **Trie**
    - String vs Trie
    - **Operations**
    - - Init
    - - Insertion
    - - Delete
    - - Search
    - Prefix and Suffix tree
    - - terminator char
    - **Compressed Trie**
    - - Radix Tree (Patricia Trie)
95. **Graph**
    - Vertex, Edge
    - - Adjacency list, matrix
    - **Types**
    - - Unidirectional (Direct graph)
    - - Bidirectional (Un DIrected graph)
    - - Cyclic
    - - Disconnected
    - - Weighted Graph
    - - Unweighted Graph
    - - Bipartite Graph
    - **Traversal**
        i. BFS
        ii. DFS

- River size problem

96. **Algorithms**
97. Greedy method
98. Kruskal's Algorithm
99. Prim's Algorithm
100. Dijkstra's Algorithm
101. Bellman-Ford Algorithm
102. Topological Sorting
103. Floyd-Warshall Algorithm
104. Bipartite Graph Checking
105. Max Flow (Ford-Fulkerson Algorithm)
106. **Question**
107. Graph vs Tree
108. Forest (in Tree)
109. Forest > Graph > Tree > Linked list
110. 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
    c. Token

W16: 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.

W17 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
15. Life Cycle
16. View Oriented
17. Memoization
18. Pure functions
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

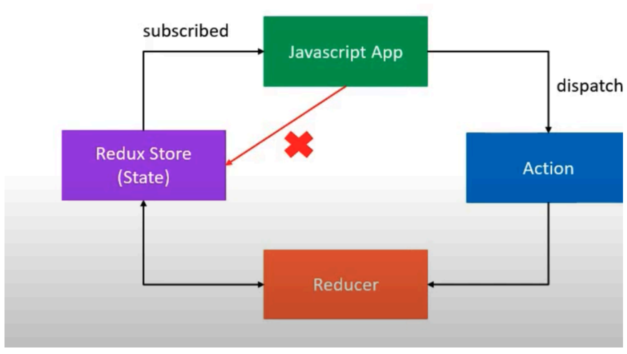
40. Immer
41. Object.entries(e)
42. Icons
43. Experimental Hooks
    a. useEffectEvent
    b. use
    c. useFormStatus
44. useOptimistic

fW18 REACT

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

W19 REDUX

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



75. **Redux principles**
    a. Store
    b. Action
    c. Reducer
76. Selectors
    a. Memoized selector
77. **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
78. **Slice**
    a. init state
    b. reducers
    c. extraReducers
79. **Redux toolkit**
    a. Nanoid
    b. Redux Query.
80. Normalizing Data
    a. Normalized state
    b. createEntityAdapter
    c. shallowEqual, reference equality
81. Serializing
82. Hydrating
83. redux vs flux
84. saga vs thunk
85. **Other**
86. Immer and the working of Immer in redux.
87. Access store outside of redux components
88. Flux by fb
89. Log rocket
90. createAsyncThunk
91. createEntityAdapter
92. createSelector
93. createListenerMiddleware
94. **JWT**
95. What?
96. **Structure**
    a. Header
    b. Payload
        i. iat
        ii. exp/eat
    c. Signature
97. Authentication working
98. Pros and cons
99. Expiration Time
100. Bearer token
101. Revocation
102. refresh token
103. Authentication vs Authorization

104. Types of Claims
    a. public
    b. registered
    c. private

FD 02: JAVASCRIPT

61. DOM

62. querySelector
63. textContent
64. addEventListener
65. Order of Parsing
66. **event Propagation**
    a. event Bubbling
    b. event Capturing/ Trickling
    c. how to add both on program
67. event.stopPropagation();
68. inst
    a. e.target
        i. id
        ii. tagName
        iii. pros and cons

69. 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. Call stack
    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

