React Interview Script - Complete Guide

Every React concept explained with: **Interviewer Question → Your Answer → Code Example → VibeVault Project Reference**

---

1. VITE BASICS

What is Vite?

Interviewer: "What is Vite?" **You:** "Vite is a next-generation frontend build tool that provides instant dev server startup and lightning-fast HMR (Hot Module Replacement)."

```
// Vite uses native ES modules for instant startup
// No bundling during development - files served directly

// Start dev server
npm run dev

// Vite serves files on-demand
// Only bundles for production
```

In VibeVault: "I use Vite for the React frontend - it starts in milliseconds vs CRA's 30+ seconds."

Why Vite vs Create React App (CRA)?

Interviewer: "Why did you choose Vite over CRA?" **You:** "Vite is faster because it uses native ES modules and esbuild. CRA bundles everything before starting."

```
| Feature           | CRA           | Vite           |
|-------------------|---------------|----------------|
| Dev startup       | 30+ seconds   | <1 second      |
| HMR               | Slow          | Instant        |
| Build tool        | Webpack       | Rollup/esbuild |
| Bundle size       | Larger        | Smaller        |
| Config            | Ejection      | Simple js file |
```

In VibeVault: "CRA took 45 seconds to start. Vite starts instantly. That's why I switched."

Vite Project Structure

Interviewer: "Explain Vite project structure." **You:** "Vite has a minimal structure with index.html at root, src folder for components, and vite.config.js for configuration."

```
vibeVault-frontend/
├── index.html           # Entry point (at root, not public)
├── vite.config.js       # Vite configuration
├── package.json
├── .env                 # Environment variables
├── public/              # Static assets (copied as-is)
│   └── favicon.ico
└── src/
    ├── main.jsx         # React entry point
    ├── App.jsx          # Root component
    ├── components/      # Reusable components
    ├── pages/           # Page components
    ├── hooks/           # Custom hooks
    ├── contexts/        # Context providers
    ├── utils/           # Helper functions
    └── api/             # API calls
```

---

vite.config.js

Interviewer: "What goes in vite.config.js?" **You:** "Plugins, path aliases, server settings, and build options."

```
// vite.config.js
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import path from 'path'

export default defineConfig({
  // Line 6: React plugin for JSX support
  plugins: [react()],

  // Line 8-12: Path aliases for clean imports
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src'),
      '@components': path.resolve(__dirname, './src/components'),
      '@hooks': path.resolve(__dirname, './src/hooks'),
    }
  },

  // Line 15-17: Dev server settings
  server: {
    port: 5173,
    proxy: {
      '/api': 'http://localhost:8000'  // Proxy to Django
    }
  }
})
```

In VibeVault: "I configured aliases so I can write `import Button from '@components/Button'` "

ENV Files

Interviewer: "How do you handle environment variables in Vite?" **You:** "Files named `.env`, `.env.local`, `.env.production`. Variables must start with `VITE_` to be exposed."

```
# .env
VITE_API_URL=http://localhost:8000/api/v1
VITE_APP_NAME=VibeVault

# .env.production
VITE_API_URL=https://api.vibevault.com/api/v1
```

```
// Usage in code
const API_URL = import.meta.env.VITE_API_URL
console.log(import.meta.env.MODE)  // 'development' or 'production'
```

In VibeVault: "I use `VITE_API_URL` to switch between local Django and production API."

Dev Server & Production Build

Interviewer: "How do you run and build a Vite project?" **You:** "npm run dev for development, npm run build for production bundle."

```
# Development (instant start, HMR)
npm run dev

# Production build (optimized, minified)
npm run build

# Preview production build locally
npm run preview

# Output structure after build:
dist/
├── index.html
└── assets/
    ├── index-abc123.js   # Hashed for cache busting
    └── index-def456.css
```

---

2. REACT BASICS

What is JSX?

