React

* Create app using npx-

npx create-react-app appname

npm start 🡪 to run

code .

rfc 🡪 auto complete

* Create app using vite-

npm create vite@latest

npm install 🡪 install anything

npm run dev 🡪 to run

* naming convention

Function,Varible -- myFunction

Component – MyComponent

* Tailwind

1.Install Tailwind CSS and Other Dependencies

npm install -D tailwindcss postcss autoprefixer

2.Generate the Configuration Files

npx tailwindcss init -p

3.Configure Source Paths -->Add this in your content section of tailwind.config.cjs file.

"./index.html",

"./src/\*\*/\*.{js,ts,jsx,tsx}",

4.Add Tailwind Directives to Your CSS -->Add the statements below to your ./src/index.css file:

@tailwind base;

@tailwind components;

@tailwind utilities;

* Redux -

npm i redux react-redux @reduxjs/toolkit

* Toastify

npm i react-toastify

app.jsx 🡪

import 'react-toastify/dist/ReactToastify.css';

import {ToastContainer } from 'react-toastify'

<ToastContainer/>

Node.js

* npm init-->package.json

type:module-->ES6

* npm i nodemon
* require in ES5
* import in ES6 (.mjs for filename)
* npm start

express js

* npm init
* npm i express nodemon

Study

* Java

First-class Function --> when functions in that language are treated like any other variable.

Arrow Functions

Callback Functions

Single Threaded

Asynchronous

Objects as first-class

closure

Prototype-based

Even-driven Architecture

scope of variable --> Done

Hoisting --> Done

shadowing --> Done

array destructor

filter

reduce

foreach

? AND ??

* react

virtual dom

diffing algorithm

react fragment

class compoents

hooks --> special functions used to use class components function in function component

tailwind --> CSS framework

undirection data flow

? AND ?? IN react

child to parent data flow

* Icon

https://fontawesome.com/

heroicons

MongoDB

* use My\_Sample\_DB 🡪 Swith DB

Insert

* db.Collection\_1.**insertOne**({name:"Anand",Age:25}) 🡪 Insert one document in a collection
* db.Collection\_1.**insertMany**([{name:"Vis",age:18,place:"Purameri"},{name:"Kukk",Age:32}]) 🡪 Insert many document in a collection

Find

* db.Collection\_1.**findOne**() 🡪 Return the first document in the collection.
* db.Collection\_1.**find**() 🡪 Return all the documents in the collection
* db.Collection\_1.**findOne**({name:"Unni"}) 🡪 Find one with condition
* db.Collection\_1.**findOne**({name:"Unni",Age:29}) 🡪 Multiple Condition
* db.Collection\_1.**find**({name:"Unni"}) 🡪 Find with condition

Comparison Operator

* **lt** 🡪less than

db.Collection\_1.find({Age:{$lt:29}})

* **lte** 🡪 less than or equal

db.Collection\_1.find({Age:{$lte:29}})

* **gt** 🡪 great than

db.Collection\_1.find({Age:{$gt:29}})

* **gte** 🡪 great than or equal

db.Collection\_1.find({Age:{$gte:29}})

* **eq** 🡪 equal

db.Collection\_1.find({name:{$eq:"Unni"}})

* **ne** 🡪 not equal

db.Collection\_1.find({name:{$ne:"Unni"}})

* **in** 🡪 Value is matched within an array

db.Collection\_1.find({Age:{$in:[32,18]}})

Logical Operators

* **and**

db.Collection\_1.find({$and : [{Age:{$lte:30}},{Age:{$gte:25}}]})

* **or**

db.Collection\_1.find({$or:[{Age:{$eq:18}},{Age:{$eq:32}}]})

* **nor** 🡪Returns documents where both queries fail to match

db.Collection\_1.find({$nor:[{Age:{$gt:30}},{Age:{$lt:20}}]})

db.Collection\_1.find({$nor:[{name:"Unni"},{name:"Kukk"}]})

* **not**🡪Returns documents where the query does not match

db.Collection\_1.find({Age:{$not:{$gt:30}}})

Delete

* **deleteOne** 🡪 Delete one document

db.Collection\_1.deleteOne({name:"Shel"})

* **deleteMany**

db.Collection\_1.deleteMany({Age:32})

Update

* **updateOne**

db.Collection\_1.updateOne({Age:23},{$set:{name:"Unni Updated",place:"Mangalad"}})

* **updateMany**

**db.Collection\_1.updateMany({Age:23},{$set:{name:"Unni up",place:"villi up"}})**

Other Operator

* inc 🡪 Increment

db.Collection\_1.updateMany({name:"Anand"},{$inc:{Age:10}})

* Decrement 🡪 inc -10

db.Collection\_1.updateMany({name:"Anand"},{$inc:{Age:-20}})

**Note 🡪 Store Value from another field**

db.Collection\_1.updateMany({name:"Anand"},{$inc:{Age:-20}})

**Mongoose in NodeJS**

**npm i mongoose**

**Regex** : The $regex operator in MongoDB is a powerful tool that provides regular expression capabilities for pattern matching within strings in queries.

        const result = await todoModel.find({

            Name:{

                $regex : search,

                $options : "i"  // case insensitive search  "i" means ignore the case

            }

        })

**Sort :** // sort=1 ==> Sort by ascending && sort=-1 ==> Sort by descen

const result = await todoModel.find({

$and:[{Age:{$gte:22}},{Age:{$lte:35}}]

}).sort({Age:1})