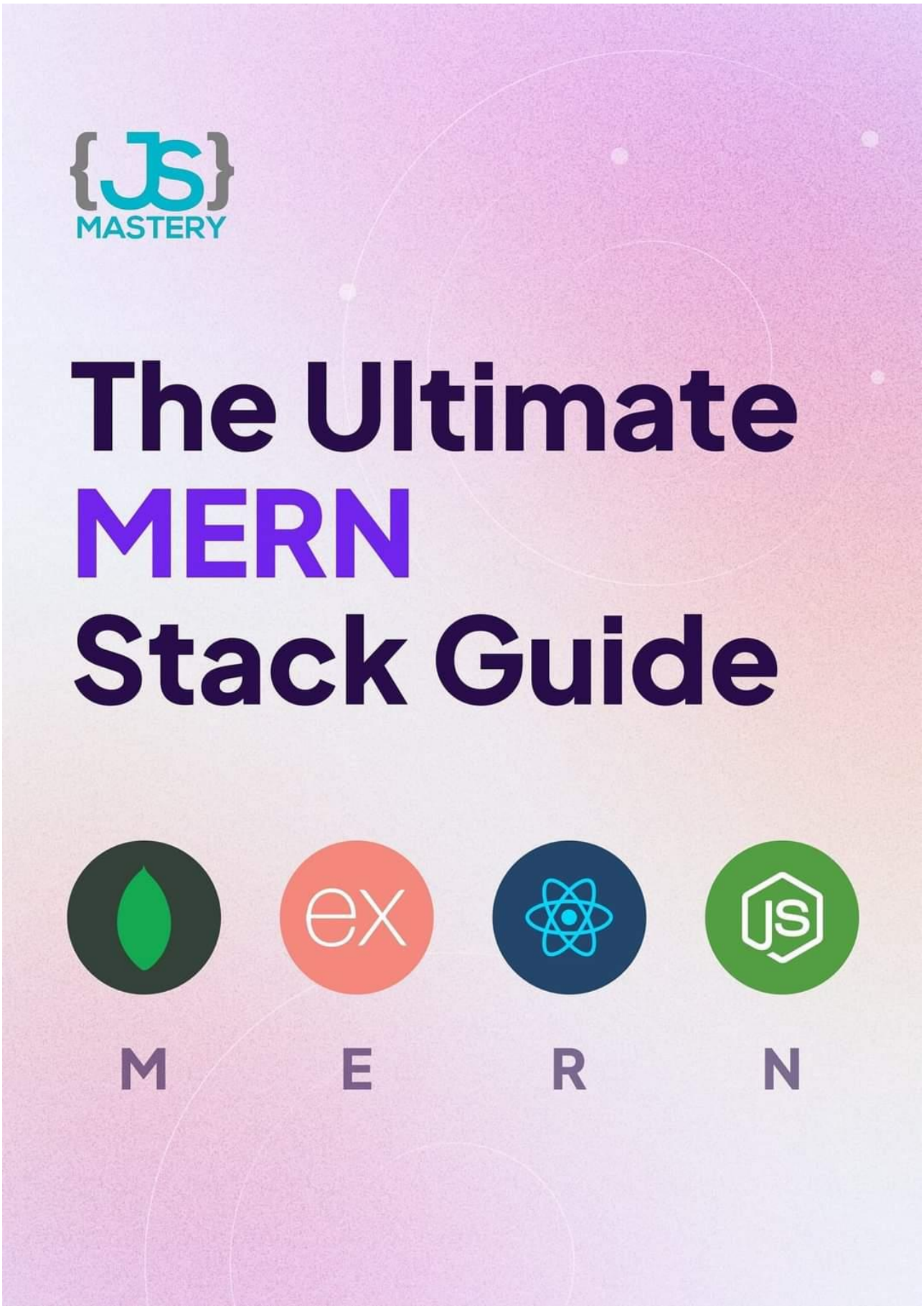
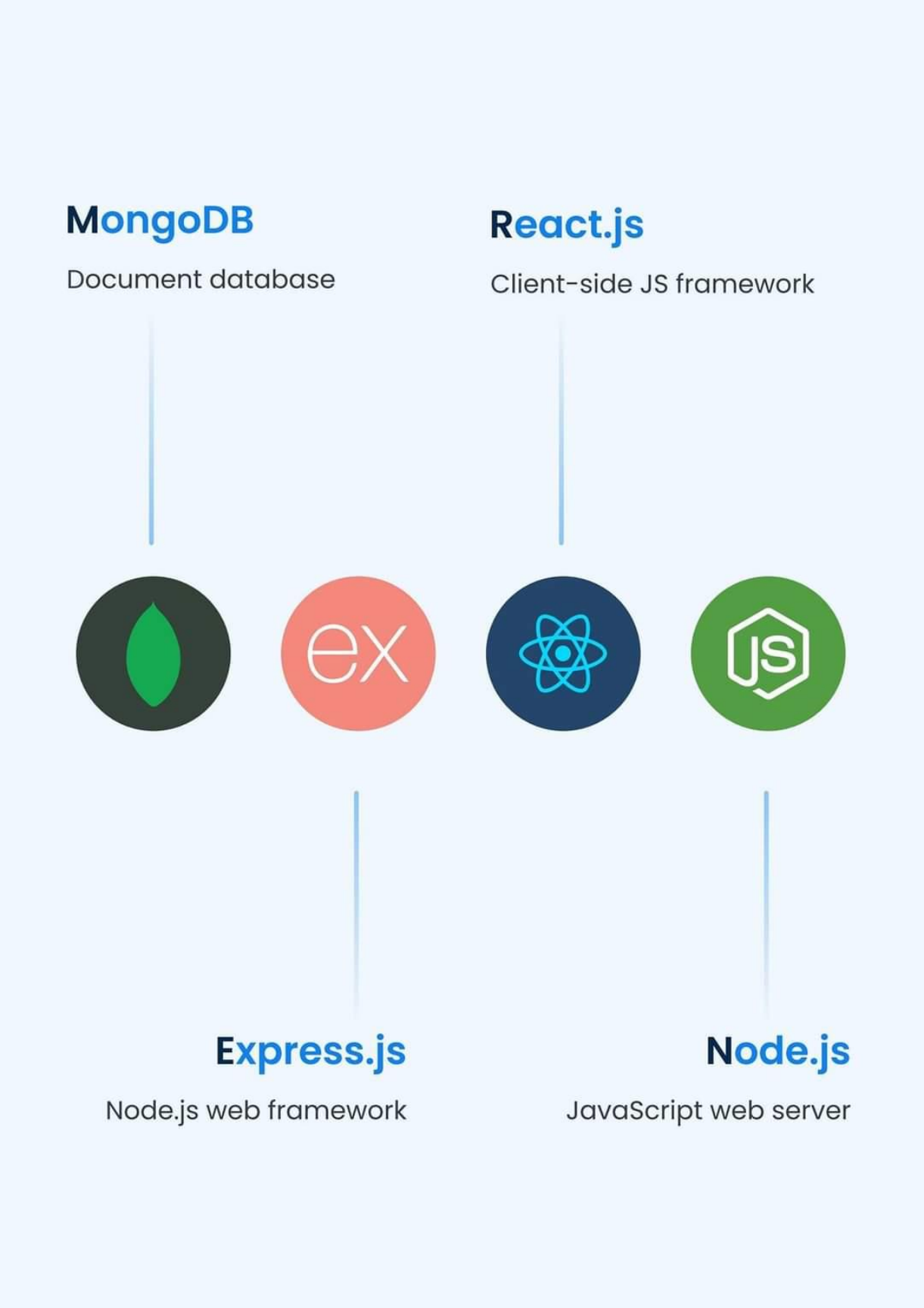
Chama ITP



Chama ITP


MongoDB
Document database
React.js
Client-side JS framework
ex
JS
Express.js
Node.js web framework
Node.js
JavaScript web server

What is MERN Stack?

MERN Stack is a Javascript Stack that is used for easier and faster deployment of full-stack web applications. MERN Stack comprises of 4 technologies namely: MongoDB, Express, React and Node. js. It is designed to make the development process smoother and easier

The MERN architecture allows you to easily construct a 3-tier architecture (frontend, backend, database) entirely using JavaScript and JSON

Using these four technologies you can create absolutely any application that you can think of everything that exists in this world today.