Interviewer: "What is JSX?" **You:** "JSX is JavaScript XML - a syntax extension that lets you write HTML-like code in JavaScript."

```
// JSX
const element = <h1 className="title">Hello, {userName}!</h1>

// Compiles to
const element = React.createElement(
  'h1',
  { className: 'title' },
  'Hello, ', userName, '!'
)

// Key JSX rules:
// 1. className instead of class
// 2. Expressions in curly braces {}
// 3. Must have single parent element
// 4. Self-closing tags: <img />
```

Functional Components

Interviewer: "What are functional components?" **You:** "JavaScript functions that return JSX. They're the modern way to build React components."

```
// VibeVault MemoryCard component
function MemoryCard({ memory }) {
  return (
    <div className="memory-card">
      <h3>{memory.title}</h3>
      <p>{memory.text}</p>
      <span className="emotion">{memory.emotion}</span>
    </div>
  )
}

// Arrow function syntax
const MemoryCard = ({ memory }) => (
  <div className="memory-card">
    {/* content */}
  </div>
)

export default MemoryCard
```

In VibeVault: "All my components are functional - MemoryCard, MemoryList, Dashboard."

Props

Interviewer: "What are props?" **You:** "Props are read-only inputs passed from parent to child components."

```
// Parent passes props
<MemoryCard
  memory={memoryData}
  onDelete={handleDelete}
  isEditable={true}
/>
```

```
// Child receives props
function MemoryCard({ memory, onDelete, isEditable }) {
  return (
    <div>
      <h3>{memory.title}</h3>
      {isEditable && (
        <button onClick={() => onDelete(memory.id)}>Delete</button>
      )}
    </div>
  )
}

// Default props
MemoryCard.defaultProps = {
  isEditable: false
}
```

In VibeVault: "I pass memory data as props: `<MemoryCard memory={m} />` "

State

Interviewer: "What is state in React?" **You:** "State is component-owned data that can change over time. When state changes, the component re-renders."

```
import { useState } from 'react'

function MemoryForm() {
  // Line 4: useState returns [value, setter]
  const [title, setTitle] = useState('')
  const [isSubmitting, setIsSubmitting] = useState(false)

  const handleSubmit = async () => {
    setIsSubmitting(true)
    await createMemory({ title })
    setIsSubmitting(false)
  }

  return (
    <input
      value={title}
      onChange={(e) => setTitle(e.target.value)}
      disabled={isSubmitting}
    />
  )
}
```

In VibeVault: "I use useState for form inputs, loading states, and UI toggles."

Events

Interviewer: "How do you handle events in React?" **You:** "Using camelCase event handlers like onClick, onChange, onSubmit."

```
function SearchBar({ onSearch }) {
  const [query, setQuery] = useState('')

  // Handle input change
  const handleChange = (e) => {
    setQuery(e.target.value)
  }

  // Handle form submit
  const handleSubmit = (e) => {
    e.preventDefault()  // Prevent page reload
    onSearch(query)
  }

  return (
    <form onSubmit={handleSubmit}>
      <input
        value={query}
        onChange={handleChange}
        placeholder="Search memories..."
      />
      <button type="submit">Search</button>
    </form>
  )
}
```

Conditional Rendering

Interviewer: "How do you conditionally render elements?" **You:** "Using ternary operators, && for short-circuit, or early returns."

```
function MemoryCard({ memory, isLoading, error }) {
  // Early return
  if (isLoading) return <Spinner />
  if (error) return <ErrorMessage error={error} />

  return (
    <div>
      {/* Ternary operator */}
      {memory.emotion
        ? <EmotionBadge emotion={memory.emotion} />
        : <span>No emotion detected</span>
      }

      {/* Short-circuit && */}
      {memory.isAnalyzed && <AnalysisComplete />}
      {/* Nullish coalescing */}
      <p>{memory.text ?? 'No content'}</p>
    </div>
  )
}
```

Lists & Keys

Interviewer: "How do you render lists? Why are keys important?" **You:** "Using .map() to render arrays. Keys help React identify which items changed."

```
function MemoryList({ memories }) {
  return (
    <ul>
      {memories.map((memory) => (
        // Key must be unique and stable
        <li key={memory.id}>
          <MemoryCard memory={memory} />
        </li>
      ))}
    </ul>
  )
}

// DON'T use index as key if list can reorder
{items.map((item, index) => (
  <li key={index}>...</li>  // BAD - causes issues on reorder
))}

// DO use unique IDs
{items.map((item) => (
  <li key={item.id}>...</li>  // GOOD
))}
```

In VibeVault: "I use `memory.id` as key when rendering the memory list."

