

A Redux-like Pattern for Managing State with Angular signals



Deborah Kurata
12.9K subscribers

Subscribed

11

1

Share

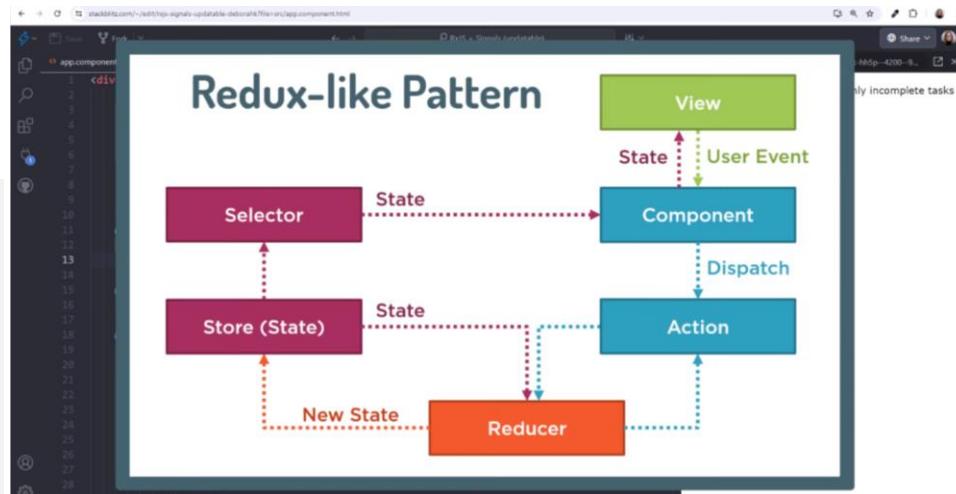
Download

45 views Jun 19, 2024 #angular #angulsarsignals #angularstatemanagementwithsignals

Now that Angular signals have been out for a while, there are patterns emerging.

In this video, we'll walk through one such pattern for managing state with signals. This Redux-like pattern leverages the best of signals and RxJS.

00:00 Redux-like pattern
00:20 Sample application
01:38 Define the state
02:23 Set the initial state
03:09 Define the selectors
04:16 Identify the actions
04:56 Define a Subject
05:55 Creating the Observable pipeline
08:59 Defining reducers
12:30 Accessing state from the component
12:48 Kicking off actions from the component
13:35 Following the pattern (again)
16:13 Modifying the component
17:13 Wrap up



The screenshot shows the StackBlitz editor with the file `app.component.html`. The code implements the Redux-like pattern:

```
<div class='container'>
  <select (change)="onSelected($event.target)">
    <option value="">--Select a team member--</option>
    @for(user of users()); track user.id {
      <option [value]='user.id'>{{ user.name }}</option>
    }
  </select>
  <input type='checkbox' (change)="onFilter($event.target)">Show only incomplete tasks

  <h2>ToDos</h2>
  @if (currentMember()) {
    for: {{ currentMember()?.name }}
  }
  </h2>
  @if (isLoading()) {
    <div>Loading ...</div>
  }
  @if (todosForMember().length > 0) {
    <div class='grid'>
      <div class='cellHeader'>Todo</div>
      <div class='cellHeader'>Completed?</div>

      @for(t of todosForMember(); track t.id) {
        <div class='cell'>{{ t.title }}</div>
        <div class='cellCheck'>
          <input type='checkbox' [checked]='t.completed'
            (change)='onChangeStatus(t, $event.target)'>
        </div>
      } @empty {
    </div>
  }
</div>
```

VS Code interface showing the `app.component.html` file. The status bar at the bottom right displays the path `/projects/rxjs-signals-redux-deborahk`.

```

<div class='container'>
  <select (change)="onSelected($event.target)">
    <option value="">-Select a team member--</option>
    @for(user of users()); track user.id {
      <option [value]='user.id'>{{ user.name }}</option>
    }
  </select>
  <input type='checkbox' (change)='onFilter($event.target)'>Show only incomplete tasks

```

VS Code interface showing the `app.component.html` file. A dropdown menu in the status bar lists team members:

- Select a team member-
- Selected a team member - Leanne Graham
- Ervin Howell
- Clementine Bauch
- Patricia Lebsack
- Chelsey Dietrich
- Mrs. Dennis Schulist
- Kurtis Weissnat
- Nicholas Runolfsdottir V
- Glenne Reichert
- Clementina DuBuque

VS Code interface showing the `app.component.html` file. A modal window titled "Todos for: Leanne Graham" is displayed, showing the message "Loading ...".

Leanne Graham

ToDos for: Leanne Graham

Todo	Completed?
delectus aut autem quis ut nam facilis fugiat veniam minus	<input type="checkbox"/>
et porro tempora laboriosam mollitia qui ullam ratione qu	<input checked="" type="checkbox"/>
illo expedita conseq quo adipisci enim qu	<input type="checkbox"/>
molestiae persipciat illo est ratione dol	<input type="checkbox"/>
vero rerum temporib ipsa repellendus fug	<input checked="" type="checkbox"/>
et doloremque nulla repellendus sunt dol	<input type="checkbox"/>
ab voluptatum amet v accusamus eos facili	<input checked="" type="checkbox"/>
quo laboriosam delen dolorum est consequa	<input checked="" type="checkbox"/>
molestiae ipsa aut v ullam nobis libero s	<input checked="" type="checkbox"/>

Leanne Graham

ToDos for: Leanne Graham

Todo	Completed?
fugiat veniam minus	<input type="checkbox"/>
laboriosam mollitia	<input type="checkbox"/>
qui ullam ratione qu	<input type="checkbox"/>
illo expedita conseq	<input type="checkbox"/>
molestiae persipciat	<input type="checkbox"/>
et doloremque nulla	<input type="checkbox"/>
dolorum est consequa	<input type="checkbox"/>

Now let us start from the beginning and code this app

The screenshot shows a code editor with three tabs: `app.component.html`, `app.component.ts`, and `todo.service.ts`. The `app.component.html` tab is active, displaying the following template code:

```
1 <div class='container'>
2   <select (change)="onSelected($event.target)">
3     <option value="">--Select a team member--</option>
4     @for(user of users(); track user.id) {
5       <option [value]:'user.id'>{{ user.name }}</option>
6     }
7   </select>
8   <input type='checkbox' (change)="onFilter($event.target)">Show only incomplete tasks
9
10  <h2>Todos
11  @if (!currentMember()) {
12    for: {{ currentMember()?.name}}
13  }
14  </h2>
15  @if (isLoading()) {
16    <div>Loading ...</div>
17  }
18  @if (todosForMember().length > 0) {
19    <div class='grid'>
20      <div class='cellHeader'>Todo</div>
21      <div class='cellHeader'>Completed?</div>
22
23      @for(t of todosForMember(); track t.id) {
24        <div class='cell'>{{ t.title }}</div>
25        <div class='cellCheck'>
26          <input type='checkbox' [checked]='t.completed'
27            (change)="onChangeStatus(t, $event.target)">
28        </div>
29      } @empty {
```

The screenshot shows the same code editor environment. A dropdown menu has been opened over the first `<option>` element in the `<select>` tag. The menu contains the following list of names:

- Leanne Graham
- Ervin Howell
- Clementine Bauch
- Patricia Lebsack
- Chelsey Dietrich
- Mrs. Dennis Schulist
- Kurtis Weissnat
- Nicholas Runolfsdotir V
- Glenna Reichert
- Clementine DuBuque

```
app.component.html
1 <div class='container'>
2   <select (change)="onSelected($event.target)">
3     <option value="">-Select a team member--</option>
4     @for(user of users(); track user.id) {
5       <option [value]=`user.id`>{{ user.name }}</option>
6     }
7   </select>
8   <input type='checkbox' (change)="onFilter($event.target)">Show only incomplete tasks
9
10  <h2>ToDos
11  @if (!currentMember()) {
12    for: {{ currentMember()?.name}}
13  }
14  </h2>
15  @if (isLoading()) {
16    <div>Loading ...</div>
17  }
18  @if (todosForMember().length > 0) {
19    <div class='grid'>
20      <div class='cellHeader'>Todo</div>
21      <div class='cellHeader'>Completed?</div>
22
23      @for(t of todosForMember(); track t.id) {
24        <div class='cell'>{{ t.title }}</div>
25        <div class='cellCheck'>
26          <input type='checkbox' [checked]=`t.completed`>
27          (change)="onChangeStatus(t, $event.target)">
28        </div>
29      } @empty {

```

```
app.component.ts
5 @Component({
6   selector: 'app-root',
7   standalone: true,
8   templateUrl: 'app.component.html'
9 })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = signal(false);
20   currentMember = signal<User | undefined>(undefined);
21   todosForMember = signal<ToDo[]>([[]]);
22   errorMessage = signal(null);
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26   }
27
28   onSelected(ele:EventTarget | null) {
29   }
30
31   onChangeStatus(task: ToDo, ele: EventTarget | null) {
32   }