Now let's understand each technology one by one.

MERN

MongoDB forms the M of the MERN stack and works pretty well with the JavaScript ecosystem.

MongoDB is a NoSQL database in which data is stored in documents that consist of key-value pairs, sharing a lot of resemblance to JSON.

The data is not stored in the form of tables and that's how it differs from other database programs.

This is how the data stored in MongoDB looks like:

```
{
    _id: ObjectId("2af3b3a615571a1015d782e9")
    name: "JavaScript Mastery"
    email: "jsm@jsmasterypro.com"
    password: "useStrongPasswords!"
    createdAt: 2022-04-04T16:02:32.146+00:00
    updatedAt: 2022-04-05T08:06:10.425+00:00
    __v: 0
}
```

Chama ITP

Express JS MERN

Express is a flexible and clean Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based web applications.

Express helps build the backend very easily while staying in JavaScript ecosystem. It is preferred for self-projects as it helps focus on learning development and building projects pretty fast.

In MERN stack, Express will be used as backend API server which interacts with mongoDB database to serve data to client (React) application.

Chama ITP

React JS

MERN

React is an open-source JavaScript library that is used for building user interfaces specifically for single-page applications. It's used for handling the view layer for web and mobile apps.

React lets you build up complex interfaces through simple Components, connect them to data on your backend server, and render them as HTML.

Almost all the modern tech companies from early-stage startups to the biggest tech companies like Microsoft and Facebook use React.

The prime reason why react is used, is for Single Page Applications(SPA). SPA means rendering the entire website on one page rather than different pages of the websites.

Chama ITP

Node JS

MERN

NodeJS is a cross-platform JavaScript runtime environment, it's built on Chrome's V8 engine to run JavaScript code outside the browser, for easily building fast and scalable applications.

The main purpose of NodeJS is simple, it allows us to write our backend in JavaScript, saving us the trouble of learning a new programming language capable of running the backend.

Node.js is the platform for the application layer (logic layer). This will not be visible to the clients. This is where client applications (React) will make requests for data or webpages.

Chama ITP

MERN Roadmap

Now let's dive into what you need to learn in MERN.

First and foremost before you dive into any advanced topics, you first need to learn the core of the web i.e. HTML, CSS, and JavaScript.

HTML

HTML provides the basic structure.

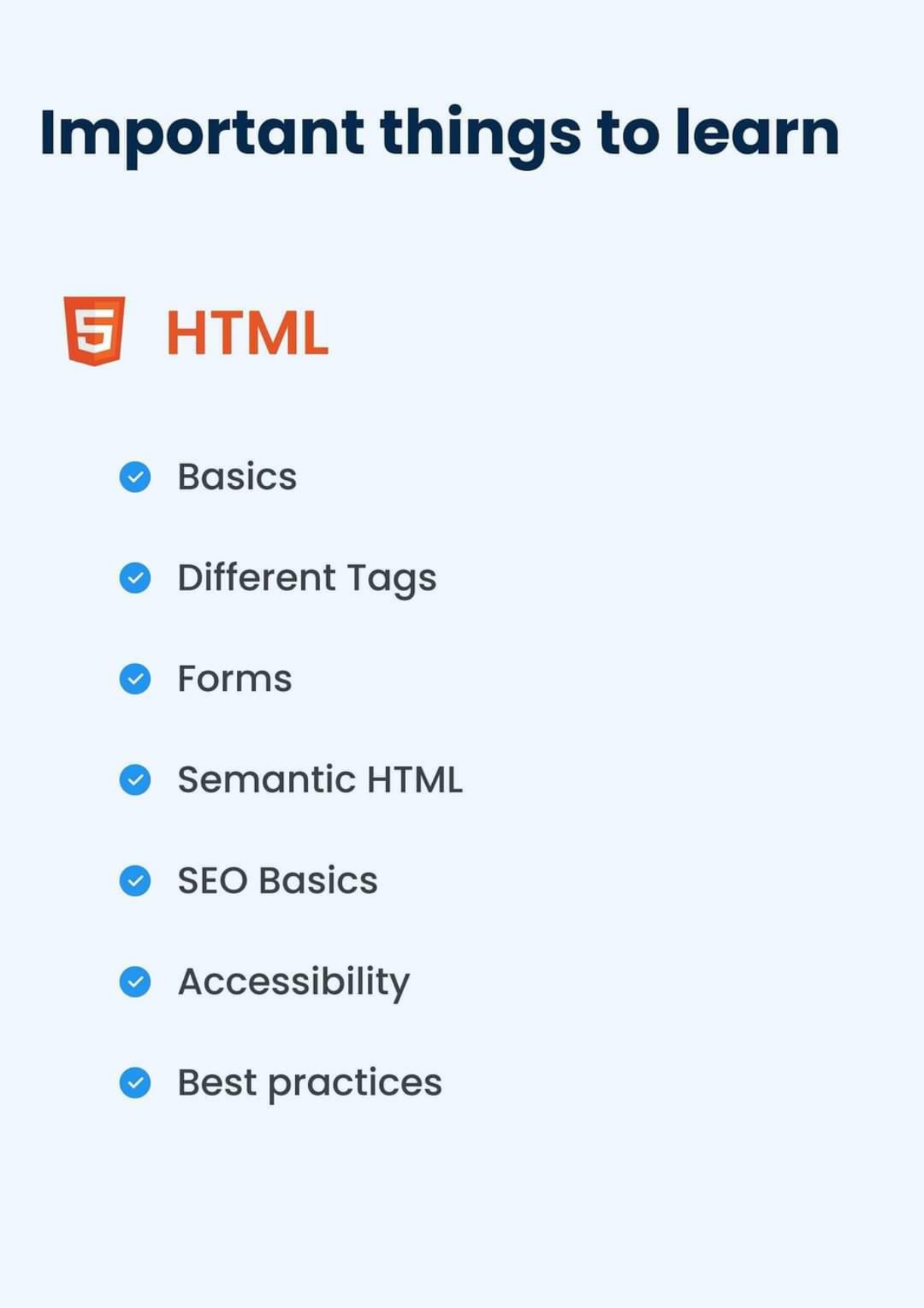
CSS

CSS provides the skin to the structure in the form of design, formatting, and layout.

