S1 :

1️⃣ Sites Web statiques vs dynamiques

React (by Facebook) : Très rapide grâce au Virtual DOM,
idéal pour des applications interactives.

Pourquoi utiliser un framework ?
✔Code plus structuré
✔Réutilisation des composants
✔Performances améliorées

map()

```
const nombres = [1, 2, 3, 4];
const doubles = nombres.map(num => num * 2);
console.log(doubles); // [2, 4, 6, 8]
```

filter()

```
const ages = [12, 18, 21, 17, 25];
const majeurs = ages.filter(age => age >= 18);
console.log(majeurs); // [18, 21, 25]
```

reduce()

```
const nombres = [1, 2, 3, 4];
const somme = nombres.reduce((total, num) =>
 total + num, 0);
console.log(somme); // 10
```

5️⃣ Programmation fonctionnelle

Fonctions pures = Elles ne modifient pas les variables extérieures.
Fermetures = Une fonction qui se souvient des variables de son environnement.
Recursion = Une fonction qui s'appelle elle-même.

S2 :

1️⃣ Différence entre SPA et MPA

MPA (Multi-Page Application) : Chaque clic recharge
toute la page.
SPA (Single Page Application) : Charge une seule fois puis met à jour dynamiquement.

Pourquoi choisir une SPA ?
✔Plus rapide
✔Expérience fluide ✓
✔Moins de requêtes au serveur

Problèmes des SPA ?
✖Mauvais référencement SEO
✖Historique du navigateur non géré correctement
✖Sécurité (risque XSS)

S3 :

1️⃣ Qu'est-ce que React ?

React est une bibliothèque JavaScript qui permet de créer des interfaces dynamiques avec des composants réutilisables

Pourquoi React ?
✔Mise à jour rapide du DOM
✔Meilleure organisation du code
✔Très performant grâce au Virtual DOM

2️⃣ Écosystème React

npm : Gestion des dépendances
ES6 : Syntaxe moderne de JavaScript
Babel : Convertit du code ES6 en code compréhensible par tous les navigateurs
Webpack : Gère les fichiers du projet

Un bundler (ou module bundler) est un outil qui prend tous les fichiers JavaScript, CSS, images et autres ressources d'un projet React et les regroupe en un ou plusieurs fichiers optimisés pour le navigateur.

Les bundlers les plus utilisés :
1️⃣ Webpack (le plus populaire)
2️⃣ Parcel (plus rapide, moins de configuration)
3️⃣ Vite (optimisé pour React et Vue, ultra rapide)

Webpack prend plusieurs fichiers et les regroupe en un fichier unique :

Comment React utilise un bundler ?

```
npx create-react-app mon-projet
cd mon-projet
npm start
```

3️⃣ Structure d'un projet React

public/ →Contient index.html
src/ →Contient les fichiers React (App.js, index.js, components/)
package.json →Liste des dépendances et scripts npm

npm install : installe le projet sur votre machine.
npm start : démarre le projet.
npm test : lance les tests du projet.
npm run dev : s'occupe de lancer un environnement de développement agréable.

1. Introduction à Redux

Redux est une bibliothèque JavaScript créée en 2015 par Dan Abramov, inspirée par l'architecture Flux de Facebook et le langage ELM.
Objectif : gérer l'état des applications de manière prévisible grâce à un flux de données unidirectionnel.

Avantages

Centralisation des états.
Débogage simplifié grâce à des outils comme Redux DevTools.
Gestion des modifications d'état prévisible.

2.Architecture Redux

Action : Objet décrivant ce qui s'est passé (type, données).
Store : Contient l'état global de l'application. C'est la source unique de vérité.
Reducer : Fonction pure qui reçoit l'état actuel et une action, et retourne un nouvel état.
Middleware : Intercepte les actions avant qu'elles atteignent le reducer, permettant des tâches comme la journalisation ou les appels asynchrones.

Méthodes principales :

dispatch : Envoie une action au store.
getState : Récupère l'état actuel.
subscribe : Permet d'écouter les changements dans le store.

S13 :

Définition du routage : Permet de naviguer entre différentes pages d'une application en fonction de l'URL

1️⃣ Installation

```
npm install react-router-dom
```

2️⃣ Créer des routes de base

```
import { BrowserRouter, Routes, Route }
from 'react-router-dom';
import Home from './Home';
import About from './About';

function App() {
 return (
  <BrowserRouter>
   <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
   </Routes>
  </BrowserRouter>
 );
}

export default App;
```

3️⃣ Navigation avec <Link>

```
import { Link } from 'react-router-dom';

<Link to="/">Accueil</Link>
<Link to="/about">À propos</Link>
```

4️⃣ Navigation dynamique avec useNavigate()

```
import { useNavigate } from 'react-router-dom';

const navigate = useNavigate();
<button onClick={() => navigate('/about')}>Aller à la
page À propos</button>
```