```

```
app.component.html
1 <div class='container'>
2   <select (change)="onSelected($event.target)">
3     <option value="">-Select a team member--</option>
4     <option value="Leanne Graham">Leanne Graham</option>
5   </select>
6   <input type='checkbox' (change)="onFilter($event.target)" checked="checked">Show only incomplete tasks
7
8   <h2>ToDos
9
10  @if (!currentMember()) {
11    for: {{ currentMember()?.name}}
12  }
13  </h2>
14  @if (isLoading()) {
15    <div>Loading ...</div>
16  }
17  @if (todosForMember().length > 0) {
18    <div class='grid'>
19      <div class='cellHeader'>Todo</div>
20      <div class='cellHeader'>Completed?</div>
21
22      @for(t of todosForMember(); track t.id) {
23        <div class='cell'>{{ t.title }}</div>
24        <div class='cellCheck'>
25          <input type='checkbox' [checked]=`t.completed`>
26          (change)="onChangeStatus(t, $event.target)">
27        </div>
28      } @empty {

```

```
app.component.ts
5 @Component({
6   selector: 'app-root',
7   standalone: true,
8   templateUrl: 'app.component.html'
9 })
10 export class App {
11   name = 'Angular';
12
13   // Signals
14   users = this.userService.members;
15   isLoading = signal(false);
16   currentMember = signal<User | undefined>(undefined);
17   todosForMember = signal<ToDo[]>([[]]);
18   errorMessage = signal(null);
19
20   // Actions
21   onFilter(ele:EventTarget | null) {
22   }
23
24   onSelected(ele:EventTarget | null) {
25   }
26
27   onChangeStatus(task: ToDo, ele: EventTarget | null) {
28   }

```

```
import { HttpClient, HttpHeadersResponse } from '@angular/common/http';
import { Injectable, computed, inject, signal } from '@angular/core';
import { takeUntilDestroyed } from '@angular/core/rxjs-interop';
import { Observable, Subject, catchError, delay, filter, map, tap, of, switchMap } from 'rxjs';
import { User, UserService } from './user.service';

@Injectable({
  providedIn: 'root'
})
export class TodoService {
  todoUrl = 'https://jsonplaceholder.typicode.com/todos';

  // Services
  private http = inject(HttpClient);
  private userService = inject(UserService);

  constructor() {}

  export interface ToDo {
    userId: number;
    id: number;
    title: string;
    completed: boolean;
  }
}
```

```
import { HttpClient, HttpHeadersResponse } from '@angular/common/http';
import { Injectable, computed, inject, signal } from '@angular/core';
import { takeUntilDestroyed } from '@angular/core/rxjs-interop';
import { Observable, Subject, catchError, delay, filter, map, tap, of, switchMap } from 'rxjs';
import { User, UserService } from './user.service';

@Injectable({
  providedIn: 'root'
})
export class TodoService {
  todoUrl = 'https://jsonplaceholder.typicode.com/todos';

  // Services
  private http = inject(HttpClient);
  private userService = inject(UserService);

  constructor() {}

  export interface ToDo {
    userId: number;
    id: number;
    title: string;
    completed: boolean;
  }
}
```

What state or data does our ToDo service need to manage?

```

10 export class TodoService {
11   todoUrl = 'https://jsonplaceholder.typicode.com/todos';
12
13   // Services
14   private http = inject(HttpClient);
15   private userService = inject(UserService);
16
17   constructor() {
18   }
19
20 }
21
22 export interface ToDo {
23   userId: number;
24   id: number;
25   title: string;
26   completed: boolean;
27 }
28
29 export interface ToDoState {
30   isLoading: boolean;
31   currentMember: User | undefined;
32   memberToDos: ToDo[];
33   error: string | null;
34 }
35
36
37
38

```

```

10 export class TodoService {
11   todoUrl = 'https://jsonplaceholder.typicode.com/todos';
12
13   // Services
14   private http = inject(HttpClient);
15   private userService = inject(UserService);
16
17   private state = signal<ToDoState>({
18     isLoading: false,
19     currentMember: undefined,
20     memberToDos: [],
21     error: null
22   });
23
24   constructor() {
25   }
26
27 }
28
29 export interface ToDo {
30   userId: number;
31   id: number;
32   title: string;
33   completed: boolean;
34 }
35
36 export interface ToDoState {
37   isLoading: boolean;
38 }

```

```

10 export class TodoService {
11   todoUrl = 'https://jsonplaceholder.typicode.com/todos';
12
13   // Services
14   private http = inject(HttpClient);
15   private userService = inject(UserService);
16
17   private state = signal<ToDoState>({
18     isLoading: false,
19     currentMember: undefined,
20     memberToDos: [],
21     error: null
22   });
23
24   // Selectors
25   isLoading = computed(() => this.state().isLoading);
26   currentMember = computed(() => this.state().currentMember);
27   toDos = computed(() => this.state().memberToDos);
28   errorMessage = computed(() => this.state().error);
29
30   constructor() {
31   }
32
33 }
34
35
36 export interface ToDo {
37   userId: number;
38   id: number;
39 }

```

The diagram illustrates the Redux architecture with the following components and their interactions:

- View**: Represented by a green box.
- Component**: Represented by a blue box.
- Action**: Represented by a teal box.
- Reducer**: Represented by an orange box.
- Store (State)**: Represented by a purple box.
- Selector**: Represented by a yellow box.

The flow of data is as follows:

- The **View** sends a **User Event** to the **Component**.
- The **Component** dispatches an **Action** to the **Reducer**.
- The **Reducer** produces a **New State**.
- The **Store (State)** receives the **New State** and updates its internal state.
- The **Selector** retrieves the **State** from the **Store (State)**.
- The **Selector** provides the **State** back to the **Component**.

A Selector selects a specific piece or slice of our state as a computed signal as seen above.

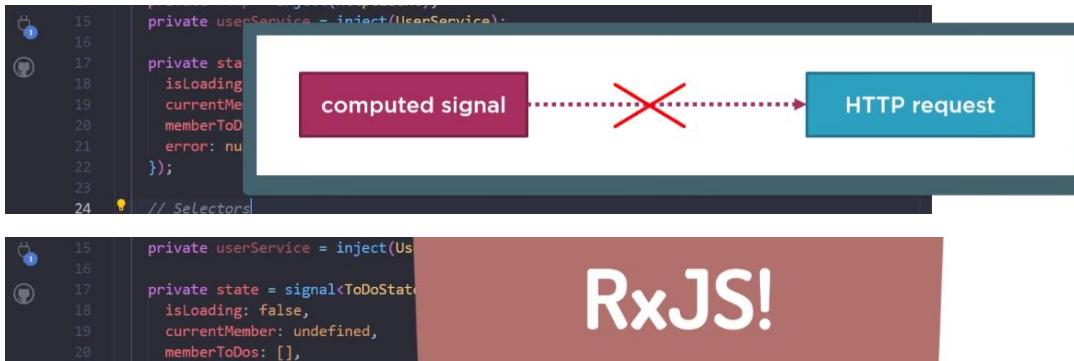
The screenshot shows a code editor on the left with a file named `todo.service.ts` open. The code defines a service with methods for getting todos from a placeholder URL and managing todo states. A yellow arrow points from the code editor to a browser window on the right. The browser displays a dropdown menu with the placeholder text "Select a team member".

```

10 export class TodoService {
11   todoUrl = 'https://jsonplaceholder.typicode.com/todos';
12
13   // Services
14   private http = inject(HttpClient);
15   private userService = inject(UserService);
16
17   private state = signal<ToDoState>({
18     isLoading: false,
19     currentMember: undefined,
20     memberTodos: [],
21     error: null
22   });
23
24   // Selectors
25   isLoading = computed(() => this.state().isLoading);
26   currentMember = computed(() => this.state().currentMember);
27   todos = computed(() => this.state().memberTodos);
28   errorMessage = computed(() => this.state().error);
29
30   constructor() {
31   }
32
33 }
34
35 export interface ToDo {
36   userId: number;
37   id: number;
38 }

```

Next is our actions, starting with the action of selecting a team member from the dropdown box



Anytime we need to react to an action by performing an async operation, we use RxJS

The screenshot shows a code editor on the left with `todo.service.ts` open. A yellow arrow points to the `selectedId$` subject definition. The browser window on the right shows a dropdown menu with the placeholder text "Select a team member".

```

17 private state = signal<ToDoState>({
18   isLoading: false,
19   currentMember: undefined,
20   memberTodos: [],
21   error: null
22 });
23
24 // Selectors
25 isLoading = computed(() => this.state().isLoading);
26 currentMember = computed(() => this.state().currentMember);
27 todos = computed(() => this.state().memberTodos);
28 errorMessage = computed(() => this.state().error);
29
30 private selectedIdSubject = new Subject<number>();
31 private selectedId$ = this.selectedIdSubject.asObservable();
32
33 constructor() {
34   this.selectedId$.pipe(
35     ...
36   );
37 }
38
39 getTodosForMember(memberId: number) {
40   this.selectedIdSubject.next(memberId);
41 }
42
43 export interface ToDo {
44   userId: number;
45 }

```

app.component.html TS app.component.ts TS todo.service.ts 2

```
17 private state = signal<ToDoState>({
18   isLoading: false,
19   currentMember: undefined,
20   memberToDos: [],
21   error: null
22 });
23
24 // Selectors
25 isLoading = computed(() => this.state().isLoading);
26 currentMember = computed(() => this.state().currentMember);
27 toDos = computed(() => this.state().memberToDos);
28 errorMessage = computed(() => this.state().error);
29
30 private selectedIdSubject = new Subject<number>();
31 private selectedId$ = this.selectedIdSubject.asObservable();
32
33 constructor() {
34   this.selectedId$.pipe(
35     tap(() => this.setLoadingIndicator(true)),
36     tap(id => this.setCurrentMember(id))
37   );
38 }
39
40 getTodosForMember(memberId: number) {
41   this.selectedIdSubject.next(memberId);
42 }
43
44 export interface ToDo {
45 }
```