Fragments

Interviewer: "What are Fragments?" **You:** "Fragments let you group elements without adding extra DOM nodes."

```
// Problem: Extra div in DOM
function UserInfo() {
  return (
    <div>  {/* Unnecessary wrapper */}
      <span>Name</span>
      <span>Email</span>
    </div>
  )
}
```

```
// Solution: Fragment
function UserInfo() {
  return (
    <>  {/* Short syntax */}
      <span>Name</span>
      <span>Email</span>
    </>
  )
}

// With key (needed in lists)
{items.map(item => (
  <React.Fragment key={item.id}>
    <dt>{item.term}</dt>
    <dd>{item.description}</dd>
  </React.Fragment>
))}
```

---

3. STYLING

CSS Modules

Interviewer: "What are CSS Modules?" **You:** "CSS files where class names are scoped locally by default, preventing conflicts."

```
/* MemoryCard.module.css */
.card {
  padding: 1rem;
  border-radius: 8px;
}

.title {
  font-size: 1.5rem;
  color: var(--primary);
}
```

```
// MemoryCard.jsx
import styles from './MemoryCard.module.css'

function MemoryCard({ memory }) {
  return (
    // Classes become unique: card_abc123
    <div className={styles.card}>
      <h3 className={styles.title}>{memory.title}</h3>
    </div>
  )
}
```

In VibeVault: "I use CSS Modules for component-specific styles to avoid conflicts."

Inline Styles

Interviewer: "When do you use inline styles?" **You:** "For dynamic styles that change based on props or state."

```
function ProgressBar({ percentage }) {
  // Dynamic style based on props
  const barStyle = {
    width: `${percentage}%`,
    backgroundColor: percentage > 80 ? 'green' : 'yellow',
    transition: 'width 0.3s ease'
  }

  return (
    <div className="progress-container">
      <div style={barStyle} />
    </div>
  )
}

// Emotion color based on sentiment
<span style={{
  color: sentiment === 'positive' ? '#10b981' : '#ef4444'
}}>
  {sentiment}
</span>
```

---

4. HOOKS – BASIC

---

useState

Interviewer: "Explain useState." **You:** "useState adds state to functional components. It returns [currentValue, setterFunction]."

```
import { useState } from 'react'

function Counter() {
  // Primitive state
  const [count, setCount] = useState(0)

  // Object state
  const [user, setUser] = useState({ name: '', email: '' })

  // Array state
  const [memories, setMemories] = useState([])

  // Update primitive
  setCount(count + 1)
  setCount(prev => prev + 1)  // Functional update (preferred)

  // Update object (must spread)
  setUser({ ...user, name: 'John' })
  setUser(prev => ({ ...prev, name: 'John' }))

  // Update array
  setMemories([...memories, newMemory])
  setMemories(prev => prev.filter(m => m.id !== id))
}
```

---

useEffect

Interviewer: "Explain useEffect." **You:** "useEffect handles side effects - API calls, subscriptions, DOM operations. Runs after render."

```
import { useEffect, useState } from 'react'

function MemoryList() {
  const [memories, setMemories] = useState([])

  // Run on mount (empty deps)
  useEffect(() => {
    fetchMemories().then(setMemories)
  }, [])

  // Run when userId changes
  useEffect(() => {
    fetchUserMemories(userId).then(setMemories)
  }, [userId])

  // Cleanup (WebSocket, subscriptions)
  useEffect(() => {
    const ws = connectWebSocket()

    return () => {
      ws.close()  // Cleanup on unmount
    }
  }, [])
}
```

In VibeVault: "I use useEffect to fetch memories on mount and for WebSocket connections."

---

useRef

Interviewer: "What is useRef?" **You:** "useRef holds a mutable value that persists across renders without causing re-renders."

```
import { useRef, useEffect } from 'react'

function AudioPlayer({ audioUrl }) {
  // DOM reference
  const audioRef = useRef(null)

  // Persist value without re-render
  const playCountRef = useRef(0)

  const handlePlay = () => {
    audioRef.current.play()
    playCountRef.current += 1  // Doesn't trigger re-render
  }

  // Focus input on mount
  const inputRef = useRef(null)
  useEffect(() => {
    inputRef.current.focus()
  }, [])

  return (
    <>
      <audio ref={audioRef} src={audioUrl} />
      <input ref={inputRef} />
    </>
  )
}
```

In VibeVault: "I use useRef for audio/video player controls."

