NgRx SignalStore Workshop

by Marko Stanimirović, Alex Okrushko, and Brandon Roberts

NG Rome

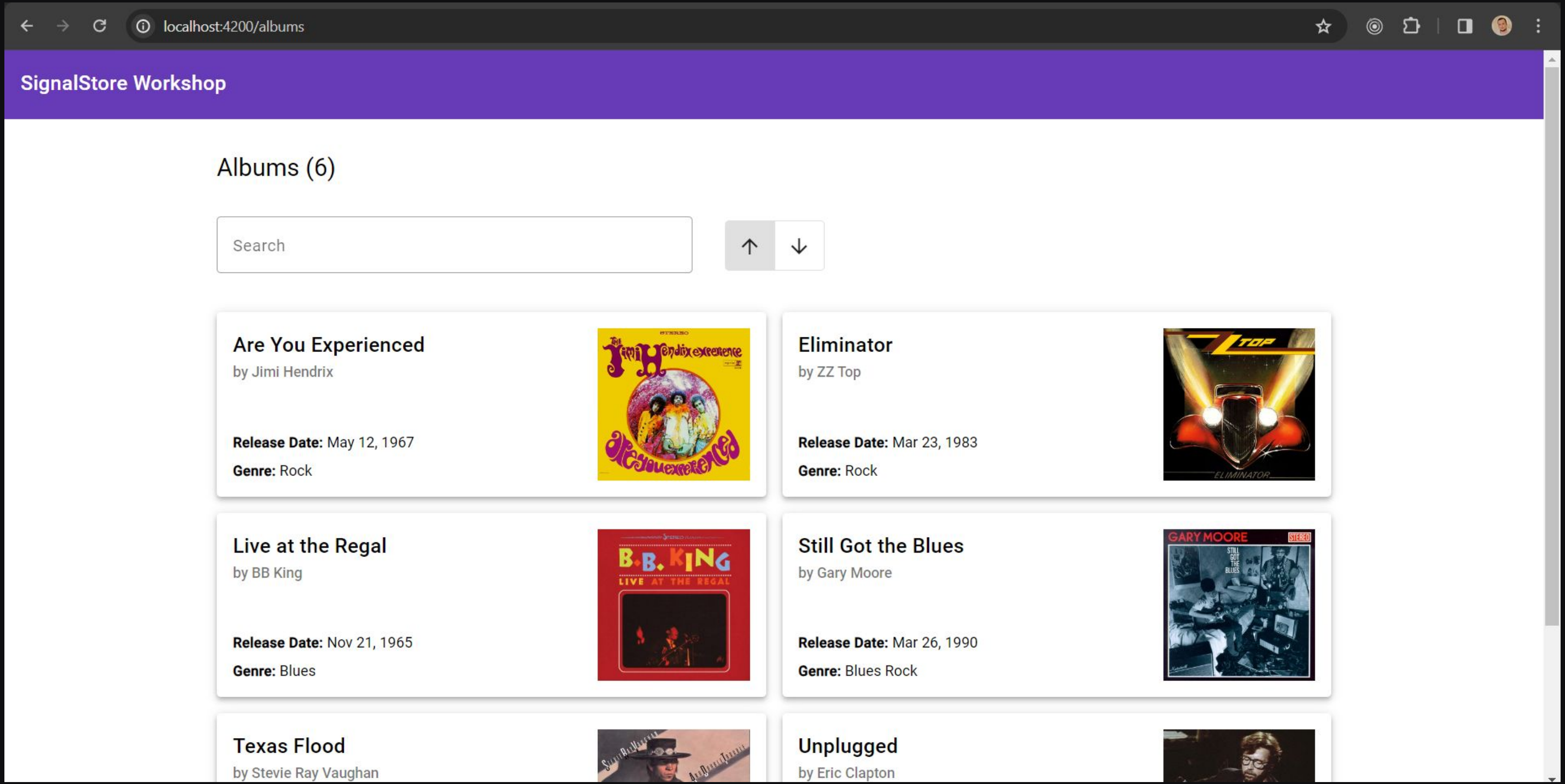
Slides

https://tinyurl.com/ngrx-workshop-ngrome

Setup

1. `git clone https://github.com/ngrx/signal-store-workshop.git`
2. `cd signal-store-workshop`
3. `git fetch --all`
4. `git checkout challenge`
5. `yarn install`

Challenge: Examine the App



localhost:4200/albums

SignalStore Workshop

Albums (6)

Search

Are You Experienced
by Jimi Hendrix
Release Date: May 12, 1967
Genre: Rock

Eliminator
by ZZ Top
Release Date: Mar 23, 1983
Genre: Rock

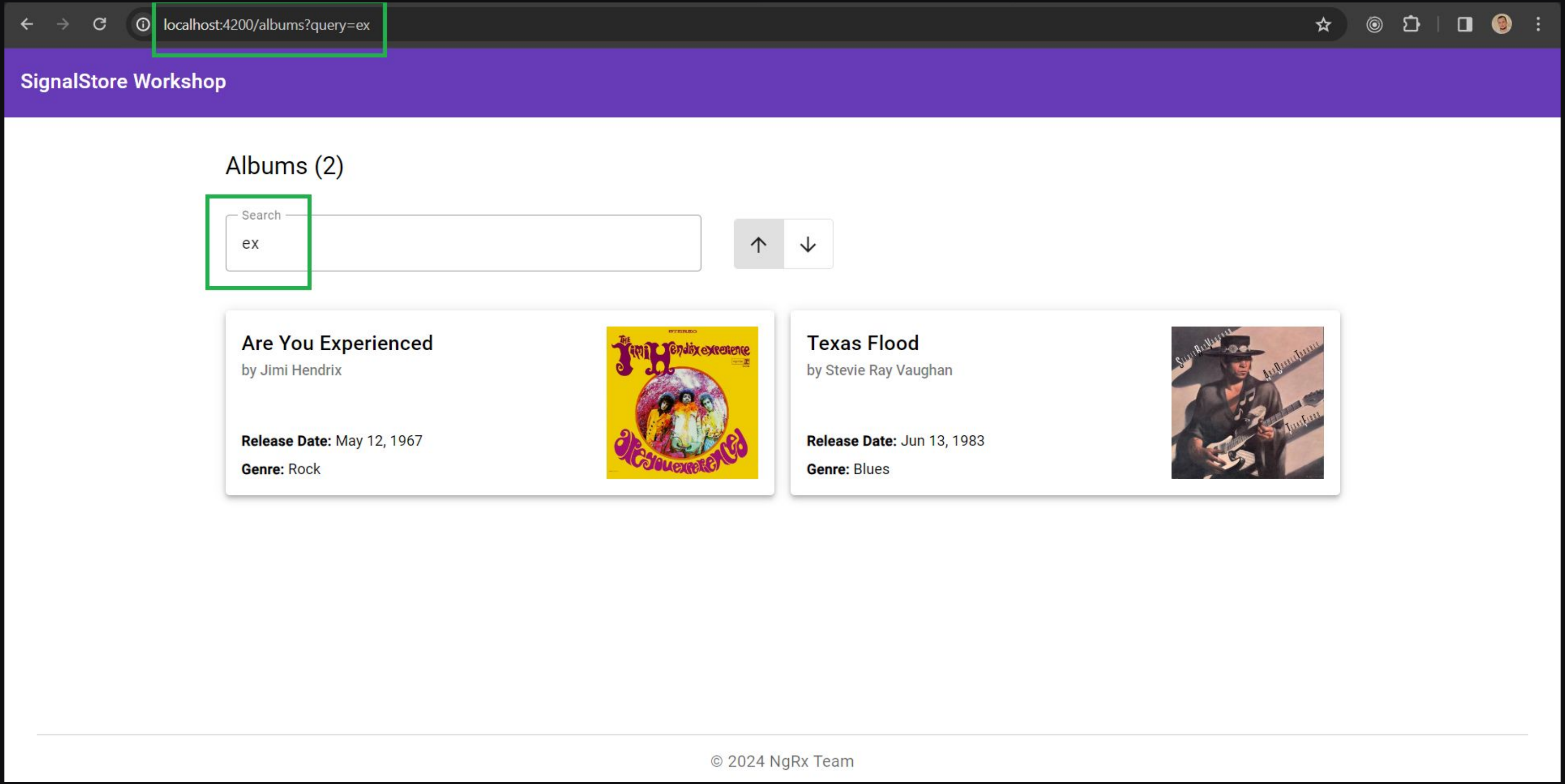
Live at the Regal
by BB King
Release Date: Nov 21, 1965
Genre: Blues

Still Got the Blues
by Gary Moore
Release Date: Mar 26, 1990
Genre: Blues Rock

Texas Flood
by Stevie Ray Vaughan

Unplugged
by Eric Clapton

Challenge: Examine the App



localhost:4200/albums?query=ex

SignalStore Workshop

Albums (2)

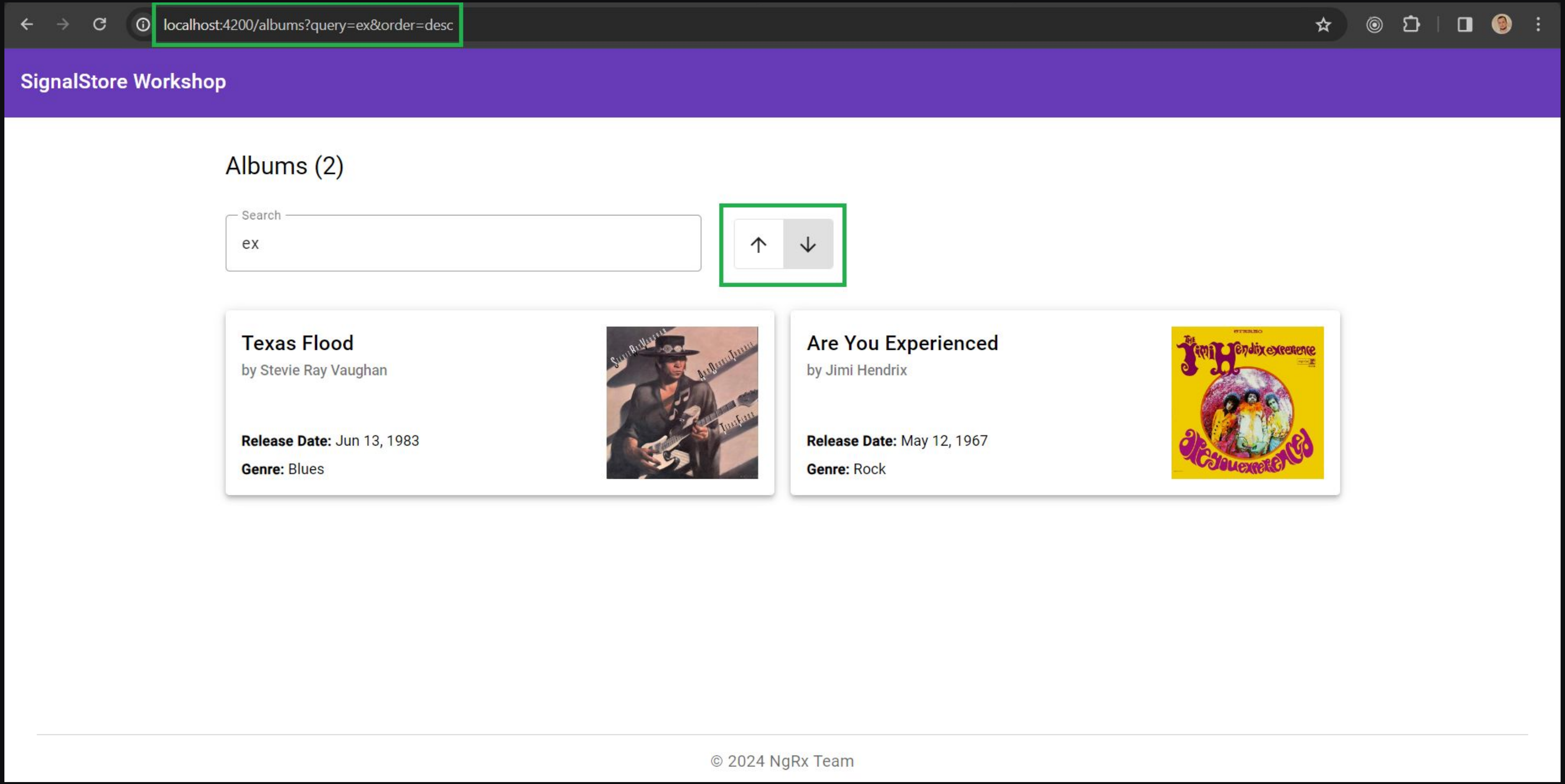
Search
ex

Are You Experienced
by Jimi Hendrix
Release Date: May 12, 1967
Genre: Rock

Texas Flood
by Stevie Ray Vaughan
Release Date: Jun 13, 1983
Genre: Blues

© 2024 NgRx Team

Challenge: Examine the App



localhost:4200/albums?query=ex&order=desc

SignalStore Workshop

Albums (2)