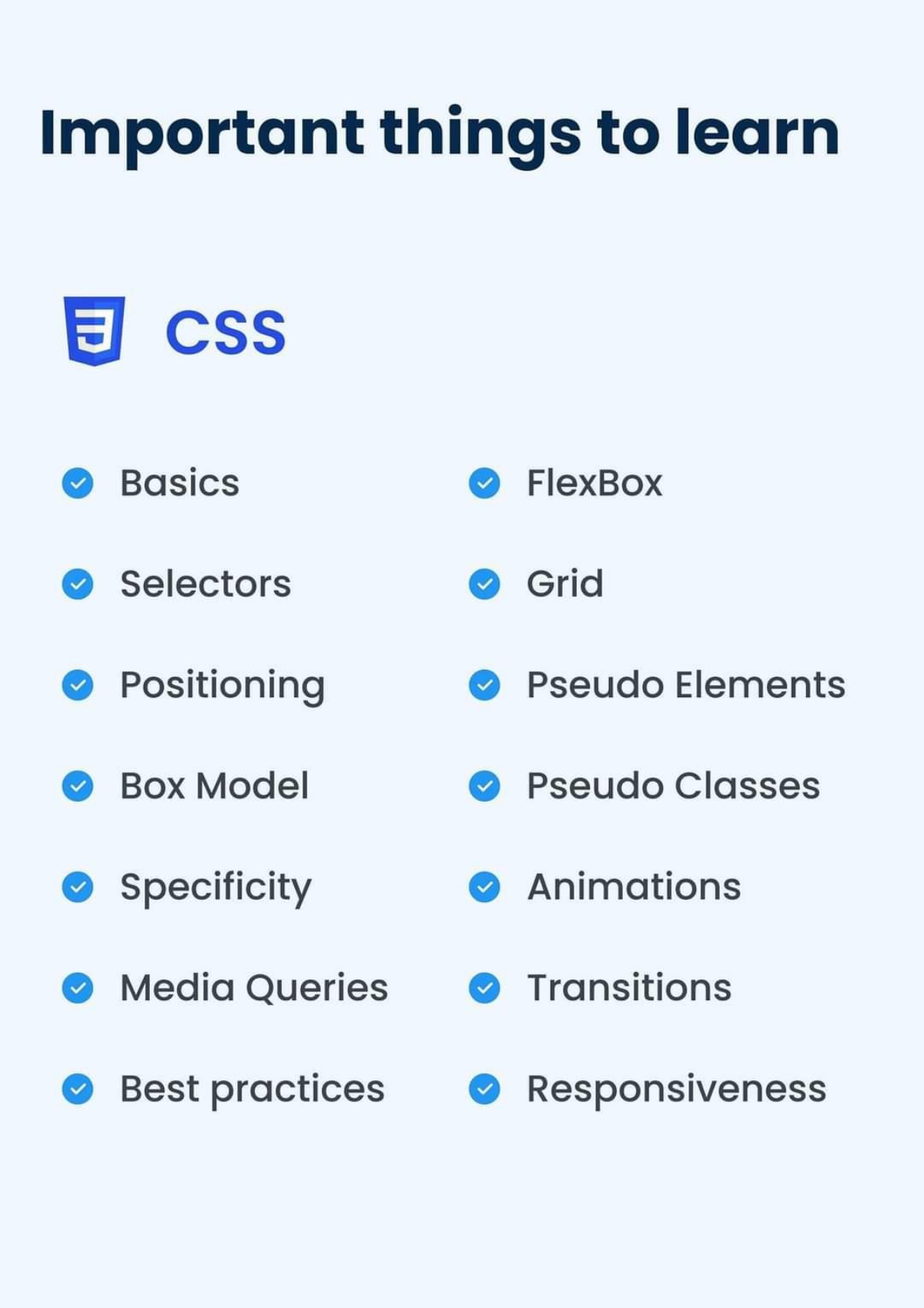
JavaScript

JavaScript adds interactivity and logic to the website.

Chama ITP


Important things to learn
HTML
Basics
Different Tags
Forms
Semantic HTML
SEO Basics
Accessibility
Best practices

Chama ITP


Important things to learn
CSS
Basics
Selectors
Positioning
Box Model
Specificity
Media Queries
Best practices
FlexBox
Grid
Pseudo Elements
Pseudo Classes
Animations
Transitions
Responsiveness

Chama ITP


Important things to learn
JS
JavaScript
Basics
DOM
Fetch API
Async Await
Event Listeners
ES6+ features
Best practices
Promises
Classes
Array Methods
String Methods
Scoping
Hoisting
Closures

MERN Roadmap

Once you know enough front-end development, where you can build great projects, then you should consider diving into learning React.js.

You might be wondering, what are the prerequisites to learn such a great JavaScript library?

There's only one prerequisite and that is - JavaScript.

Keep in mind, do not directly jump into learning React.js, before learning JavaScript, you should be confident with JavaScript because JavaScript is the main thing you'll be working with.

Most importantly, you must understand

- Syntax
- ES6+ features

Chama ITP

MERN Roadmap

- Arrow functions
- Template literals
- Array Methods
- Object property shorthand
- Destructuring
- Rest operator
- Spread operator
- Promises
- Async/Await syntax
- Import and export syntax

Chama ITP

MERN Roadmap

Basic things to learn in React.js

- File & Folder structure
- Components
- JSX
- Props
- State
- Events
- Styling
- Conditional Rendering

Chama ITP

MERN Roadmap

Learn about React.js Hooks - the essential hooks to learn:

- useState
- useEffect
- useRef
- useContext
- useReducer
- useMemo
- useCallback

Chama ITP

MERN Roadmap

Also learn these essential things:

- Fetching data from APIs
- Routing
- Context API
- Learn to create custom hooks
- Handling form submits
- Use cases of less common hooks
- Higher-Order Components
- React DevTools

MERN Roadmap

Then learn some of the React.js UI Frameworks

- ⭐ Material UI
- ⭐ Ant Design
- ⭐ Chakra UI
- React Bootstrap
- Rebass
- Blueprint
- Semantic UI React

