1. React: Вступ

- **React** — JS-бібліотека для створення **інтерфейсів користувача (UI)**.
- Основна ідея: **компоненти + Virtual DOM + односпрямований потік даних**.
- **JSX** — синтаксис, що дозволяє писати HTML-подібний код у JavaScript.
- Старт: `npm create vite@latest my-app --template react`.

```
function App() {
  return <h1>Привіт, React!</h1>;
}
```

---

2. Стан (state) та пропси (props)

Props

- **Props** — вхідні дані компонента, передаються **з батька в дитину**.
- ТІЛЬКИ для читання (immutable).

```
function Greeting({ name }) {
  return <h1>Привіт, {name}!</h1>;
}

// Виклик:
<Greeting name="Тимур" />
```

State

- **State** — внутрішній стан компонента, який може змінюватися.
- При зміні стану компонент **перерендерюється**.

```
const [count, setCount] = useState(0);
setCount(count + 1);
```

Головне:

- `props` — приходять ЗЗОВНІ;
- `state` — живе ВСЕРЕДИНІ компонента.

---

3. Хуки: `useState`, `useEffect`

`useState`

- Додає стан у функціональний компонент.

```
const [value, setValue] = useState(початковеЗначення);
```

`useEffect`

- Для **побічних ефектів**: запити, таймери, підписки, робота з DOM.

```
useEffect(() => {
  // ефект
}, [dependencies]); // коли залежності змінюються
```

Варіанти:

- Без другого аргумента → після кожного рендера.
- `[]` → тільки один раз (монтування).
- `[x]` → при зміні `x`.

---

4. React Router (маршрутизація)

- Дає змогу створювати **SPA** з різними “сторінками” без перезавантаження.

Основні елементи:

```
<BrowserRouter>  // обгортка всього застосунку
  <Routes>       // контейнер маршрутів
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
  </Routes>
</BrowserRouter>
```

Навігація:

```
<Link to="/about">Про нас</Link>
<NavLink to="/" className={({ isActive }) => isActive ? "active" : ""} />
```

Динамічні маршрути:

```
<Route path="/user/:id" element={<UserPage />} />

const { id } = useParams();
```

---

5. Управління станом: Context, Redux

Context API

- Для **глобальних даних** (тема, мова, користувач), щоб не робити prop drilling.

```
const ThemeContext = createContext();

<ThemeContext.Provider value={value}>
  <App />
</ThemeContext.Provider>

const theme = useContext(ThemeContext);
```

Redux (Redux Toolkit)

- **Централізоване сховище (store)** всього стану застосунку.

- Використовується у великих проєктах.

Ключові поняття: `store`, `reducer`, `action`, `dispatch`.

```
const counterSlice = createSlice({
  name: "counter",
  initialState: { value: 0 },
  reducers: {
    increment(state) { state.value += 1; },
  },
});

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

У компоненті:

```
const value = useSelector(state => state.counter.value);
const dispatch = useDispatch();
dispatch(increment());
```

---

6. Оптимізація продуктивності

Основні інструменти:

- **`React.memo`** — не рендерить компонент заново, якщо props не змінилися.
- **`useCallback`** — мемоізація функцій (щоб не створювати нові при кожному рендері).
- **`useMemo`** — мемоізація важких обчислень.

```
const MemoComponent = React.memo(Component);

const handleClick = useCallback(() => {
  // ...
}, [deps]);

const result = useMemo(() => {
  return heavyCalc(data);
}, [data]);
```

Інші ідеї:

- Правильні `key` у списках (`key={item.id}`);
- Віртуалізація великих списків (`react-window`);
- Code splitting: `React.lazy` + `Suspense`.

---

7. Міні-висновок по React

- React дозволяє створювати **швидкі та динамічні** інтерфейси на базі компонентів.
- Стан (`state`) і пропси (`props`) керують тим, що й як відображається.

- Хуки (`useState`, `useEffect`) дають функціональним компонентам “мозок” і “поведінку”.
- React Router, Context, Redux і оптимізація продуктивності роблять додаток **масштабованим і зручним**.