70. THEORY

71. 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
72. Operators
73. enum
    a. how to get enum in javascript
74. **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
75. Advantages and disadvantages of JS
76. **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()
77. **Operators**
    a. Nullish operator
        i. Nullish coalescing assignment
    b. Optional chaining
    c. Ternary operator
    d. Type Operators
    e. **Unary operators**
        i. delete
        ii. typeof
        iii. !, ++, -, +
    f. **Bitwise Operators**
        i. bitwise OR
        ii. bitwise AND
        iii. uses
78. **Scope**
    a. Global scope
    b. Module scope
    c. Function scope
    d. Lexical scope
    e. Block scope
79. Shadowing & Illegal shadowing
80. **Prototype**
81. Types of error
    a. syntax, logic
82. **Closure**
    a. Disadvantage
    b. Uses
    c. lexical scope vs closure
    d. IIFE
83. **Garbage collection**
    a. How does it work?
    b. mark-and-sweep
    c. reachability
    d. **Optimizations**
        i. - Generational collection
        ii. - Incremental collection
        iii. - Idle-time collection
84. **Hoisting**
    a. TDZ
    b. let, const vs var
    c. Function vs arrow function
85. **Call Apply Bind**
    a. function borrowing
    b. call vs apply vs bind
    c. polyfills
86. This Keyword
87. **String Methods**
    a. Length, toUpperCase, LowerCase, Trim, Pad, charAt, Split, Concat, substring, indexOf, lastIndexOf, localeCompare
88. **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
89. **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
90. Symbol
    a. properties
    b. global symbol registry

c. for, keyFor, iterator, toPrimitive

91. **Loop**
    a. for
    b. do while vs while
    c. labelled statements
    d. - break
    e. - continue
    f. for...in
    g. for...of
92. **Callback**
    a. callback hell
    b. inversion of control
93. **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
94. **Async await**
    a. async always return a promise
    b. error handling in async await
95. **Debouncing & Throttling**
    a. both are used for optimising performance of a web app
    b. by limiting the rate of API calls
96. Spread and Rest Operator
97. DOM, BOM
98. **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
99. **Primitive and non-primitive**
    a. Pass by value and pass by reference
100. Message queue
101. Life
102. Generator
103. **Prototype**
    a. Prototype chain
    b. Prototypal Inheritance
    c. uses?
    d. Circular reference
    e. Object.key
104. **Recursion**
    a. recursive call to function
    b. condition to exit
    c. pros and cons
    d. display the fibonacci sequence
    e. use
105. JavaScript is dynamically types
106. **Currying**
    a. function inside function
107. **Type Casting**
    a. Implicite (Coercion)
    b. Explicit (Conversion)
108. Microtask queue
109. **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()
110. TCP/IP
111. DNS
112. **IIFE**
    a. pros and cons
113. **Composition vs Inheritance**
114. Function recursion
115. [Symbol.iterator]
116. Truthy and falsy value
117. Strict mode in JS
118. this substitution

119. VS

a. == and ===

   b. Let, const, var
   c. Synchronous vs asynchronous
   d. While vs do while
   e. Foreach Vs Map
   f. Parameters, Arguments
   g. for in, for of
   h. Undefined, Null
   i. Keywords & Identifiers
   j. Type casting vs Type coercion
   k. textContent vs innerText
   l. identifiers vs variables
   m. defer vs async

120. GOOD TO KNOW

121. interpreted and compiled doe
122. Server-side vs client-side code

FD 02: NODE.JS EXPRESS

THEORY

91. What is Node.js
92. why v8 Engine
93. Advantages & Disadvantages of Node.js
94. How node works
95. Node Module System
96. Concurrency vs parallelism
97. REPL, Cli
    a. _
98. NPX
99. Globals
    a. __dirname
    b. __filename
    c. **Module**
    d. Process
100. **Modules**
    a. **Core Modules.**
    b. local Modules.
    c. Third-party Modules.
    d. module.exports:{}
    e. require
    f. ESM
        i. import and export
101. **NPM**
    a. local and global
    b. npm init
    c. npm install or i
102. Nodemon
    a. scripts
        i. start
        ii. dev
    b. npm run dev
103. package.json
104. package-lock.json
105. Event loop
106. Event Queue
107. Events
    a. **Events emitter**
    b. Http module
108. **Streams**
    a. type of streams
        i. writable, readable, duplex, transform
    b. createReadStream()
    c. pipe()
    d. Buffers
109. **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
110. **CORS**
    a. preflight request
        i. header
        ii. accept-control-allow-origin: *
        iii. accept-control-allow-methods: *
111. Cluster
112. Multithreading in node.js
    a. require('worker_theads')
    b. new Worker
113. thread pool
114. worker thread
    a. creating worker,
    b. parent port
115. cluster vs workerthread
116. child process
    a. methods
    b. - fork
    c. - exec
    d. - execFile
    e. - spawn
    f. spawn vs fork