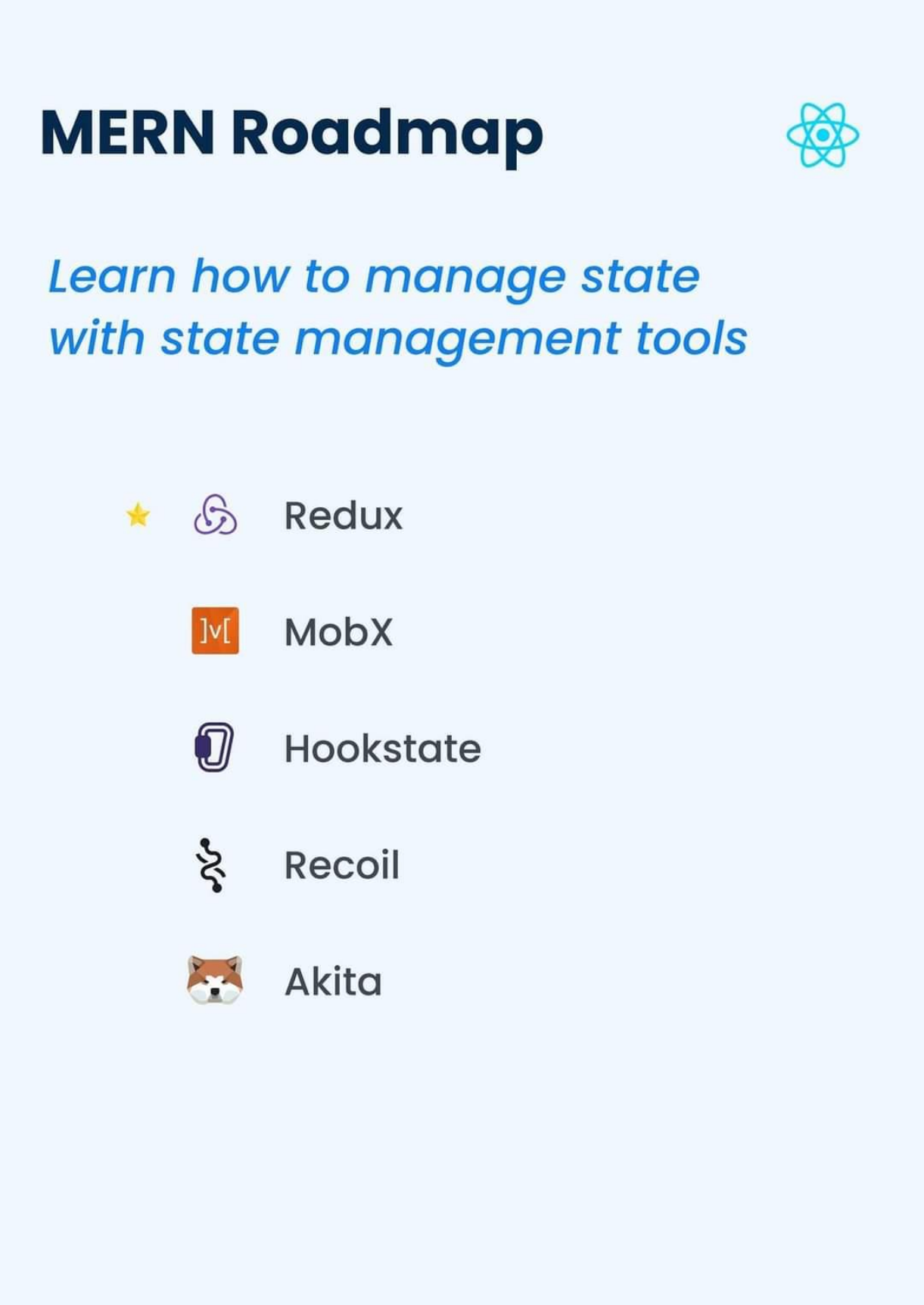
Chama ITP

MERN Roadmap

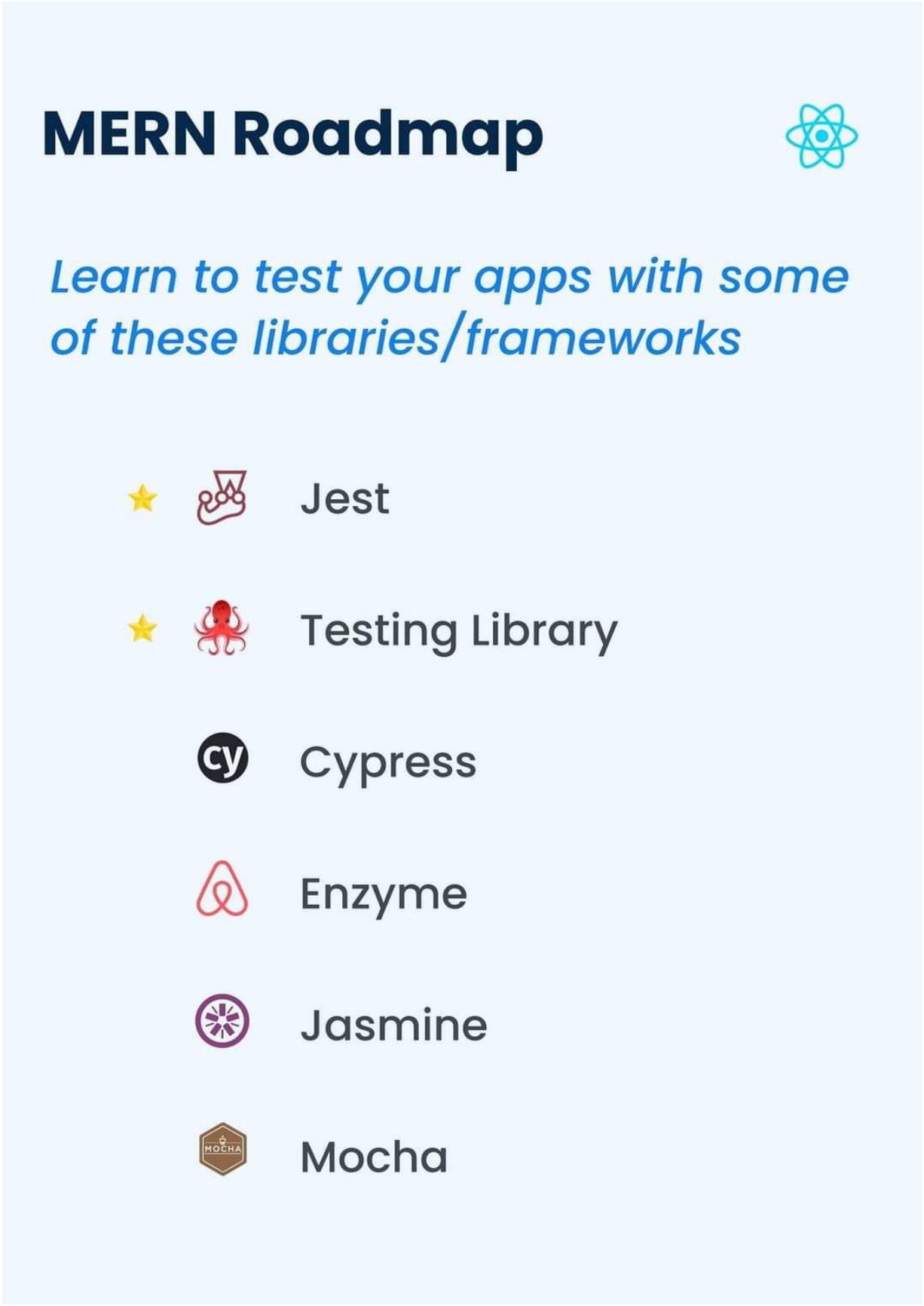
Learn to use some of the most popular React.js packages

- React Router
- Axios
- Styled Components
- React Hook Form
- React Query
- Storybook
- Framer Motion

Chama ITP


MERN Roadmap
Learn how to manage state
with state management tools
Redux
MobX
Hookstate
Recoil
Akita

Chama ITP


MERN Roadmap
Learn to test your apps with some of these libraries/frameworks
Jest
Testing Library
Cypress
Enzyme
Jasmine
Mocha

Chama ITP

MERN Roadmap

Before you dive into backend, you should first consider learning or atleast understanding some of these concepts mentioned below:

- HTTP/HTTPS
- RESTful APIs
- CRUD
- CORS
- JSON
- Package Manager
- MVC architecture
- GraphQL

MERN Roadmap

Now if you feel confident with React.js & concepts mentioned, you can jump into learning Node.js.

As mentioned earlier, NodeJS is a cross-platform JavaScript runtime environment that means we can use JavaScript on the server.

Isn't that amazing? With the help of Node.js we don't have to learn any other programming language. We already know one: **JavaScript**.

Now let's take a look at, what you need to learn in Node.js, well there's not much specific to NodeJS that you have to learn to build a MERN stack application, here are some related things you should take a look at

- Initialising a npm package

Chama ITP

MERN Roadmap

- Installing npm packages through npm or yarn
- Understanding the package.json file
- Create a basic http server in Node.js
- Importing and exports modules
- Working with FileSystem in Node.js
- HTTP Protocols
- Events & Event Emitters
- Global object

Chama ITP

MERN Roadmap ex

After you know the basics of Node.js you can start learning Express.js.

It's the most popular web application framework which uses NodeJS. In MERN stack applications, Express's role is to manage our backend API server.

Express is used to listen to a particular port of our server for requests from the client or the frontend. We can create different routes for each endpoint.

For example:

```
GET http://domain.com/blogs    → Fetches all blogs
GET https://domain.com/blog/1  → Fetches the blog with ID of 1
```

Chama ITP

MERN Roadmap ex

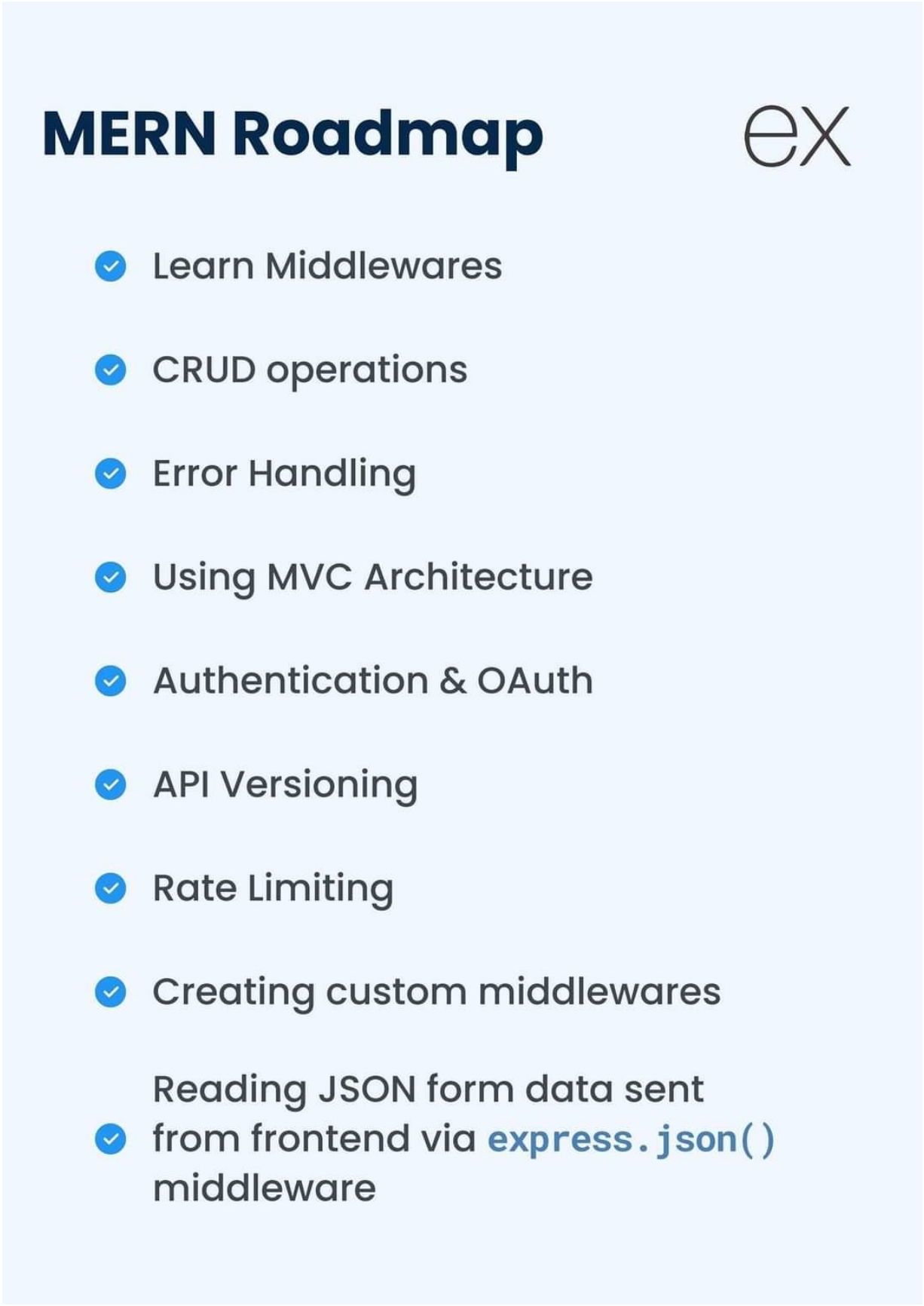
A simple example of Express.js Server.