TS2339: Property 'setLoadingIndicator' does not exist on type 'ToDo'.
src/todo.service.ts:35:23

Click outside, press Esc key, or fix the code to dismiss



app.component.html TS app.component.ts TS todo.service.ts 4

```
17 private state = signal<ToDoState>({
18   isLoading: false,
19   currentMember: undefined,
20   memberToDos: [],
21   error: null
22 });
23
24 // Selectors
25 isLoading = computed(() => this.state().isLoading);
26 currentMember = computed(() => this.state().currentMember);
27 toDos = computed(() => this.state().memberToDos);
28 errorMessage = computed(() => this.state().error);
29
30 private selectedIdSubject = new Subject<number>();
31 private selectedId$ = this.selectedIdSubject.asObservable();
32
33 constructor() {
34   this.selectedId$.pipe(
35     tap(() => this.setLoadingIndicator(true)),
36     tap(id => this.setCurrentMember(id)),
37     switchMap(id => this.getTodos(id)),
38     delay(1000),
39     takeUntilDestroyed(),
40   ).subscribe(todos => this.setMemberToDos(todos));
41 }
42
43 getTodosForMember(memberId: number) {
44   this.selectedIdSubject.next(memberId);
45 }
```

TS2339: Property 'setLoadingIndicator' does not exist on type 'ToDo'.
src/todo.service.ts:35:23

Click outside, press Esc key, or fix the code to dismiss

```

17  private state = sign
18    isLoading: false,
19    currentMember: unc
20    memberToDos: [],
21    error: null
22  );
23
24  // Selectors
25  isLoading = computed(
26    currentMember = comp
27    todos = computed()
28    errorMessage = comput
29
30  private selectedIdsSub
31  private selectedId$Sub
32
33  constructor() {
34

```

```

33  constructor() {
34    this.selectedId$.pipe(
35      tap(() => this.setLoadingIndicator(
36        tap(id => this.setCurrentMember(id)),
37        switchMap(id => this.getTodos(id)),
38        delay(1000),
39        takeUntilDestroyed(),
40      ).subscribe(todos => this.setMemberTodos(todos));
41    )
42
43    private setLoadingIndicator(isLoading: boolean) {
44      this.state.update(state => ({
45        ...state,
46        isLoading: isLoading
47      }));
48    }
49
50    getTodosForMember(memberId: number) {
51      this.selectedIdSubject.next(memberId);
52    }
53
54    export interface ToDo {
55      userId: number;
56      id: number;
57      title: string;
58      completed: boolean;
59    }
60  }
61

```

A Reducer defines how an action should update state

```

33  constructor() {
34    this.selectedId$.pipe(
35      tap(() => this.setLoadingIndicator(true)),
36      tap(id => this.setCurrentMember(id)),
37      switchMap(id => this.getTodos(id)),
38      delay(1000),
39      takeUntilDestroyed(),
40    ).subscribe(todos => this.setMemberTodos(todos));
41  }
42
43  private setLoadingIndicator(isLoading: boolean) {
44    this.state.update(state => ({
45      ...state,
46      isLoading: isLoading
47    }));
48  }
49  private setCurrentMember(id: number) {
50    const member = this.userService.getCurrentMember(id);
51    this.state.update(state => ({
52      ...state,
53      currentMember: member,
54      memberTodos: []
55    }));
56  }
57
58  getTodosForMember(memberId: number) {
59    this.selectedIdSubject.next(memberId);
60  }

```

app.component.html app.components.ts ts todo.service.ts

```
46     isLoading: isLoading
47   );
48
49   private setCurrentMember(id: number) {
50     const member = this.userService.getCurrentMember(id);
51     this.state.update(state => ({
52       ...state,
53       currentMember: member,
54       memberToDos: []
55     }));
56   }
57
58   private getTodos(id: number): Observable<ToDo[]> {
59     return this.http.get<ToDo[]>(`${this.todoUrl}?userId=${id}`).pipe(
60       // Cut the length of the long strings
61       map(data => data.map(t =>
62         t.title.length > 20 ? ({...t, title: t.title.substring(0, 20)}) : t
63       )),
64       catchError(err => this.setError(err))
65     );
66   }
67   private setError(err: HttpErrorResponse): Observable<ToDo[]> {
68     this.state.update(state => ({
69       ...state,
70       error: setErrorMessage(err)
71     }));
72     return of([]);
73   }

```

Output in terminal window: Show only incomplete tasks

TS2339: Property 'setMemberToDos' does not exist in type 'ToDoState'. src/todo.service.ts:40:32

Click outside, press Esc key, or fix the code to dismiss

app.component.html app.component.ts ts todo.service.ts

```
76   getTodosForMember(memberId: number) {
77     this.selectedIdSubject.next(memberId);
78   }
79
80   export interface ToDo {
81     userId: number;
82     id: number;
83     title: string;
84     completed: boolean;
85   }
86
87   export interface ToDoState {
88     isLoading: boolean;
89     currentMember: User | undefined;
90     memberToDos: ToDo[];
91     error: string | null;
92   }
93
94
95   // This should be somewhere reusable
96   export function setErrorMessage(err: HttpErrorResponse): string {
97     let errorMessage: string;
98     if (err.error instanceof ErrorEvent) {
99       // A client-side or network error occurred. Handle it accordingly.
100      errorMessage = `An error occurred: ${err.error.message}`;
101    } else {
102      // The backend returned an unsuccessful response code.
103      // The response body may contain clues as to what went wrong,
104      errorMessage = `Backend returned code ${err.status}: ${err.message}`;
105    }
106    console.error(err);
107    return errorMessage;
108  }
109
```

Output in terminal window: Show only incomplete tasks

TS2339: Property 'setMemberToDos' does not exist in type 'ToDoState'. src/todo.service.ts:40:32

Click outside, press Esc key, or fix the code to dismiss

app.component.html app.component.ts ts todo.service.ts

```
86
87   export interface ToDoState {
88     isLoading: boolean;
89     currentMember: User | undefined;
90     memberToDos: ToDo[];
91     error: string | null;
92   }
93
94
95   // This should be somewhere reusable
96   export function setErrorMessage(err: HttpErrorResponse): string {
97     let errorMessage: string;
98     if (err.error instanceof ErrorEvent) {
99       // A client-side or network error occurred. Handle it accordingly.
100      errorMessage = `An error occurred: ${err.error.message}`;
101    } else {
102      // The backend returned an unsuccessful response code.
103      // The response body may contain clues as to what went wrong,
104      errorMessage = `Backend returned code ${err.status}: ${err.message}`;
105    }
106    console.error(err);
107    return errorMessage;
108  }
109
```

Output in terminal window: Show only incomplete tasks

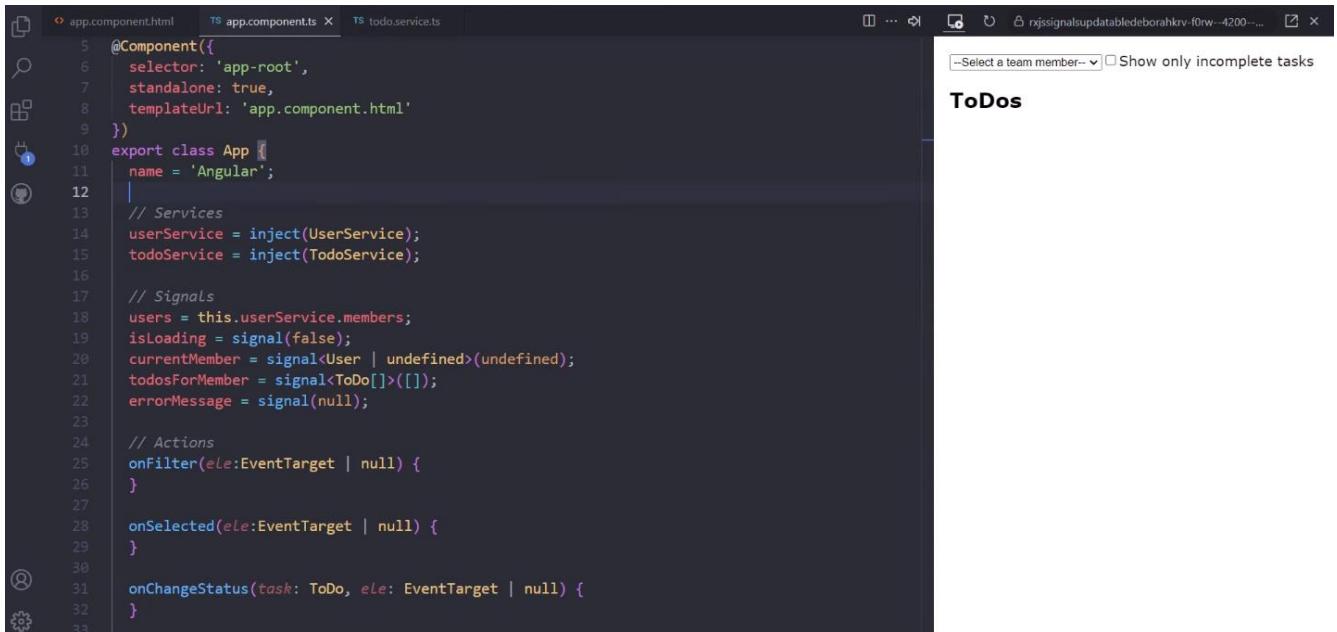
TS2339: Property 'setMemberToDos' does not exist in type 'ToDoState'. src/todo.service.ts:40:32