Search

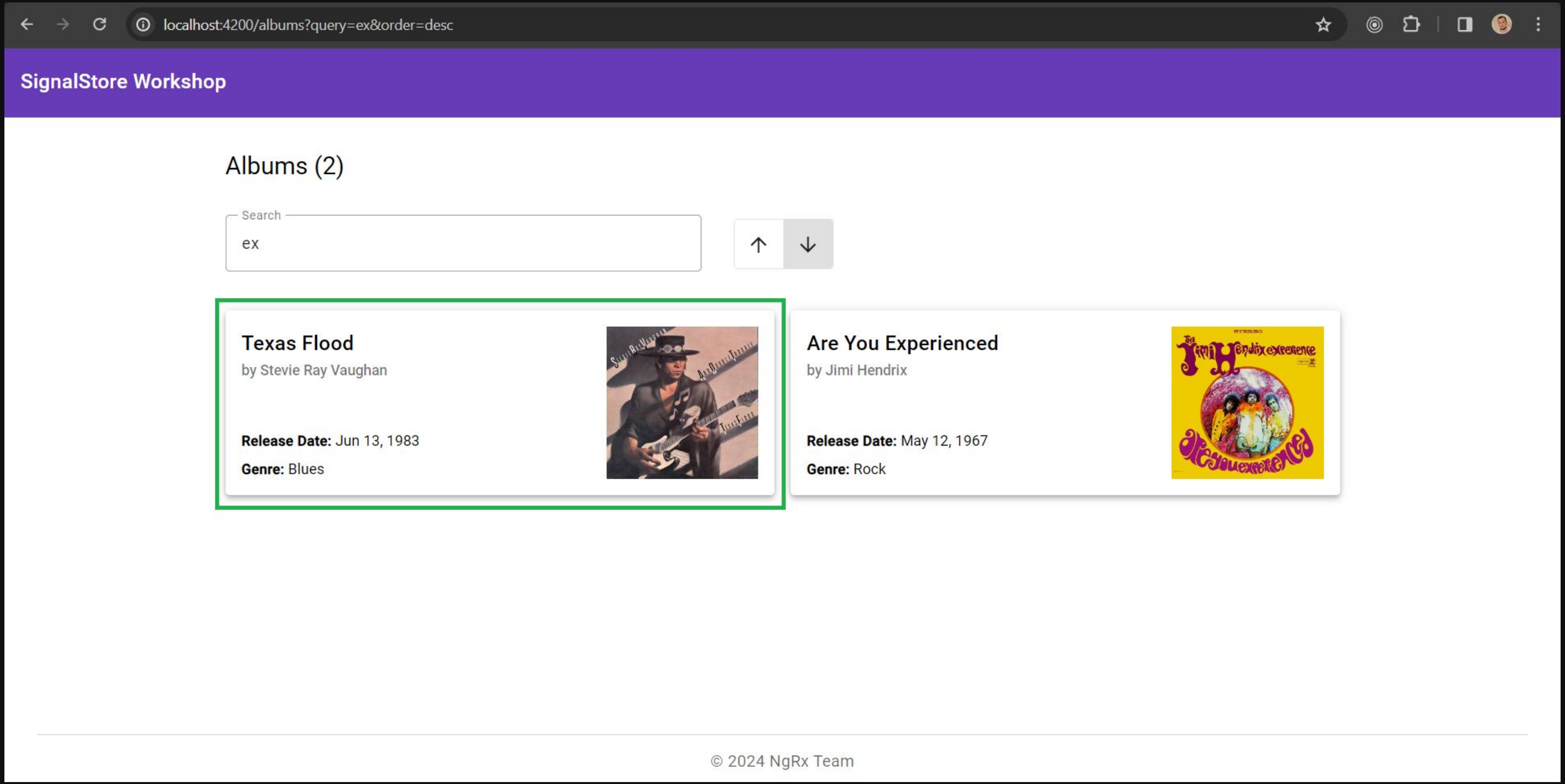
ex

Texas Flood
by Stevie Ray Vaughan
Release Date: Jun 13, 1983
Genre: Blues

Are You Experienced
by Jimi Hendrix
Release Date: May 12, 1967
Genre: Rock

© 2024 NgRx Team

Challenge: Examine the App


localhost:4200/albums?query=ex&order=desc

SignalStore Workshop

Albums (2)

Search

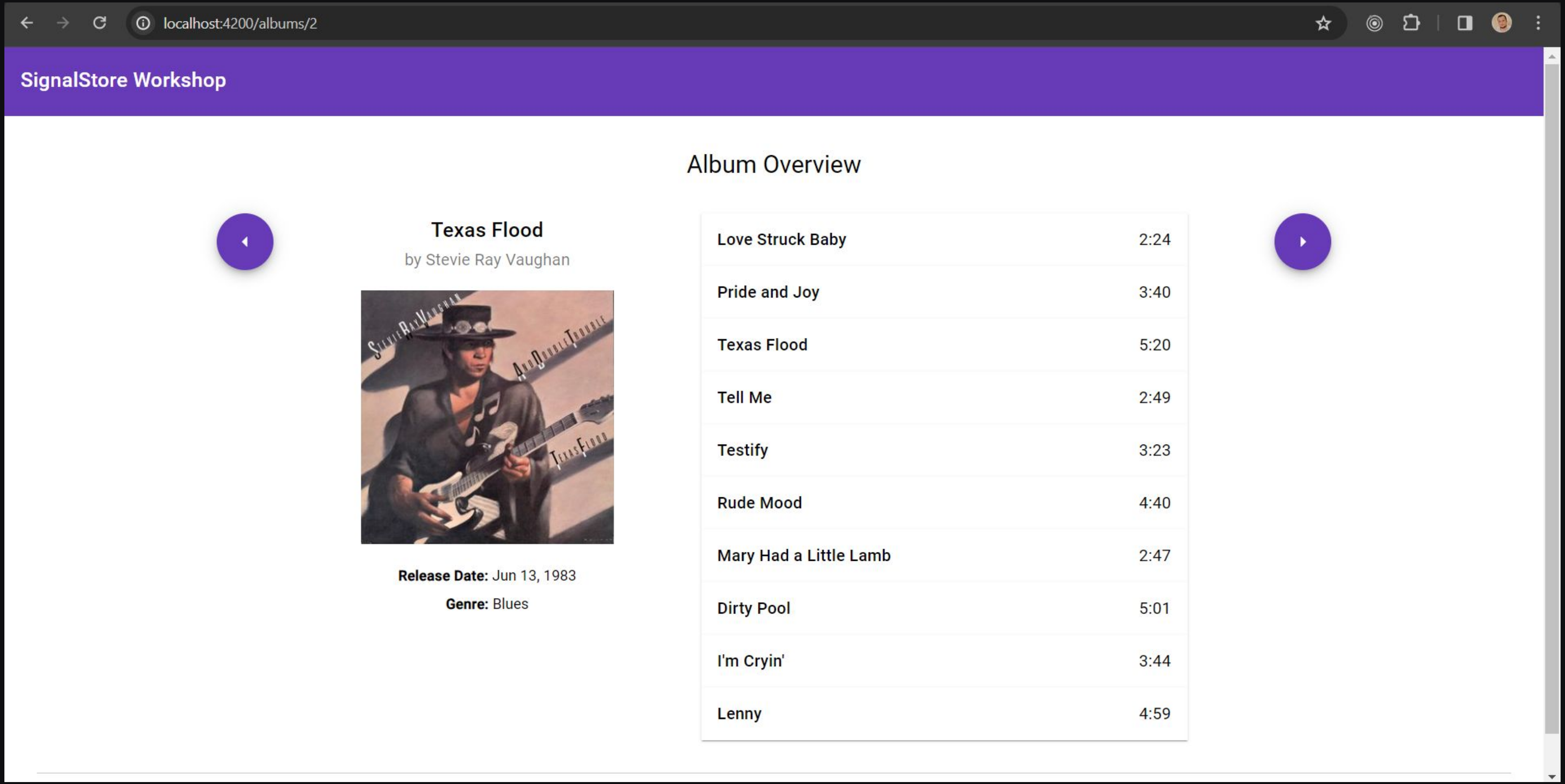
ex

Texas Flood
by Stevie Ray Vaughan
Release Date: Jun 13, 1983
Genre: Blues

Are You Experienced
by Jimi Hendrix
Release Date: May 12, 1967
Genre: Rock

© 2024 NgRx Team

Challenge: Examine the App



localhost:4200/albums/2

SignalStore Workshop

Album Overview

Texas Flood

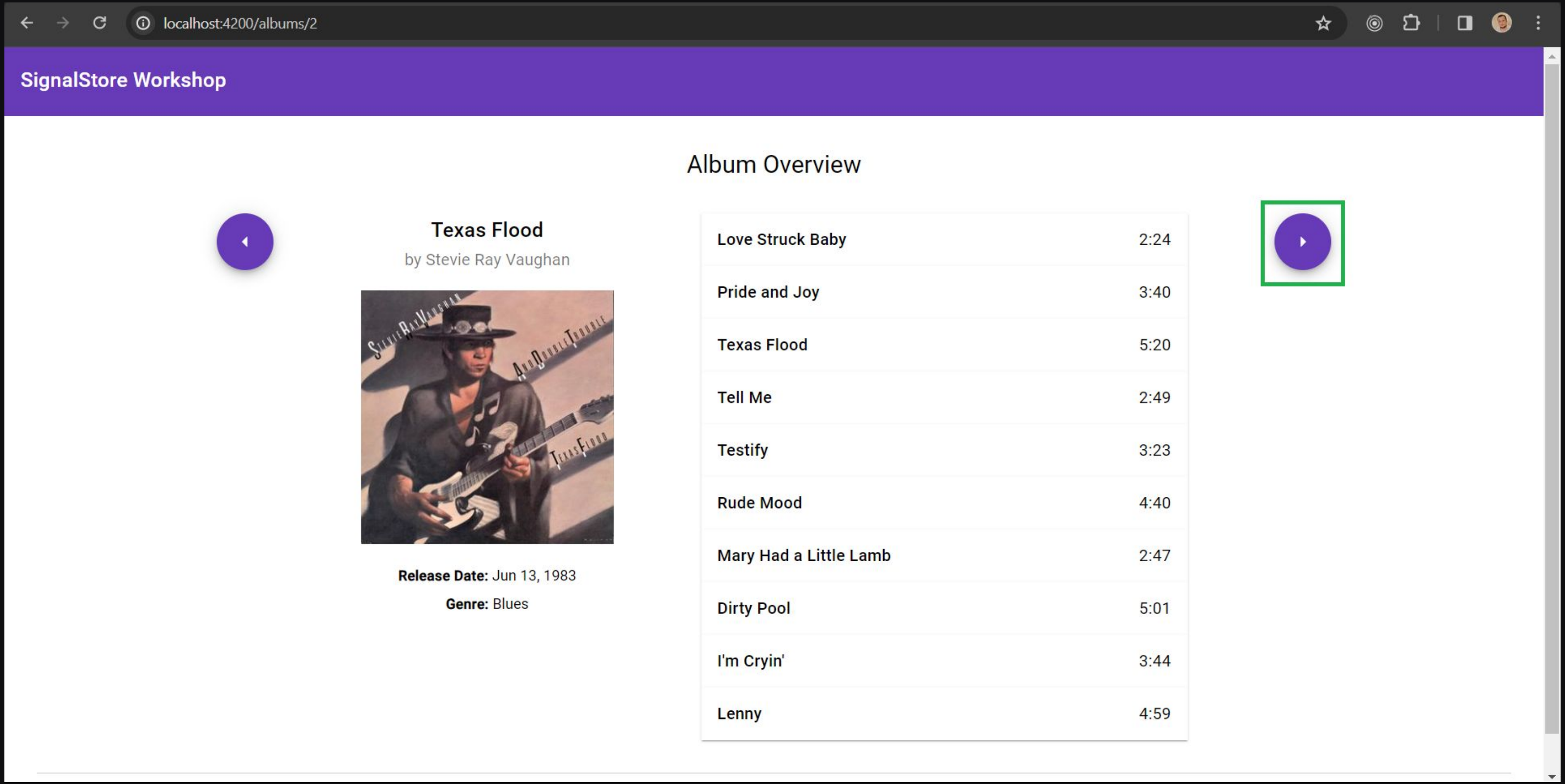
by Stevie Ray Vaughan

Release Date: Jun 13, 1983

Genre: Blues

Love Struck Baby	2:24
Pride and Joy	3:40
Texas Flood	5:20
Tell Me	2:49
Testify	3:23
Rude Mood	4:40
Mary Had a Little Lamb	2:47
Dirty Pool	5:01
I'm Cryin'	3:44
Lenny	4:59

Challenge: Examine the App



localhost:4200/albums/2

SignalStore Workshop

Album Overview

Texas Flood

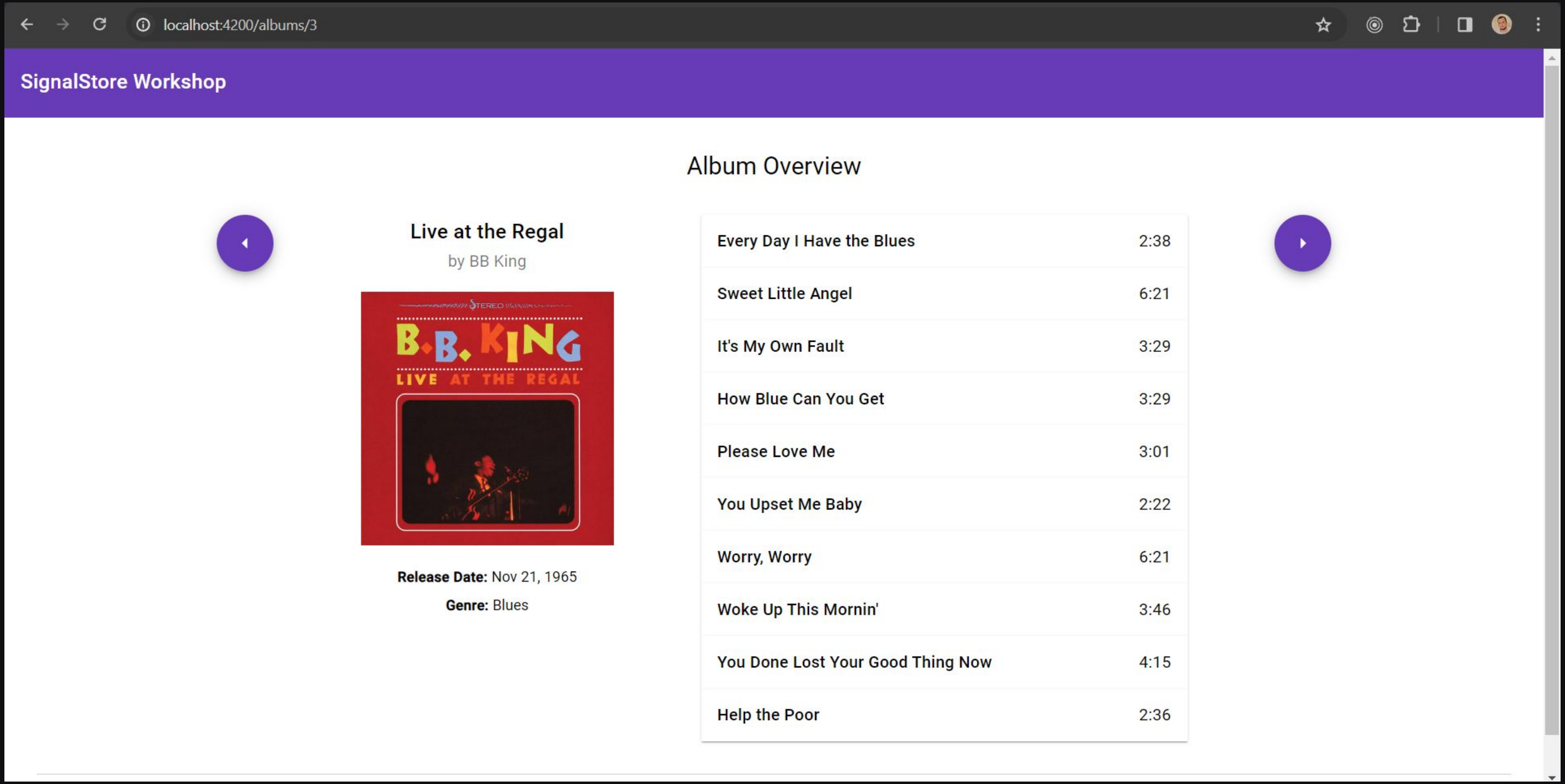
by Stevie Ray Vaughan

Release Date: Jun 13, 1983

Genre: Blues

Love Struck Baby	2:24
Pride and Joy	3:40
Texas Flood	5:20
Tell Me	2:49
Testify	3:23
Rude Mood	4:40
Mary Had a Little Lamb	2:47
Dirty Pool	5:01
I'm Cryin'	3:44
Lenny	4:59

