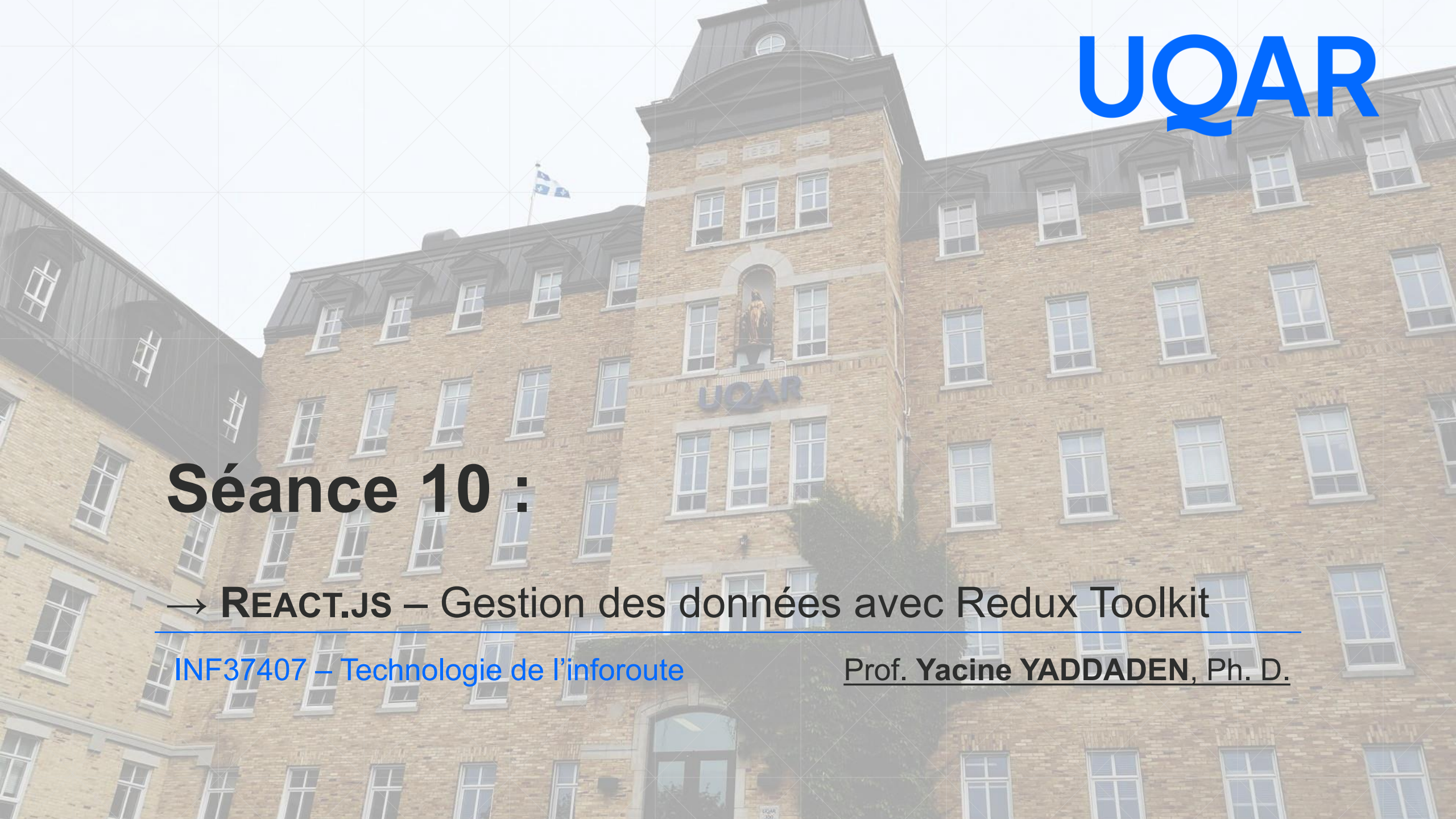


Séance 10 :

→ REACT.JS – Gestion des données avec Redux Toolkit

INF37407 – Technologie de l'inforoute

Prof. **Yacine YADDADEN**, Ph. D.

UQAR

Plan

1. Notions fondamentales de la gestion d’état
2. Modèles d’architecture d’état
3. Mise en pratique avec **Redux Toolkit**
4. **Zustand** — gestion d’état simplifiée
5. Zustand vs Redux Toolkit
6. Questions & Discussion

UQAR

Notions fondamentales de la gestion d'état

→ Qu'est-ce qu'un « état » ?

- L'état représente **l'ensemble des données** nécessaires pour déterminer :
  - ☑ ce que l'interface affiche
  - ☑ comment elle réagit aux interactions
- Il évolue dans le temps → **state = data + temporalité**
- Exemples :
  - Texte d'un champ de formulaire
  - Position d'un menu ouvert/fermé
  - Résultats récupérés via une API
  - Etc.