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.json({ message: "Hello World!" });
});

app.listen(5000);
```

Now, let's dive into things that you should learn concerning Express, as a MERN stack developer:

- Basic server in Express
- Creating & handling routes

Chama ITP


MERN Roadmap
ex
Learn Middlewares
CRUD operations
Error Handling
Using MVC Architecture
Authentication & OAuth
API Versioning
Rate Limiting
Creating custom middlewares
Reading JSON form data sent from frontend via express.json() middleware

MERN Roadmap

Once you know how to use Express.js and you'll start creating projects, you'll notice that you need some kind of database to stores data of your applications, data such as (*user profiles, content, comments, uploads, etc.*)

And that's where MongoDB comes into play. We use MongoDB because it's the most popular one, and it works well in the JavaScript ecosystem, because it's just similar to JSON.

JSON documents created in your React.js front end can be sent to the Express.js server, where they can be processed and (assuming they're valid) stored directly in MongoDB for later retrieval.

Concepts to learn in mongoDB are:

Chama ITP

MERN Roadmap

- SQL Vs NoSQL
- MongoDB database structure
- Setting up local MongoDB or cloud MongoDB Atlas database
- Perform CRUD *(Create, Read, Update & Delete)* operations on the database
- Indexing
- Aggregations
- Ad-hoc query
- Creating models and schemas

Chama ITP

MERN Roadmap

To interact with MongoDB better, we generally use an ODM or Object Data Modelling library like Mongoose.

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

Important things to learn in Mongoose:

- Defining the Schema
- Mongoose data validation
- Understanding mongoose pre and post hooks

Chama ITP

MERN Roadmap

Also there are some essential things you should learn to become a fantastic MERN stack developer.


Git


GitHub


Terminal (CLI)


Postman


Payment Gateways


Testing

MERN Roadmap

Learn to deploy your frontend

Some popular free websites that you can use to deploy your frontend application or client-side code.

Netlify

Vercel

Firebase

Github Pages

Render

MERN Roadmap

Learn to deploy your backend

The 2 popular free services that you can use to deploy your backend application or server-side code.

Vercel

Heroku

MERN Roadmap

You can also use any popular cloud service, almost all cloud services offer 1 year free trial, but you need to put your card information. So it's up to you.


Google Cloud


Digital Ocean

AWS


Azure


Linode

Chama ITP

CRUD

CRUD Stands for

Create Read Update Delete

In a REST environment, CRUD often corresponds to the HTTP methods GET, POST, PUT/PATCH, and DELETE.

If you think about it, every app is a CRUD app in a way. You can find this pattern everywhere:

Building a blog:

create posts, read them, and delete them.

Building a social media network with users

create users, update user profiles, and delete users

Building a movie app

add your favorites, rate, and delete them

Chama ITP

CRUD

In regards to its use in RESTful APIs, CRUD is the standardized use of HTTP Action Verbs.

This means that if you want to create a new record you should be using "POST". If you want to read a record, you should be using "GET". To update a record use "PUT" or "PATCH". And to delete a record, use "DELETE".

Create	→	POST
Read	→	GET
Update	→	PUT / PATCH
Delete	→	DELETE

CRUD

Let's look at some code examples of our Memories application, to understand CRUD better:

```
export const createPost = async (req, res) => {
    const { title, message, selectedFile, creator, tags } = req.body;

    const newPostMessage = new PostMessage(
      { title, message, selectedFile, creator, tags })

    try {
        await newPostMessage.save();
        res.status(201).json(newPostMessage );
    } catch (error) {
        res.status(409).json({ message: error.message });
    }
}
```

CreatePost:

We tell the createPost function that here is the data object for the new Post, please insert it into the database. Or we can say, we're **creating** a new post.

Chama ITP

CRUD

```
export const getPosts = async (req, res) => {
    try {
        const postMessages = await PostMessage.find();
        res.status(200).json(postMessages);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
}
```

GetPosts

Return all the posts from the database. That means we're **reading** all the post from the database.

```
export const updatePost = async (req, res) => {
    const { id } = req.params;
    const { title, message, creator, selectedFile, tags } = req.body;

    const updatedPost =
         { creator, title, message, tags, selectedFile, _id: id };

    await PostMessage.findByIdAndUpdate(id, updatedPost, { new: true });
    res.json(updatedPost);
}
```

UpdatePosts

Find a post with this ID and then update the post with the new data. So we're **updating** the post.

CRUD

```
export const deletePost = async (req, res) => {
    const { id } = req.params;

