Study Guide / Cheat Sheet: Fullstack Web Development — Week 1

React Foundations, State Management, and Express.js

---

1. React Quick Reference

Functional Component Syntax

```
// Named function
function MyComponent(props) {
  return <div>Hello, {props.name}</div>;
}

// Arrow function with destructured props
const MyComponent = ({ name, age = 25 }) => {
  return <div>Hello, {name}! Age: {age}</div>;
};

export default MyComponent;
```

JSX Rules

Rule	Description	Example
One Root Element	Must return a single root	`<>...</>` (Fragment) or `<div>...</div>`
`className`	Use instead of `class`	`<div className="header">`
Expressions in `{}`	Embed JS expressions	`<p>Count: {count * 2}</p>`
Self-closing tags	Required for void elements	`<img src="..." />` `<br />`
camelCase attributes	HTML attrs use camelCase	`onClick`, `htmlFor`, `tabIndex`

Props: Passing, Destructuring, Default Values

```
// Parent passes props
function Parent() {
  return <Child name="Bob" age={30} />;
}

// Child destructures with defaults
function Child({ name, age, status = "active" }) {
  return <p>{name}, {age}, {status}</p>;
}
```

useState

```
import { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  // Direct update
  const reset = () => setCount(0);

  // Functional update (use when new state depends on previous)
  const increment = () => setCount(prev => prev + 1);
  const decrement = () => setCount(prev => prev - 1);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}
```

Rules:

- Never mutate state directly: `count++` is WRONG, use `setCount(prev => prev + 1)`
- For arrays: `setItems([...items, newItem])` not `items.push(newItem)`
- For objects: `setUser({...user, name: 'New'})` not `user.name = 'New'`

useEffect

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

function DataFetcher({ userId }) {
  const [data, setData] = useState(null);

  // Runs ONCE on mount (empty dependency array)
  useEffect(() => {
    console.log('Component mounted');
    return () => console.log('Component unmounted'); // Cleanup
  }, []);

  // Runs when userId changes
  useEffect(() => {
    console.log(`Fetching user ${userId}`);
    // Simulate fetch
    setTimeout(() => setData({ id: userId, name: 'User' }), 1000);
  }, [userId]); // Dependency array

  // NO dependency array = runs on EVERY render (usually a mistake!)
  // useEffect(() => { ... }); // DANGER: can cause infinite loops
  return <div>{data ? data.name : 'Loading...'}</div>;
}
```

Dependency Array Rules:

Syntax	When it runs
`useEffect(fn, [])`	Once on mount
`useEffect(fn, [a, b])`	When `a` or `b` changes
`useEffect(fn)`	Every render (usually wrong!)

Controlled Forms

```
function MyForm() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault(); // CRITICAL: prevents page reload
    console.log({ name, email });
    setName('');
    setEmail('');
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Name"
      />
      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Email"
      />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Conditional Rendering

```
function Display({ isLoading, items, error }) {
  // Early return
  if (error) return <p className="error">{error}</p>;
  // Ternary operator
  if (isLoading) return <p>Loading...</p>;

  return (
    <div>
      {/* Logical && */}
      {items.length === 0 && <p>No items found.</p>}

      {/* Ternary in JSX */}
      {items.length > 0 ? (
        <ul>{items.map(item => <li key={item.id}>{item.name}</li>)}</ul>
      ) : null}
    </div>
  );
}
```

Lists and Keys

```
function TaskList({ tasks }) {
  return (
    <ul>
      {tasks.map(task => (
        <li key={task.id}> {/* key must be unique and stable */}
          {task.title} - {task.status}
        </li>
      ))}
    </ul>
  );
}

// WRONG: Don't use array index as key if list can reorder
// tasks.map((task, index) => <li key={index}>...)  // BAD
```

React Router v6

```
import { BrowserRouter, Routes, Route, Link, useNavigate, useParams } from 'react-router-dom';

// Navigation links
function Nav() {
  return (
    <nav>
      <Link to="/">Home</Link>
      <Link to="/about">About</Link>
      <Link to="/users/42">User 42</Link>
    </nav>
  );
}

// Dynamic route parameter
function UserPage() {
  const { id } = useParams(); // extracts :id from URL
  return <h2>User #{id}</h2>;
}

// Programmatic navigation
function LoginPage() {
  const navigate = useNavigate();
  const handleLogin = () => {
    // ... login logic
    navigate('/dashboard'); // redirect after login
  };
  return <button onClick={handleLogin}>Login</button>;
}

// Route setup (in App.jsx)
function App() {
  return (
    <BrowserRouter>
      <Nav />
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/about" element={<AboutPage />} />
        <Route path="/users/:id" element={<UserPage />} />
        <Route path="*" element={<NotFoundPage />} /> {/* 404 catch-all */}
      </Routes>
    </BrowserRouter>
  );
}
```