---

useContext

Interviewer: "What is useContext?" **You:** "useContext consumes values from a Context without prop drilling."

```
// 1. Create context
const AuthContext = createContext(null)

// 2. Provider wraps app
function AuthProvider({ children }) {
  const [user, setUser] = useState(null)

  return (
    <AuthContext.Provider value={{ user, setUser }}>
      {children}
    </AuthContext.Provider>
  )
}
```

```
// 3. Consume with useContext
function Navbar() {
  const { user } = useContext(AuthContext)

  return (
    <nav>
      {user ? `Welcome, ${user.name}` : 'Please login'}
    </nav>
  )
}
```

In VibeVault: "I use AuthContext to share user state across all components."

---

5. HOOKS – ADVANCED

---

useReducer

Interviewer: "When do you use useReducer over useState?" **You:** "For complex state logic with multiple sub-values or when next state depends on previous."

```
import { useReducer } from 'react'

// Reducer function
function memoryReducer(state, action) {
  switch (action.type) {
    case 'FETCH_START':
      return { ...state, loading: true, error: null }
    case 'FETCH_SUCCESS':
      return { ...state, loading: false, memories: action.payload }
    case 'FETCH_ERROR':
      return { ...state, loading: false, error: action.payload }
    case 'ADD_MEMORY':
      return { ...state, memories: [...state.memories, action.payload] }
    case 'DELETE_MEMORY':
      return {
        ...state,
        memories: state.memories.filter(m => m.id !== action.payload)
      }
    default:
      return state
  }
}

function MemoryList() {
  const [state, dispatch] = useReducer(memoryReducer, {
    memories: [],
    loading: false,
    error: null
  })

  // Dispatch actions
  dispatch({ type: 'FETCH_START' })
  dispatch({ type: 'FETCH_SUCCESS', payload: data })
}
```

---

useCallback

Interviewer: "What is useCallback?" **You:** "useCallback memoizes a function so it doesn't get recreated on every render."

```
import { useCallback, useState } from 'react'

function MemoryList() {
  const [memories, setMemories] = useState([])

  // Without useCallback - new function every render
  const handleDelete = (id) => {
    setMemories(prev => prev.filter(m => m.id !== id))
  }

  // With useCallback - same function reference
  const handleDeleteMemo = useCallback((id) => {
    setMemories(prev => prev.filter(m => m.id !== id))
  }, [])  // Empty deps = never recreated

  // Pass to child - prevents unnecessary re-renders
  return memories.map(m => (
    <MemoryCard
      key={m.id}
      memory={m}
      onDelete={handleDeleteMemo}  // Stable reference
    />
  ))
}
```

---

useMemo

Interviewer: "What is useMemo?" **You:** "useMemo memoizes a computed value so expensive calculations don't run on every render."

```
import { useMemo, useState } from 'react'

function MemoryStats({ memories }) {
  // Expensive calculation
  const stats = useMemo(() => {
    console.log('Computing stats...')
    return {
      total: memories.length,
      positive: memories.filter(m => m.sentiment === 'positive').length,
      avgScore: memories.reduce((a, m) => a + m.score, 0) / memories.length
    }
  }, [memories])  // Only recompute when memories change

  return (
    <div>
      <p>Total: {stats.total}</p>
      <p>Positive: {stats.positive}</p>
    </div>
  )
}
```

Custom Hooks

Interviewer: "What are custom hooks?" **You:** "Functions that start with 'use' and can call other hooks. They encapsulate reusable logic."

```
// Custom hook for API fetch
function useMemories() {
  const [memories, setMemories] = useState([])
  const [loading, setLoading] = useState(true)
  const [error, setError] = useState(null)

  useEffect(() => {
    fetchMemories()
      .then(setMemories)
      .catch(setError)
      .finally(() => setLoading(false))
  }, [])

  return { memories, loading, error }
}

// Custom hook for auth
function useAuth() {
  const [user, setUser] = useState(null)

  const login = async (credentials) => {
    const response = await api.login(credentials)
    setUser(response.user)
    localStorage.setItem('token', response.token)
  }

  const logout = () => {
    setUser(null)
    localStorage.removeItem('token')
  }
  return { user, login, logout }
}

// Usage
function Dashboard() {
  const { memories, loading } = useMemories()
  const { user, logout } = useAuth()
}
```

In VibeVault: "I have useAuth, useMemories, and useWebSocket custom hooks."