Challenge: Examine the App



localhost:4200/albums/3

SignalStore Workshop

Album Overview

Live at the Regal

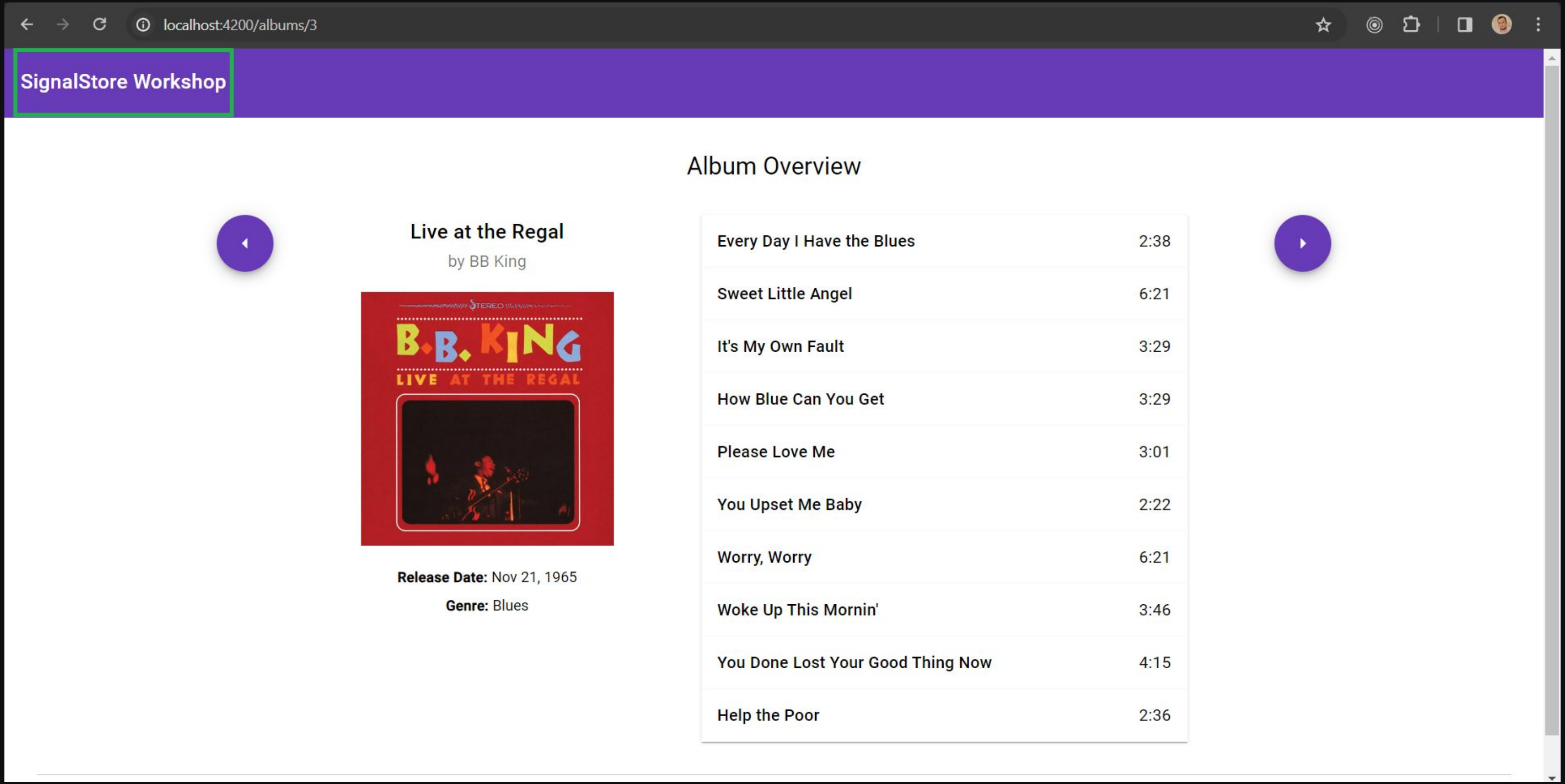
by BB King

Release Date: Nov 21, 1965

Genre: Blues

Every Day I Have the Blues	2:38
Sweet Little Angel	6:21
It's My Own Fault	3:29
How Blue Can You Get	3:29
Please Love Me	3:01
You Upset Me Baby	2:22
Worry, Worry	6:21
Woke Up This Mornin'	3:46
You Done Lost Your Good Thing Now	4:15
Help the Poor	2:36

Challenge: Examine the App



Album Overview

Live at the Regal

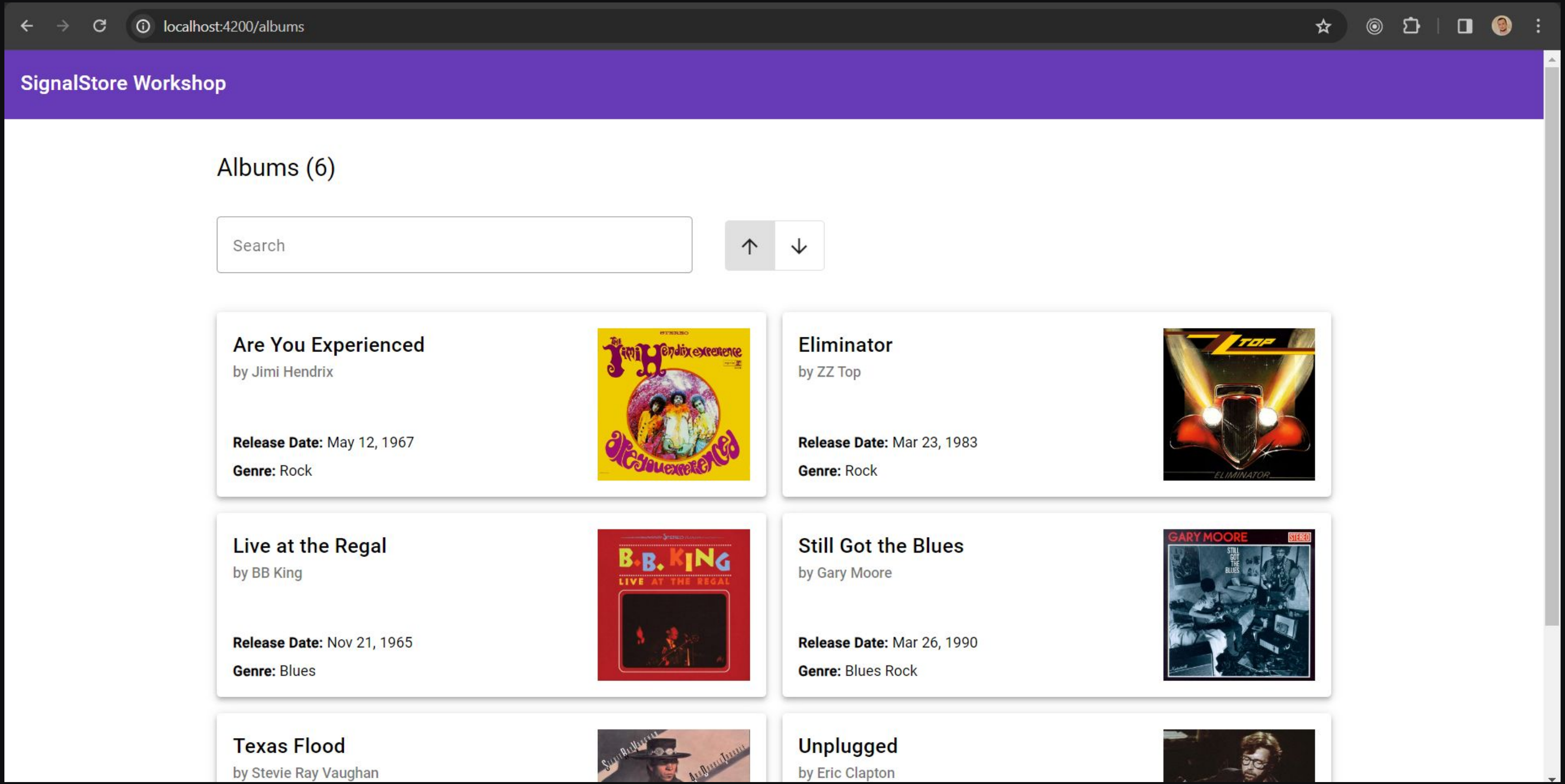
by BB King

Release Date: Nov 21, 1965

Genre: Blues

Every Day I Have the Blues	2:38
Sweet Little Angel	6:21
It's My Own Fault	3:29
How Blue Can You Get	3:29
Please Love Me	3:01
You Upset Me Baby	2:22
Worry, Worry	6:21
Woke Up This Mornin'	3:46
You Done Lost Your Good Thing Now	4:15
Help the Poor	2:36

Challenge: Examine the App



localhost:4200/albums

SignalStore Workshop

Albums (6)

Search

Are You Experienced
by Jimi Hendrix
Release Date: May 12, 1967
Genre: Rock

Eliminator
by ZZ Top
Release Date: Mar 23, 1983
Genre: Rock

Live at the Regal
by BB King
Release Date: Nov 21, 1965
Genre: Blues

Still Got the Blues
by Gary Moore
Release Date: Mar 26, 1990
Genre: Blues Rock

Texas Flood
by Stevie Ray Vaughan

Unplugged
by Eric Clapton

Signals

Signal

A wrapper around a value that notifies interested consumers when that value changes.

- ★ Writable Signals
- ★ Computed Signals

Writable Signals

```
import { signal } from '@angular/core';

const count = signal(0);
```

```
import { signal } from '@angular/core';

const count = signal(0);

console.log('count value', count());
// console output: count value 0
```

```
import { signal } from '@angular/core';

const count = signal(0);

console.log('count value', count());
// console output: count value 0

count.set(10);

console.log('count value', count());
// console output: count value 10
```

```
import { signal } from '@angular/core';

const count = signal(0);

console.log('count value', count());
// console output: count value 0

count.set(10);

console.log('count value', count());
// console output: count value 10

count.update((val) => val + 1);

console.log('count value', count());
// console output: count value 11
```

Computed Signals

```
import {           signal } from '@angular/core';

const count = signal(1);
```

```typescript
import { computed, signal } from '@angular/core';

const count = signal(1);
const doubleCount = computed(() => count() * 2);
```

```typescript
import { computed, signal } from '@angular/core';

const count = signal(1);
const doubleCount = computed(() => count() * 2);

console.log('double count value', doubleCount());
// console output: double count value 2
```

```
import { computed, signal } from '@angular/core';

const count = signal(1);
const doubleCount = computed(() => count() * 2);

console.log('double count value', doubleCount());
// console output: double count value 2

count.set(10);

console.log('count value', doubleCount());
// console output: double count value 20
```

```
import { computed, signal } from '@angular/core';

const count = signal(1);
const doubleCount = computed(() ⇒ count() * 2);

console.log('double count value', doubleCount());
// console output: double count value 2

count.set(10);

console.log('count value', doubleCount());
// console output: double count value 20

doubleCount.set(100); ✖
```

Effects

```
import {          signal } from '@angular/core';

const count = signal(1);
```

```
import { effect, signal } from '@angular/core';

const count = signal(1);

effect(() => {
  console.log('current count value', count());
});
// console output: current count value 1
```

```
import { effect, signal } from '@angular/core';

const count = signal(1);

effect(() => {
  console.log('current count value', count());
});
// console output: current count value 1

count.set(10);
// console output: current count value 10
```

```
import { effect, signal } from '@angular/core';

const count = signal(1);

effect(() => {
  console.log('current count value', count());
});
// console output: current count value 1

count.set(10);
// console output: current count value 10

count.update((val) => val + 1);
// console output: current count value 11
```

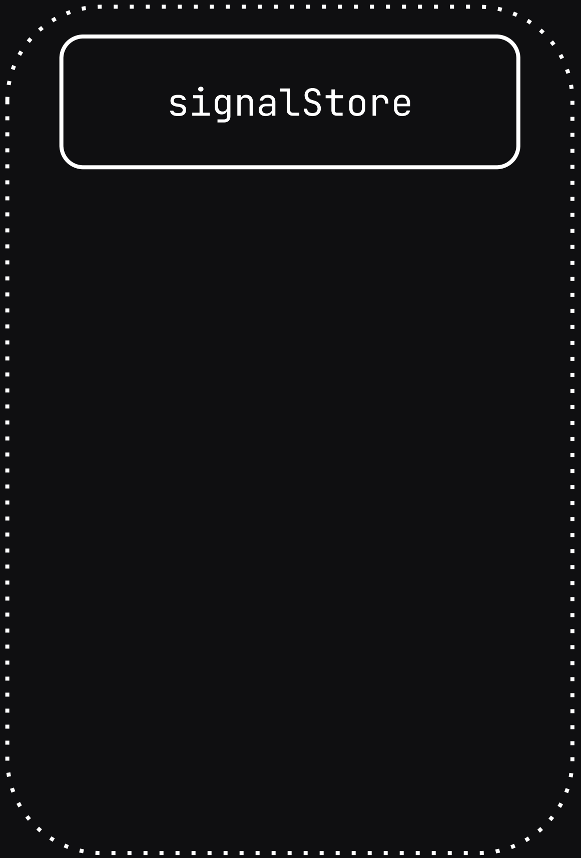
Demo

@ngrx/signals

@ngrx/signals

A standalone library that provides a reactive state management solution and a set of utilities for Angular Signals.