117. **HTTP**

a. https
b. How does it work?
c. default port
d. request response cycle
e. Stateless protocol
    i. Local storage, Sessions and Cookies
f. Request
    i. General (start line)
        1. method/target/version
    ii. header
    iii. body
g. Response
    i. General (start line)
        1. version/statuscode/statustext
    ii. header
        1. content type
    iii. body
        1. requested resource
h. **HTTP Methods**
    i. GET
    ii. POST
    iii. PUT
    iv. PATCH
    v. DELETE
    vi. HEAD
    vii. CONNECT
    viii. OPTIONS
    ix. TRACE
i. Idempotent
j. Safe Methods
k. User-Agent
l. Headers
m. writeHead vs setHead
n. 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
o. MIME type
p. HTTP v2
q. TCP and IP

118. XSS

119. CSRF
    a. referral header

120. SQL injection
    a. prepared statements

121. **EXPRESS**

122. npm install express –save

123. 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()

124. Static files
    a. public
    b. express.static()

125. **API**
    a. json()

126. **Params, Query String**

127. Route Parameter

128. Query string/url Parameter

129. Path params

130. **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

131. **Routing**
   a. router
   b. express.Router()

132. 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

133. EJS

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

134. **Rest API**
   a. RESTful

135. fragment identifier

136. VS

137. API vs HTTP
138. API vs SSR
139. HTTP vs HTTPS
140. URIs vs URLs vs URNs
141. Session vs Cookies
142. GET vs POST
143. PUT vs PATCH
144. SSL vs TLS
145. **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. **fs async**
      v. - readFile( )
      vi. - writeFile()
   d. http
      i. createServer()

FD 02: MONGODB

72. **THEORY**
73. SQL(relational) vs
74. NoSQL ()
75. What is MongoDB?
76. Run on JS Engine
77. How does mongoDB work?
78. Non-relational Document based
79. Advantage and Disadvantages
80. BSON
81. MongoDB Structure
82. MongoDB architecture
83. JSON vs BSON
84. MongoDB shell
85. CRUD Operations
86. Cursor, Iterate a Cursor
87. Time to Leave
88. Maximum Document Size : 16Mb
    a.
89. **Storage engines**
    a. **types**
        i. WiredTiger engine
        ii. In-memory engine
        iii. MMAPv1
    b. GridFS
    c. Journal
90. **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
91. Cursor
    a. cursor methods
    b. - toArray
    c. - forEach
92. **Collection**
    a. db
    b. db.createCollection(collection Name)
    c. show collections
    d. renaming Collection
93. **Documents**
    a. adding new Documents
    b. Nested Documents
        i. advantage
94. **Inserting Document**
95. Insert One and Many
96. what are the additional methods used for inserting
97. **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
98. **Filtering**
    a. find(*filter*)
    b. find(*{filter}, {fieldsToGet}*)
99. **Method Chaining**
    a. count()
    b. limit()
    c. sort(1 or -1)
    d. skip()

100. **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, $unset
    h. **$elemMatch**
    i. $slice
    j. $size
    k. $inc: 1, $inc: -1
    l. $pull, $push
    m. $each [1, 2]
    n. $eq, $ne
    o. $currentDate
    p. $exists
    q. **$expr**
    r. **$cond**
    s. $rename
    t. $min, $max
    u. $mul
    v. $ifNull
    w. **Array Operator**
        i. $push
        ii. $each
        iii. $pull
        iv. $pullAll
        v. $pop
        vi. $
        vii. $elemMatch
101. **Deleting**
    a. deleteOne({ *field:value* })
    b. deleteMany()
    c. remove()
    d. delete vs remove
102. **Updating**
    a. updateOne({*whichObject*} , {$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
103. **bulkWrite()**
    a. ordered: false
    b. ordered vs unordered
    c. advantages and disadvantages
104. **Commands**
    a. mongosh
    b. db
    c. show dbs
    d. db.stats
105. **Aggregation**
    a. How does it work
    b. advantages
    c. types of aggregation
    d. distinct
    e. **Aggregate stages**
        i. $match
        ii. $group
            1. grouping by
            2. -nested field
            3. -multiple field
        iii. $sort
        iv. $count
        v. - other ways to count
        vi. - client and server side counting
        vii. $limit, $skip
        viii. $out
        ix. $project
        x. $lookup
        xi. $unwind
        xii. allowDiskUse: true
    f. “$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