---

6. FOLDER ARCHITECTURE

Project Structure

Interviewer: "How do you organize a React project?" **You:** "By feature/functionality with clear separation of concerns."

```
src/
├── components/          # Reusable UI components
│   ├── common/          # Button, Input, Modal
│   ├── memory/          # MemoryCard, MemoryList
│   └── layout/          # Header, Sidebar, Footer
│
├── pages/               # Route-level components
│   ├── Dashboard.jsx
│   ├── Login.jsx
│   └── MemoryDetail.jsx
│
├── hooks/               # Custom hooks
│   ├── useAuth.js
│   ├── useMemories.js
│   └── useWebSocket.js
│
├── contexts/            # Context providers
│   ├── AuthContext.jsx
│   └── ThemeContext.jsx
│
├── utils/               # Helper functions
│   ├── formatDate.js
│   └── validators.js
│
├── api/                 # API layer
│   ├── api.js           # Axios instance
│   ├── auth.js          # Auth endpoints
│   └── memories.js      # Memory endpoints
│
├── layouts/             # Page layouts
│   ├── MainLayout.jsx
│   └── AuthLayout.jsx
│
└── styles/              # Global styles
    └── index.css
```

Aliased Imports

Interviewer: "What are aliased imports?" **You:** "Path shortcuts configured in vite.config.js to avoid relative path hell."

```
// Without aliases (messy)
import Button from '../../../components/common/Button'
import useAuth from '../../../../hooks/useAuth'

// With aliases (clean)
import Button from '@/components/common/Button'
import useAuth from '@/hooks/useAuth'

// vite.config.js setup
resolve: {
  alias: {
    '@': path.resolve(__dirname, './src'),
  }
}

// jsconfig.json for IDE support
{
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "@/*": ["src/*"]
    }
  }
}
```

---

7. REACT ROUTER

BrowserRouter, Routes, Route

Interviewer: "How do you set up routing?" **You:** "Using react-router-dom with BrowserRouter, Routes, and Route."

```
// main.jsx
import { BrowserRouter } from 'react-router-dom'

ReactDOM.createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
)
```

```
// App.jsx
import { Routes, Route } from 'react-router-dom'

function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/login" element={<Login />} />
      <Route path="/dashboard" element={<Dashboard />} />
      <Route path="/memory/:id" element={<MemoryDetail />} />
      <Route path="*" element={<NotFound />} />
    </Routes>
  )
}
```

---

useNavigate & useParams

Interviewer: "How do you navigate programmatically and get URL params?" **You:** "useNavigate for navigation, useParams for URL parameters."

```
import { useNavigate, useParams } from 'react-router-dom'

function MemoryCard({ memory }) {
  const navigate = useNavigate()

  const handleClick = () => {
    navigate(`/memory/${memory.id}`)
  }

  const handleBack = () => {
    navigate(-1)  // Go back
  }

  return <div onClick={handleClick}>...</div>
}

function MemoryDetail() {
  // Get :id from URL /memory/:id
  const { id } = useParams()

  useEffect(() => {
    fetchMemory(id)
  }, [id])
}
```

Protected Routes

Interviewer: "How do you protect routes?" **You:** "Create a wrapper component that checks auth and redirects if not authenticated."

```
import { Navigate, Outlet } from 'react-router-dom'
import { useAuth } from '@/hooks/useAuth'

function ProtectedRoute() {
  const { user, loading } = useAuth()

  if (loading) return <LoadingSpinner />

  // Redirect to login if not authenticated
  if (!user) {
    return <Navigate to="/login" replace />
  }

  // Render child routes
  return <Outlet />
}

// Usage in App.jsx
<Routes>
  <Route path="/login" element={<Login />} />

  {/* Protected routes */}
  <Route element={<ProtectedRoute />}>
    <Route path="/dashboard" element={<Dashboard />} />
    <Route path="/memories" element={<MemoryList />} />
    <Route path="/memory/:id" element={<MemoryDetail />} />
  </Route>
</Routes>
```

In VibeVault: "All routes except login/register are wrapped with ProtectedRoute."