@ngrx/signals



@ngrx/signals

signalStore

withState

withComputed

withMethods

withHooks

@ngrx/signals

signalStore

withState

withComputed

withMethods

withHooks

signalStoreFeature

@ngrx/signals

- signalStore
- withState
- withComputed
- withMethods
- withHooks
- signalStoreFeature

- signalState
- patchState

@ngrx/signals

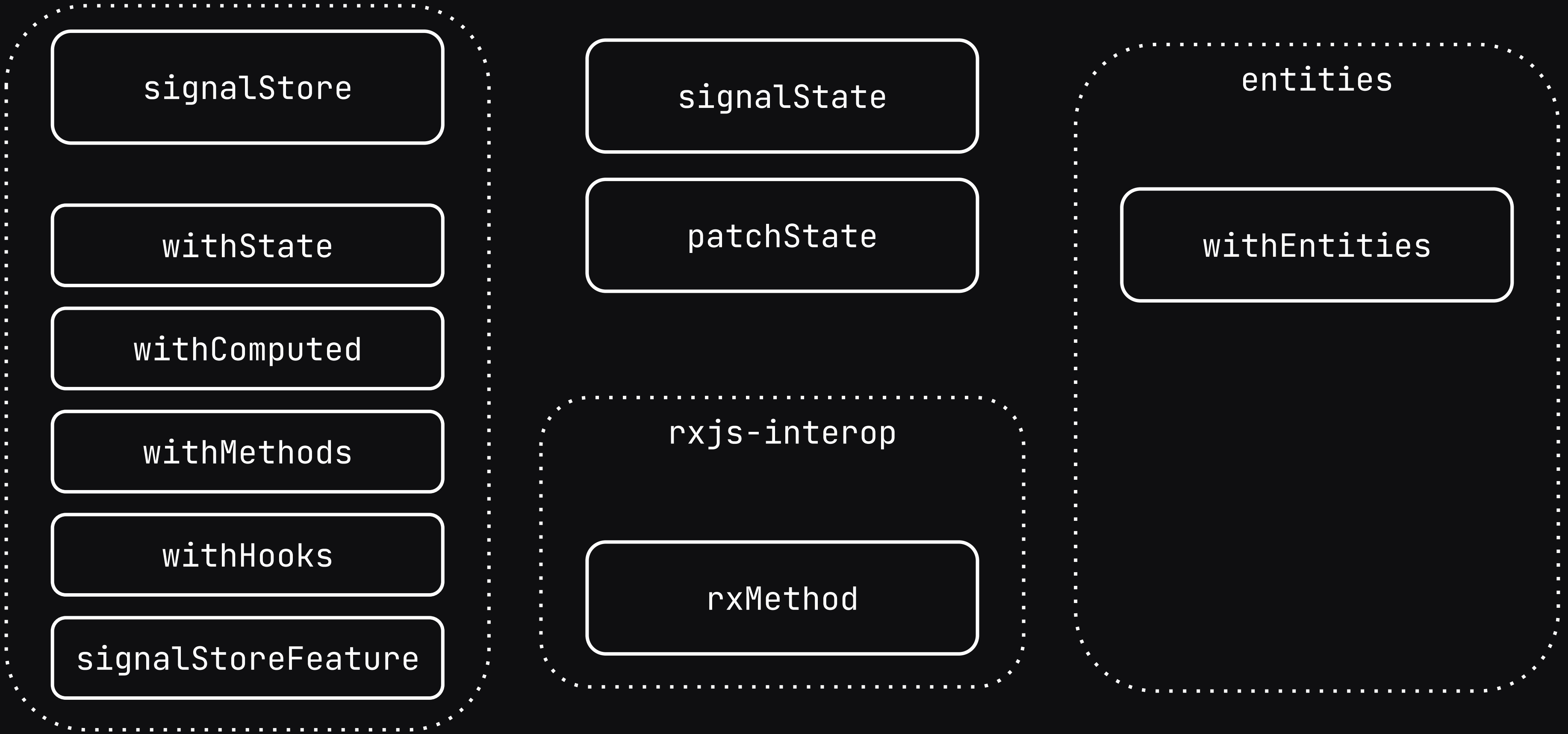
- signalStore
- withState
- withComputed
- withMethods
- withHooks
- signalStoreFeature

- signalState
- patchState

rxjs-interop

- rxMethod

@ngrx/signals



@ngrx/signals

- signalStore
- withState
- withComputed
- withMethods
- withHooks
- signalStoreFeature

- signalState
- patchState

rxjs-interop

- rxMethod

entities

- withEntities
- addEntity
- updateEntity
- ...

SignalState

SignalState

A lightweight utility for managing signal-based state in Angular components and services in a concise and minimalistic manner.

```
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});
```

```typescript
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});
```

initial state

```typescript
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
}); // type: SignalState<UserState>
```

```
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
}); // type: SignalState<UserState>

console.log(userState()); // logs the initial state
```

`userState` is a signal

```ts
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
}); // type: SignalState<UserState>

console.log(userState()); // logs the initial state

const user = userState.user; // type: DeepSignal<User>
console.log(user()); // logs: { firstName: 'Eric', lastName: 'Clapton' }
```

```ts
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
}); // type: SignalState<UserState>

console.log(userState()); // logs the initial state

const user = userState.user; // type: DeepSignal<User>
console.log(user()); // logs: { firstName: 'Eric', lastName: 'Clapton' }

const firstName = user.firstName; // type: Signal<string>
const lastName = user.lastName; // type: Signal<string>
```

```typescript
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
}); // type: SignalState<UserState>

console.log(userState()); // logs the initial state

const user = userState.user; // type: DeepSignal<User>
console.log(user()); // logs: { firstName: 'Eric', lastName: 'Clapton' }

const firstName = user.firstName; // type: Signal<string>
const lastName = user.lastName; // type: Signal<string>

console.log(firstName()); // logs: 'Eric'
console.log(lastName()); // logs: 'Clapton'
```

Demo

Updating State

```
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});
```

```typescript
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});

patchState(userState, { isAdmin: true }); // partial state object
```

```
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});

patchState(userState, { isAdmin: true }); // partial state object

// partial state updater
patchState(userState, (state) => ({
  user: { ...state.user, firstName: 'Jimi' },
}));
```

```
type UserState = { user: User; isAdmin: boolean };

const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});

patchState(userState, { isAdmin: true }); // partial state object

// partial state updater
patchState(userState, (state) ⇒ ({
  user: { ...state.user, firstName: 'Jimi' },
}));

// a sequence of partial state objects and/or updaters
patchState(
  userState,
  { isAdmin: false },
  (state) ⇒ ({ user: { ...state.user, lastName: 'Hendrix' } })
);
```

Custom State Updaters

```
const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});
```

```typescript
const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});

function setFirstName(
  firstName: string
): PartialStateUpdater<{ user: User }> {
  return (state) ⇒ ({ user: { ...state.user, firstName } });
}
```

```
const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});

function setFirstName(
  firstName: string
): PartialStateUpdater<{ user: User }> {
  return (state) ⇒ ({ user: { ...state.user, firstName } });
}

const setAdmin = () ⇒ ({ isAdmin: true });
```

```ts
const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});

function setFirstName(
  firstName: string
): PartialStateUpdater<{ user: User }> {
  return (state) => ({ user: { ...state.user, firstName } });
}

const setAdmin = () => ({ isAdmin: true });
```

```ts
patchState(
  userState,
  (state) => ({
    user: { ...state.user, firstName: 'Stevie' },
    isAdmin: true,
  }),
);
```

```
const userState = signalState<UserState>({
  user: { firstName: 'Eric', lastName: 'Clapton' },
  isAdmin: false,
});

function setFirstName(
  firstName: string
): PartialStateUpdater<{ user: User }> {
  return (state) => ({ user: { ...state.user, firstName } });
}

const setAdmin = () => ({ isAdmin: true });
```

```
patchState(
  userState,
  (state) => ({
    user: { ...state.user, firstName: 'Stevie' },
    isAdmin: true,
  }),
);
```

```
patchState(
  userState,
  setFirstName('Stevie'),
  setAdmin()
);
```

Demo

Milestone 01: SignalState

1. Use `signalState` to manage the state of the `AlbumSearchComponent`.
   💡 State properties: `albums`, `showProgress`, `query`, `order`.
2. Create computed signal `filteredAlbums` that filters `albums` by `query` and sorts them by `order`.
   💡 Utilities `searchAlbums` and `sortAlbums` are exported from the `album.model.ts` file.
3. Create computed signal `totalAlbums` that should calculate the length of `filteredAlbums`.
4. Create computed signal `showSpinner` that should be true when `showProgress` is true and `albums` length is 0.
5. Adjust the template to consume created signals.
6. Implement `updateQuery` and `updateOrder` methods by using the `patchState` function.
7. Inject `AlbumsService` and use the `getAll` method to fetch all albums from the API when `AlbumSearchComponent` is initialized.
   💡 Set `showProgress` to false when the request succeeds or fails.
   💡 Use `MatSnackBar` to show an error when the request fails.

RxMethod

RxMethod

A standalone factory function designed for managing side effects by utilizing RxJS APIs. It takes a chain of RxJS operators as input and returns a reactive method.

```
@Component({ /* ... */ })
export class NumbersComponent implements OnInit {

  readonly logDoubledNumber = rxMethod<number>(
    pipe(
      map((num) => num * 2),
      tap((doubledNum) => console.log(doubledNum)),
    ),
  );









}
```

```
@Component({ /* ... */ })
export class NumbersComponent implements OnInit {

  readonly logDoubledNumber = rxMethod<number>(
    pipe(
      map((num) => num * 2),
      tap((doubledNum) => console.log(doubledNum)),
    ),
  );









}
```

RxJS operators can be chained together using the `pipe` function.

```
@Component({ /* ... */ })
export class NumbersComponent implements OnInit {

  readonly logDoubledNumber = rxMethod<number>(
    pipe(
      map((num) => num * 2),
      tap((doubledNum) => console.log(doubledNum)),
    ),
  );








}
```

Input can be typed by providing a generic argument.

```
@Component({ /* ... */ })
export class NumbersComponent implements OnInit {

  readonly logDoubledNumber = rxMethod<number>(
    pipe(
      map((num) => num * 2),
      tap((doubledNum) => console.log(doubledNum)),
    ),
  );

}
```

A reactive method will have an input argument of type `number | Signal<number> | Observable<number>`

```
@Component({ /* ... */ })
export class NumbersComponent implements OnInit {

  readonly logDoubledNumber = rxMethod<number>(
    pipe(
      map((num) => num * 2),
      tap((doubledNum) => console.log(doubledNum)),
    ),
  );

  ngOnInit(): void {















  }
}
```

```
@Component({ /* ... */ })
export class NumbersComponent implements OnInit {

  readonly logDoubledNumber = rxMethod<number>(
    pipe(
      map((num) => num * 2),
      tap((doubledNum) => console.log(doubledNum)),
    ),
  );

  ngOnInit(): void {
    this.logDoubledNumber(1);
    // console output: 2










  }
}
```

```typescript
@Component({ /* ... */ })
export class NumbersComponent implements OnInit {

  readonly logDoubledNumber = rxMethod<number>(
    pipe(
      map((num) => num * 2),
      tap((doubledNum) => console.log(doubledNum)),
    ),
  );

  ngOnInit(): void {
    this.logDoubledNumber(1);
    // console output: 2

    const num$ = interval(2_000);
    this.logDoubledNumber(num$);
    // console output: 0, 2, 4, 6... every 2 seconds




  }
}
```

```
@Component({ /* ... */ })
export class NumbersComponent implements OnInit {

  readonly logDoubledNumber = rxMethod<number>(
    pipe(
      map((num) => num * 2),
      tap((doubledNum) => console.log(doubledNum)),
    ),
  );

  ngOnInit(): void {
    this.logDoubledNumber(1);
    // console output: 2

    const num$ = interval(2_000);
    this.logDoubledNumber(num$);
    // console output: 0, 2, 4, 6... every 2 seconds

    const num = signal(100);
    this.logDoubledNumber(num);
    // console output: 200



  }
}
```

```
@Component({ /* ... */ })
export class NumbersComponent implements OnInit {

  readonly logDoubledNumber = rxMethod<number>(
    pipe(
      map((num) => num * 2),
      tap((doubledNum) => console.log(doubledNum)),
    ),
  );

  ngOnInit(): void {
    this.logDoubledNumber(1);
    // console output: 2

    const num$ = interval(2_000);
    this.logDoubledNumber(num$);
    // console output: 0, 2, 4, 6... every 2 seconds

    const num = signal(100);
    this.logDoubledNumber(num);
    // console output: 200

    num.set(200);
    // console output: 400
  }
}
```

Milestone 02: RxMethod

1. Create reactive method `loadAllAlbums` by using the `rxMethod` function that fetches all albums from the API.
   💡 Use `exhaustMap` to prevent parallel calls when the reactive method is called multiple times.
   💡 Use the `tapResponse` operator from the `@ngrx/operators` package to keep the reactive method subscription alive if the request fails.