UQAR

Notions fondamentales de la gestion d'état

→ Types d'état dans une application *front-end*

1. **Local State** (*état local*)
   - Déclaré dans un composant via *useState* ou *useReducer*
   - Idéal pour :
2. **Shared / Global State** (*état partagé*)
   - Utilisé par plusieurs composants
   - Exemples : utilisateur connecté, thème global, panier
3. **Server State**
   - Données provenant d'une API externe
   - Exemples :
     - fetch de produits
     - informations utilisateur via backend
   - Problème clé : **synchro client ↔ serveur**
4. **UI State**
   - États purement visuels
   - Exemples :
     - Modals
     - sidebar ouverte
     - Etc.

UQAR

Notions fondamentales de la gestion d'état

→ Exemple simple en **React** : *useState*

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

return (
  <>
    <p>Valeur : {count}</p>
    <button onClick={() => setCount(count + 1)}>+</button>
  </>
);
```

UQAR

Notions fondamentales de la gestion d'état

→ Pourquoi aller vers des modèles plus robustes ?

- **Limites des solutions natives** (React uniquement)
  - ☒ *useState* → trop local
  - ☒ *Context* → utile mais mauvaise pefromances si trop utilisé
  - ☒ Pas de structure pour les gros projets
  - ☒ Pas de log centralisé des actions (debugging faible)
- **Besoins réels d'une application moderne**
  - ☑ Un **store global** clair
  - ☑ Une architecture **prévisible** (state → action → new state)
  - ☑ Des outils :
    → devtools

UQAR

Modèles d’architecture d’état

→ Flux de données : unidirectionnel vs bidirectionnel

- **Flux unidirectionnel** (**React**, Vue)
  - ☑ **Data** → **View** → **Events** → **New Data**
  - ☑ L’état descend, les actions remontent
  - ☑ Prévisible, simple à déboguer
  - ☑ Favorise la structure en composants
- **Flux bidirectionnel** (Angular, MVVM)
  - ☑ **Data** ↔ **View** (two-way binding)
  - ☑ Mise à jour automatique dans les deux sens
  - ☑ Très rapide pour développer, mais risque de **liaisons multiples difficiles à suivre**

UQAR

Modèles d’architecture d’état

→ Architecture centralisée : Store et immutabilité

- **Store centralisé**
  - Une **source de vérité unique** pour l’application
  - Stocke les états critiques : utilisateur, panier, préférences, données métiers
- **Immutabilité**
  - L’état **n’est jamais modifié directement**
  - On crée un **nouvel état** à chaque mise à jour

  ☑ **Avantages :** historique possible, meilleure optimisation des re-renders, logique plus prévisible.
- **Pourquoi c’est important ?**
  - Permet de comprendre et suivre l’évolution de l’état même dans de gros projet

UQAR

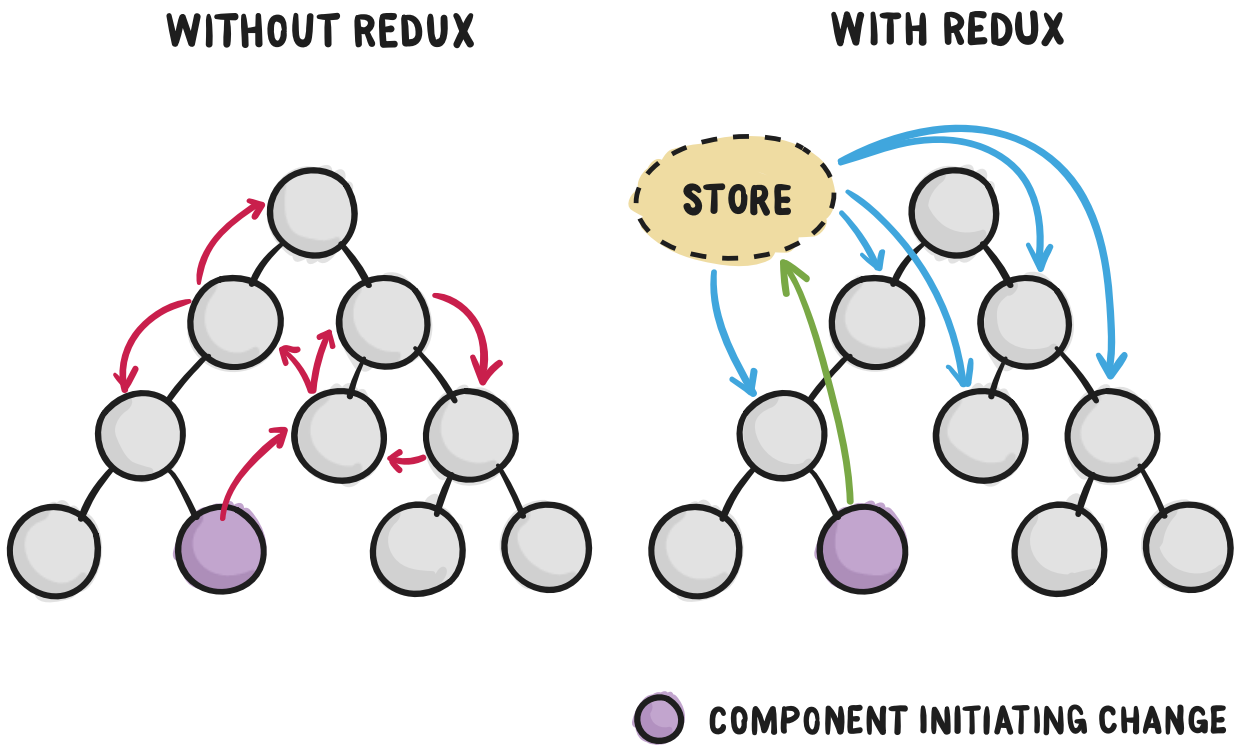
Modèles d’architecture d’état

→ Comparaison des approches de gestion d’état

Approche	Points forts	Limites	Usage recommandé
useState	Simple, local, rapide	Pas de partage natif	État isolé
Context API	Simple global	Mauvaises performances	Petits états globaux
Redux Toolkit	Structuré, scalable, outils	Plus de code	Projets moyens/grands
Zustand	Minimal, flexible, très simple	Moins structurant	Prototypes, apps moyennes

Modèles d’architecture d’état

→ Avec Redux vs. Sans Redux



UQAR

Modèles d’architecture d’état

→ Choisir la bonne architecture selon le projet

- **UseState**
  - Application simple
  - État local isolé
  - Composants indépendants
- **Context**
  - Besoin d’un petit global state
  - Thème, langue, utilisateur connecté
  - Éviter pour des données très dynamiques
- **Redux Toolkit**
  - Projet structuré, équipe multiple
  - États complexes et nombreux
  - Besoin de DevTools, middlewares, …
- **Zustand**
  - Projet de petite à moyenne taille
  - Besoin de rapidité de développement
  - Store minimaliste, logique claire

UQAR

Mise en pratique avec Redux Toolkit

→ Concepts clés de Redux Toolkit

1. **Store**
   - Contient l'état global de l'application
   - Une *source de vérité* unique
2. **Slice**
   - Une portion du state + les fonctions pour le modifier
   - Contient : *initialState*, *reducers*, *actions*
3. **Reducer**
   - Fonction pure qui génère le **nouvel état**
   - Reçoit : (state, action)
   - Retourne : nouveau state

- **Pourquoi RTK ?**
  - ☑ Simplifie drastiquement la configuration Redux
  - ☑ Mécanismes intégrés : immutabilité, *devtools*, actions automatiques

UQAR

Mise en pratique avec Redux Toolkit

→ createSlice() expliqué

- **Génère automatiquement :**
  - un *reducer*
  - des *actions* (*increment*, *decrement*, …)
- Permet de modifier l'état **de manière immuable** mais avec une syntaxe mutable.
- **Résultats :**
  - *slice.reducer* → utilisé dans le store
  - *slice.actions* → utilisé les composants