106. **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. **Types of Indexes**
        i. Single Field Index
        ii. Compound Index
        iii. Multikey Index
        iv. Text Index
        v. Geospatial, Hashed, Clustered Index
107. **Schema**
    a. pros and cons of using schema
    b. optional schema
    c. validation action
108. **Relationships**
    a. embedding
    b. referencing
    c. one-to-one
    d. one-to-many
    e. one-to-squillions
    f. many-to-many
109. **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…

110. **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
111. **Cluster**
    a. types of cluster
    b. config servers
112. **Data Modeling**
    a. embedded data model
    b. reference data model
    c. linking vs embedding
113. **Transactions**
    a. ACID Transaction

   b. A- Atomicity
   c. C- Consistency
   d. I - Isolation
   e. D - Durability

114. **VS**
   a. $or vs $in
   b. $all vs $in
   c. drop() vs remove()
   d. findAndModify() vs findOneAndUpdate()
   e. Primary key vs secondary key
   f. join vs lookup
   g. dot notation vs nested form
   h. $currentDate vs $$NOW
   i. delete() vs remove()
   j. bulkWrite vs InsertMany
   k. replace vs update
   l. shard vs node vs cluster
   m. Aggregation Pipeline vs Map Reduce
   n. vertical scalability vs horizontal scalability
   o. load balancer vs sharding
   p. odm vs driver
   q. stage operator vs accumulator operator
   r. normal shard key vs hashed shard key
   s. aggregate([$count:"tota"]) vs find({}).count()
   t. replication vs replica set
   u. transaction vs query
   v. scaling up vs scaling down vs scaling out?
   w. config servers vs mongos
   x. load balancer vs auto balancer
   y. countdocument vs count
115. What is a MongoDB driver?
116. Capped collection and it's advantages
117. Profiler
118. Explain
119. Soft deleting
120. **INTERVIEW QUESTION**
121. What to do when your quireing becomes slow?
122. What to do when your files are getting very big?
123. How to condense large volumes of data?
124. How to search for text in MongoDB?
125. How does MongoDB schema change?
126. How can we Backup and Restore in MongoDB?
127. What are the pros and cons of Normalising Data in MongoDB
128. **GOOD TO KNOW**
129. Atomicity
130. Type Bracketing
131. Dot Notation
132. Cursor behaviour
133. Aggregation Pipeline
134. Retryable Writes and Reads
135. MongoDB CRUD Concepts
136. B-Tree
137. ACID compliance
138. Mongoose
139. Network Components
   a. load balancer
   b. firewall
140. **CAP Theorem**
   a. consistency
   b. availability
   c. partition tolerance
141. Firewall
142. **Mongo Utilities**
   a. mongoexport
   b. mongoimport
   c. mongodump
   d. mongorestore
   e. mongostat
   f. mongotop
   g. mongooplog
143. Clustered collections
144. WAL

