? JavaScript Fundamentals

? 1. Hoisting

- **Definition**: JavaScript's default behavior of moving declarations to the top.
- **Example**:

```
js
CopyEdit
console.log(a); // undefined (not ReferenceError)
var a = 5;
```

- `let` and `const` are hoisted but not initialized — accessing them before declaration gives **ReferenceError**.

? 2. Scope

- **Global**, **Function**, and **Block** scope.
- `var` ? function scoped
- `let/const` ? block scoped
- **Example**:

```
js
CopyEdit
{
  var x = 10;
  let y = 20;
}
console.log(x); // 10
console.log(y); // ReferenceError
```

? 3. Type Coercion

- JS automatically converts data types when needed.
- Example:

```
js
CopyEdit
'5' + 1 // "51" (number converted to string)
'5' - 1 // 4 (string converted to number)
```

? 4. Callback Function

- A function passed as an argument and invoked later.
- Example:

```
js
CopyEdit
function greet(name, callback) {
  callback(`Hello, ${name}`);
}
```

```
greet('Vignesh', console.log);
```

? 5. Map vs forEach

- `map` ? returns a new array
- `forEach` ? just iterates, doesn't return
- Use `map` for transformations.

? React & React Native

```
? 6. Redux (Quick Summary)
1?/?What problem does Redux solve?
Concern How Redux addresses it
Single source of truth Global store object holds all app state.
Predictability State can change only when an action is dispatched and handled by
Debuggability The exact sequence?prev?state ? action ? next?
state is recorded; time?travel debugging possible.

Technically, Redux is just ~200?
LOC of JavaScript; everything else (middlewares, dev?
tools, etc.) sits around that tiny core.

2?/?Classic Redux (2015?2019 style)
js
CopyEdit
// counter/actions.js
export const INCREMENT = 'INCREMENT';
export const increment = () => ({ type: INCREMENT });

// counter/reducer.js
import { INCREMENT } from './actions';
const initialState = { value: 0 };

export default function counterReducer(state = initialState, action) {
  switch (action.type) {
    case INCREMENT:
      return { ...state, value: state.value + 1 };   // manual immutability
    default:
      return state;
  }
}

// store.js
import { createStore, applyMiddleware, combineReducers } from 'redux';
import thunk from 'redux-thunk';

const rootReducer = combineReducers({ counter: counterReducer });
export const store = createStore(
  rootReducer,
  applyMiddleware(thunk)              // devtools wired manually too
);

Downsides you can call out
```

Boilerplate explosion: action types, action creators, switch?
cases, combineReducers, manual immutability spreads.

Manual wiring of middleware & DevTools.

Pattern drift: teams invent different folder structures / naming conventions.

3?/?Redux?
Toolkit (RTK) – the current standard (recommended docs path since 2020)
Why RTK?
Classic Redux pain RTK fix
Lots of repetitive code createSlice generates action creators & reducer in one ca
Immutable updates are verbose Immer is baked in – you can “mutate” draft state.
Async logic patterns vary createAsyncThunk (and RTKQuery) give a first?
class solution.
Manual store setup configureStore autowires thunk, DevTools, serializable check,
Data?fetching boilerplate createApi (RTK Query) handles caching, re?
fetch, polling.
Minimal RTK counter
js
CopyEdit

```
// features/counterSlice.js
import { createSlice, configureStore } from '@reduxjs/toolkit';

const counterSlice = createSlice({
  name: 'counter',
  initialState: { value: 0 },
  reducers: {
    increment: state => {           // <– can “mutate”; Immer generates immutable
      state.value += 1;
    },
  },
});

export const { increment } = counterSlice.actions;

export const store = configureStore({
  reducer: { counter: counterSlice.reducer },
});
```

Usage in a functional component (React Native or web):

js
CopyEdit

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { increment } from './features/counterSlice';

export default function Counter() {
  const value = useSelector(state => state.counter.value);
  const dispatch = useDispatch();

  return (
    <Button title={`Count: ${value}`} onPress={() => dispatch(increment())} />
  );
}
```

4?/?Async logic comparison
Classic Redux “thunk” pattern
js
CopyEdit
```
export const fetchUsers = () => async dispatch => {
  dispatch({ type: 'users/loading' });
  try {
    const res = await fetch('/users').then(r => r.json());
    dispatch({ type: 'users/loaded', payload: res });
  } catch (e) {
    dispatch({ type: 'users/error', error: e.toString() });
  }
};
```

RTK createAsyncThunk
js
CopyEdit
```
// usersSlice.js
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

export const fetchUsers = createAsyncThunk(
  'users/fetch',
  async () => (await fetch('/users')).json()
);

const usersSlice = createSlice({
  name: 'users',
  initialState: { entities: [], status: 'idle' },
  reducers: {},
  extraReducers: builder =>
    builder
      .addCase(fetchUsers.pending,  state => { state.status = 'loading'; })
      .addCase(fetchUsers.fulfilled, (state, action) => {
        state.status = 'succeeded';
        state.entities = action.payload;
      })
      .addCase(fetchUsers.rejected, state => { state.status = 'failed'; }),
});
```

Even nicer: RTK?Query
js
CopyEdit
```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';

export const usersApi = createApi({
  reducerPath: 'usersApi',
  baseQuery: fetchBaseQuery({ baseUrl: '/api' }),
  endpoints: builder => ({
    getUsers: builder.query({
      query: () => '/users',
    }),
  }),
});

export const { useGetUsersQuery } = usersApi;
```

In a component:

```
js
CopyEdit
const { data = [], isLoading } = useGetUsersQuery();
```

Talking point: RTK Query removes 90?
% of CRUD boilerplate, adds caching, and is transport?
agnostic (works in React Native just as well).

5?/?Performance & dev?experience notes you can mention

Area	Classic Redux	Redux Toolkit
Boilerplate	High	Low
TypeScript	Manual types	Auto?generated action & slice types
Mutation safety	Manual spread / Object.assign	Immer draft syntax
DevTools	Extra setup	Enabled by default in configureStore
Serializability check	DIY	Built?in middleware warns about non? serializable state
Bundle size	Slightly smaller core	RTK adds Immer & Reselect? like utils, but tree?shakable
Learning curve	Concept heavy	RTK docs show “start here” pattern? faster onboarding
Data fetching	Custom thunks, sagas, etc.	First?class RTK Query

6?/?How to answer “Do you still need to learn classic Redux?”

“Not really. The Redux core is still there under the hood, so understanding its p
? reducer ? store) is vital. But in day?to?day React?
Native projects we write almost zero raw Redux code – we use Redux?
Toolkit APIs, which are officially recommended by the Redux team and documented a

7?/?Three quick demo lines you can write on a whiteboard

```
js
CopyEdit
// 1?? define slice
const todosSlice = createSlice({ name: 'todos', initialState: [], reducers: {
  addTodo: (state, { payload }) => void state.push(payload),
}});

// 2?? export actions
export const { addTodo } = todosSlice.actions;

// 3?? configure store
export const store = configureStore({ reducer: { todos: todosSlice.reducer } });
```

That one file replaces half a dozen files in the 2016?
style Redux architecture.

Key takeaway sentence

“Redux?
Toolkit is still Redux – it just automates the wiring, lets me write mutable
?looking code with Immer, and gives me batteries?
included async and API layers, so I spend time on business logic instead of boil

? 7. HOC (Higher Order Component)

- A function that takes a component and returns a new one.
- Example:

```
js
CopyEdit
const withLogger = (Component) => (props) => {
  console.log('Rendering:', Component.name);
  return <Component {...props} />;
};
```

? 8. Hooks Syntax (useEffect, useState, useCallback, etc.)

```
js
CopyEdit
const [count, setCount] = useState(0);

useEffect(() => {
  console.log('Component mounted or count changed');
}, [count]);

const memoizedCallback = useCallback(() => {
  console.log('Will only recreate if deps change');
}, []);
```

? 9. useEffect

- Side-effect handler (like API calls, subscriptions)
- **Mounting**: `[]`
- **On state/prop change**: `[dep]`
- **Cleanup**:

```
js
CopyEdit
useEffect(() => {
  const id = setInterval(...);
  return () => clearInterval(id);
}, []);
```

? 10. FlatList vs SectionList

- **FlatList**: Renders flat, scrollable list.
- **SectionList**: List grouped by sections.
- Both are performant with virtualization.

? React Native Specific Topics

? 11. Main Components of React Native

- `View`, `Text`, `Image`, `TextInput`, `ScrollView`, `FlatList`, `TouchableOpacity`, etc.

? 12. Recent Things in React Native

- **Fabric renderer** (new architecture)
- **TurboModules**
- **Hermes** as default JS engine
- **JSI** for better bridging
- **Expo Router** (React Navigation alternative)
- **React Native Skia** (for custom rendering)

? 13. Bridge (JS ? Native)

- Communication layer between JS and Native (Java/Kotlin/Swift).
- New architecture: moves from traditional bridge to **JSI (JavaScript Interface)** for faster sync.

? 14. Styling

- React Native uses a style system similar to CSS-in-JS.
- Example:

```
js
CopyEdit
const styles = StyleSheet.create({
  box: {
    margin: 10,
    padding: 10,
    backgroundColor: 'lightblue'
  }
});
```

? 15. Padding vs Margin

- **Padding**: space inside the element.
- **Margin**: space outside the element.
- ? Analogy: Padding is **inside the box**, Margin is **outside the box**.

? Security & Storage in RN

? 16. Security Aspects

- ? Use HTTPS for APIs
- ? Enable **Proguard** (Android) & Bitcode (iOS)
- ? Secure sensitive data using:
- `react-native-keychain`
- `SecureStore` (Expo)
- ? Avoid logging sensitive info
- ? Obfuscate JS code (e.g., Metro config)

? 17. Storage Options

- `AsyncStorage` (non-secure key-value store)
- Secure alternatives:
- `react-native-encrypted-storage`
- `SecureStore` (Expo)
- **SQLite** for structured data
- **MMKV** for fast, persistent key-value

? Navigation Peer Dependencies

? 18. React Navigation Peer Deps (v6+)

Make sure you install:

```
bash
CopyEdit
npm install @react-navigation/native
npm install react-native-screens react-native-safe-area-context react-native-ges
```

For stack navigation:

```
bash
CopyEdit
npm install @react-navigation/native-stack
```

Remember to link and wrap your app with `NavigationContainer`.