5️⃣ Récupérer des paramètres d'URL avec useParams()

```
import { useParams } from 'react-router-dom';

function Profile() {
  const { id } = useParams();
  return <h2>Profil de l'utilisateur {id}</h2>;
}
```

Définition de la route dynamique :

```
<Route path="/profile/:id" element={<Profile />} />
```

6️⃣ Créer une structure avec <Outlet>

```
import { Link, Outlet } from 'react-router-dom';

function Layout() {
 return (
  <div>
   <nav>
    <Link to="/">Accueil</Link>
    <Link to="/about">À propos</Link>
   </nav>
   <Outlet />
  </div>
 );
}
```

Utilisation dans App.js :

```
<Route path="/" element={<Layout />}>
 <Route index element={<Home />} />
 <Route path="about" element={<About />} />
</Route>
```

7️⃣ Gérer une page 404 (Route Inconnue)

```
<Route path="*" element={<h2>❌Page non </h2>} />
```

S14 :

Le test permet de vérifier si une application React fonctionne correctement avant de la mettre en production.
On utilise Jest et Enzyme pour tester les comp osants React.

1️⃣ Installer

```
npm install --save-dev enzyme enzyme-adapter-react-16
```

✓Jest : Outil de test intégré à React.
✓Enzyme : Bibliothèque pour tester facilement les composants React.

Utilisez la commande suivante pour exécuter les scénarios de test
```
npm test
```

Concept	Description
shallow(<App />)	: Charge le composant sans ses enfants.
.find('#ClickMe')	: Sélectionne un élément dans le DOM virtuel.
.simulate('click')	: Simule un événement (clic, saisie...).
.text()	: Récupère le texte d'un élément.
.prop('disabled')	: Vérifie une propriété d'un élément (ex: disabled).
expect(...).toEqual(...)	:Vérifie si le résultat est correct.

Immer :

Librairie utilisée pour gérer les états immuables.
Exemple : Modifier un tableau ou un objet sans changer sa référence.

Événements asynchrones

Redux ne gère pas directement les événements asynchrones, mais des middlewares comme redux-thunk permettent de les intégrer.

Redux DevTools :

Outil de débogage pour explorer l'état et les actions en temps réel.
Permet de visualiser les états sous forme de Tree, Chart ou Raw.
Fonctionnalités : _Revenir en arrière dans les actions.
_Lancer des actions manuellement.
Bloquer ou persister les états.

S16 - Redux :

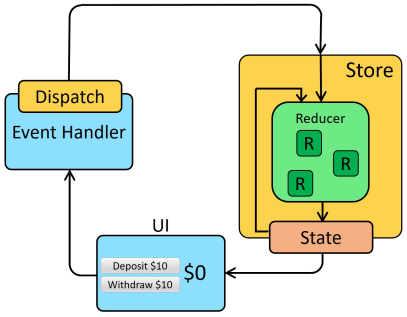
Redux est une bibliothèque qui permet de gérer l'état global d'une application React. Il est particulièrement utile lorsque plusieurs composants doivent partager les mêmes données.

Pourquoi Redux est utile ?

Stockage centralisé : Toutes les données sont stockées dans un seul endroit appelé store.
Facilité de gestion : Les composants n'ont pas besoin de gérer leur état indépendamment.
Debugging simplifié : Grâce aux outils comme Redux DevTools, on peut facilement voir les changements d'état.
Type your text

1. Les concepts de Redux

Concept	Explication
Store	Stocke l'état global de l'application
Actions	Objet décrivant une intention de modification d'état ({ type: "INCREMENT" })
Reducers	Fonctions qui modifient l'état en fonction des actions reçues
Dispatch	Fonction utilisée pour envoyer une action et modifier l'état



Les 2 Composants de Flux
VIEW (Vue) : Interface utilisateur avec des composants React.
ACTION (Action) : Objet représentant un événement utilisateur (ex: clic sur un bouton).
DISPATCHER (Distributeur) : Centralise les actions et les envoie aux Stores.
STORE (Magasin de données) : Gère l'état de l'application et met à jour la Vue en fonction des actions reçues.

Explication du code :
✔useSelector(state => state.count): Accède à l'état global pour afficher la valeur du compteur.
✔useDispatch(): Permet d'envoyer une action pour modifier l'état.
✔dispatch({ type: "INCREMENT" }): Augmente la valeur du compteur.
✔dispatch({ type: "DECREMENT" }): Diminue la valeur du compteur.

