React

- React is a function module & class module
- React is a Library

Create project cmd: npm create vite@latest.

Eg: App.jsx

```
function APP(){
  return (
    <div>
      <h1> boomadevi </h1>
    </div>
  )
}

export default app
```

Note
We should export the function to run the server.

none
about
contact
~~Pages~~

./ → folder in
../ → folder out

keyword to create a function structure
rafce

State

user get inputs.

Props

Parents to child get their input

class components

```
import {Component} from 'react';
class Skills extends Component {
  render() {
    return (
      <div> Skills </div>
    )
  }
}
export default Skills.
```

Props

parent -> child.

Ex

child.jsx

```
const child = (props) => {
  return (
    <div>
      <h1> Name : {props.name} </h1>
    </div>
  )
}
export default child
```

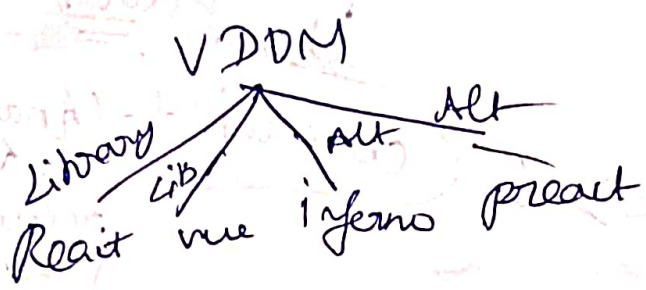
Destructuring props

```
const child = ({name}) => {
  return (
    <div>
      <h1> Name : {name} </h1>
    </div>
  )
}
```

App.jsx

```
const App = () => {
  return (
    <div>
      <Child name = "Bonadari" />
    </div>
  )
}
```

VDOM (Virtual document object model)



Page Navigation (Router)

npm i react-router-dom.

App.jsx

import {Router, Route} for react-router-dom.

```
<Routes>
  <Route path='/home' element={<Home />}/>
</Routes>
```

Navabar.jsx

import {Link} from react-router-dom.

```
<Link to='/home' classname='link'>Home</Link>
```

Hooks

- use State → Manage & update dynamic data
- use effect → Perform the "side effects" in functional documents.
- use Ref → ~~Don't refresh the page~~ without causing re-render when its value changes.

Note:
Render - Process of converting React components into a representation that can be displayed in the browser as a UI

→ use Reducer - that takes current state and an action as input and returns a new state.

Hooks

State → Manage mutable data with a component, causing re-renders when updated

Effect → Synchronizes a component with external systems or Performs side effects after rendering

Ref → Provides a way to access DOM nodes or React elements directly.

Reducer → Centralizes complex state update logic into a predictable, pure function

Context → Shares data across the component tree without Prop drilling.

React

Key components

- Create React project
  Cmd : npm create ~~vite@latest~~
- Install modules
  Cmd : npm install
- ~~Install~~ Run the server
  cmd : npm run dev
- Router or navigation
  Cmd : npm i react-router-dom
- Functional components (+)
- Class components (not frequently used)
- Hooks.
  - → State (
  - → Effect
  - → Ref
  - → Reducer } (not frequently used)
  - → context
- Props

MongoDB

@ insertone
→ to insert the items.

@ Create database

* Create Collection

* use database ~~name~~
  Cmd: use database_name

* insert one → ~~insertOne~~
  → Cmd: db.student.insertOne(

React Deployment

~~Step1~~: Push the project in github.

Step2: Go to vercel website

Step 2-1: Click "start deployment" button.

Step 2-2: Login with github

Step 2-3:

Step3: Import the repository

Step4: Edit the Root directory
(Note: It should be where the react project occurs)

Step 5: Press the deploy button

Step 6: React deployed successfully Hehe 😊

MongoDB

* TO show the all the database

  Cmd: show dbs

* How to use the database

  Cmd: use database_name

  Eg: use student

* How to create a collection

  Cmd: db.createCollection('collection_name')

  Eg: db.createCollection('users')

* TO insert one data in collection.

  cmd: db.collection_name.insertOne({title: "-"})

  Eg: db.student.insertOne({Name: "Boomadevi", Dept: "IT"})

* To insert many data in collection

  cmd: db.collection_name.insertMany([{title: "-"}, {title: "-"}])

  Eg: db.student.insertMany([{Name: "Boomadevi"}, {Name: "Dharanya"}])

* To update one data in collection

  Cmd: db.collection_name.updateOne({title: "-"}, {$set: {title: "-"}})

  Eg: db.student.updateOne({Name: "boomadevi"}, {$set: {age: 20}})

* To update many data in collection

  Cmd: db.collection_name.updateMany({title: "-"}, {$set: {title: "-", title: "-"}})

  Eg: db.student.updateMany({Dept: "CS"}, {$set: {dem: {sb: 5, c: 9}}})

* To find the data

cmd: db.collectionname.find()

Eg: db.student.find()

* Limited datas.

cmd: db.collectionname.find().limit(number)

Eg: db.student.find().limit(2)

* Greater than.

cmd: $gt

Eg: db.student.find({age:{$gt:20}})