FD 02: REACT

105. **Set up**
106. npx create-react-app <appName >
107. components
    a. default is App
108. rafce, tsrafce
109. calling function on button click
    a. without parameter
    b. with parameter
110. Fragments
111. Children Prop
112. **Theory**
113. What is React
114. DOM
    a. DOM vs Virtual DOM
    b. Reconciliation
        i. working
    c. Diffing Algorithm
    d. React Fibre
        i. incremental rendering
    e. Shadow DOM
115. Dynamic rendering
116. props vs state
117. Server Side vs Client Side Rendering in React
118. Synthetic Events
119. Life Cycle
120. View Oriented
121. Memoization
122. Pure functions
123. Strict Mode
124. SPAs vs MPAs
125. CSR vs SSR
126. Static vs Dynamic rendering
    a. ISR, SPA
127. **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
128. **JSX**
    a. Rules of JSX
    b. Fragment
    c. JavaScript in JSX
    d. HTML VS JSX
129. Conditional rendering
130. Key
131. **UI as a tree**
    a. Render trees
    b. Module Dependency Tree
    c. Bundler
        i. eg: Webpack
        ii. Compiling
        iii. Loader
        iv. Code splitting
132. **Rendering steps**
    a. Triggering
    b. Rendering
    c. Committing
133. Rerendering
134. Batching updates
135. **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
136. Declarative vs Imperative UI
137. **Event handlers**
    a. onClick, onSubmit etc…d
    b. Stopping propagation
    c. Preventing default
138. 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

139. **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

140. **Props**
  - a. default prop
  - b. PropDrilling
  - c. Children

141. **Components**
  - a. Creating Components
  - b. Controlled vs Uncontrolled Components
    - i. Inputs
  - c. Higher order components
  - d. Pure components

142. **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

143. Good to Know

144. Immer
145. Object.entries(e)
146. Icons
147. Experimental Hooks
  a. useEffectEvent
  b. use
  c. useFormStatus
148. useOptimistic

149. Week 2

150. Render props
151. Higher order components
152. Custom hooks
153. Code splitting
  a. Route based
  b. Component based
  c. React.lazy
154. Higher order comps
155. Lazy Loading
  i. fallback ui
  ii. suspense
  iii. Error boundaries
    iv. componentDidCatch
    v. Fallback UI
    vi. Nested & Propagation
156. useReducer
  a. dispatch
  b. useReducer vs useState
  c. useReducer vs redux
  d. payload
157. PropTypes
  a. types => name, string, any
  b. required, optional,
  c. node, element type
  d. oneof, shape
  e. PropTypes vs Typescript
158. useMemo vs useCallback
  a. React.Memo vs useMemo
  b. Object reference
  c. Pros and cons of memoization
159. Context API
  a. Provider
  b. Consumer
  c. useContext
  d. useReducer
160. Webpack
  a. Module Bundler
  b. Code Splitting
  c. Webpack Dev Server
  d. Hot Module Replacement (HMR)
  e. Tree Shaking
161. Babel
  a. Transpilation
  b. Plugins
  c. Runtime Polyfills
  d. Dynamic Import
162. useDeferedValue
163. useTransition

164. OTHERS

  a. forward ref
  b. useDebugValue
  c. useImperativeHandle
  d. Axios interceptor
  e. Concurrent Requests
    i. axios.all(), axios.spread()
cancel Token

FD 02: DS & Algorithms

111. **Algorithms**
    - **Search**
    - Binary Search(recursive also)
    - Linear Search
112. Recursion
113. Iterative & recursive
114. Virtual memory
115. Amortised residing
116. Dynamic programing
    - Memoize approach
    - Bottom up approach
117. **Problems**
    - Factorial, fibonacci, prime number (with and without recursion)
118. **Complexity Analysis**
    - Time complexity
    - Space complexity
119. **Asymptotic Notations**
    - Ranking
    - Big O notation
    - Omega Notation
    - Theta Notation
120. **Memory**
121. **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
122. **Memory Leak**
    - Symptoms
    - **Garbage Collections**
        i. Process
    - Reasons for memory leak
    - How to debug
123. **Big O Notation**
    - Linear time complexity
    - Constant time complexity
    - Quadratic time complexity
    - Qubic
    - Logarithmic complexity
    - Exponential complexity
124. **Operations in normal array**
    - Init
    - Set
    - Get
    - Traverse
    - Insert
    - Delete
125. **Data Structures**
126. What is DS?
127. Advantages and Disadvantages
128. Examples
    - DOM
    - Undu & Redo
    - Os job scheduling
129. **Dynamic Array**
    - It's working and memory allocation?
    - Set
130. **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