2. Express Quick Reference

Basic Server Setup

```
const express = require('express');
const cors = require('cors');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 3000;

// Built-in middleware
app.use(express.json());  // Parse JSON request bodies
app.use(cors());           // Enable Cross-Origin Resource Sharing

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

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Route Methods

```
// GET - Read data
app.get('/api/items', (req, res) => {
  res.json(items); // 200 is default
});

// POST - Create data
app.post('/api/items', (req, res) => {
  const newItem = { id: Date.now(), ...req.body };
  items.push(newItem);
  res.status(201).json(newItem); // 201 Created
});

// PUT - Update data
app.put('/api/items/:id', (req, res) => {
  const item = items.find(i => i.id === parseInt(req.params.id));
  if (!item) return res.status(404).json({ message: 'Not found' });
  Object.assign(item, req.body);
  res.json(item);
});

// DELETE - Remove data
app.delete('/api/items/:id', (req, res) => {
  const index = items.findIndex(i => i.id === parseInt(req.params.id));
  if (index === -1) return res.status(404).json({ message: 'Not found' });
  items.splice(index, 1);
  res.status(204).send(); // 204 No Content
});
```

Request Parameters

```
// URL params: /api/users/42
app.get('/api/users/:id', (req, res) => {
  const userId = req.params.id; // "42"
});

// Query string: /api/search?q=react&limit=10
app.get('/api/search', (req, res) => {
  const query = req.query.q;     // "react"
  const limit = req.query.limit; // "10"
});

// Request body (POST/PUT): { "name": "New Item" }
app.post('/api/items', (req, res) => {
  const { name, description } = req.body;
});
```

Middleware Pattern

```
// Custom logging middleware
const logger = (req, res, next) => {
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.path}`);
  next(); // MUST call next() to continue to next middleware/route
};
app.use(logger); // Apply to all routes

// Route-specific middleware
const validateTask = (req, res, next) => {
  const { title } = req.body;
  if (!title) return res.status(400).json({ message: 'Title is required' });
  if (title.length > 255) return res.status(400).json({ message: 'Title too long' });
  next();
};
app.post('/api/tasks', validateTask, createTaskHandler);
```

Error Handling

```
// 404 handler (AFTER all routes)
app.use((req, res) => {
  res.status(404).json({ message: `Route ${req.path} not found` });
});

// Global error handler (MUST have 4 parameters)
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ message: 'Internal server error' });
});

// Triggering error handler from a route
app.get('/api/risky', (req, res, next) => {
  try {
    // risky operation
  } catch (err) {
    next(err); // Passes to error handler
  }
});
```

HTTP Status Codes

Code	Name	When to Use
200	OK	Successful GET, PUT
201	Created	Successful POST (resource created)

204	No Content	Successful DELETE
400	Bad Request	Invalid input / validation error
401	Unauthorized	Missing or invalid auth token
403	Forbidden	Valid auth but insufficient permissions
404	Not Found	Resource doesn't exist
409	Conflict	Duplicate resource (e.g., email already registered)
500	Internal Server Error	Unhandled server error

---

3. Common Mistakes

Top 10 React Mistakes

#	Mistake	Fix
1	Mutating state directly (`count++`)	Use setter: `setCount(prev => prev + 1)`
2	Missing `key` prop in lists	Add unique `key={item.id}` to mapped elements
3	Infinite loop in useEffect (no deps)	Always provide dependency array: `useEffect(fn, [])`
4	Forgetting `e.preventDefault()` in forms	First line of onSubmit: `e.preventDefault()`
5	Using `class` instead of `className`	JSX uses `className` for CSS classes
6	Not wrapping Router correctly	`<BrowserRouter>` must wrap `<Routes>`
7	Returning multiple root elements	Wrap in `<>...</>` (Fragment)
8	Using index as key for dynamic lists	Use stable unique IDs from data
9	Forgetting to import hooks	`import { useState, useEffect } from 'react'`
10	Calling setState in render body	Put state updates in event handlers or useEffect

Top 10 Express Mistakes

#	Mistake	Fix
1	Forgetting `next()` in middleware	Always call `next()` to continue the chain
2	`req.body` is undefined	Add `app.use(express.json())` before routes
3	404 handler before routes	Place 404 handler AFTER all route definitions
4	Sending multiple responses	Only send one `res.json()` or `res.send()` per request
5	Hardcoding port number	Use `process.env.PORT || 3000`
6	No CORS middleware	Add `app.use(cors())` for frontend access
7	Not handling async errors	Wrap async handlers in try/catch, call `next(err)`

8	Error handler with 3 params	Must have exactly 4: `(err, req, res, next)`
9	`req.params.id` is a string	Parse with `parseInt(req.params.id)` for number comparison
10	Forgetting `app.listen()`	Server won't start without it