* Lesser than

cmd: $lt

Eg: db.student.find({age:{$lt:20}})

* Lesser than or equal

cmd: $lte

Eg: db.student.find({age:{$lte:20}})

* Not equal

cmd: $ne

Eg: db.student.find({age:{$ne:20}})

* AND operation

cmd: $and (or) not neccesary to mention

Eg: db.student.find({Dept:"CT", age:20})

* OR operation

cmd: $or

Eg: db.student.find({$or:[{Dept:"CS"}, age:20}])

* Sort the data

Ascending order

Cmd: db.collection_name.find().sort({title:-1})

Eg: db.student.find().sort({age:-1})

Descending order

Cmd: db.collection_name.find().sort({title:1})

Eg: db.student.find().sort({age:1})

* delete to one data

Cmd: db.collection_name.deleteOne({title:""})

Eg: db.Student.deleteOne({Dept:"CSE"})

* delete to many data

cmd: db.collection_name.deleteMany({title:""})

Eg: db.Student.deleteMany({Dept:"IT"})

* How to Export data

* click "Export data" button

* Select the "Download" folder in our PC

* And then click

* click "Export full collection"

* Select "JSON"

* Click 'Export' button

* How to import the data

* Select "import Data" button

* And select the file import it.

How to connect with MongoDB

Step 1: Go to MongoDB Atlas

Step 2: Sign in

Step 3: Create cluster.

Step 4: Click free

Step 5: Give password

Step 5-1: Create username/password

Step 5-2: Set the connection method

Step 6: (I have MongoDB compass)
Button click.
Copy the string in this
[Note: It is the one we
can able to connect
the mongoDB cloud]

Step 7: Done.

Node & Express

* Initialize Node
  cmd : npm init -y
* Install Express.
  cmd : npm install express.
* Install Packages like package.json, package-lock.json
  cmd : npm install.
* ~~git ego~~ Run the Node
  cmd : Node server.js
* Listen → Server listening
  Eg: app.listen(3000)
* -- watch instead of ~~Node nodemon~~ nodemon.
  * Auto project server running.
* Nodemon install akro
  cmd : ~~node~~ npm i nodemon
  * Third party library.
* .env
  installation cmd : npm i dotenv
  → mongoDB link have to use through this .env

Methods

* Get → Client wants data → server responds with data from MongoDB
Eg: Model.find({}, (err, data) => console.log(data));
* POST → Clients sends data → Server inserts new record into MongoDB
Eg: new model({name: "Bonaventure", skills: "Dev"}).save()
* PUT → Client sends updated data → Server updates existing MongoDB document.
Eg: Model.findByIdAndUpdate(id, {name: "Full"}, {new: true}, (err, data) => console.log(data));
* Delete → Delete it from the collection
Eg: Model.findByIdAndDelete(id);

Install mongoose

Cmd: npm i mongoose

Router & controller

MongoDB connection

Data and Modification

controller.js (controller folder)

```
const user = require('../models/userModel')
// Get Method

exports.getRoute = async (req, res) => {
    const response = await user.find();
    res.status(201).json({data: response});
    res.send("Get route working");
}
// Post method
exports.postRoute = async(req, res) => {
    const {username, password} = req.body;
    const exist = await user.findOne({username})
    if (exists) {
        return res.status(401).json({
            message: "user already exist"
        })
    } else {
    const newUser = new user({username, password});
    await newUser.save();
    res.status(201).json({user: newuser});

    }
}
// Put method
exports.putRoute = async (req, res) => {
    const {username, password} = req.body;
    const updateuser = await user.findByIdAndUpdate
                                    Update
    (req.params.id, req.body
    {new: true})
```

```
if (!update){
    return res.status(404).json({message: "user not exist"})
  } else {
    res.status(201).json({update});
    }
}

// Delete method

export.deleteRoute = async (req, res) => {
  const deleteUser = await
                  User.findByIdandDelete
                  (req.params.id);
  if (!deleteUser) {
      return res.status(401).json({message:
                  "user not exist"})
    }
  else {
      res.status(201).json({deleteUser});
    }
}
```

Router.js (Routers folder).

```
const express = require('express');
const router = express.Router();
const {getRoute, postRoute, putRoute, deleteRoute} =
  require('../controllers/controller');
router.get('/get', getRoute)
router.post('/post', postRoute)
router.put('/put', putRoute)
router.delete('/delete', deleteRoute)

module.export = router;
```

Bcrypt installation

cmd: npm i bcrypt.

* It helps to hide a password in Database

Cors

cmd: npm i cors

* We used when we getting the multiple server

Axios

cmd: npm i axios

* It helps to fetch the data

Render

backend connection.

Step 1: Import the repository

Step 2: Give the path
Ex: src. unload / backend

Step 3: npm i -> download node_modules

Step 4: npm start (or) node server.js
(what how we run the server)

Step 5: Environment variable (.env).

Step 6: copy the .env in the code
Paste it in it.

Step 7: deploy.

Step 8: After deployment it will show the link. copy it and past it in the code (Ex: localhost:8000 instead use this link).

Step 9: Reload and deployment