2. Invoke the `loadAllAlbums` method when `AlbumSearchComponent` is initialized.

SignalStore

SignalStore

A fully-featured state management solution that provides native support for Angular Signals and offers a robust way to manage application state.

Key Principles

Key Principles

- Simple and Intuitive

Key Principles

- Simple and Intuitive
- Lightweight and Performant

Key Principles

- Simple and Intuitive
- Lightweight and Performant
- Declarative

Key Principles

- Simple and Intuitive
- Lightweight and Performant
- Declarative
- Modular, Extensible, and Scalable

Key Principles

- Simple and Intuitive
- Lightweight and Performant
- Declarative
- Modular, Extensible, and Scalable
- Opinionated, but Flexible

Key Principles

- Simple and Intuitive
- Lightweight and Performant
- Declarative
- Modular, Extensible, and Scalable
- Opinionated, but Flexible
- Type-safe

Creating a Store

```
const TodosStore = signalStore(















);
```

```
type TodosState = { todos: Todo[] };

const TodosStore = signalStore(
  withState<TodosState>({ todos: [] }),














);
```

```
type TodosState = { todos: Todo[] };

const TodosStore = signalStore(
  withState<TodosState>({ todos: [] }),
  withComputed(({ todos }) ⇒ ({




  })),







);
```

```
type TodosState = { todos: Todo[] };

const TodosStore = signalStore(
  withState<TodosState>({ todos: [] }),
  withComputed(({ todos }) => ({
    completedTodos: computed(() =>
      todos().filter((todo) => todo.completed)
    ),
  })),









);
```

```
type TodosState = { todos: Todo[] };

const TodosStore = signalStore(
  withState<TodosState>({ todos: [] }),
  withComputed(({ todos }) => ({
    completedTodos: computed(() =>
      todos().filter((todo) => todo.completed)
    ),
  })),
  withMethods((store) => ({







  }))
);
```

```
type TodosState = { todos: Todo[] };

const TodosStore = signalStore(
  withState<TodosState>({ todos: [] }),
  withComputed(({ todos }) => ({
    completedTodos: computed(() =>
      todos().filter((todo) => todo.completed)
    ),
  })),
  withMethods((store) => ({
    addTodo(todo: Todo): void {
      patchState(store, {
        todos: [...store.todos(), todo],
      });
    },
  }))
);
```

```
@Component({
  template: `
    <h1>Add Todo</h1>
    <todo-form (addTodo)="store.addTodo($event)" />

    <h1>Todos</h1>
    <todo-list [todos]="store.todos()" />

    <h1>Completed Todos</h1>
    <todo-list [todos]="store.completedTodos()" />
  `,
  providers: [TodosStore],
})
export class TodosComponent {
  readonly store = inject(TodosStore);
}
```

```
@Component({
  template: `
    <h1>Add Todo</h1>
    <todo-form (addTodo)="store.addTodo($event)" />

    <h1>Todos</h1>
    <todo-list [todos]="store.todos()" />

    <h1>Completed Todos</h1>
    <todo-list [todos]="store.completedTodos()" />
  `,
  providers: [TodosStore],
})
export class TodosComponent {
  readonly store = inject(TodosStore);
}
```

```
@Component({
  template: `
    <h1>Add Todo</h1>
    <todo-form (addTodo)="store.addTodo($event)" />

    <h1>Todos</h1>
    <todo-list [todos]="store.todos()" />

    <h1>Completed Todos</h1>
    <todo-list [todos]="store.completedTodos()" />
  `,
  providers: [TodosStore],
})
export class TodosComponent {
  readonly store = inject(TodosStore);
}
```

Lifecycle Hooks

```
type TodosState = { todos: Todo[] };

const TodosStore = signalStore(
  withState<TodosState>({ todos: [] }),
  withComputed(({ todos }) => ({
    completedTodos: computed(() =>
      todos().filter((todo) => todo.completed)
    ),
  })),
  withMethods((store) => ({
    addTodo(todo: Todo): void {
      patchState(store, {
        todos: [...store.todos(), todo],
      });
    },
  }))









);
```

```
type TodosState = { todos: Todo[] };

const TodosStore = signalStore(
  withState<TodosState>({ todos: [] }),
  withComputed(({ todos }) => ({
    completedTodos: computed(() =>
      todos().filter((todo) => todo.completed)
    ),
  })),
  withMethods((store) => ({
    addTodo(todo: Todo): void {
      patchState(store, {
        todos: [...store.todos(), todo],
      });
    },
  }))
  withHooks(({ todos }) => ({







  }))
);
```

```
type TodosState = { todos: Todo[] };

const TodosStore = signalStore(
  withState<TodosState>({ todos: [] }),
  withComputed(({ todos }) => ({
    completedTodos: computed(() =>
      todos().filter((todo) => todo.completed)
    ),
  })),
  withMethods((store) => ({
    addTodo(todo: Todo): void {
      patchState(store, {
        todos: [...store.todos(), todo],
      });
    },
  }))
  withHooks(({ todos }) => ({
    onInit() {
      console.log('todos on init', todos());
    },




  }))
);
```

```typescript
type TodosState = { todos: Todo[] };

const TodosStore = signalStore(
  withState<TodosState>({ todos: [] }),
  withComputed(({ todos }) => ({
    completedTodos: computed(() =>
      todos().filter((todo) => todo.completed)
    ),
  })),
  withMethods((store) => ({
    addTodo(todo: Todo): void {
      patchState(store, {
        todos: [...store.todos(), todo],
      });
    },
  }))
  withHooks(({ todos }) => ({
    onInit() {
      console.log('todos on init', todos());
    },
    onDestroy() {
      console.log('todos on destroy', todos());
    },
  }))
);
```

Why functional approach?

Limitations of class-based approach

Limitations of class-based approach

- **Typing:** It’s not possible to define dynamic class properties or methods that are strongly typed.

```
@Injectable()
export class BooksStore extends ComponentStore<BooksState> {









  constructor() {
    super({ books: [], query: '', isPending: false });
  }
}
```

```
@Injectable()
export class BooksStore extends ComponentStore<BooksState> {




  readonly filteredBooks = this.selectSignal(
    this.books, ❌
    this.query, ❌
    (books, query) ⇒ books.filter(({ title }) ⇒ title.includes(query)),
  );

  constructor() {
    super({ books: [], query: '', isPending: false });
  }
}
```

```typescript
@Injectable()
export class BooksStore extends ComponentStore<BooksState> {
  readonly books = this.selectSignal((s) => s.books);
  readonly query = this.selectSignal((s) => s.query);

  readonly filteredBooks = this.selectSignal(
    this.books,
    this.query,
    (books, query) => books.filter(({ title }) => title.includes(query)),
  );

  constructor() {
    super({ books: [], query: '', isPending: false });
  }
}
```

Limitations of class-based approach

- **Typing:** It’s not possible to define dynamic class properties or methods that are strongly typed.
- **Tree-shaking:** Unused class methods and properties won’t be removed from the final bundle.

Limitations of class-based approach

- **Typing:** It’s not possible to define dynamic class properties or methods that are strongly typed.
- **Tree-shaking:** Unused class methods and properties won’t be removed from the final bundle.
- **Extensibility:** Multiple inheritance is not supported.

Limitations of class-based approach

- **Typing:** It’s not possible to define dynamic class properties or methods that are strongly typed.
- **Tree-shaking:** Unused class methods and properties won’t be removed from the final bundle.
- **Extensibility:** Multiple inheritance is not supported.
- **Modularity:** Splitting selectors, updaters, and effects into different classes is possible, but not provided out of the box.

Demo

Milestone 03: SignalStore

1. Create the `album-search.store.ts` file and initialize `AlbumSearchStore` by using the `signalStore` function.
2. Remove `signalState` from `AlbumSearchComponent` and move the state to `AlbumSearchStore` using the `withState` feature.
3. Move all computed signals to the `AlbumSearchStore` using the `withComputed` feature.
   💡 Computed signals: `filteredAlbums`, `totalAlbums`, `showSpinner`.
4. Move all methods to the `AlbumSearchStore` using the `withMethods` feature.
   💡 Methods: `updateQuery`, `updateOrder`, `loadAllAlbums`.
   💡 `AlbumsService` and `MatSnackBar` can be injected within the `withMethods` factory function.
5. Remove the `ngOnInit` method from `AlbumSearchComponent` and invoke the `loadAllAlbums` method when `AlbumSearchStore` is initialized using the `withHooks` feature.
6. Provide `AlbumSearchStore` at the `AlbumSearchComponent` level and inject it into the component.
7. Adjust the template to consume signals and methods from the injected store.

Custom Store Features

```
export function withRequestStatus() {









}
```

```
export function withRequestStatus() {
  return signalStoreFeature(









  );
}
```

```
export type RequestStatus = 'idle' | 'pending' | 'fulfilled' | { error: string };

export type RequestStatusState = { requestStatus: RequestStatus };

export function withRequestStatus() {
  return signalStoreFeature(
    withState<RequestStatusState>({ requestStatus: 'idle' }),









  );
}
```

```typescript
export type RequestStatus = 'idle' | 'pending' | 'fulfilled' | { error: string };

export type RequestStatusState = { requestStatus: RequestStatus };

export function withRequestStatus() {
  return signalStoreFeature(
    withState<RequestStatusState>({ requestStatus: 'idle' }),
    withComputed(({ requestStatus }) => ({
      isPending: computed(() => requestStatus() === 'pending'),
      isFulfilled: computed(() => requestStatus() === 'fulfilled'),
      error: computed(() => {
        const status = requestStatus();
        return typeof status === 'object' ? status.error : null;
      }),
    })),
  );
}
```

```
export const BooksStore = signalStore(
  withState({ books: [] as Book[], isPending: false }), ❌







);
```

```
export const BooksStore = signalStore(
  withState({ books: [] as Book[] }),
  withRequestStatus(), ✓









);
```

```
export const BooksStore = signalStore(
  withState({ books: [] as Book[] }),
  withRequestStatus(),




);
```

State properties:

- `requestStatus: Signal<RequestStatus>`

Computed properties:

- `isPending: Signal<boolean>`
- `isFulfilled: Signal<boolean>`
- `error: Signal<string | null>`

```typescript
export function setPending(): RequestStatusState {
  return { requestStatus: 'pending' };
}

export function setFulfilled(): RequestStatusState {
  return { requestStatus: 'fulfilled' };
}

export function setError(error: string): RequestStatusState {
  return { requestStatus: { error } };
}
```

```
export const BooksStore = signalStore(
  withState({ books: [] as Book[] }),
  withRequestStatus(),
  withMethods((store, booksService = inject(BooksService)) => ({
    async loadAll() {
      patchState(store, { isPending: true }); ❌

      const books = await booksService.getAll();
      patchState(store, { books, isPending: false }); ❌
    },
  })),
);
```

```
export const BooksStore = signalStore(
  withState({ books: [] as Book[] }),
  withRequestStatus(),
  withMethods((store, booksService = inject(BooksService)) ⇒ ({
    async loadAll() {
      patchState(store, setPending()); ✓

      const books = await booksService.getAll();
      patchState(store, { books }, setFulfilled()); ✓
    },
  })),
);
```

Milestone 04: Custom Store Features

1. Create the `shared/state/request-status.feature.ts` file and implement the `withRequestStatus` feature with the corresponding updaters.
2. Remove the `showProgress` state property from `AlbumSearchStore` and use the `withRequestStatus` feature instead.
3. Adjust computed signals to use the `isPending` signal from the `withRequestStatus` feature.
4. Use `setPending`, `setFulfilled`, and `setError` updaters to update the state in the `loadAllAlbums` reactive method.

Entities

```
export const TodosStore = signalStore(









);
```

```
export const TodosStore = signalStore(
  withEntities<Todo>(),











);
```

```
export const TodosStore = signalStore(
  withEntities<Todo>(),



);
```

State properties:

- `entityMap: Signal<EntityMap<Todo>>`
- `ids: Signal<EntityId[]>`

Computed properties:

- `entities: Signal<Todo[]>`

```
export const TodosStore = signalStore(
  withEntities<Todo>(),
  withMethods((store) => ({
    addTodo(todo: Todo): void {
      patchState(store, addEntity(todo));
    },







  })),
);
```

```
export const TodosStore = signalStore(
  withEntities<Todo>(),
  withMethods((store) => ({
    addTodo(todo: Todo): void {
      patchState(store, addEntity(todo));
    },
    removeTodo(id: number): void {
      patchState(store, removeEntity(id));
    },
    completeAllTodos(): void {
      patchState(store, updateAllEntities({ completed: true }));
    },
  })),
);
```

```
export const TodosStore = signalStore(
  withEntities<Todo>(),
  withMethods((store) => ({
    addTodo(todo: Todo): void {
      patchState(store, addEntity(todo));
    },
    removeTodo(id: number): void {
      patchState(store, removeEntity(id));
    },
    completeAllTodos(): void {
      patchState(store, updateAllEntities({ completed: true }));
    },
  })),
);
```

