



Gestione dello Stato
con Redux e Redux Toolkit



REACT AVANZATO

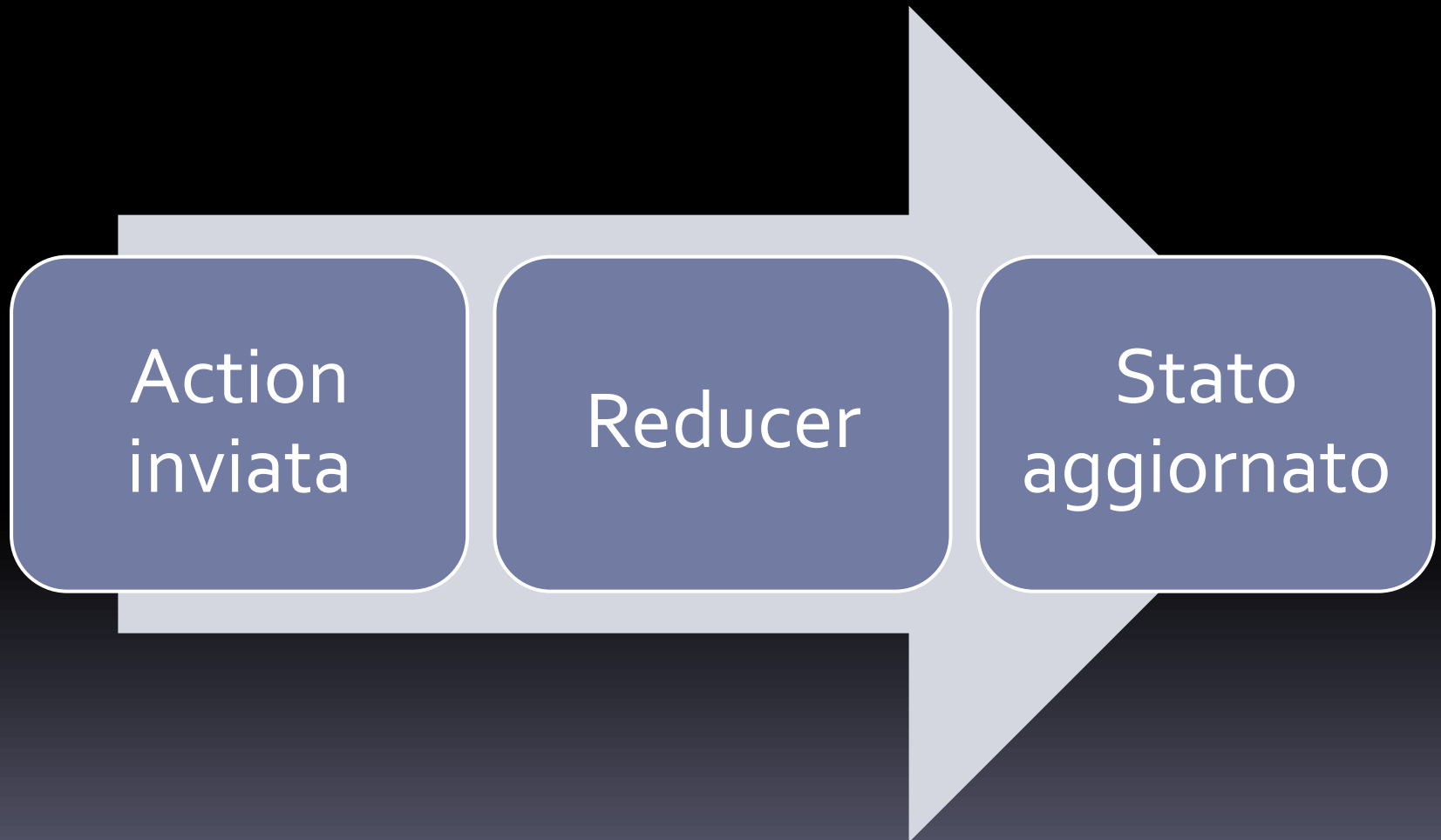
Obiettivi

- Approfondimento di **Redux** e **Redux Toolkit**
- Applicazione di concetti avanzati con **TypeScript**
- Cosa vedremo
 - Concetti fondamentali di **Redux**
 - Configurazione ed implementazione
 - **Middleware** avanzati
 - Utilizzo di **Redux Toolkit**

Redux

- Cos'è **Redux**?
 - Libreria per la gestione dello stato
 - Elementi fondamentali
 - **Store**: contiene l'intero stato
 - **Actions**: descrivono cosa accade
 - **Reducers**: funzioni pure che aggiornano lo stato

Redux



Configurazione

```
npm install redux react-redux @types/react-redux
```

```
import { createStore } from 'redux';  
const store = createStore(reducer);
```

Implementazione di Redux

- Connessione di Redux a React

```
import { Provider } from 'react-redux';  
<Provider store={store}>  
  <App />  
</Provider>
```

- Utilizzo degli hook:

- `useSelector` per leggere lo stato
- `useDispatch` per inviare le azioni

Esempio Pratico

- Azioni:

```
const addTask = (task) => ({ type: 'ADD_TASK', payload: task });
```

- Reducer:

```
const taskReducer = (state = [], action) => {  
  switch (action.type) {  
    case 'ADD_TASK':  
      return [...state, action.payload];  
    default:  
      return state;  
  }  
};
```