131. OTHERS

132. Build in DS in JS

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

133. Custom DS

- Stacks
- Queue
- Circular queues
- Linked lists
- Hash tables
- Trees
- Graphs

134. Trees

- **Binary tree**
    1. Complete binary tree
    2. Full binary tree
    3. Perfect binary tree
- **Heap**
    1. Features
    2. Min Heap
        1. Creating Heap
        2. Insrt
        3. Dlt
    3. Max Heap

Week 2

135. Algorithms

- **Sorting**
- Bubble sort
- Insertion sort
- Quick sort
    1. Divide and conquer
    2. Partition method
    3. **Pivot selection**
    4. Last, first
    5. average/median
- Heap sort
- Merge sort
    1. Divide and conquer
- Merge vs Quick sort

136. Data Structures

137. Stacks

- LIFO
- Push, pop
- Stack underflow
- Stack overflow
- Use cases
- **Types of Stack**
- Linear Stack
- Dynamic Stack
- Array-based
- Linked list based

138. Queue

- FIFO
- Enqueue
- Dequeue
- Peek
- Priority queue
- Circular queue
- Uses

- **Types of Queue**
- - Linear Queue
- - Circular Queue
- - Priority Queue
- - DEqueue (Double ended queue)
    1. Input restricted
    2. Output restricted
- - Blocking Queue
- - Concurrent Queue
- - Delay Queue

139. **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
    1. Linear Probing
    2. Quadratic Probing
    3. Double Hashing
    4. Clustering
- - Cuckoo hashing
- - Robin Hood hashing

140. **SHA: Secure Hashing Algorithm**

Week 3

141. Linear, non-linear, hierarchical

142. **Data Structures**

143. **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 tre
- **- Degrenarted**
    1. Left-skew
    2. Right-skew

144. **Binary Search Tree (BST)**
- BST vs BT
- Uses
- Balanced vs unbalanced tree
- Properties of BST
- **Operations**
- - Inserting
- - Deletion
- **- Traversal**
    1. **DFS**
    2. - InOrder
    3. - PreOrder
    4. - PostOrder
    5. **BFS**

145. **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

146. **Heap**
- Min Heap
    1. **To get value of**
    2. - Left child
    3. - Right child
    4. - Parent
    5. **Operations**

vi. - Init/ Heapify
vii. - Insert
viii. - Delete

- Max Heap
- Heapfity
  i. Bottom-up
  ii. Top-down
- DEPQ

147. **Trie**

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

148. **Graph**

- Vertex, Edge
- - Adjacency list, matrix
- **Types**
- - Unidirectional (Direct graph)
- - Bidirectional (Un DIrected graph)
- - Cyclic
- - Disconnected
- - Weighted Graph
- - Unweighted Graph
- - Bipartite Graph
- **Traversal**
  i. BFS
  ii. DFS
- River size problem

149. Algorithms

150. Greedy method
151. Kruskal's Algorithm
152. Prim's Algorithm
153. Dijkstra's Algorithm
154. Bellman-Ford Algorithm
155. Topological Sorting
156. Floyd-Warshall Algorithm
157. Bipartite Graph Checking
158. Max Flow (Ford-Fulkerson Algorithm)

159. Question

160. Graph vs Tree
161. Forest (in Tree)
162. Forest > Graph > Tree > Linked list
163. Operators
  - Binary operators
  - Priority
  - Infix
  - Prefix (Polish notation)
  - Postfix (Reverse Polish notation)

General

20. How does Logarithms work
21. File structure vs Data Structure
22. Where is the DS used?
23. Void vs null
24. Dynamic data structure
  a. Uses
  b. Example
25. Dynamic memory management/ allocations
26. Heap be used over a stack
27. Data abstraction
28. Post fix expression
29. Signed number
30. Pointers in DS
  a. Uses
31. Huffman’s algorithm working
32. What is recursive algorithm
  a. Divide and conquer on recursion
33. Which is the fastest sorting algorithm available?
34. Multi linked
35. Sparse matrices
36. Disadvantages of implementing queues using arrays
37. Void pointer
38. Lexical analysis
  a. Lexeme
  b. Pattern