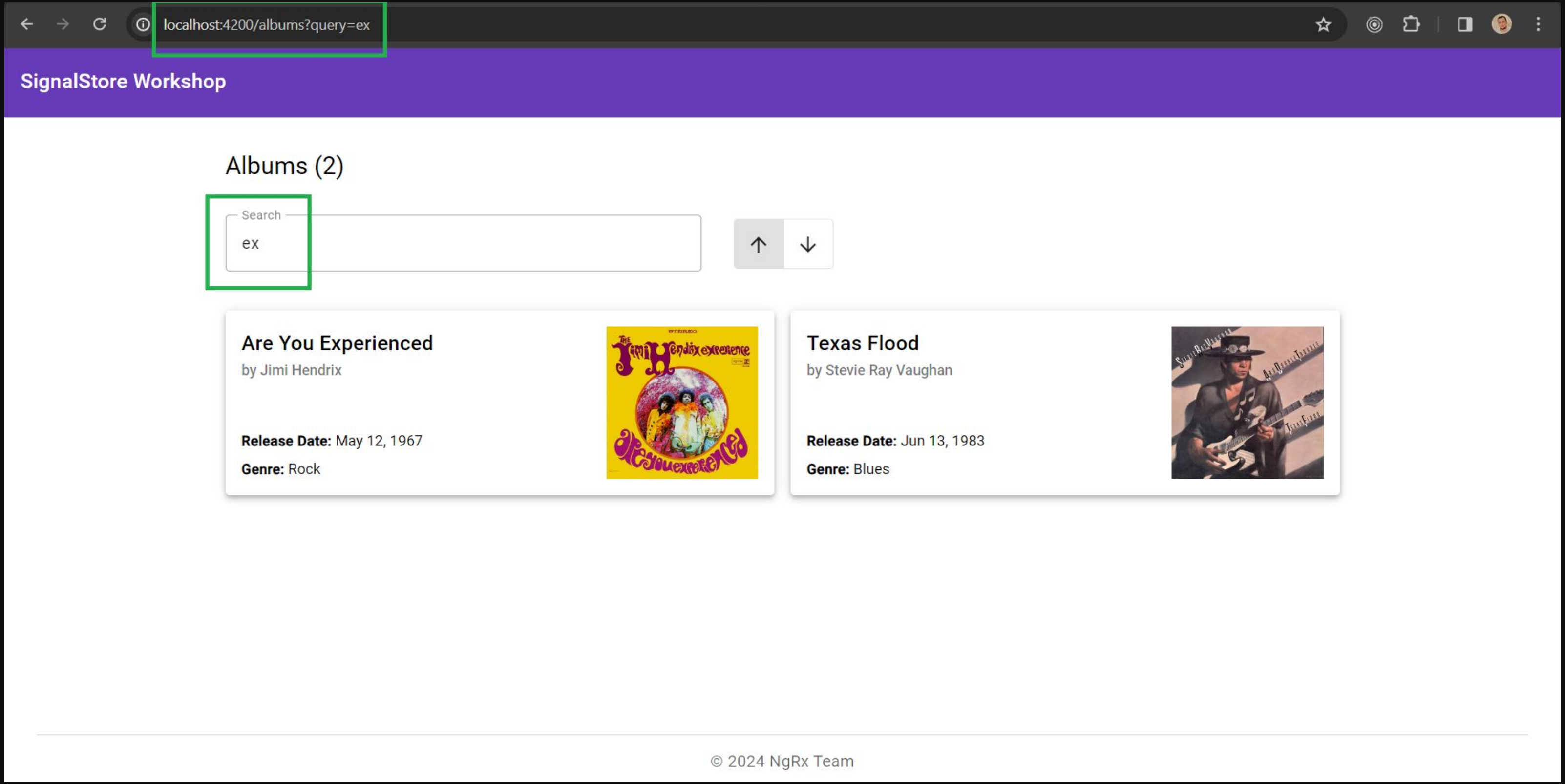
standalone functions

Milestone 05: Entities

1. Remove the `albums` state property from `AlbumSearchStore` and use the `withEntities` feature instead.
2. Adjust computed signals to use the `entities` signal from the `withEntities` feature.
3. Use `setAllEntities` updater to update the state in the `loadAllAlbums` reactive method.

Router State

Router State



localhost:4200/albums?query=ex

SignalStore Workshop

Albums (2)

Search

ex

Are You Experienced

by Jimi Hendrix

Release Date: May 12, 1967

Genre: Rock

Texas Flood

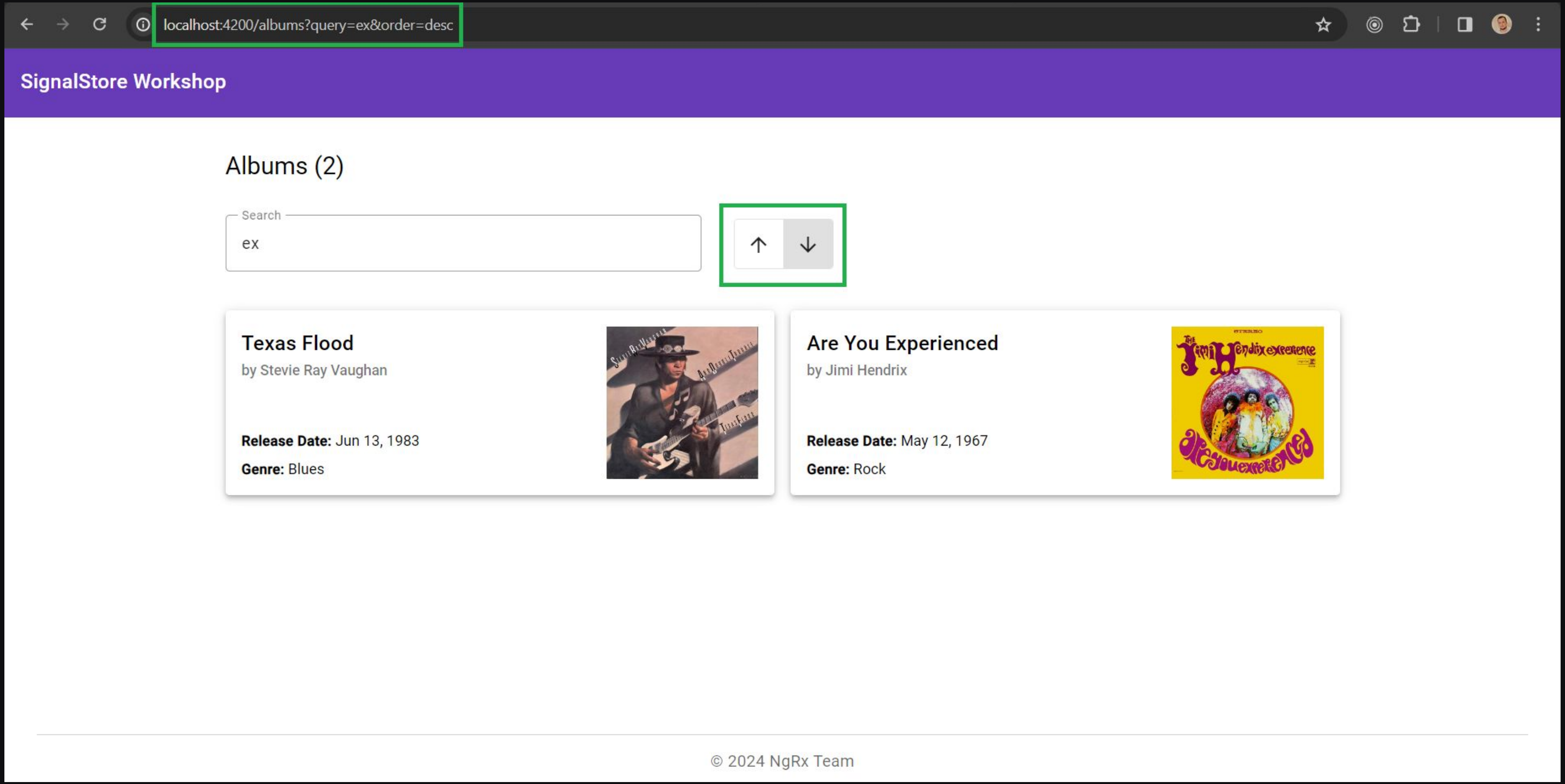
by Stevie Ray Vaughan

Release Date: Jun 13, 1983

Genre: Blues

© 2024 NgRx Team

Router State



localhost:4200/albums?query=ex&order=desc

SignalStore Workshop

Albums (2)