Lazy Loading Routes

Interviewer: "How do you lazy load routes?" **You:** "Using React.lazy() and Suspense for code splitting."

```
import { lazy, Suspense } from 'react'

// Lazy load components
const Dashboard = lazy(() => import('@/pages/Dashboard'))
const MemoryDetail = lazy(() => import('@/pages/MemoryDetail'))
const Settings = lazy(() => import('@/pages/Settings'))

function App() {
  return (
    <Suspense fallback={<LoadingSpinner />}>
      <Routes>
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="/memory/:id" element={<MemoryDetail />} />
        <Route path="/settings" element={<Settings />} />
      </Routes>
    </Suspense>
  )
}
```

---

8. FORMS & VALIDATION

Controlled Components

Interviewer: "What are controlled components?" **You:** "Form elements where React controls the value via state."

```
function MemoryForm() {
  const [formData, setFormData] = useState({
    title: '',
    text: '',
    mediaType: 'text'
  })

  const handleChange = (e) => {
    const { name, value } = e.target
    setFormData(prev => ({ ...prev, [name]: value }))
  }

  const handleSubmit = (e) => {
    e.preventDefault()
    createMemory(formData)
  }

  return (
    <form onSubmit={handleSubmit}>
      <input
        name="title"
        value={formData.title}
        onChange={handleChange}
      />
      <textarea
        name="text"
        value={formData.text}
        onChange={handleChange}
      />
      <button type="submit">Create</button>
    </form>
  )
}
```

File Upload & Image Preview

Interviewer: "How do you handle file uploads?" **You:** "Using input type='file' with FileReader for preview."

```
function FileUpload({ onFileSelect }) {
  const [preview, setPreview] = useState(null)

  const handleFileChange = (e) => {
    const file = e.target.files[0]
    if (!file) return

    // Create preview URL
    const reader = new FileReader()
    reader.onloadend = () => {
      setPreview(reader.result)
    }
    reader.readAsDataURL(file)

    // Pass file to parent
    onFileSelect(file)
  }

  return (
    <div>
      <input
        type="file"
        accept="image/*,audio/*"
        onChange={handleFileChange}
      />
      {preview && <img src={preview} alt="Preview" />}
    </div>
  )
}
```

In VibeVault: "Users can upload photos/audio. I show preview before submission."

---

9. API INTEGRATION

Axios Instance & Interceptors

Interviewer: "How do you set up Axios for API calls?" **You:** "Create an instance with base URL and interceptors for auth tokens."

```
// api/api.js
import axios from 'axios'

const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL,
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json'
  }
})

// Request interceptor - add auth token
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('accessToken')
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})

// Response interceptor - handle errors
api.interceptors.response.use(
  (response) => response,
  async (error) => {
    if (error.response?.status === 401) {
      // Try to refresh token or logout
      localStorage.removeItem('accessToken')
      window.location.href = '/login'
    }
    return Promise.reject(error)
  }
)

export default api
```

In VibeVault: "I use interceptors to automatically attach JWT and handle 401 errors."

Loading & Error States

Interviewer: "How do you handle loading and error states?" **You:** "Using state variables and conditional rendering."

```
function MemoryList() {
  const [memories, setMemories] = useState([])
  const [loading, setLoading] = useState(true)
  const [error, setError] = useState(null)

  useEffect(() => {
    fetchMemories()
      .then(data => setMemories(data))
      .catch(err => setError(err.message))
      .finally(() => setLoading(false))
  }, [])

  if (loading) return <Spinner />
  if (error) return <ErrorMessage message={error} />
  if (memories.length === 0) return <EmptyState />
  return (
    <div>
      {memories.map(m => <MemoryCard key={m.id} memory={m} />)}
    </div>
  )
}
```