Click outside, press Esc key, or fix the code to dismiss

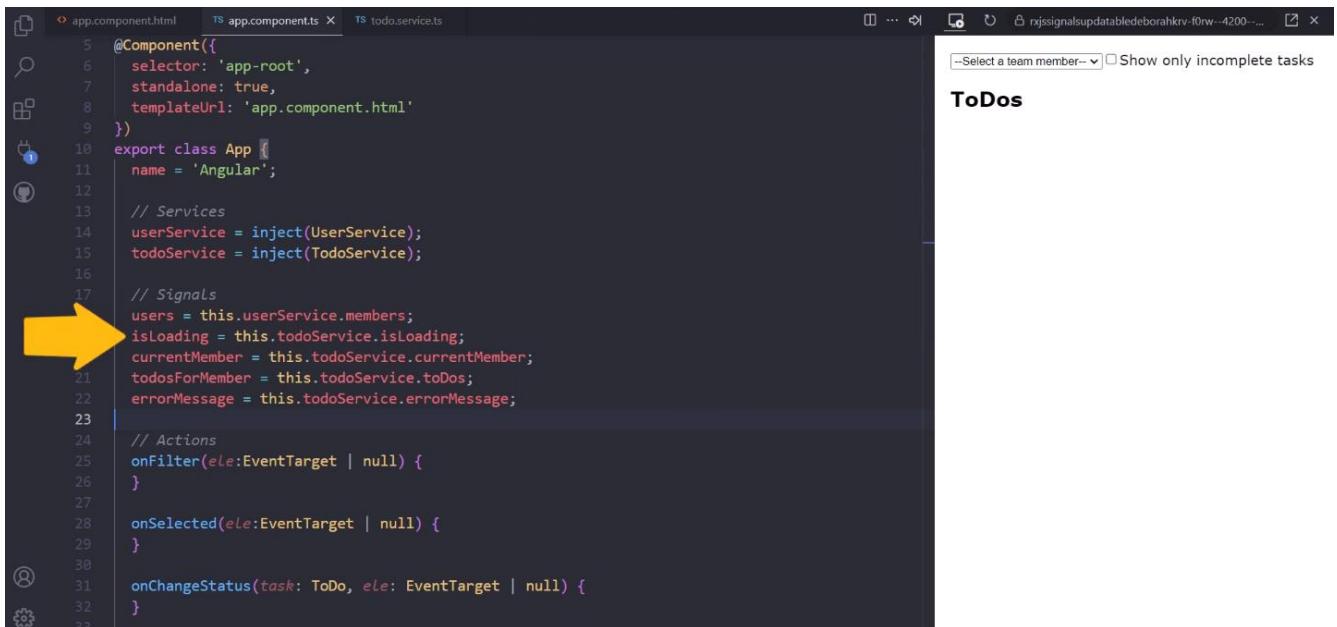
```
54     memberToDos: []
55   });
56 }
57 private getTodos(id: number): Observable<ToDo[]> {
58   return this.http.get<ToDo[]>(`${this.todoUrl}?userId=${id}`).pipe(
59     // Cut the length of the long strings
60     map(data => data.map(t =>
61       t.title.length > 20 ? ({...t, title: t.title.substring(0, 20)}) : t
62     )),
63     catchError(err => this.setError(err))
64   );
65 }
66 private setError(err: HttpErrorResponse): Observable<ToDo[]> {
67   this.state.update(state => ({
68     ...state,
69     error: setErrorMessage(err)
70   }));
71   return of([]);
72 }
73 private setMemberToDos(todos: ToDo[]): void {
74   this.state.update(state => ({
75     ...state,
76     memberToDos: todos,
77     isLoading: false
78   }));
79 }
80
```

```
33 constructor() {
34   this.selectedId$.pipe(
35     tap(() => this.setLoadingIndicator(true)),
36     tap(id => this.setCurrentMember(id)),
37     switchMap(id => this.getTodos(id)),
38     delay(1000),
39     takeUntilDestroyed(),
40     .subscribe(todos => this.setMemberToDos(todos));
41 }
42
43 private setLoadingIndicator(isLoading: boolean) {
44   this.state.update(state => ({
45     ...state,
46     isLoading: isLoading
47   }));
48 }
49 private setCurrentMember(id: number) {
50   const member = this.userService.getCurrentMember(id);
51   this.state.update(state => ({
52     ...state,
53     currentMember: member,
54     memberToDos: []
55   }));
56 }
57 private getTodos(id: number): Observable<ToDo[]> {
58   return this.http.get<ToDo[]>(`${this.todoUrl}?userId=${id}`).pipe(
59     // Cut the length of the long strings
60     map(data => data.map(t =>
61       t.title.length > 20 ? ({...t, title: t.title.substring(0, 20)}) : t
62     )));
63 }
```

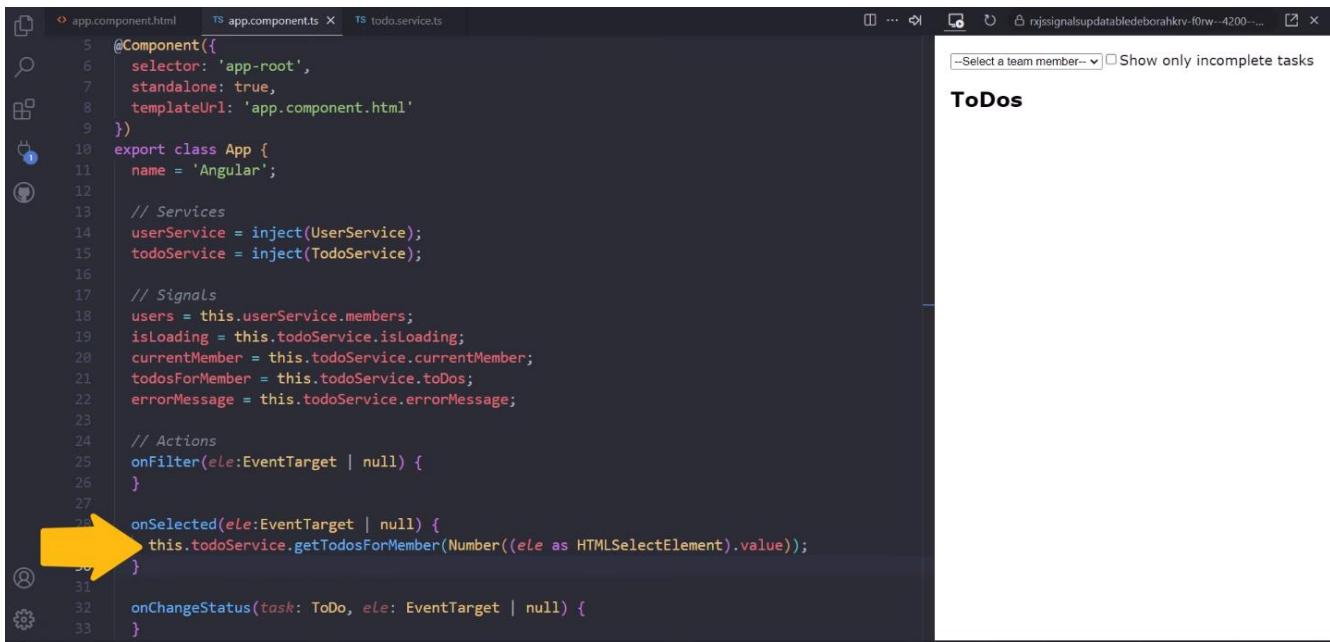
Next, let us modify the component to use our ToDo service