Search

ex

Texas Flood

by Stevie Ray Vaughan

Release Date: Jun 13, 1983

Genre: Blues

Are You Experienced

by Jimi Hendrix

Release Date: May 12, 1967

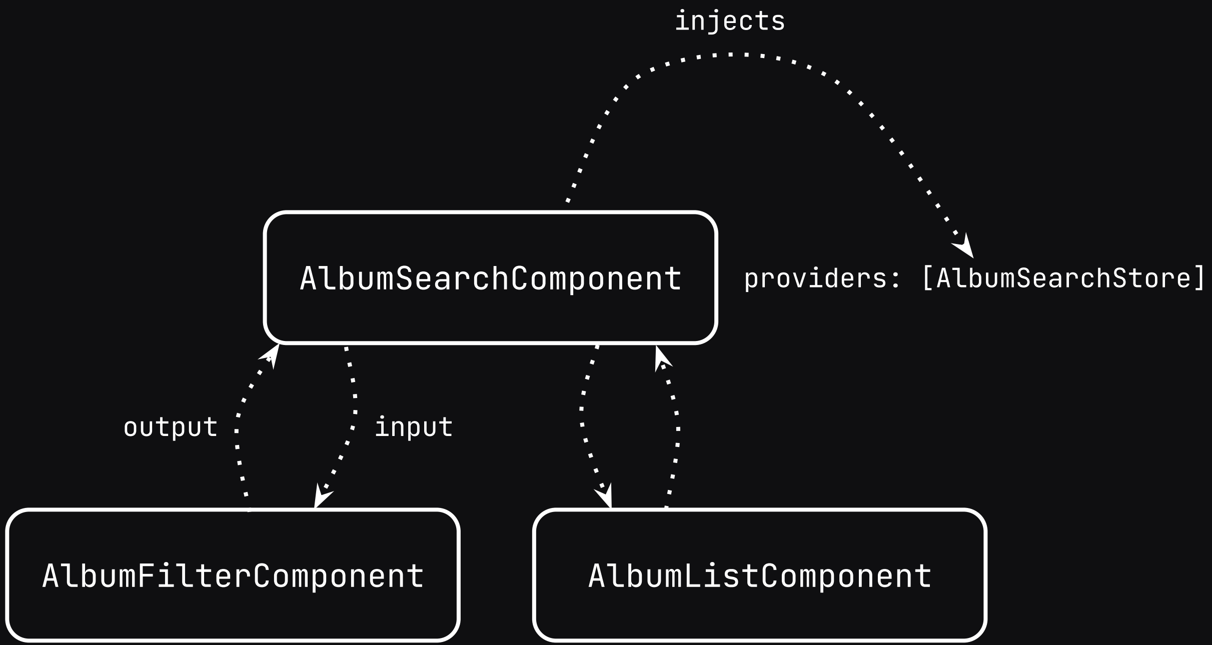
Genre: Rock

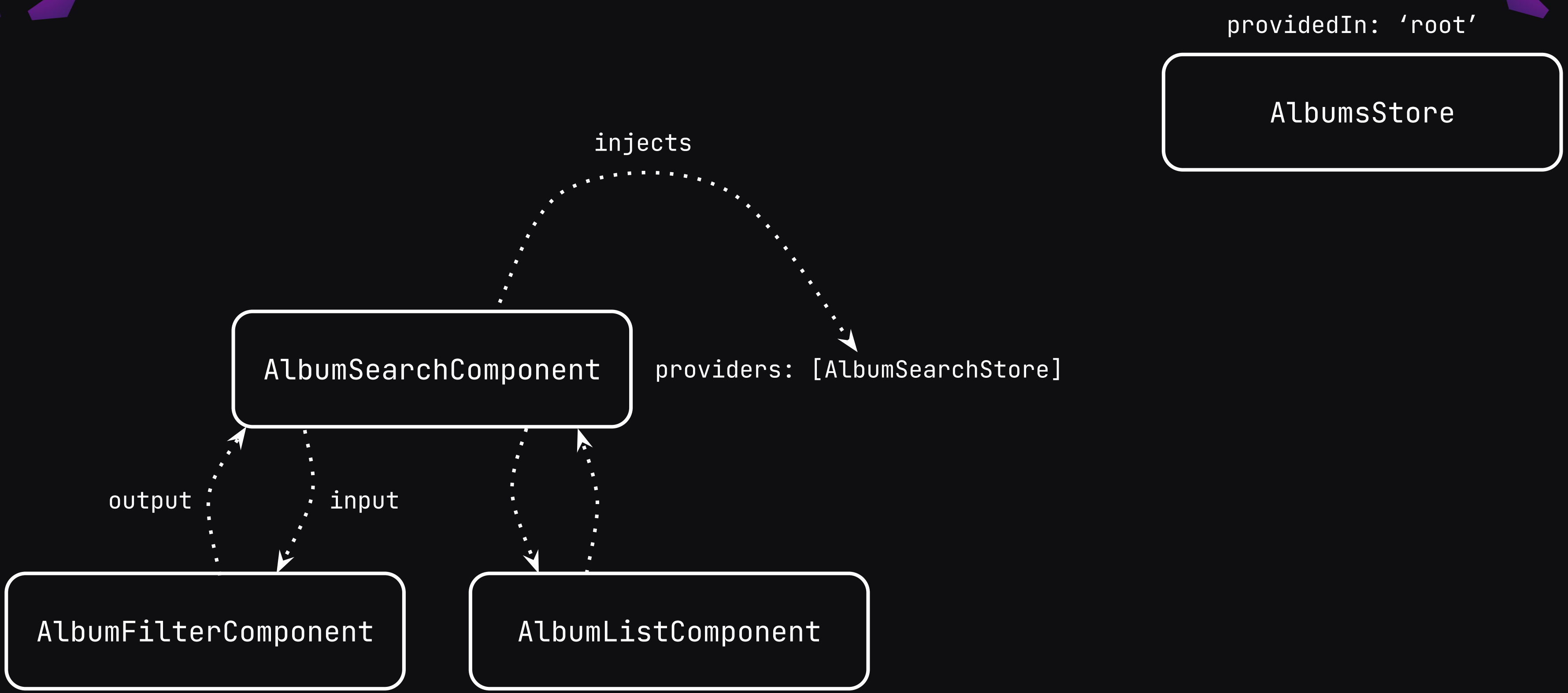
© 2024 NgRx Team

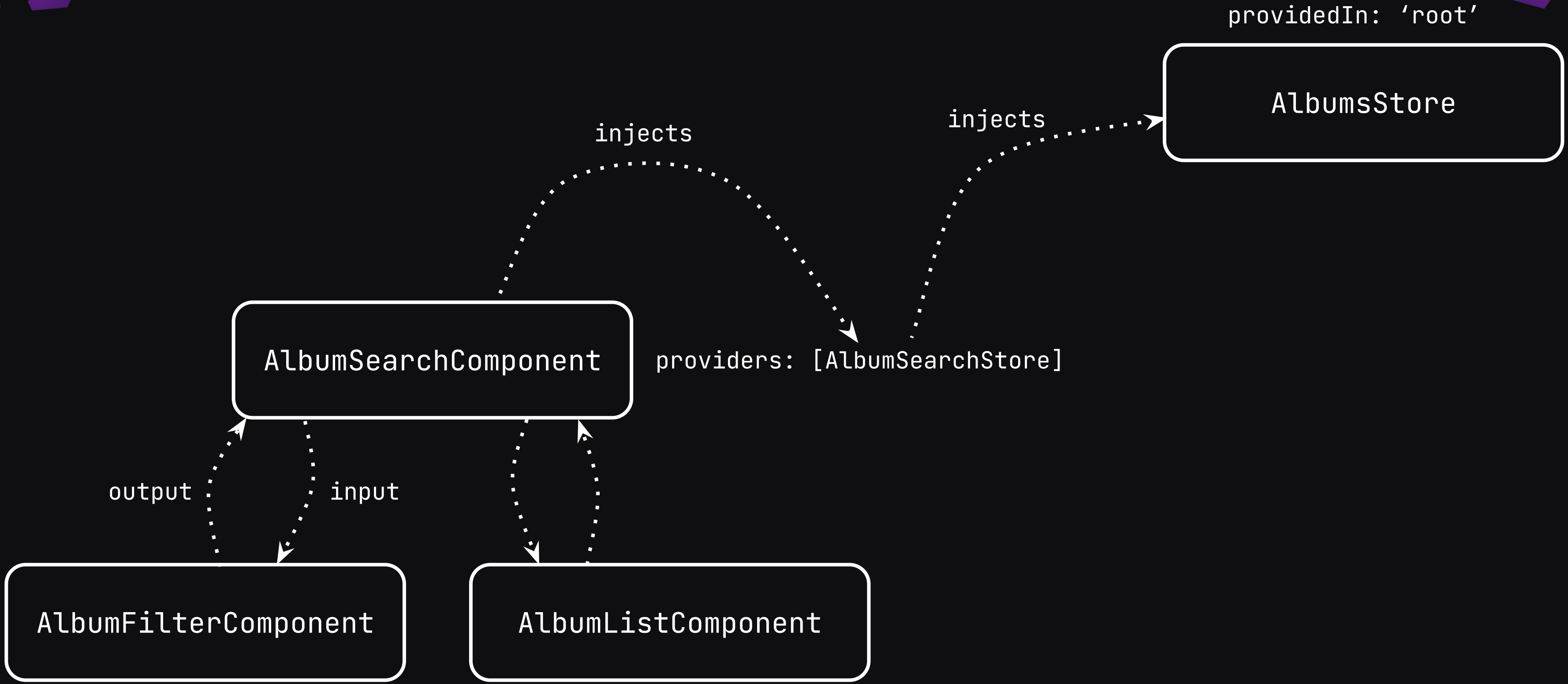
Milestone 06: Router State

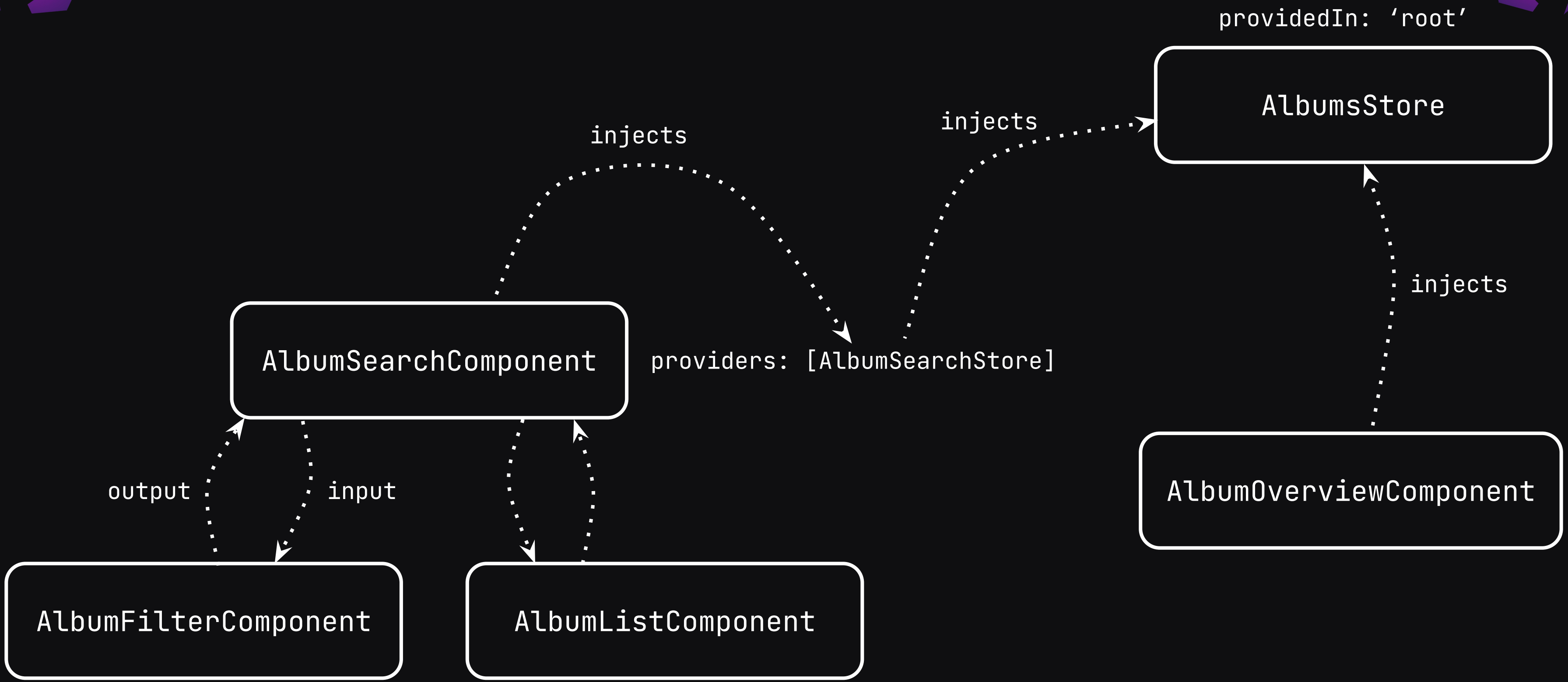
1. Remove `query` and `order` state properties together with `updateQuery` and `updateOrder` methods.
2. Use the `withQueryParams` feature to synchronize the album filter with `query` and `order` query parameters.

Global State


injects
AlbumSearchComponent
providers: [AlbumSearchStore]
output
input
AlbumFilterComponent
AlbumListComponent


providedIn: ‘root’
AlbumsStore
injects
AlbumSearchComponent
providers: [AlbumSearchStore]
output
input
AlbumFilterComponent
AlbumListComponent


providedIn: 'root'
AlbumsStore
injects
injects
AlbumSearchComponent
providers: [AlbumSearchStore]
output
input
AlbumFilterComponent
AlbumListComponent


providedIn: ‘root’
AlbumsStore
injects
injects
injects
AlbumSearchComponent
providers: [AlbumSearchStore]
output
input
AlbumOverviewComponent
AlbumFilterComponent
AlbumListComponent

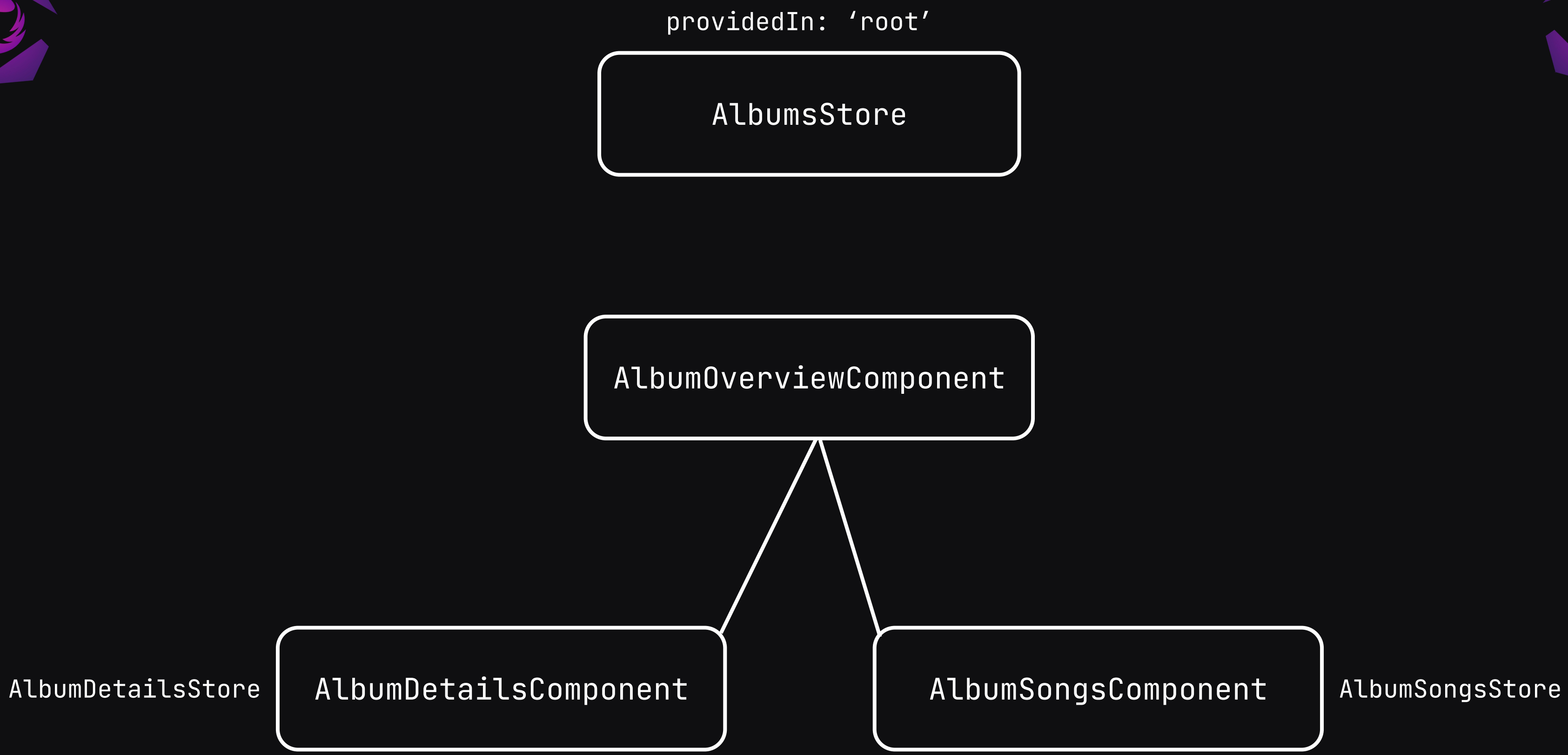
Providing SignalStore at the root level

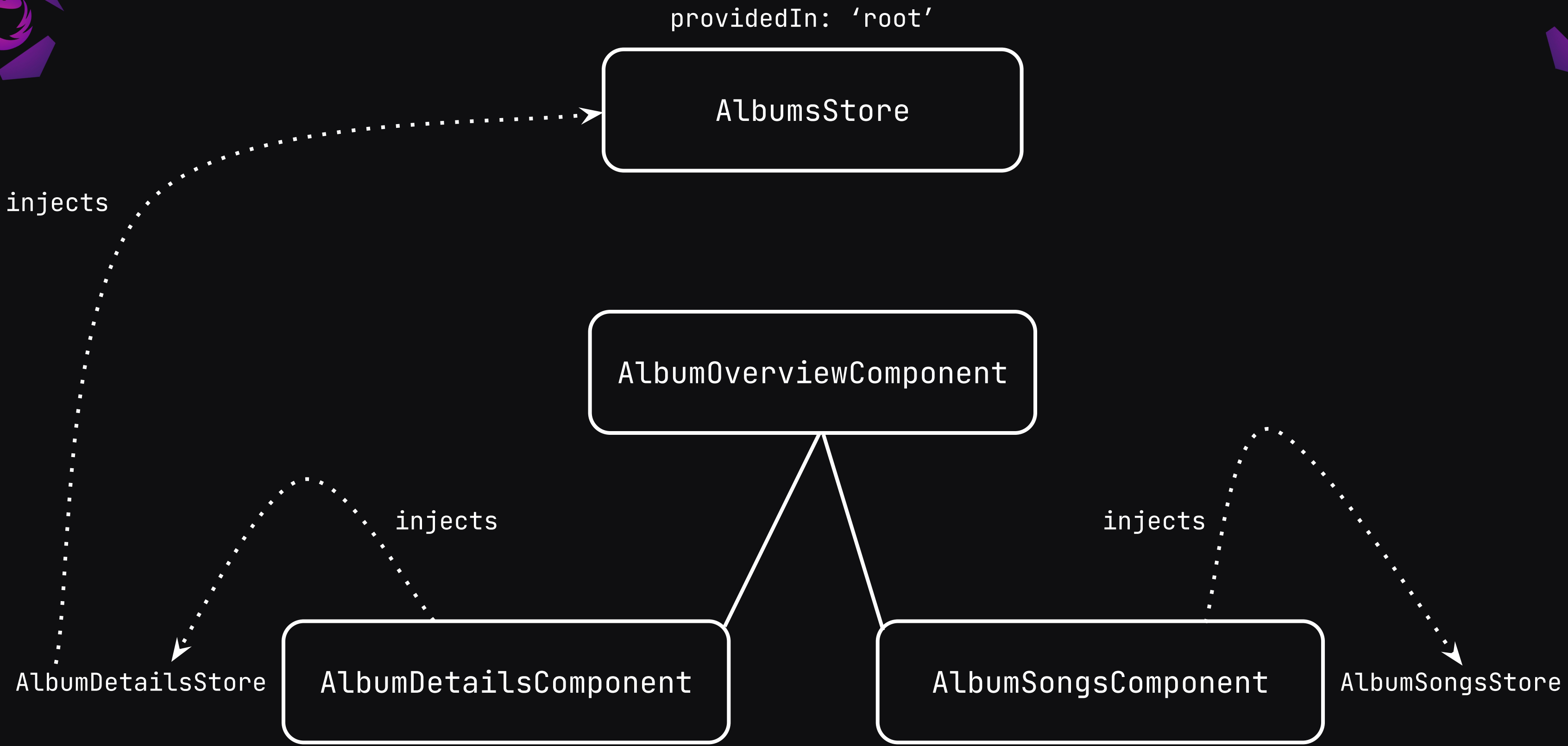
```
export const AlbumsStore = signalStore(
  { providedIn: 'root' },
  /* ... */
);
```

Milestone 07: Global State

1. Create `AlbumsStore` in the `albums/albums.store.ts` file and provide it at the root level.
2. Move entity collection, request status, and `loadAllAlbums` method from `AlbumSearchStore` to `AlbumsStore`.
3. Adjust computed signals from the `AlbumSearchStore` to use `entities` and `isPending` signals from the `AlbumsStore`.
4. Invoke the `loadAllAlbums` method from the `AlbumsStore` when `AlbumSearchStore` is initialized.