Pagination

Interviewer: "How do you implement pagination?" **You:** "Track current page in state, fetch data with page parameter."

```
function MemoryList() {
  const [memories, setMemories] = useState([])
  const [page, setPage] = useState(1)
  const [totalPages, setTotalPages] = useState(1)

  useEffect(() => {
    fetchMemories(page).then(response => {
      setMemories(response.results)
      setTotalPages(Math.ceil(response.count / 20))
    })
  }, [page])

  return (
    <>
      {memories.map(m => <MemoryCard key={m.id} memory={m} />)}

      <div className="pagination">
        <button
          onClick={() => setPage(p => p - 1)}
          disabled={page === 1}
        >
          Previous
        </button>
        <span>Page {page} of {totalPages}</span>
        <button
          onClick={() => setPage(p => p + 1)}
          disabled={page === totalPages}
        >
          Next
        </button>
      </div>
    </>
  )
}
```

---

10. AUTHENTICATION

JWT Token Storage & Auth Flow

Interviewer: "How do you handle JWT authentication?" **You:** "Store tokens in localStorage, attach to API requests via interceptor."

```
// contexts/AuthContext.jsx
const AuthContext = createContext()

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null)
  const [loading, setLoading] = useState(true)

  // Check for existing token on mount
  useEffect(() => {
    const token = localStorage.getItem('accessToken')
    if (token) {
      api.get('/auth/profile/')
        .then(res => setUser(res.data))
        .catch(() => localStorage.removeItem('accessToken'))
        .finally(() => setLoading(false))
    } else {
      setLoading(false)
    }
  }, [])

  const login = async (credentials) => {
    const response = await api.post('/auth/login/', credentials)
    localStorage.setItem('accessToken', response.data.access)
    localStorage.setItem('refreshToken', response.data.refresh)
    setUser(response.data.user)
  }

  const logout = () => {
    localStorage.removeItem('accessToken')
    localStorage.removeItem('refreshToken')
    setUser(null)
  }

  return (
    <AuthContext.Provider value={{ user, login, logout, loading }}>
      {children}
    </AuthContext.Provider>
  )
}
```

In VibeVault: "JWT flow: Login → Store tokens → Attach to requests → Refresh on expiry."

11. STATE MANAGEMENT

Context API vs Redux

Interviewer: "When do you use Context vs Redux?" **You:** "Context for simple global state (auth, theme). Redux for complex state with many actions."

```
// Context - simple, built-in
const ThemeContext = createContext()

// Redux Toolkit - for complex state
import { createSlice, configureStore } from '@reduxjs/toolkit'

const memoriesSlice = createSlice({
  name: 'memories',
  initialState: { items: [], loading: false },
  reducers: {
    setLoading: (state, action) => { state.loading = action.payload },
    setMemories: (state, action) => { state.items = action.payload },
    addMemory: (state, action) => { state.items.push(action.payload) },
    deleteMemory: (state, action) => {
      state.items = state.items.filter(m => m.id !== action.payload)
    }
  }
})
```

In VibeVault: "I use Context for auth/theme. Redux would be overkill for this app size."

12. PERFORMANCE OPTIMIZATION

React.memo

Interviewer: "What is React.memo?" **You:** "A HOC that memoizes a component - it only re-renders if props change."

```
// Without memo - re-renders on every parent render
function MemoryCard({ memory }) {
  return <div>{memory.title}</div>
}

// With memo - skips render if props same
const MemoryCard = React.memo(function MemoryCard({ memory }) {
  console.log('Rendering:', memory.title)
  return <div>{memory.title}</div>
})

// Custom comparison
```

```
const MemoryCard = React.memo(
  function MemoryCard({ memory }) { ... },
  (prevProps, nextProps) => prevProps.memory.id === nextProps.memory.id
)
```

Avoid Re-renders

Interviewer: "How do you avoid unnecessary re-renders?" **You:** "Use React.memo, useMemo, useCallback, and proper state structure."

```
// BAD - creates new object every render
<ChildComponent style={{ color: 'red' }} />

// GOOD - stable reference
const style = useMemo(() => ({ color: 'red' }), [])
<ChildComponent style={style} />

// BAD - new function every render
<Button onClick={() => handleClick(id)} />

// GOOD - stable callback
const handleClickMemo = useCallback(() => handleClick(id), [id])
<Button onClick={handleClickMemo} />
```