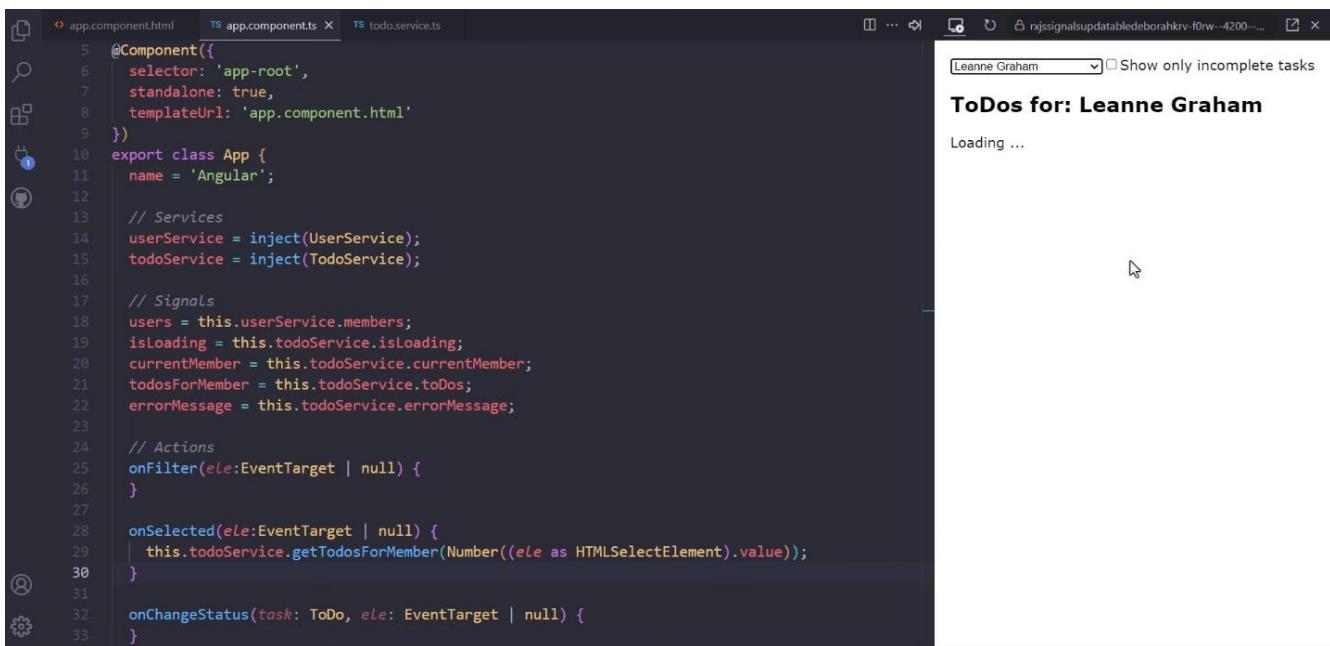
```
app.component.html    TS app.component.ts    TS todo.service.ts
5  @Component({
6    selector: 'app-root',
7    standalone: true,
8    templateUrl: 'app.component.html'
9  })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = signal(false);
20   currentMember = signal<User | undefined>(undefined);
21   todosForMember = signal<ToDo[]>([[]]);
22   errorMessage = signal(null);
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26   }
27
28   onSelected(ele:EventTarget | null) {
29   }
30
31   onChangeStatus(task: ToDo, ele: EventTarget | null) {
32   }
```



```
app.component.html    TS app.component.ts    TS todo.service.ts
5  @Component({
6    selector: 'app-root',
7    standalone: true,
8    templateUrl: 'app.component.html'
9  })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = this.todoService.isLoading;
20   currentMember = this.todoService.currentMember;
21   todosForMember = this.todoService.todos;
22   errorMessage = this.todoService.errorMessage;
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26   }
27
28   onSelected(ele:EventTarget | null) {
29   }
30
31   onChangeStatus(task: ToDo, ele: EventTarget | null) {
32   }
```



```
app.component.html    TS app.component.ts    TS todo.service.ts
5  @Component({
6    selector: 'app-root',
7    standalone: true,
8    templateUrl: 'app.component.html'
9  })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = this.todoService.isLoading;
20   currentMember = this.todoService.currentMember;
21   todosForMember = this.todoService.todos;
22   errorMessage = this.todoService.errorMessage;
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26   }
27
28   onSelected(ele:EventTarget | null) {
29     this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
30   }
31
32   onChangeStatus(task: ToDo, ele: EventTarget | null) {
33 }
```



--Select a team member-- Show only incomplete tasks

ToDos

Leanne Graham

ToDos for: Leanne Graham

Loading ...

```
app.component.html    TS app.component.ts    TS todo.service.ts
5  @Component({
6    selector: 'app-root',
7    standalone: true,
8    templateUrl: 'app.component.html'
9  })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = this.todoService.isLoading;
20   currentMember = this.todoService.currentMember;
21   todosForMember = this.todoService.todos;
22   errorMessage = this.todoService.errorMessage;
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26   }
27
28   onSelected(ele:EventTarget | null) {
29     this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
30   }
31
32   onChangeStatus(task: ToDo, ele: EventTarget | null) {
33 }
```

app.component.html app.component.ts todo.service.ts

```

5  @Component({
6    selector: 'app-root',
7    standalone: true,
8    templateUrl: 'app.component.html'
9  })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = this.todoService.isLoading;
20   currentMember = this.todoService.currentMember;
21   todosForMember = this.todoService.todos;
22   errorMessage = this.todoService.errorMessage;
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26   }
27
28   onSelected(ele:EventTarget | null) {
29     | this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
30   }
31
32   onChangeStatus(task: ToDo, ele: EventTarget | null) {
33   }

```

Clementine Bauch Show only incomplete tasks

ToDos for: Clementine Bauch

Todo	Completed?
aliquid amet impedit	<input type="checkbox"/>
rerum perferendis er	<input type="checkbox"/>
tempore ut sint quis	<input checked="" type="checkbox"/>
cum debitibus quis acc	<input checked="" type="checkbox"/>
velit soluta adipisc	<input type="checkbox"/>
vel voluptatem repel	<input type="checkbox"/>
nam qui rerum fugiat	<input type="checkbox"/>
sit reprehenderit om	<input type="checkbox"/>
ut necessitatibus au	<input type="checkbox"/>
cupiditate necessita	<input checked="" type="checkbox"/>
distinctio exercitat	<input type="checkbox"/>
nesciunt dolorum qui	<input type="checkbox"/>
qui labore est occae	<input type="checkbox"/>
quis et est ut volup	<input checked="" type="checkbox"/>
voluptatum omnis min	<input checked="" type="checkbox"/>
deleniti ea temporib	<input checked="" type="checkbox"/>
pariatur et magnam e	<input type="checkbox"/>
est dicta totam qui	<input type="checkbox"/>
perspiciatis velit i	<input type="checkbox"/>
et sequi qui archite	<input checked="" type="checkbox"/>

app.component.html app.component.ts todo.service.ts

```

5  @Component({
6    selector: 'app-root',
7    standalone: true,
8    templateUrl: 'app.component.html'
9  })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = this.todoService.isLoading;
20   currentMember = this.todoService.currentMember;
21   todosForMember = this.todoService.todos;
22   errorMessage = this.todoService.errorMessage;
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26   }
27
28   onSelected(ele:EventTarget | null) {
29     | this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
30   }
31
32   onChangeStatus(task: ToDo, ele: EventTarget | null) {
33   }

```

```

graph TD
    Selector[Selector] --> State[State]
    State --> Component[Component]
    Component --> View[View]
    View -- "User Event" --> State
    State --> Action[Action]
    Action -- Dispatch --> Reducer[Reducer]
    Reducer --> NewState[New State]
    NewState --> Store[Store State]
    Store --> Selector

```

Clementine Bauch Show only incomplete tasks

ToDos for: Clementine Bauch

Todo	Completed?
aliquid amet impedit	<input type="checkbox"/>
rerum perferendis er	<input type="checkbox"/>
tempore ut sint quis	<input checked="" type="checkbox"/>
cum debitibus quis acc	<input checked="" type="checkbox"/>
velit soluta adipisc	<input type="checkbox"/>
vel voluptatem repel	<input type="checkbox"/>
nam qui rerum fugiat	<input type="checkbox"/>
sit reprehenderit om	<input type="checkbox"/>
ut necessitatibus au	<input type="checkbox"/>
cupiditate necessita	<input checked="" type="checkbox"/>
distinctio exercitat	<input type="checkbox"/>
nesciunt dolorum qui	<input type="checkbox"/>
qui labore est occae	<input type="checkbox"/>
quis et est ut volup	<input checked="" type="checkbox"/>
voluptatum omnis min	<input checked="" type="checkbox"/>
deleniti ea temporib	<input checked="" type="checkbox"/>
pariatur et magnam e	<input type="checkbox"/>
est dicta totam qui	<input type="checkbox"/>
perspiciatis velit i	<input type="checkbox"/>
et sequi qui archite	<input checked="" type="checkbox"/>

File structure:

- app.component.html
- app.component.ts
- todo.service.ts

```

33  constructor() {
34    this.selectedId$.pipe(
35      tap(() => this.setLoadingIndicator(true)),
36      tap(id => this.setCurrentMember(id)),
37      switchMap(id => this.getTodos(id)),
38      delay(1000),
39      takeUntilDestroyed(),
40    ).subscribe(todos => this.setMemberTodos(todos));
41  }
42
43  private setLoadingIndicator(isLoading: boolean) {
44    this.state.update(state => ({
45      ...state,
46      isLoading: isLoading
47    }));
48  }
49  private setCurrentMember(id: number) {
50    const member = this.userService.getCurrentMember(id);
51    this.state.update(state => ({
52      ...state,
53      currentMember: member,
54      memberTodos: []
55    }));
56  }
57  private getTodos(id: number): Observable<ToDo[]> {
58    return this.http.get<ToDo[]>(` ${this.todoUrl}?userId=${id}`).pipe(
59      // Cut the length of the long strings
60      map(data => data.map(t =>
61        t.title.length > 20 ? { ...t, title: t.title.substring(0, 20)} : t
62      ))
63    );
64  }
65
66  private setMemberTodos(todos: ToDo[]): void {
67    this.state.update(state => ({
68      ...state,
69      memberTodos: todos,
70      isLoading: false
71    }));
72  }
73
74  getTodosForMember(memberId: number) {
75    this.selectedIdSubject.next(memberId);
76  }
77
78  export interface ToDo {
79    userId: number;
80    id: number;
81    title: string;
82    completed: boolean;
83  }
84
85  export interface ToDoState {
86    isLoading: boolean;
87    currentMember: User | undefined;
88    memberTodos: ToDo[];
89    incompleteOnly: boolean;
90    error: string | null;
91  }
92
93  
```

Output window:

Clementine Bauch Show only incomplete tasks

ToDos for: Clementine Bauch

Todo	Completed?
aliquid amet impedit	<input type="checkbox"/>
rerum perferendis er	<input type="checkbox"/>
tempore ut sint quis	<input checked="" type="checkbox"/>
cum debitis quis acc	<input checked="" type="checkbox"/>
velit soluta adipisc	<input type="checkbox"/>
vel voluptatem repel	<input type="checkbox"/>
nam qui rerum fugiat	<input type="checkbox"/>
sit reprehenderit om	<input type="checkbox"/>
ut necessitatibus au	<input type="checkbox"/>
cupiditate necessita	<input checked="" type="checkbox"/>
distinctio exercitat	<input type="checkbox"/>
nesciunt dolorum qui	<input type="checkbox"/>
qui labore est occae	<input type="checkbox"/>
quis et est ut volup	<input checked="" type="checkbox"/>
voluptatum omnis min	<input checked="" type="checkbox"/>
deleniti ea temporib	<input checked="" type="checkbox"/>
pariatur et magnam e	<input type="checkbox"/>
est dicta totam qui	<input type="checkbox"/>
perspicatis velit i	<input type="checkbox"/>
et sequi qui archite	<input checked="" type="checkbox"/>

File structure:

- app.component.html
- app.component.ts
- todo.service.ts

```

73  private setMemberTodos(todos: ToDo[]): void {
74    this.state.update(state => ({
75      ...state,
76      memberTodos: todos,
77      isLoading: false
78    }));
79  }
80
81
82  getTodosForMember(memberId: number) {
83    this.selectedIdSubject.next(memberId);
84  }
85
86
87  export interface ToDo {
88    userId: number;
89    id: number;
90    title: string;
91    completed: boolean;
92  }
93
94  export interface ToDoState {
95    isLoading: boolean;
96    currentMember: User | undefined;
97    memberTodos: ToDo[];
98    incompleteOnly: boolean;
99    error: string | null;
100   }
101
102  
```

Output window:

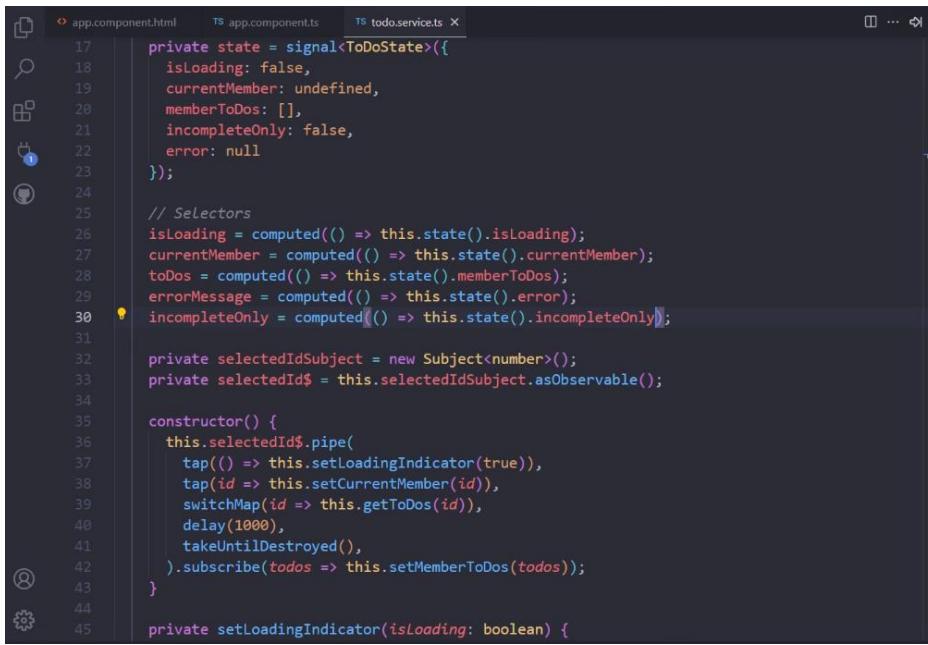
Clementine Bauch Show only incomplete tasks

ToDos for: Clementine Bauch

Todo	Completed?
aliquid amet impedit	<input type="checkbox"/>
rerum perferendis er	<input type="checkbox"/>
tempore ut sint quis	<input checked="" type="checkbox"/>
cum debitis quis acc	<input checked="" type="checkbox"/>
velit soluta adipisc	<input type="checkbox"/>
vel voluptatem repel	<input type="checkbox"/>
nam qui rerum fugiat	<input type="checkbox"/>
sit reprehenderit om	<input type="checkbox"/>
ut necessitatibus au	<input type="checkbox"/>
cupiditate necessita	<input checked="" type="checkbox"/>
distinctio exercitat	<input type="checkbox"/>
nesciunt dolorum qui	<input type="checkbox"/>
qui labore est occae	<input type="checkbox"/>
quis et est ut volup	<input checked="" type="checkbox"/>
voluptatum omnis min	<input checked="" type="checkbox"/>
deleniti ea temporib	<input checked="" type="checkbox"/>
pariatur et magnam e	<input type="checkbox"/>
est dicta totam qui	<input type="checkbox"/>
perspicatis velit i	<input type="checkbox"/>
et sequi qui archite	<input checked="" type="checkbox"/>



```
app.component.html      TS app.component.ts      TS todo.service.ts ×
17  private state = signal<ToDoState>({
18    isLoading: false,
19    currentMember: undefined,
20    memberToDos: [],
21    incompleteOnly: false,
22    error: null
23  });
24
25  // Selectors
26  isLoading = computed(() => this.state().isLoading);
27  currentMember = computed(() => this.state().currentMember);
28  toDos = computed(() => this.state().memberToDos);
29  errorMessage = computed(() => this.state().error);
30
31  private selectedIdSubject = new Subject<number>();
32  private selectedId$ = this.selectedIdSubject.asObservable();
33
34  constructor() {
35    this.selectedId$.pipe(
36      tap(() => this.setLoadingIndicator(true)),
37      tap(id => this.setCurrentMember(id)),
38      switchMap(id => this.getToDos(id)),
39      delay(1000),
40      takeUntilDestroyed(),
41    ).subscribe(todos => this.setMemberToDos(todos));
42  }
43
44  private setLoadingIndicator(isLoading: boolean) {
45    this.state.update(state => ({
```



```
app.component.html      TS app.component.ts      TS todo.service.ts ×
17  private state = signal<ToDoState>({
18    isLoading: false,
19    currentMember: undefined,
20    memberToDos: [],
21    incompleteOnly: false,
22    error: null
23  });
24
25  // Selectors
26  isLoading = computed(() => this.state().isLoading);
27  currentMember = computed(() => this.state().currentMember);
28  toDos = computed(() => this.state().memberToDos);
29  errorMessage = computed(() => this.state().error);
30  incompleteOnly = computed(() => this.state().incompleteOnly);
31
32  private selectedIdSubject = new Subject<number>();
33  private selectedId$ = this.selectedIdSubject.asObservable();
34
35  constructor() {
36    this.selectedId$.pipe(
37      tap(() => this.setLoadingIndicator(true)),
38      tap(id => this.setCurrentMember(id)),
39      switchMap(id => this.getToDos(id)),
40      delay(1000),
41      takeUntilDestroyed(),
42    ).subscribe(todos => this.setMemberToDos(todos));
43  }
44
45  private setLoadingIndicator(isLoading: boolean) {
```

app.component.html

app.component.ts

todo.service.ts

```
private state = signal<ToDoState>({
  isLoading: false,
  currentMember: undefined,
  memberToDos: [],
  incompleteOnly: false,
  error: null
});

// Selectors
isLoading = computed(() => this.state().isLoading);
currentMember = computed(() => this.state().currentMember);
toDos = computed(() => this.state().memberToDos);
errorMessage = computed(() => this.state().error);
incompleteOnly = computed(() => this.state().incompleteOnly);
filteredToDos = computed(() => {
  if (this.incompleteOnly()) {
    return this.toDos().filter(t => t.completed === false);
  }
  else {
    return this.toDos();
  }
});

private selectedIdSubject = new Subject<number>();
private selectedId$ = this.selectedIdSubject.asObservable();

constructor() {
  this.selectedId$.pipe(
    tap(() => this.setLoadingIndicator(true)),
    
```