```
const slice = createSlice({
  name: "counter",
  initialState: { value: 0 },
  reducers: {
    increment: (state) ⇒ { state.value++ },
    decrement: (state) ⇒ { state.value-- },
  },
});
```

UQAR

Mise en pratique avec Redux Toolkit

→ configureStore() expliqué

- **Rôle**
  - Crée le *store* global
  - Combine automatiquement plusieurs slices
  - **Active** : Redux DevTools, Middleware par défaut, Vérifications d'immutabilité & sérialisation

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

UQAR

Mise en pratique avec Redux Toolkit

→ Hooks Redux : *useSelector* & *useDispatch*

- **useSelector()**

  → Sert à *lire* une partie de l'état global

```
1  const count = useSelector((state) ⇒ state.counter.value);
```

- **useDispatch()**

  → Sert à *envoyer* une action

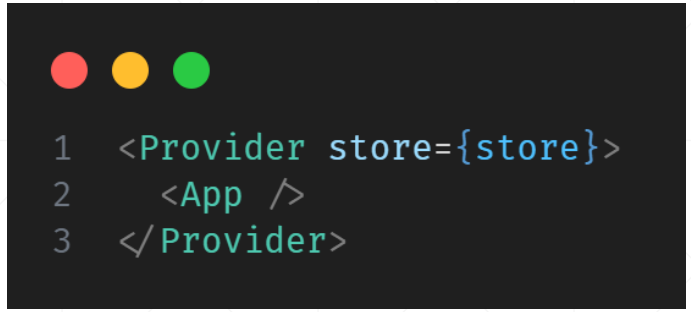
```
1  const dispatch = useDispatch();
2  dispatch(increment());
```

- Avantages des *hooks*
  - Parfaitement adaptés aux composants fonctionnels
  - Pas besoin du HOC *connect()* (ancienne API)

Mise en pratique avec Redux Toolkit

→ Structure d'un projet Redux Toolkit