3 .Redux dans une application :

1 .Installation

```
npm install redux react-redux
```

store.js

```
import { createStore } from "redux";

function compteurReducer(state = { count: 0 },
action) {
  switch (action.type) {
     case "INCREMENT":
        return { count: state.count + 1 };
     case "DECREMENT":
        return { count: state.count - 1 };
     default:
        return state;
  }
}

// Création du store Redux
const store = createStore(compteurReducer);
export default store;
```

index.js

```
import React from "react";
import ReactDOM from "react-dom";
import { Provider } from "react-redux";
import store from "./store";
import App from "./App";

ReactDOM.render(
   <Provider store={store}>
      <App />
   </Provider>,
   document.getElementById("root")
);
```

Compteur.js

```
import React from "react";
import { useSelector, useDispatch } from "react-redux"

export default function Compteur() {
   // Accéder à l'état global
   const count = useSelector(state => state.count);
   // Fonction pour envoyer des actions
   const dispatch = useDispatch();

   return (
      <div>
         <h1>Compteur : {count}</h1>
         <button onClick={() =>
dispatch({ type: "INCREMENT" })}>+</button>
         <button onClick={() =>
dispatch({ type: "DECREMENT" })}>-</button>
      </div>
   );
}
```

Redux Toolkit : Optimisation de la Gestion du State en React

2. Installation de Redux Toolkit

```
npx create-react-app mon-projet-redux
cd mon-projet-redux
npm install @reduxjs/toolkit react-redux
```

Explication :

@reduxjs/toolkit est la bibliothèque Redux améliorée.
react-redux est une bibliothèque qui permet d'intégrer Redux dans une application React.

3. Structure d'un projet Redux Toolkit

```
/mon-projet-redux
├──/src
│  ├──/features       # Dossier contenant les "slices" Redux
│  │  ├──counterSlice.js  # Un slice pour gérer un compteur
│  │  ├──usersSlice.js    # Un slice pour gérer les utilisateurs
│  ├──store.js        # Configuration du store Redux
│  ├──App.js          # Composant principal React
│  ├──index.js        # Point d'entrée de l'application
```

Explication :

features/ contient les slices, qui sont des morceaux d'état gérés indépendamment.
store.js configure Redux.
App.js est le composant principal qui affiche les données.Type your text

4. Création d'un Slice avec createSlice

Un Slice est une portion du state global. Avec createSlice,on peut définir l'état initial, les actions et les reducers en une seule étape.

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = { value: 0 };

const counterSlice = createSlice({
  name: "counter",
  initialState,
  reducers: {
    increment: (state) => {
      state.value++;  // Grâce à Immer, on peut modifier
    },                          l'état directement
    decrement: (state) => {
      state.value--;
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload;
    },
  },
});

export const { increment, decrement, incrementByAmount } =
counterSlice.actions;
 export default counterSlice.reducer;
```

5. Configuration du Store (store.js)

```
import { configureStore } from "@reduxjs/toolkit";
import counterReducer from "./features/counterSlice";

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

6. Utilisation du State Redux dans un Composant React (App.js)

```
import React from "react";
import { useSelector, useDispatch } from "react-redux";
import { increment, decrement, incrementByAmount } from
 "./features/counterSlice";
function App() {
  const count = useSelector((state) => state.counter.value);
                                          // Lire l'état
const dispatch = useDispatch();  // Permet d'envoyer des actions
    return (
      <div>
        <h1>Compteur : {count}</h1>
        <button onClick={() => dispatch(increment())}>+1</button>
        <button onClick={() => dispatch(decrement())}>-1</button>
          <button onClick={() => dispatch(incrementByAmount(5))}>
   +5</button>
      </div>
    );
  }
export default App;
```

7. Gestion des Actions Asynchrones avec createAsyncThunk

Dans une vraie application, on a souvent besoin de récupérer des données d'une API.
Avec createAsyncThunk, on peut gérer des requêtes API simplement.

1️⃣ Définition du Slice (usersSlice.js)

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import axios from "axios";

// Action asynchrone pour récupérer les utilisateurs
export const fetchUsers = createAsyncThunk("users/fetchUsers",
async () => {
   const response = await axios.get("https://jsonplaceholder.
 typicode.com/users");
 return response.data;
});
const usersSlice = createSlice({
  name: "users",
  initialState: { loading: false, users: [], error: "" },
  reducers: {},  // Pas d'actions synchrones ici
  extraReducers: (builder) => {
    builder.addCase(fetchUsers.pending, (state) => {
      state.loading = true;
    });
    builder.addCase(fetchUsers.fulfilled, (state, action) => {
      state.loading = false;
      state.users = action.payload;
    });
    builder.addCase(fetchUsers.rejected, (state, action) => {
      state.loading = false;
      state.error = action.error.message;
    });
  },
});

export default usersSlice.reducer;
```

2️⃣ Affichage des Utilisateurs dans React (App.js)

```
import { useDispatch, useSelector } from "react-redux";
import { fetchUsers } from "./features/usersSlice";
import { useEffect } from "react";

function App() {
  const dispatch = useDispatch();
  const { users, loading, error } = useSelector((state) => state.users);

  useEffect(() => {
    dispatch(fetchUsers());  // Charger les utilisateurs au démarrage
  }, [dispatch]);

  return (
    <div>
      <h1>Liste des utilisateurs</h1>
      {loading && <p>Chargement...</p>}
      {error && <p style={{ color: "red" }}>{error}</p>}
      <ul>
        {users.map((user) => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```