Tabii Erhan, **Redux'ta kullanılan terimleri en TEMELDEN ve EN BASİT mantıkla**, örneklerle, hayat benzetmeleriyle ve çok anlaşılır şekilde anlatayım.
Şu an amacımız:
Redux'u öğrenmeden önce bilmen gereken "alfabe"yi oturtmak.

Hazırsan başlıyoruz.
Sıfırdan → taş gibi bir temel kuruyoruz. 🔥

🎯 ÖĞRENMEN GEREKEN TEMEL REDUX TERİMLERİ

Redux için bilmen gereken 8 temel terim var:

1. **State**
2. **Action**
3. **Dispatch**
4. **Reducer**
5. **Store**
6. **Initial State**
7. **Slice (Redux Toolkit)**
8. **Provider**

Bunları tek tek **çocuk bile anlar gibi açıklayacağım**
Sonra **gerçek projeden örnek** vererek oturtacağız.

1️⃣ STATE — Durum (Verinin şu anki hali)

Redux'ta **state**, uygulamanın o anki verisidir.

👉 Telefonun ekran parlaklığı,
👉 Sepetteki ürün sayısı,
👉 Kullanıcının adı,
👉 Tema (dark/light)…

Hepsi **state**.

Örnek state:

```
{
  cartItems: [],
  user: { name: 'Erhan' },
  theme: 'light'
}
```

State = Uygulamanın kalbi.

2️⃣ ACTION — Bir olay bildirimi

Action, sadece bir **nesnedir**.
Ve “ne yapacağını” söyler, “nasıl yapacağını” söylemez.

Örnek action:

```
{
  type: 'cart/add',
  payload: { id: 1, name: 'Product A', price: 60 }
}
```

Bunu Türkçeye çevir:

👉 “Sepete bu ürünü eklemek istiyorum.”

Action yalnızca **istek bildirir**, işin kendisini yapmaz.

Action = Emirdir.
“Bunu yap!” demek.

3️⃣ DISPATCH — Action'ı store'a göndermek

Dispatch etmek:
Bir action’ı Redux’a göndermek demektir.

Mesela:

```
dispatch(addToCart(product));
```

Türkçesi:

👉 “Redux, bu ürünü sepete ekle!” diye haber gönder.

Dispatch = Komutu göndermek.

4️⃣ REDUCER — State’i değiştiren tek yer

Reducer, action’ı alır → yeni state'i oluşturur.

Asla eski state'i değiştirmez
(yani mutate yok → RTK bunu arka planda güvenli yapar)

Reducer örneği:

```
addToCart(state, action) {
  state.cartItems.push(action.payload);
}
```

Türkçesi:

👉 “Gelen ürünü, sepet listesine ekle.”

Reducer = İşçidir.
Action emir verir → reducer işi yapar.

5️⃣ STORE — Tüm state'in tutulduğu büyük depo

Redux'ta tüm state tek bir yerde tutulur → buna **store** denir.

Telefonun hafızası gibi düşün:

```
{
  cart: { cartItems: [] },
  user: { name: 'Erhan' }
}
```

Store = Uygulamanın merkezi veri deposu.

6️⃣ INITIAL STATE — Başlangıç durumu

Her slice’ın bir başlangıç verisi olur.

Örnek:

```
const initialState = {
  cartItems: []
}
```

Yani:

👉 “Uygulama ilk açıldığında sepet boş olsun.”

initialState = İlk açılış verisi.

7️⃣ SLICE — Redux Toolkit’te state + reducer bir arada

Redux Toolkit gelince her şeyi sadeleştirdi.

Bir slice şunları içerir:

- initialState
- reducers (addToCart, remove, clearCart)
- action creator’lar (otomatik oluşuyor)
- reducer fonksiyonları

Örnek:

```
const cartSlice = createSlice({
  name: 'cart',
  initialState,
  reducers: {
    addToCart(state, action) {},
    removeFromCart(state, action) {},
  }
});
```

Slice = Uygulamanın bir parçasını yöneten mini Redux dosyası.

8️⃣ PROVIDER — React’a store’u tanıtır

React uygulaması, Redux store’u kullanabilsin diye
uygulamanın en dışını Provider ile sararsın.

```
<Provider store={store}>
  <App />
</Provider>
```

Bu olmasa:

- Component’ler state’e ulaşamaz
- dispatch kullanamazsın
- Redux çalışmaz

Provider = Redux'u React'a bağlayan köprü.

🔥 Redux Veri Akışı (Özet Hikâye)

Bu süreci günlük hayattan bir örnekle anlatayım:

📦 Sepete ürün eklemek hikâyesi

Erhan bir ürün görüyor → sepete eklemek istiyor.

1. UI'da butona basarsın

```
Add to Cart
```

2. Component, Redux'a emir gönderir

```
dispatch(addToCart(product))
```

3. Redux action yaratır

```
{ type: 'cart/addToCart', payload: product }
```

4. Reducer bu action'ı alır

Ürünü `cartItems` içine ekler.

5. Store yeni state'i saklar

```
cartItems: [...]
```

6. Component değişen state'i görür → UI yenilenir

Sepetteki ürün sayısı artar.

🚀 En Basit Örnek (1 ürün ekleme)

1. Action yaratılır

```
addToCart(product)
```

2. Dispatch edilir

```
dispatch(addToCart(product))
```

3. Reducer çalışır

```
state.cartItems.push(product)
```

4. Store yeni state'i sunar

5. UI güncellenir