13. UTILITY SKILLS

Debounce/Throttle

Interviewer: "What is debouncing?" **You:** "Debouncing delays execution until user stops typing. Throttling limits execution rate."

```
import { useState, useEffect } from 'react'

function SearchBar({ onSearch }) {
  const [query, setQuery] = useState('')

  // Debounce - wait 500ms after last keystroke
  useEffect(() => {
    const timer = setTimeout(() => {
      if (query) onSearch(query)
    }, 500)

    return () => clearTimeout(timer)
  }, [query])

  return (
    <input
```

```
      value={query}
      onChange={(e) => setQuery(e.target.value)}
      placeholder="Search..."
    />
  )
}
```

In VibeVault: "I debounce the semantic search to avoid excessive API calls."

Toasts

Interviewer: "How do you show notifications?" **You:** "Using a toast library like react-hot-toast."

```
import toast, { Toaster } from 'react-hot-toast'

// In App.jsx
function App() {
  return (
    <>
      <Routes>...</Routes>
      <Toaster position="top-right" />
    </>
  )
}

// Usage anywhere
function MemoryForm() {
  const handleSubmit = async () => {
    try {
      await createMemory(data)
      toast.success('Memory created!')
    } catch (error) {
      toast.error('Failed to create memory')
    }
  }
}
```

---

14. TESTING

Vitest & React Testing Library

Interviewer: "How do you test React components?" **You:** "Using Vitest for test runner and React Testing Library for component testing."

```
// MemoryCard.test.jsx
import { render, screen, fireEvent } from '@testing-library/react'
import { describe, it, expect, vi } from 'vitest'
import MemoryCard from './MemoryCard'
```

```
describe('MemoryCard', () => {
  const mockMemory = {
    id: 1,
    title: 'Test Memory',
    text: 'Content here',
    emotion: 'joy'
  }

  it('renders memory title', () => {
    render(<MemoryCard memory={mockMemory} />)
    expect(screen.getByText('Test Memory')).toBeInTheDocument()
  })

  it('calls onDelete when clicked', () => {
    const handleDelete = vi.fn()
    render(<MemoryCard memory={mockMemory} onDelete={handleDelete} />)

    fireEvent.click(screen.getByText('Delete'))
    expect(handleDelete).toHaveBeenCalledWith(1)
  })
})
```

---

15. DEPLOYMENT

Vite Build for Production

Interviewer: "How do you deploy a Vite React app?" **You:** "npm run build creates optimized static files. Serve them via nginx or Django."

```
# Build for production
npm run build

# Output in dist/
dist/
├── index.html
└── assets/
    ├── index-abc123.js    # Code-split, minified
    └── index-def456.css

# Serve with Django (WhiteNoise)
# Copy dist/ to Django's staticfiles
# Configure nginx to serve index.html for all routes
```

Serve via Django + WhiteNoise

Interviewer: "How do you integrate React with Django?" **You:** "Build React, copy to Django staticfiles, use WhiteNoise to serve."

```python
# Django settings.py
STATICFILES_DIRS = [
    BASE_DIR / 'frontend' / 'dist',  # Vite build output
]

# Add WhiteNoise middleware
MIDDLEWARE = [
    'whitenoise.middleware.WhiteNoiseMiddleware',
]

# urls.py - catch-all for SPA
from django.views.generic import TemplateView

urlpatterns = [
    path('api/', include('api.urls')),
    path('', TemplateView.as_view(template_name='index.html')),
]
```

CORS Configuration

Interviewer: "How do you handle CORS between React and Django?" **You:** "Install django-cors-headers and configure allowed origins."

```python
# Django settings.py
INSTALLED_APPS = ['corsheaders', ...]

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',  # Must be high
    ...
]

CORS_ALLOWED_ORIGINS = [
    'http://localhost:5173',  # Vite dev server
]
```

In VibeVault: "In production, React is served by Django so no CORS needed. In dev, I allow localhost:5173."