    if (!mongoose.Types.ObjectId.isValid(id)) {
      return res.status(404).send(`No post with id: ${id}`);
    }
    await PostMessage.findByIdAndRemove(id);
    res.json({ message: "Post deleted successfully." });
}
```

DeletePosts

Find a post with this ID and delete it from the database. So here we're **deleting** a post.

While this application may have more complicated database queries included, without all of the basic CRUD operations this app will be nothing.

Chama ITP

MERN Project Ideas


Social Media App


Chat App


E-commerce Platform

Hotel Booking App

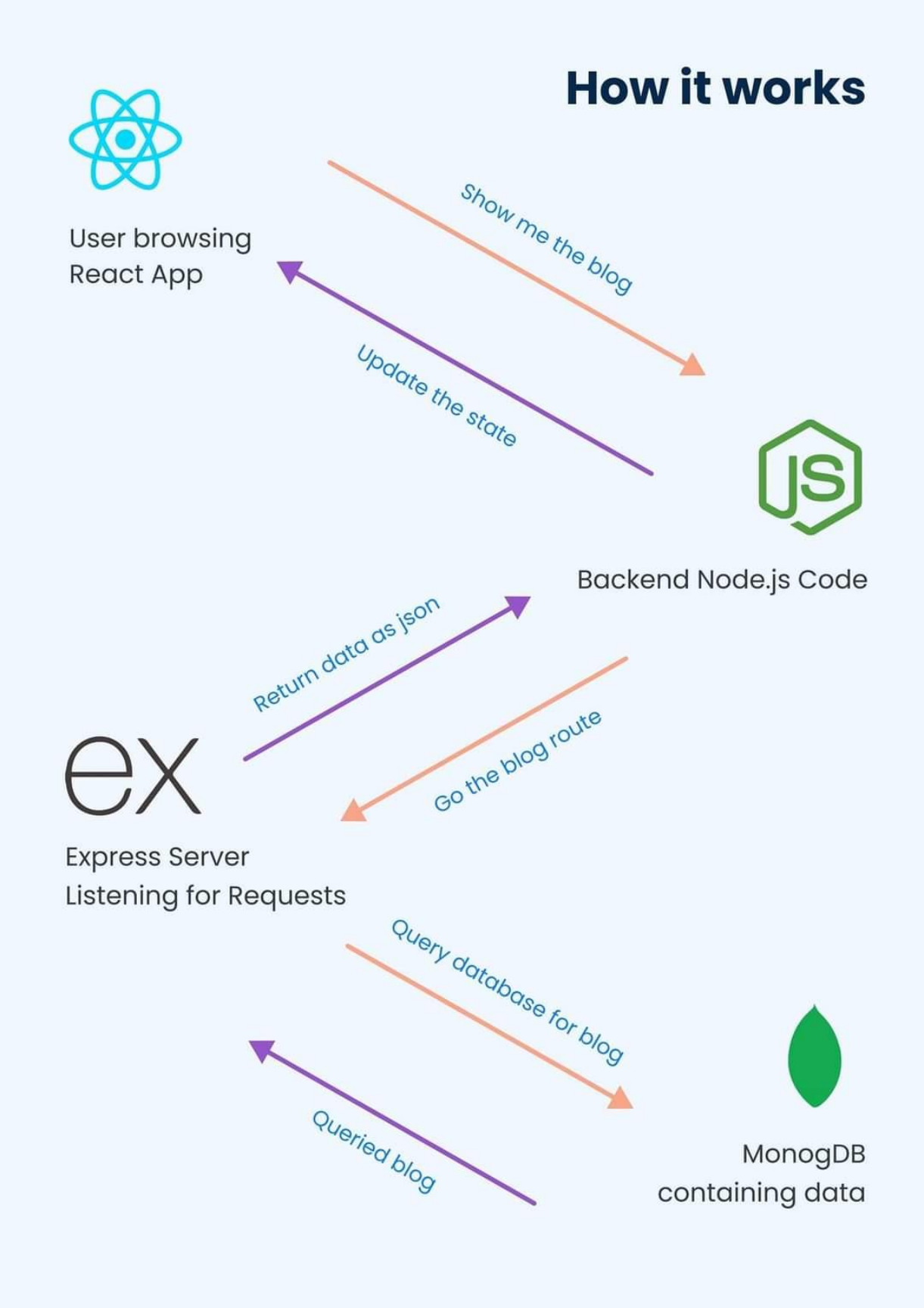

Travel Log App


Task Management Tool


Discord Clone


Bookstore Library

Chama ITP


How it works
User browsing
React App
Show me the blog
Update the state
Backend Node.js Code
Return data as json
Go the blog route
Express Server
Listening for Requests
Query database for blog
Queried blog
MonogDB
containing data

Important Note

You don't need to learn all the things mention in this roadmap to become a MERN stack developer or get a job as a MERN developer.

There is no end of learning in web development there's always something to learn.

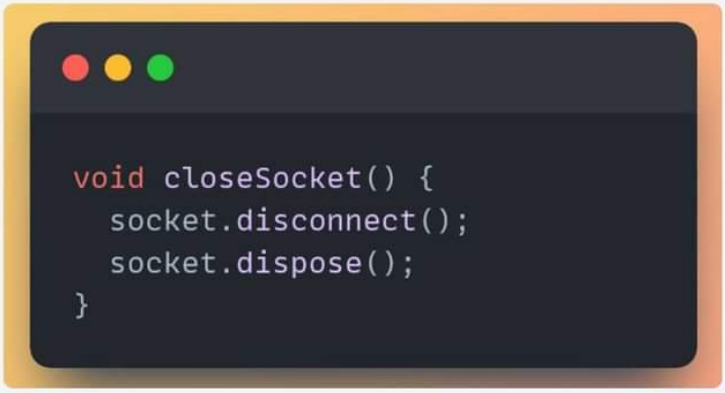
So never stop learning!

Conclusion

- You now have a basic setup to integrate Socket.IO Client in your Flutter app! Implement real-time features like chat, live tracking, and more effortlessly.
- If you encounter any errors with the socket, please leave a comment with the details. I'm happy to help!
- Happy coding!

Closing Socket

- Always close the socket when it's no longer needed.



```
void closeSocket() {
  socket.disconnect();
  socket.dispose();
}
```

Note: If the socket connection fails, verify that the versions of the server and client libraries are compatible with each other.

Emitting Events

- When you want to send data to the server or broadcast an event, use the **emit method**. This allows you to transmit messages or data from the client to the server, such as sending a new message in a chat application.

```
void sendData() {

  final messageData = {
    'messageId': 'abc123',
    'text': 'Hello, this is message!',
    'sender': 'user456',
    'timestamp': DateTime.now().toIso8601String(),
    'isImportant': true
  };

  socket.emit('emitMessageName', messageData);
}
```

- Replace **'emitMessageName'** with the appropriate event name as defined on your server.

Listening Events

- To receive and handle data from the server, use the **on method** to listen for specific events. This enables your app to react to incoming messages or updates, such as displaying new chat messages in real-time.

```
void listenForMessages() {
  socket.on('eventName', (data) {
    print('Message received: $data');
  });
}
```

- Replace **'eventName'** with the event name sent from your server.

Initialize Socket.IO

Brief descriptions for each state:

- **onConnect**: When the connection is established, the onConnect callback will be triggered.
- **onDisconnect**: When the connection is disconnected, the onDisconnect callback will be triggered.
- **onConnectError**: If there is an error during the connection attempt, the onConnectError callback will be triggered.
- **onError**: If any general error occurs, the onError callback will be triggered.

Chama ITP

Initialize Socket.IO

```
IO.Socket socket = IO.io('http://yourserveraddress',
    OptionBuilder()
      .setTransports(['websocket']) // for Flutter or Dart VM
      .disableAutoConnect()  // disable auto-connection
      .setExtraHeaders({'foo': 'bar'}) // optional
      .build()
  );

socket.connect();

// Listen for connection
socket.onConnect((_) => print('Connected to server'));

// Handle errors
socket.onConnectError((err) => print('Connection Error: $err'));
socket.onError((err) => print('Error: $err'));

// Listen for disconnection
socket.onDisconnect((_) => print('Disconnected from server'));
```

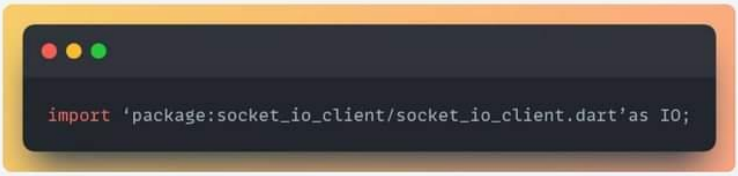
Chama ITP

Add Dependencies

- To start, add the **socket_io_client** package to your pubspec.yaml file:

Import the library

- To start, add the **socket_io_client** package to your pubspec.yaml file:



How Does Socket.IO Work?

Client-Server Communication:

- In our setup, the **Flutter app acts as the client**, while your backend server acts as the server.
- We establish a bidirectional, real-time data transmission between them using Socket.IO.

Steps:

- First, we establish a connection between the client and the server.
- The app listens for events.
- You can emit events to broadcast data to the backend.
- Close the connection between the client and server when it's no longer needed.

What is Socket.IO?

- Socket.IO is a library that enables real-time, bidirectional communication between clients and servers. It's perfect for chat applications, live tracking, online gaming and more!
- Providing real-time communication between the client and server through WebSockets.
- In this post, we will integrate the Socket.IO Client into a Flutter app.

Chama ITP

Q

What is JSX?

A

- JSX stands for JavaScript XML. It is a React extension which allows writing JavaScript code that looks similar to HTML.
- It makes HTML file easy to understand. The JSX file makes the React application robust and boosts its performance.
- JSX provides you to write XML-like syntax in the same file where you write JavaScript code, and then preprocessor (i.e., transpilers like Babel) transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children.

Chama ITP

Example: Customizing the Theme

You can customize the default theme provided by Material-UI to match your application's design needs.

```jsx
import React from 'react';
import { createTheme, ThemeProvider } from '@mui/material/styles';
import Button from '@mui/material/Button';

const theme = createTheme({
  palette: {
    primary: {
      main: '#1976d2',
    },
    secondary: {
      main: '#dc004e',
    },
  },
});

function App() {
  return (
    <ThemeProvider theme={theme}>
      <Button variant="contained" color="primary">
        Custom Primary Color
      </Button>
    </ThemeProvider>
  );
}

export default App;
```

Key Features of Material-UI:

- **Components:** Material-UI offers a wide range of components such as buttons, sliders, app bars, dialogs, icons, and more. These components are designed to be responsive, accessible, and customizable.
- **Theming:** Material-UI supports theming, allowing you to define a custom color palette, typography, spacing, and other design aspects to match your brand or application requirements.
- **Styling:** Material-UI provides several options for styling components, including using CSS-in-JS with the makeStyles and styled APIs, or using traditional CSS classes.
- **Customization:** All components are highly customizable. You can override styles, extend components, and use the sx prop for inline styling.
- **Grid System:** Material-UI includes a powerful grid system based on CSS Flexbox, making it easy to create responsive layouts.
- **Icons:** Material-UI includes a large set of Material Design icons that you can easily incorporate into your applications.
- **Accessibility:** Components are built with accessibility in mind, following WAI-ARIA guidelines to ensure they are usable by people with disabilities.

Installing and Using Material-UI:

To use Material-UI in a React project, you need to install the core package and the icons package if you want to use icons.