```
src/
├─ store/
│   ├─ counterSlice.jsx
│   └─ index.jsx (store)
├─ components/
│   └─ Counter.jsx
├─ App.jsx
└─ main.jsx (Provider)
```



UQAR

Mise en pratique avec Redux Toolkit

→ Exemple complet : compteur increment/decrement

```
1  const counterSlice = createSlice({
2    name: "counter",
3    initialState: { value: 0 },
4    reducers: {
5      increment: (state) ⇒ { state.value++ },
6      decrement: (state) ⇒ { state.value-- },
7    },
8  });
```

Slice

UQAR

Mise en pratique avec Redux Toolkit

→ Exemple complet : compteur increment/decrement

```
1  function Counter() {
2    const count = useSelector((s) ⇒ s.counter.value);
3    const dispatch = useDispatch();
4
5    return (
6      <>
7        <p>Valeur : {count}</p>
8        <button onClick={() ⇒ dispatch(increment())}>+</button>
9        <button onClick={() ⇒ dispatch(decrement())}>–</button>
10     </>
11   );
12 }
```

Composant

UQAR

Mise en pratique avec Redux Toolkit

→ Avantages & limites de Redux Toolkit

- **Avantages**
  - ☑ Architecture claire et scalable
  - ☑ Debug puissant (Redux DevTools)
  - ☑ Flux prévisible → facile à maintenir
  - ☑ Boilerplate réduit grâce à *createSlice*
- **Limites**
  - ☒ Plus de structure = plus de code qu'un simple *hook*
  - ☒ Peut être excessif pour de petits projets
  - ☒ Requiert une organisation stricte (par domain/slice)

Redux Toolkit est l'outil idéal pour des **applications moyennes à grandes**, collaboratives ou long-terme.

UQAR

Zustand — gestion d’état simplifiée

→ Philosophie de Zustand

- **Zustand** en quelques mots
  - Bibliothèque de state management **minimaliste, moderne et basée sur les hooks**.
  - Pas de reducers, pas d’actions séparées, pas de boilerplate.
  - Très proche de l’approche « vanilla JavaScript → React ».
- **Principes clés**
  - Un store = une **fonction** *create()*
  - Mise à jour simple via *set()*
  - Lecture via des **sélecteurs** sur le *hook*
  - Pas de Provider global (contrairement à Redux)
- ***Objectif :*** Fournir un **store simple et flexible**, tout en restant performant.

UQAR

Zustand — gestion d'état simplifiée

→ Création d'un store minimaliste

```
1  import { create } from "zustand";
2
3  export const useStore = create((set) ⇒ ({
4    value: 0,
5    increment: () ⇒ set((s) ⇒ ({ value: s.value + 1 })),
6    decrement: () ⇒ set((s) ⇒ ({ value: s.value - 1 })),
7  }));
```

- Le state et les actions sont définis **au même endroit**.
- Les updates peuvent être **synchrone ou async** sans contrainte.
- *set()* = mutation contrôlée de l'état.

UQAR

Zustand — gestion d’état simplifiée

→ Exemple d'utilisation dans un composant

```jsx
import { useStore } from "./store";

function Counter() {
  const { value, increment, decrement } = useStore();

  return (
    <>
      <p>Valeur : {value}</p>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>–</button>
    </>
  );
}
```

UQAR

Zustand vs Redux Toolkit

Zustand

- Minimal, flexible
- Peu de règles → liberté totale
- Idéal pour projets petits/moyens
- Setup ultra rapide
- Moins structurant → dépend beaucoup de la discipline du développeur

Redux Toolkit

- Architecture claire : actions → reducers → store
- Idéal pour grandes équipes et gros projets
- DevTools + middlewares + prédictibilité
- Boîte à outils complète et structurante

UQAR

Questions & Discussion

UQAR

Bibliographie

1. Templier, Thierry & Gougeon, Arnaud (2007). JavaScript pour le Web 2.0 Programmation objet, DOM, Ajax, Prototype, Dojo, Script.aculo.us, Rialto. Éditions Eyrolles.
2. Porteneuve, Christophe (2008). Bien développer pour le Web 2.0 : Bonnes pratiques Ajax. Éditions Eyrolles.
3. Engels, Jean (2012). HTML5 et CSS3 : Cours et exercices corrigés. Éditions Eyrolles.
4. Martin, Michel (2014). HTML5, CSS3 & jQuery : Créez votre premier site web. Éditions Pearson.