Local State

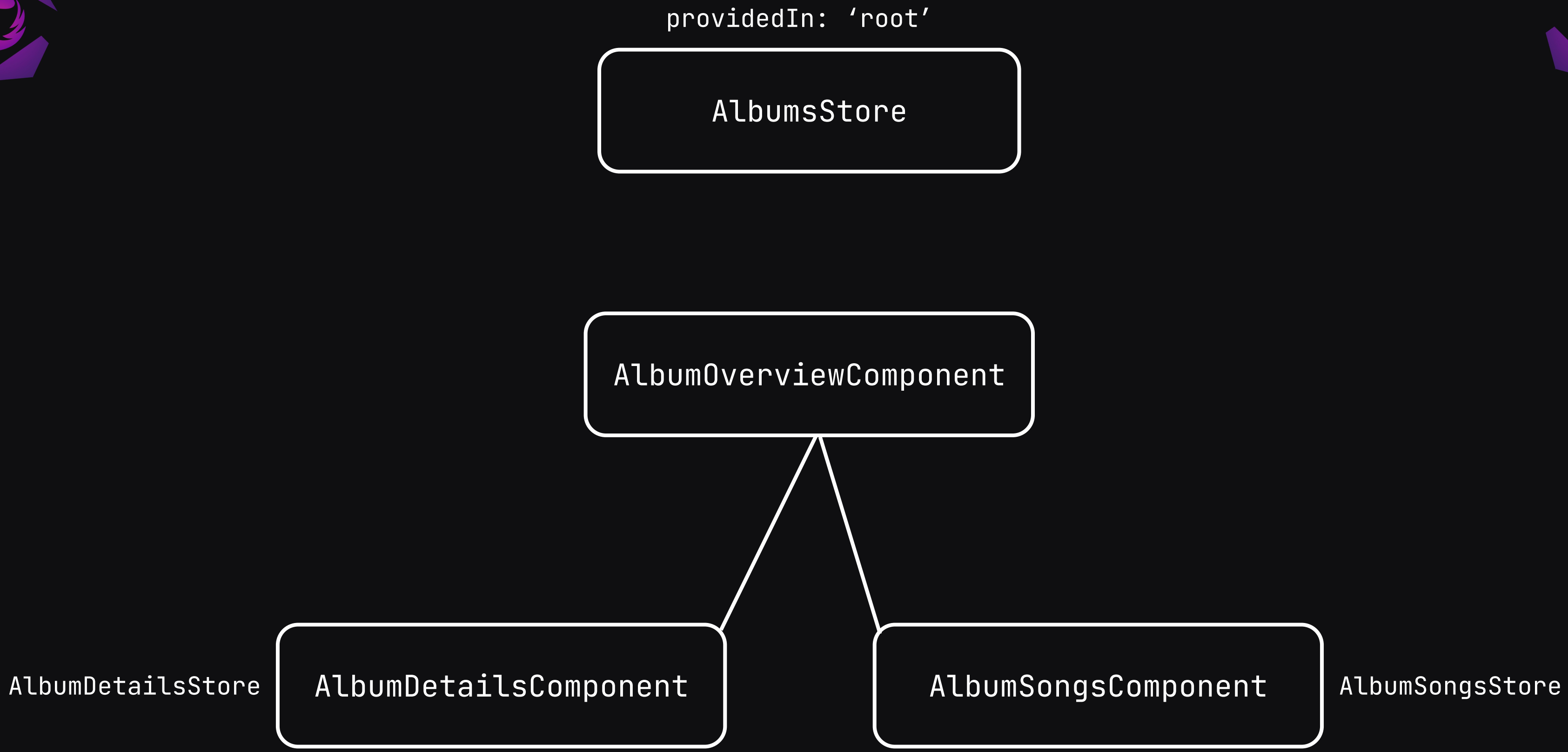

providedIn: ‘root’
AlbumsStore
AlbumOverviewComponent
AlbumDetailsStore
AlbumDetailsComponent
AlbumSongsComponent
AlbumSongsStore

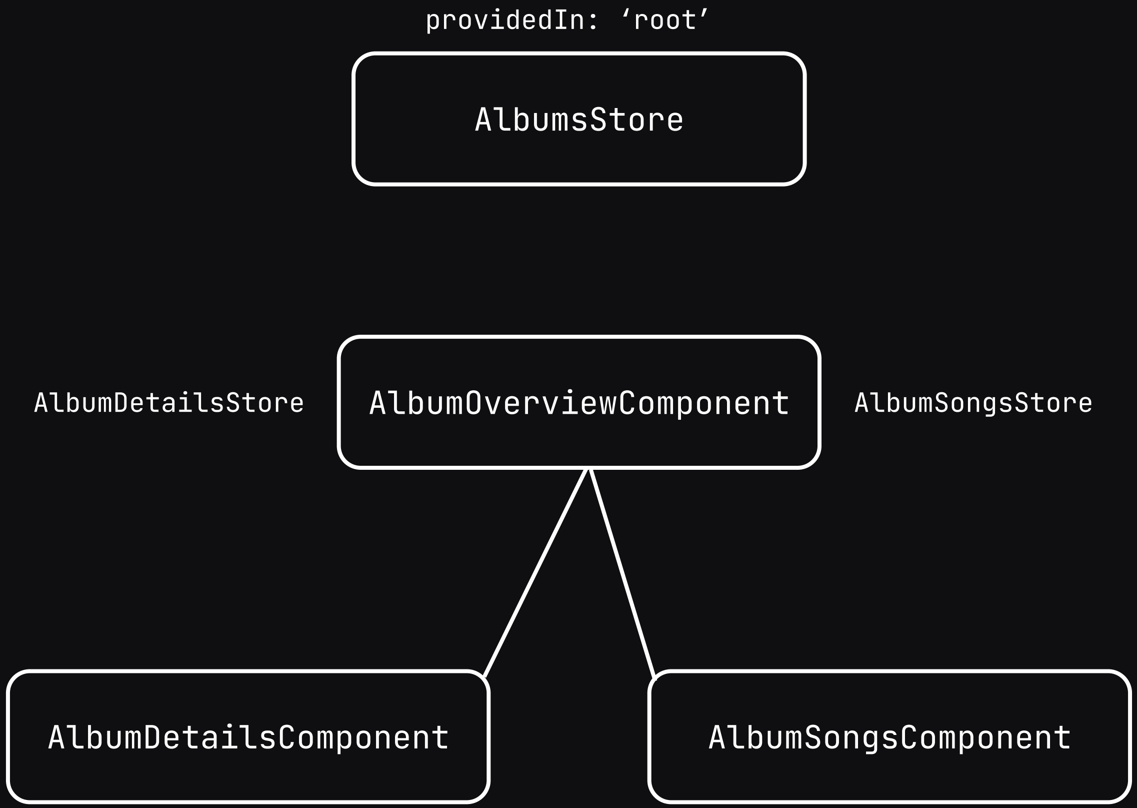

providedIn: 'root'
AlbumsStore
injects
AlbumOverviewComponent
injects
injects
AlbumDetailsStore
AlbumDetailsComponent
AlbumSongsComponent
AlbumSongsStore

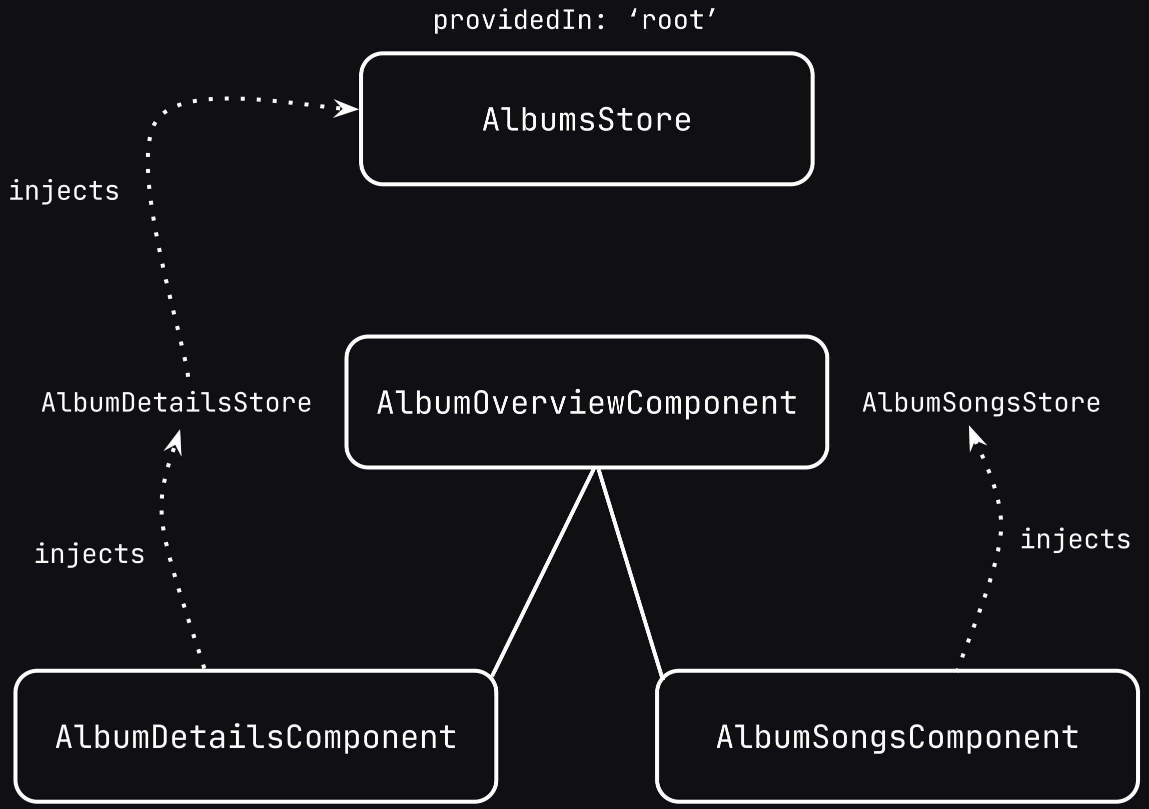
Milestone 08: Local State

1. Create `AlbumDetailsStore` that takes the `albumId` route parameter, injects `AlbumsStore`, and loads an album by id if it is not already loaded.
2. Provide `AlbumDetailsStore` at the `AlbumDetailsComponent` level and adjust the component template to consume signals from the store.
3. Create `AlbumSongsStore` that takes the `albumId` route parameter and loads songs by album id.
4. Provide `AlbumSongsStore` at the `AlbumSongsComponent` level and adjust the component template to consume signals from the store.

Branch State


providedIn: ‘root’
AlbumsStore
AlbumOverviewComponent
AlbumDetailsStore
AlbumDetailsComponent
AlbumSongsComponent
AlbumSongsStore


providedIn: ‘root’
AlbumsStore
AlbumDetailsStore
AlbumOverviewComponent
AlbumSongsStore
AlbumDetailsComponent
AlbumSongsComponent


providedIn: ‘root’
AlbumsStore
injects
AlbumDetailsStore
AlbumOverviewComponent
AlbumSongsStore
injects
injects
AlbumDetailsComponent
AlbumSongsComponent

Milestone 09: Branch State

1. Provide `AlbumDetailsStore` and `AlbumSongsStore` at the `AlbumOverviewComponent` level.
2. Inject both stores in the `AlbumOverviewComponent` and create the `showProgress` computed signal by combining `isPending` signals from these stores.
3. Adjust the template to consume the `showProgress` signal.
4. Implement `goToNextAlbum` and `goToPreviousAlbum` methods in the `AlbumOverviewComponent`.
   💡 Inject `Router` and use `router.navigate` to implement these methods.

Store vs ComponentStore vs SignalStore

Store vs ComponentStore vs SignalStore

- Store and ComponentStore also have integration with Angular Signals.

Store vs ComponentStore vs SignalStore

- Store and ComponentStore also have integration with Angular Signals.
- SignalStore is the successor of ComponentStore.

Store vs ComponentStore vs SignalStore

- Store and ComponentStore also have integration with Angular Signals.
- SignalStore is the successor of ComponentStore.
- NgRx Store is still a great choice for global state management if the Redux pattern is preferred.

Store vs ComponentStore vs SignalStore

- Store and ComponentStore also have integration with Angular Signals.
- SignalStore is the successor of ComponentStore.
- NgRx Store is still a great choice for global state management if the Redux pattern is preferred.
- SignalStore can be used to manage both local and global state.

Twitter: @ngrx_io

LinkedIn: NgRx

Discord: discord.gg/ngrx

Docs: ngrx.io

Blog: dev.to/ngrx

Workshops: ti.to/ngrx

GitHub: github.com/ngrx/platform