app.component.html

app.component.ts

todo.service.ts

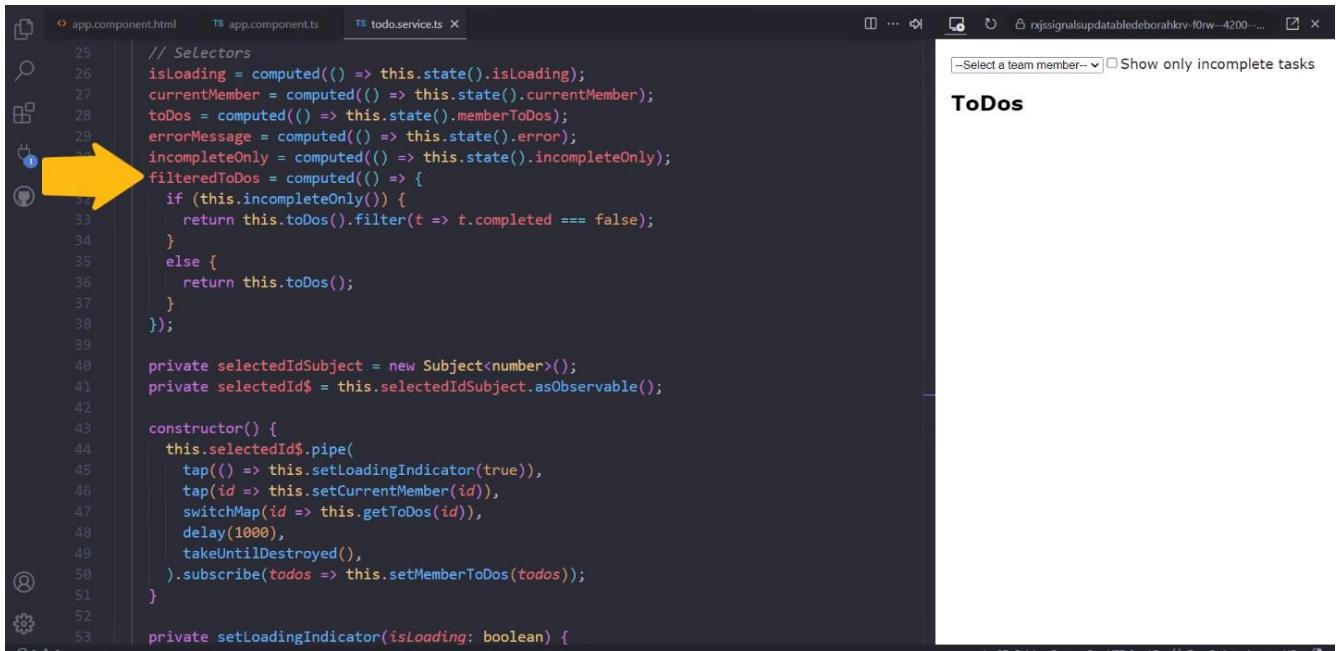
```
private setMemberToDos(todos: ToDo[]): void {
  this.state.update(state => ({
    ...state,
    memberToDos: todos,
    isLoading: false
  }));
}

filterTodos(filter: boolean) {
  this.state.update(state => ({
    ...state,
    incompleteOnly: filter
  }));
}

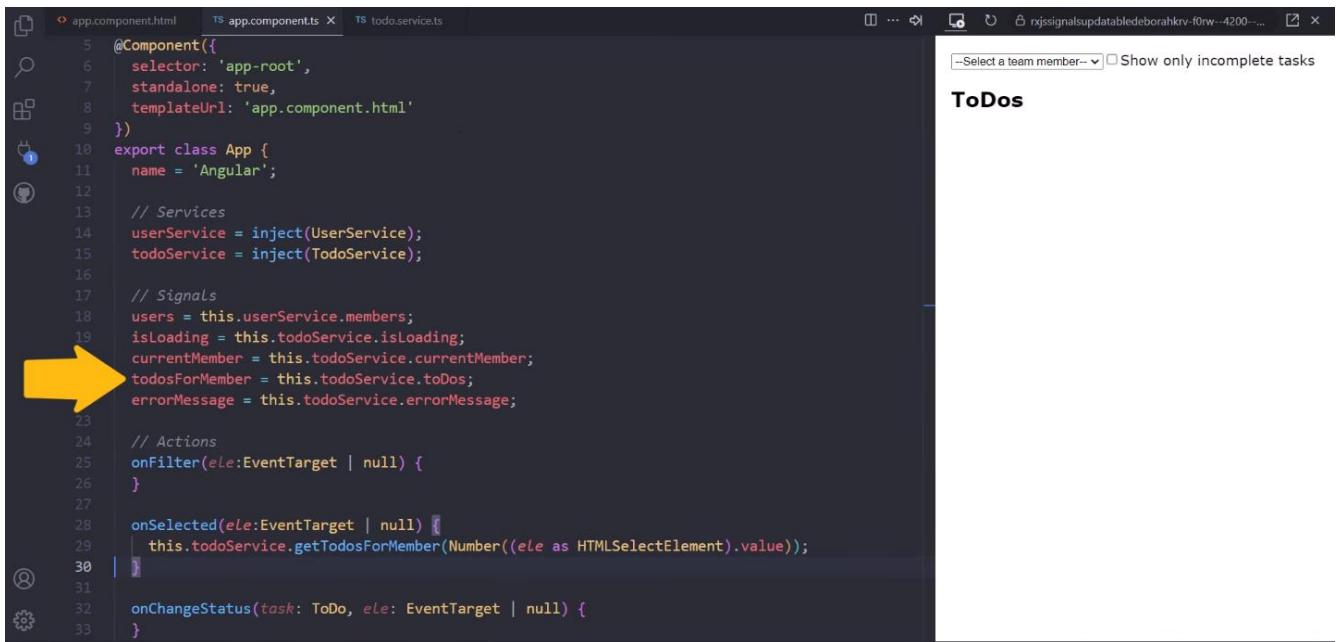
getTodosForMember(memberId: number) {
  this.selectedIdSubject.next(memberId);
}

export interface ToDo {
  userId: number;
  id: number;
  title: string;
  completed: boolean;
}

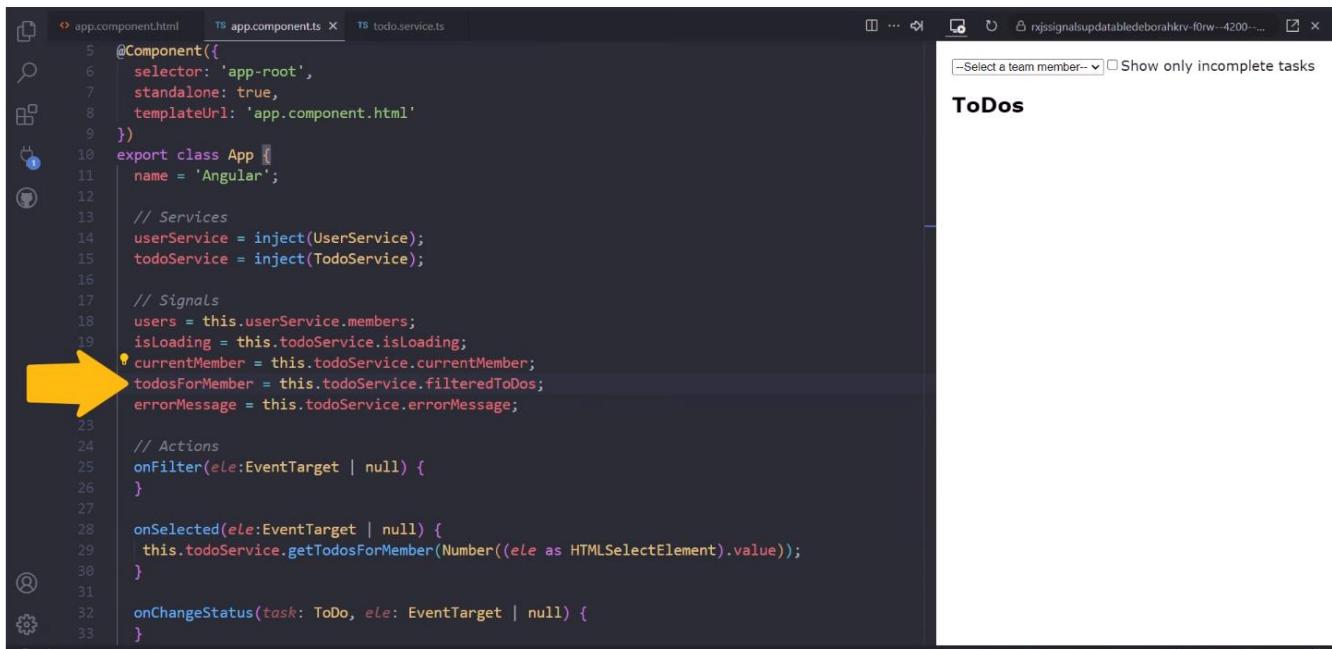
export interface ToDoState {
```