Token

W21: MICROSERVICES

Concepts & Theory

39. What is a service?
40. Monolithic arch
    a. pros and cons
41. Microservice arch
    a. pros and cons
42. **Monolithic vs Microservice**
    a. deployment, scalability, reliability, development, flexibility, debugging
43. Security
44. **Cloud computing**
    a. Public IP address
    b. On-premises
    c. Iaas, Cass, Pass, Faas (Server less computer), Saas
    d. Private could
    e. Hybridge cloud
45. Scaling
46. Blue Green Deployment
47. Cloud Native vs Cloud Ready
48. Event-Driven Architecture
    a. Event producer
    b. Event broker
    c. consumer
    d. pub/sub
    e. eventual consistency
    f. cache layer
    g. idempotent
49. 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
50. Load balancing
    a. Round robin
    b. Least connection
    c. IP hash
51. Service Registry
52. Failed fast
53. Service Discovery
54. **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
55. **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
56. **Security**

    a. Defence in depth mechanism
    b. Token and API gateway
    c. Distributed tracing
    d. First session
    e. Mutual SSL
    f. OAuth
57. API gateway
    a. client performance
    b. security
    c. rate limiting
    d. monitoring logging
    e. BFF
58. SOA vs Microservices
59. **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

W21: 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

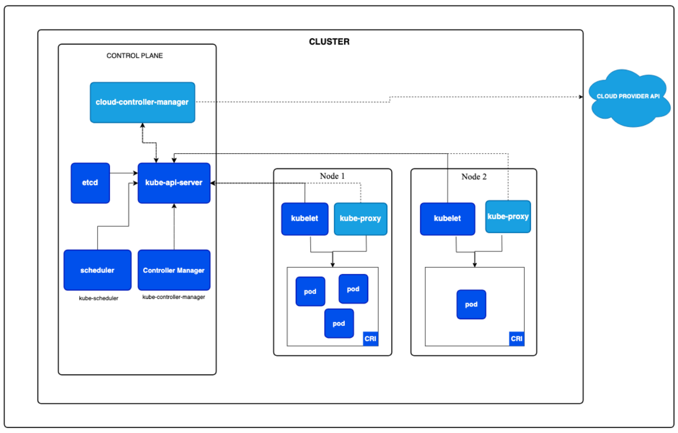
27. **Docker compose**
    a. docker compose
        i. up
        ii. down
        iii. watch
        iv. ps
    b. services
        i. image
        ii. ports
        iii. environment
        iv. restart
            1. always
            2. on-failure
            3. unless-stopped
        v. depends_on
        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

W21: 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

W21: 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

W21: MICROSERVICES 2

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

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

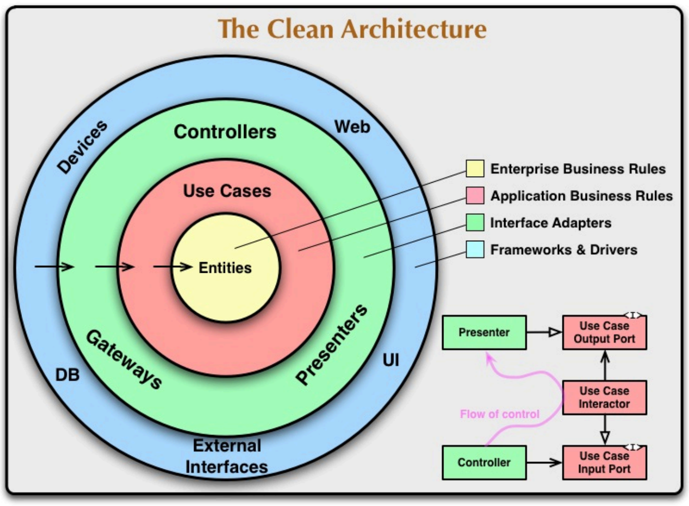
37. Deduplication
38. Caching
    a. ISR (Incremental site generation)
    b. {cache: force-cache}
    c. {cache: no-store}
    d. {next: {revalidate: 60}}
39. Dynamic params

Sequze

1. getters and setters
2. Validations
3. Paranoid
4. Associations
    a. one to one, one to many, many to many
    b. hasOne()
    c. belongsOne()

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 M...

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