```
npm install @mui/material @emotion/react @emotion/styled
npm install @mui/icons-material
```

Once installed, you can start using Material-UI components in your React application:

```
import React from 'react';
import Button from '@mui/material/Button';

function App() {
  return (
    <div>
      <Button variant="contained" color="primary">
        Hello World
      </Button>
    </div>
  );
}

export default App;
```

Chama ITP

Example: Customizing the Theme

You can customize the default theme provided by Material-UI to match your application's design needs.

```jsx
import React from 'react';
import { createTheme, ThemeProvider } from '@mui/material/styles';
import Button from '@mui/material/Button';

const theme = createTheme({
  palette: {
    primary: {
      main: '#1976d2',
    },
    secondary: {
      main: '#dc004e',
    },
  },
});

function App() {
  return (
    <ThemeProvider theme={theme}>
      <Button variant="contained" color="primary">
        Custom Primary Color
      </Button>
    </ThemeProvider>
  );
}

export default App;
```


Material-UI is a popular React component library that implements Google's Material Design. It provides a set of pre-built components that follow Material Design guidelines, allowing developers to create consistent, visually appealing, and responsive user interfaces quickly and easily.
@coding__comics

What is Material Design?

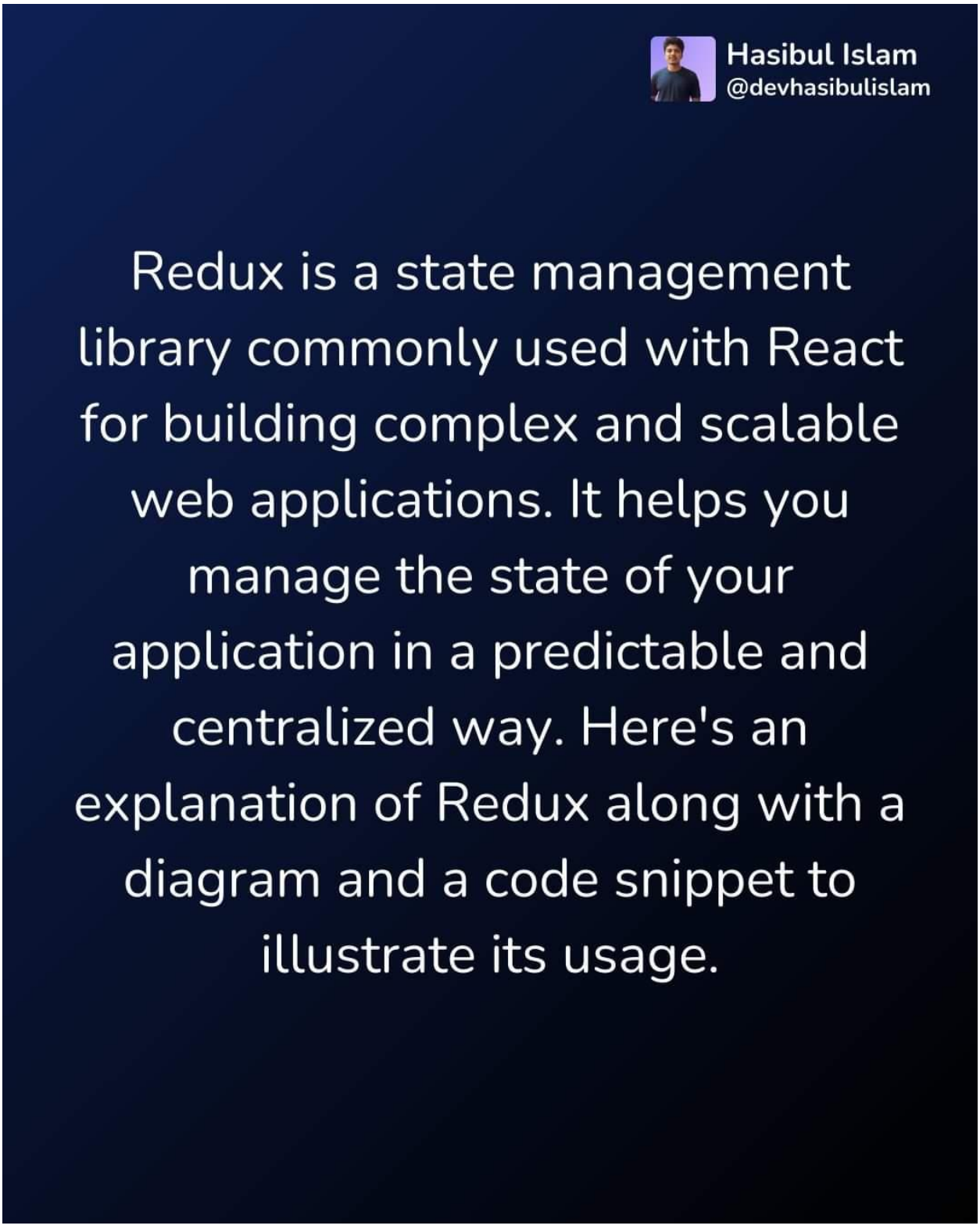
Material Design is a design language developed by Google in 2014. It aims to create a unified experience across all platforms and device sizes, emphasizing the following principles:

- **Material Metaphor:** Surfaces and edges provide visual cues to help users understand the function of an element.
- **Bold, Graphic, Intentional:** Use of deliberate color choices, edge-to-edge imagery, large-scale typography, and intentional white space to create a more engaging UI.
- **Motion Provides Meaning:** Transitions and animations are used to focus attention and maintain continuity.

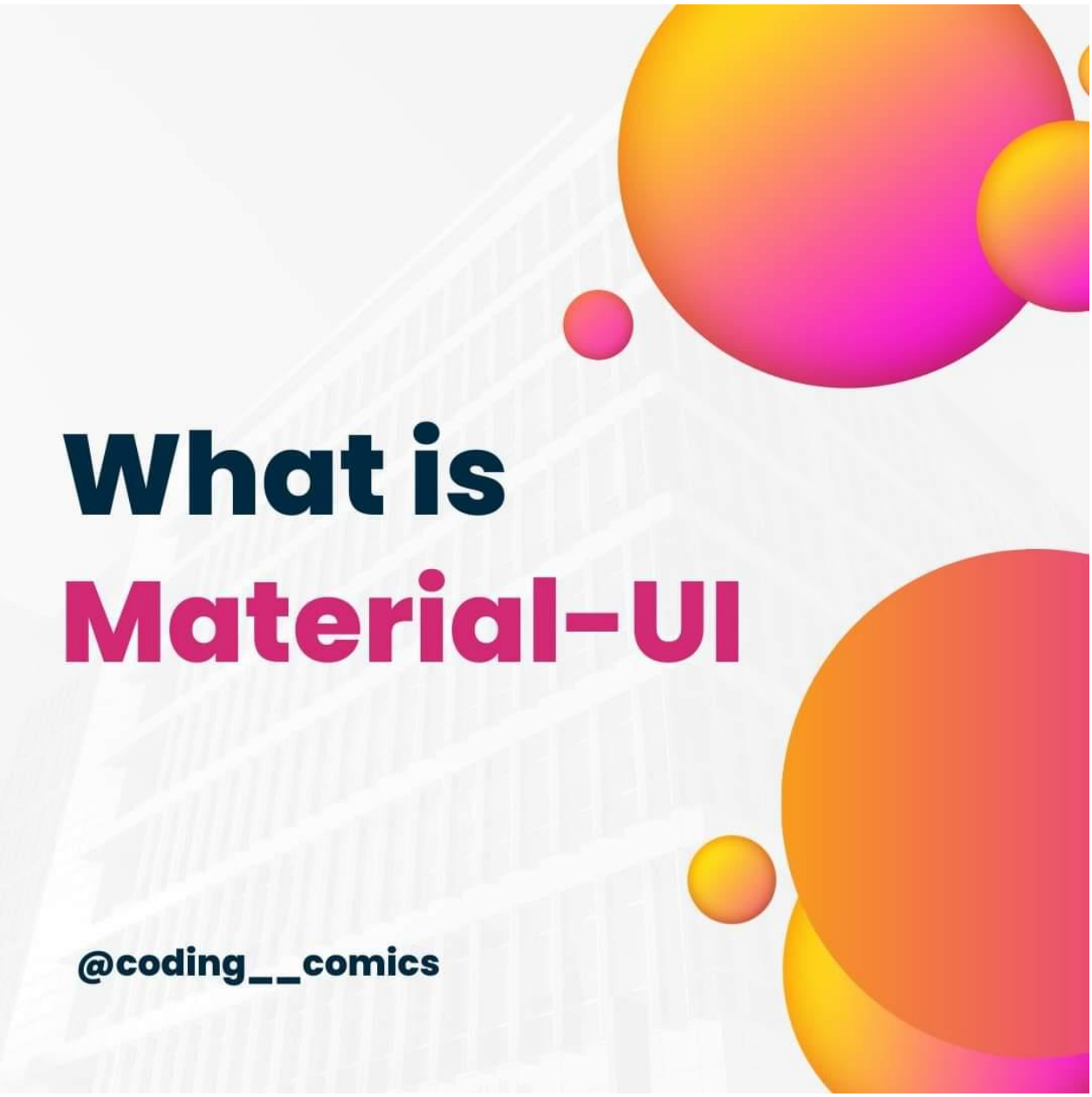
What is Material-UI?

Material-UI is a React-based framework that implements Google's Material Design. It allows developers to build React applications with a consistent design language by providing a set of reusable and customizable components.

Chama ITP


Hasibul Islam
@devhasibulislam