Middleware

- Cos'è un **Middleware**?
 - Intercetta le azioni tra dispatch e reducer
 - Esegue operazioni asincrone o aggiuntive
- **Redux Thunk**

```
const fetchTasks = createAsyncThunk('tasks/fetch', async () => {  
  const response = await fetch('/tasks');  
  return response.json();  
});
```


Redux Thunk

```
npm install redux-thunk @types/redux-thunk
```

```
import { configureStore } from '@reduxjs/toolkit';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const store = configureStore({
  reducer: rootReducer,
  middleware: (getDefaultMiddleware) => getDefaultMiddleware().concat(thunk),
});

export default store;
```

Middleware Avanzati

```
const logger: Middleware = (store) => (next) => (action) => {  
  console.log('Azione:', action);  
  return next(action);  
};
```

```
import { Middleware } from 'redux';  
import { RootState } from './store';  
  
// Middleware per bloccare azioni riservate agli admin  
const permissionMiddleware: Middleware<{}, RootState> = (store) => (next)  
  const state = store.getState();  
  const userRole = state.user.role; // Assume che il ruolo sia nello stato  
  
  // Blocca l'azione se l'utente non è autorizzato  
  if (action.type === 'ADMIN_ACTION' && userRole !== 'admin') {  
    console.warn('Azione bloccata: Permessi insufficienti per eseguire que  
    return; // Non inoltra l'azione al reducer  
  }  
  
  return next(action); // L'azione viene inoltrata se permessa  
};
```



Redux Toolkit

- Vantaggi
 - Semplifica la configurazione
 - Azioni e reducer in un unico contesto
 - Store caratterizzato da tanti “pezzi”
 - Ognuno dei quali rappresenta una parte dello stato dell'applicazione



Slices

- Lo stato dell'applicazione è diviso in **slices**
 - Rappresentano ognuna un contesto di stato a sé stante
 - Contengono:
 - Lo stato iniziale
 - Le azioni
 - I reducers configurati per il singolo slice

Configurazione di Slice

- Definizione dello stato iniziale

```
interface Task {  
  id: number;  
  description: string;  
  completed: boolean;  
}  
  
const initialState: Task[] = [];
```

Configurazione di Slice

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit';

// Stato iniziale
const initialState: Task[] = [];

// Creazione della slice
const taskSlice = createSlice({
  name: 'tasks', // Nome della slice
  initialState, // Stato iniziale
  reducers: {
    // Aggiungi un task
    addTask: (state, action: PayloadAction<Task>) => {
      state.push(action.payload);
    },

    // Rimuovi un task tramite l'ID
    removeTask: (state, action: PayloadAction<number>) => {
      return state.filter(task => task.id !== action.payload);
    },

    // Cambia stato "completato" di un task
    toggleTask: (state, action: PayloadAction<number>) => {
      const task = state.find(task => task.id === action.payload);
      if (task) {
        task.completed = !task.completed;
      }
    },
  },
});

// Esporta azioni e reducer
export const { addTask, removeTask, toggleTask } = taskSlice.actions;
export default taskSlice.reducer;
```

Redux Toolkit - Dispatch

```
const store = configureStore({  
  reducer: {  
    tasks: taskReducer,  
  },  
});
```

```
dispatch(addTask({ id: 1, description: 'Nuova Task' }));
```

Async Thunk

- Cos'è una **AsyncThunk**?
 - È una funzione asincrona che consente di gestire uno strato di logica complessa
 - Ad esempio una chiamata a Web API
 - Viene creata in Redux Toolkit tramite **createAsyncThunk**

Async Thunk

```
// Thunk per recuperare i task da un'API
export const fetchTasks = createAsyncThunk<Task[]>(
  'tasks/fetchTasks', // Nome della thunk
  async () => {
    const response = await fetch('https://api.example.com/tasks');
    if (!response.ok) {
      throw new Error('Errore nel recupero dei task');
    }
    return response.json();
  }
);
```



Creazione di
Async Thunk



Aggiunta
alla Slice

```
reducers: {},
extraReducers: (builder) => {
  builder
    .addCase(fetchTasks.pending, (state) => {
      state.loading = true;
      state.error = null;
    })
    .addCase(fetchTasks.fulfilled, (state, action: PayloadAction<Task[]>) => {
      state.loading = false;
      state.tasks = action.payload; // Popolamento dello stato con i task
    })
    .addCase(fetchTasks.rejected, (state, action) => {
      state.loading = false;
      state.error = action.error.message || 'Errore sconosciuto';
    });
},
```

Async Thunk

- Richiamo dell'azione asincrona con **dispatch**:

```
const TaskList: React.FC = () => {  
  const dispatch = useAppDispatch();  
  const { tasks, loading, error } = useAppSelector((state) => state.ta  
  
  useEffect(() => {  
    dispatch(fetchTasks()); // Recupero asincrono dei task al caricame  
  }, [dispatch]);  
}
```




Conclusione

- Punti chiave:
 - Redux gestisce uno stato complesso in grandi applicazioni
 - I middleware permettono l'esecuzione di operazioni avanzate
 - Ad esempio operazioni asincrone
 - Redux Toolkit semplifica l'organizzazione dello stato e lo sviluppo



Domande e Discussione



Grazie per l'attenzione!