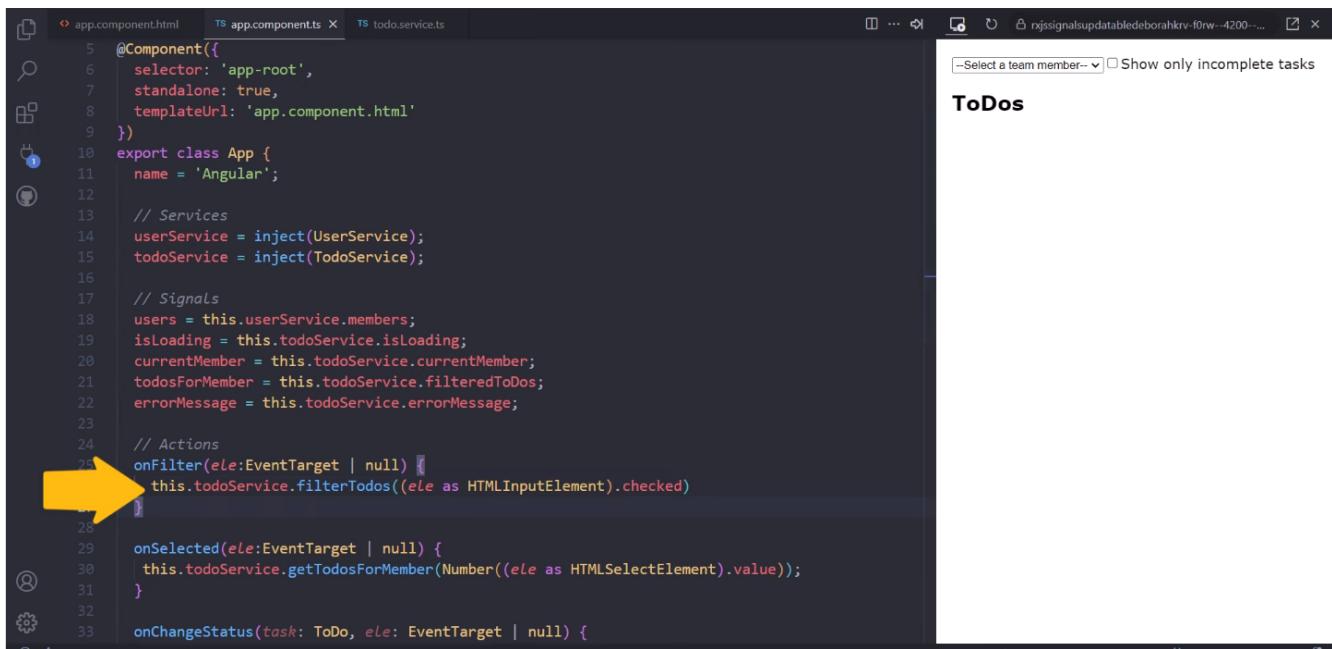
```
25 // Selectors
26 isLoading = computed(() => this.state().isLoading);
27 currentMember = computed(() => this.state().currentMember);
28 toDos = computed(() => this.state().memberToDos);
29 errorMessage = computed(() => this.state().error);
30 incompleteOnly = computed(() => this.state().incompleteOnly);
31 filteredToDos = computed(() => {
32   if (this.incompleteOnly()) {
33     return this.toDos().filter(t => t.completed === false);
34   }
35   else {
36     return this.toDos();
37   }
38 });
39
40 private selectedIdSubject = new Subject<number>();
41 private selectedId$ = this.selectedIdSubject.asObservable();
42
43 constructor() {
44   this.selectedId$.pipe(
45     tap(() => this.setLoadingIndicator(true)),
46     tap(id => this.setCurrentMember(id)),
47     switchMap(id => this.getTodos(id)),
48     delay(1000),
49     takeUntilDestroyed(),
50   ).subscribe(todos => this.setMemberToDos(todos));
51 }
52
53 private setLoadingIndicator(isLoading: boolean) {
```



```
5 @Component({
6   selector: 'app-root',
7   standalone: true,
8   templateUrl: 'app.component.html'
9 })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = this.todoService.isLoading;
20   currentMember = this.todoService.currentMember;
21   todosForMember = this.todoService.toDos;
22   errorMessage = this.todoService.errorMessage;
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26   }
27
28   onSelected(ele:EventTarget | null) {
29     this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
30   }
31
32   onChangeStatus(task: ToDo, ele: EventTarget | null) {
33 }
```



```
app.component.html    TS app.components    TS todo.services
5  @Component({
6    selector: 'app-root',
7    standalone: true,
8    templateUrl: 'app.component.html'
9  })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = this.todoService.isLoading;
20   currentMember = this.todoService.currentMember;
21   todosForMember = this.todoService.filteredTodos;
22   errorMessage = this.todoService.errorMessage;
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26   }
27
28   onSelected(ele:EventTarget | null) {
29     this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
30   }
31
32   onChangeStatus(task: ToDo, ele: EventTarget | null) {
33 }
```



```
app.component.html    TS app.component.ts    TS todo.service.ts
5  @Component({
6    selector: 'app-root',
7    standalone: true,
8    templateUrl: 'app.component.html'
9  })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = this.todoService.isLoading;
20   currentMember = this.todoService.currentMember;
21   todosForMember = this.todoService.filteredTodos;
22   errorMessage = this.todoService.errorMessage;
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26     this.todoService.filterTodos((ele as HTMLInputElement).checked)
27   }
28
29   onSelected(ele:EventTarget | null) {
30     this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
31   }
32
33   onChangeStatus(task: ToDo, ele: EventTarget | null) {
```

Leanne Graham

Show only incomplete tasks

ToDos for: Leanne Graham

Todo	Completed?
delectus aut autem	<input checked="" type="checkbox"/>
quis ut nam facilis	<input type="checkbox"/>
fugiat veniam minus	<input type="checkbox"/>
laboriosam mollitia	<input type="checkbox"/>
qui ullam ratione qu	<input type="checkbox"/>
illo expedita conseq	<input type="checkbox"/>
molestiae persipiat	<input type="checkbox"/>
et doloremque nulla	<input type="checkbox"/>
dolorum est consequa	<input type="checkbox"/>

```

5  @Component({
6    selector: 'app-root',
7    standalone: true,
8    templateUrl: 'app.component.html'
9  })
10 export class App {
11   name = 'Angular';
12
13   // Services
14   userService = inject(UserService);
15   todoService = inject(TodoService);
16
17   // Signals
18   users = this.userService.members;
19   isLoading = this.todoService.isLoading;
20   currentMember = this.todoService.currentMember;
21   todosForMember = this.todoService.filteredTodos;
22   errorMessage = this.todoService.errorMessage;
23
24   // Actions
25   onFilter(ele:EventTarget | null) {
26     | this.todoService.filterTodos((ele as HTMLInputElement).checked)
27   }
28
29   onSelected(ele:EventTarget | null) {
30     | this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
31   }
32
33   onChangeStatus(task: ToDo, ele: EventTarget | null) {

```

private state = signal<ToDoState>({
 isLoading: false,
 currentMember: undefined,
 memberTodos: [],
 incompleteOnly: false,
 error: null
});

// Selectors
isLoading = computed(() => this.state().isLoading);
currentMember = computed(() => this.state().currentMember);
toDos = computed(() => this.state().memberTodos);
errorMessage = computed(() => this.state().error);
incompleteOnly = computed(() => this.state().incompleteOnly);
filteredToDos = computed(() => {
 if (this.incompleteOnly()) {
 return this.toDos().filter(t => t.completed === false),
 }
})

private state = signal<ToDoState>({
 isLoading: false,
 currentMember: undefined,
 memberTodos: [],
 incompleteOnly: false,
 error: null
});

// Selectors
isLoading = computed(() => this.state().isLoading);
currentMember = computed(() => this.state().currentMember);
toDos = computed(() => this.state().memberTodos);
errorMessage = computed(() => this.state().error);
incompleteOnly = computed(() => this.state().incompleteOnly);
filteredToDos = computed(() => {
 if (this.incompleteOnly()) {
 return this.toDos().filter(t => t.completed === false),
 }
 else {
 return this.toDos();
 }
});

private selectedIdSubject = new Subject<number>();
private selectedId\$ = this.selectedIdSubject.asObservable();

constructor() {
 this.selectedId\$.pipe(
 tap(() => this.setLoadingIndicator(true)),

```

graph TD
    View[View] -- "User Event" --> Component[Component]
    Component -- Dispatch --> Action[Action]
    Action --> Reducer[Reducer]
    Reducer --> NewState[New State]
    NewState --> Store[Store State]
    Store -- State --> Selector[Selector]
    Selector --> Component

```

The diagram illustrates the Redux flow. It starts with a **View** component that emits a **User Event**. This event triggers a **Component**, which then dispatches an **Action**. The **Action** is processed by a **Reducer** to produce a **New State**. This new state is stored in a **Store (State)**, which provides the **State** to a **Selector**. The **Selector** then provides the **State** back to the **Component**.

```

app.component.html
app.component.ts
todo.service.ts

private selectedIdSubject = new Subject<number>();
private selectedId$ = this.selectedIdSubject.asObservable();

constructor() {
  this.selectedId$.pipe(
    tap(() => this.setLoadingIndicator(true)),
    tap(id => this.setCurrentMember(id)),
    switchMap(id => this.getTodos(id)),
    delay(1000),
    takeUntilDestroyed(),
  ).subscribe(todos => this.setMemberTodos(todos));
}

private setLoadingIndicator(isLoading: boolean) {
  this.state.update(state => ({
    ...state,
    isLoading: isLoading
  }));
}

private setCurrentMember(id: number) {
  const member = this.userService.getCurrentMember(id);
  this.state.update(state => ({
    ...state,
    currentMember: member,
    memberTodos: []
  }));
}

private getTodos(id: number): Observable<ToDo[]> {
  return this.http.get<ToDo[]>(`${this.todoUrl}?userId=${id}`).pipe(
    ...
  );
}

```

A yellow arrow points to the `this.state.update` line in the code, highlighting the point where the state is being modified.

```

app.component.html
app.component.ts
todo.service.ts

private selectedIdSubject = new Subject<number>();
private selectedId$ = this.selectedIdSubject.asObservable();

constructor() {
  this.selectedId$.pipe(
    tap(() => this.setLoadingIndicator(true)),
    tap(id => this.setCurrentMember(id)),
    switchMap(id => this.getTodos(id)),
    delay(1000),
    takeUntilDestroyed(),
  ).subscribe(todos => this.setMemberTodos(todos));
}

private setLoadingIndicator(isLoading: boolean) {
  this.state.update(state => ({
    ...state,
    isLoading: isLoading
  }));
}

private setCurrentMember(id: number) {
  const member = this.userService.getCurrentMember(id);
  this.state.update(state => ({
    ...state,
    currentMember: member,
    memberTodos: []
  }));
}

private getTodos(id: number): Observable<ToDo[]> {
  return this.http.get<ToDo[]>(`${this.todoUrl}?userId=${id}`).pipe(
    ...
  );
}

```