Redux is a state management library commonly used with React for building complex and scalable web applications. It helps you manage the state of your application in a predictable and centralized way. Here's an explanation of Redux along with a diagram and a code snippet to illustrate its usage.

Chama ITP

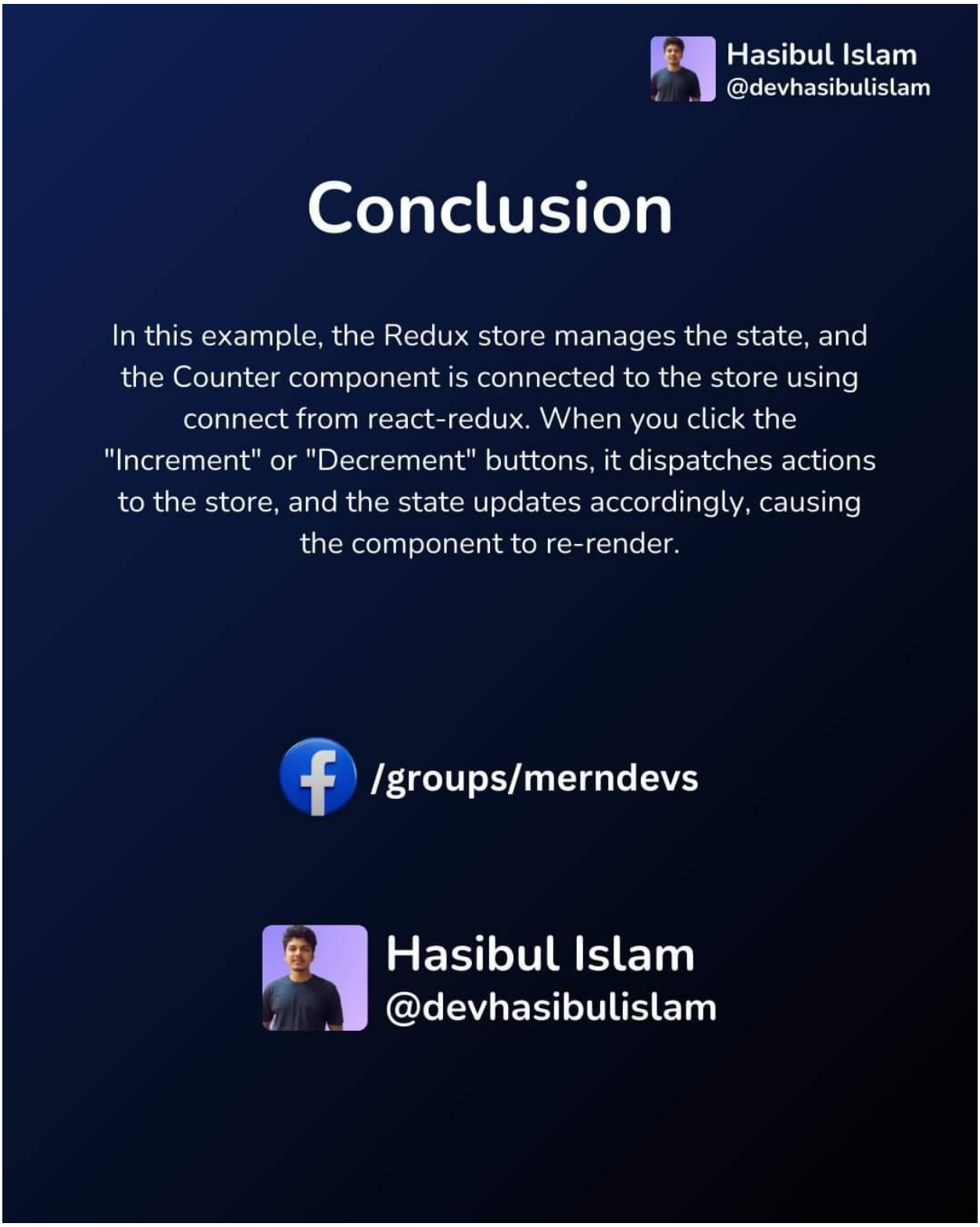



Hasibul Islam
@devhasibulislam
Step 04:
Wrap your app with the Redux Provider
// App.js
import React from 'react';
import { Provider } from 'react-redux';
import store from './store';
import Counter from './Counter';
function App() {
return (
<Provider store={store}>

<Counter />

</Provider>
);
}
export default App;

Chama ITP


Hasibul Islam
@devhasibulislam
Conclusion
In this example, the Redux store manages the state, and the Counter component is connected to the store using connect from react-redux. When you click the "Increment" or "Decrement" buttons, it dispatches actions to the store, and the state updates accordingly, causing the component to re-render.
/groups/merndevs
Hasibul Islam
@devhasibulislam

Chama ITP


Hasibul Islam
@devhasibulislam
Step 03:
Use the store in a React component
// Counter.js
import React from 'react';
import { connect } from 'react-redux';
class Counter extends React.Component {
render() {
return (

Count: {this.props.count}
<button onClick={this.props.increment}>Increment</button>
<button onClick={this.props.decrement}>Decrement</button>

);
}
}
// Define mapStateToProps to map state to props
const mapStateToProps = (state) => ({
count: state.count,
});
// Define mapDispatchToProps to map actions to props
const mapDispatchToProps = (dispatch) => ({
increment: () => dispatch({ type: 'INCREMENT' }),
decrement: () => dispatch({ type: 'DECREMENT' }),
});
// Connect the component to the Redux store
export default connect(mapStateToProps, mapDispatchToProps)(Counter);

Chama ITP


Hasibul Islam
@devhasibulislam
Step 01 & 02
First, you need to install Redux and React-Redux
npm install redux react-redux
Create a Redux store with a reducer:
// store.js
import { createStore } from 'redux';
// Define the initial state
const initialState = {
count: 0,
};
// Define a reducer function
const reducer = (state = initialState, action) => {
switch (action.type) {
case 'INCREMENT':
return { ...state, count: state.count + 1 };
case 'DECREMENT':
return { ...state, count: state.count - 1 };
default:
return state;
}
};
// Create the Redux store
const store = createStore(reducer);
export default store;

Chama ITP