A Redux-like Pattern for Managing State with Angular signals

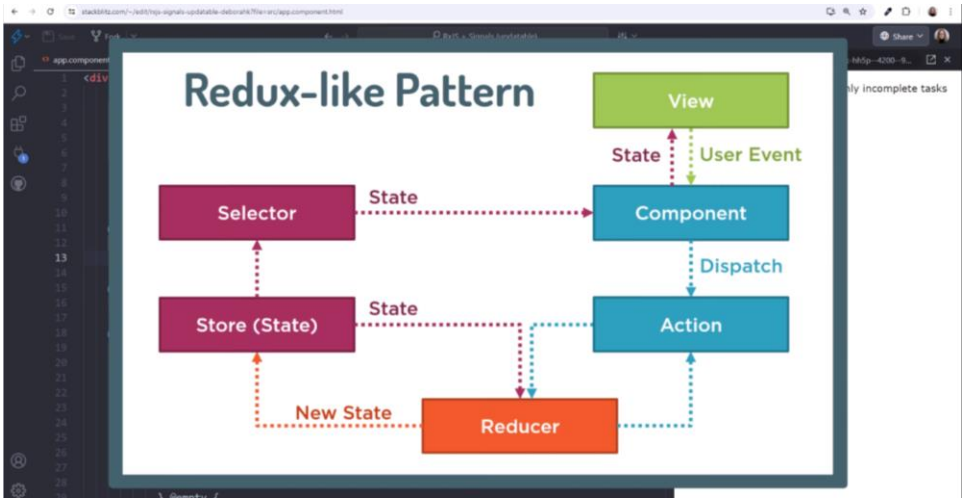
Deborah Kurata
12.9K subscribers

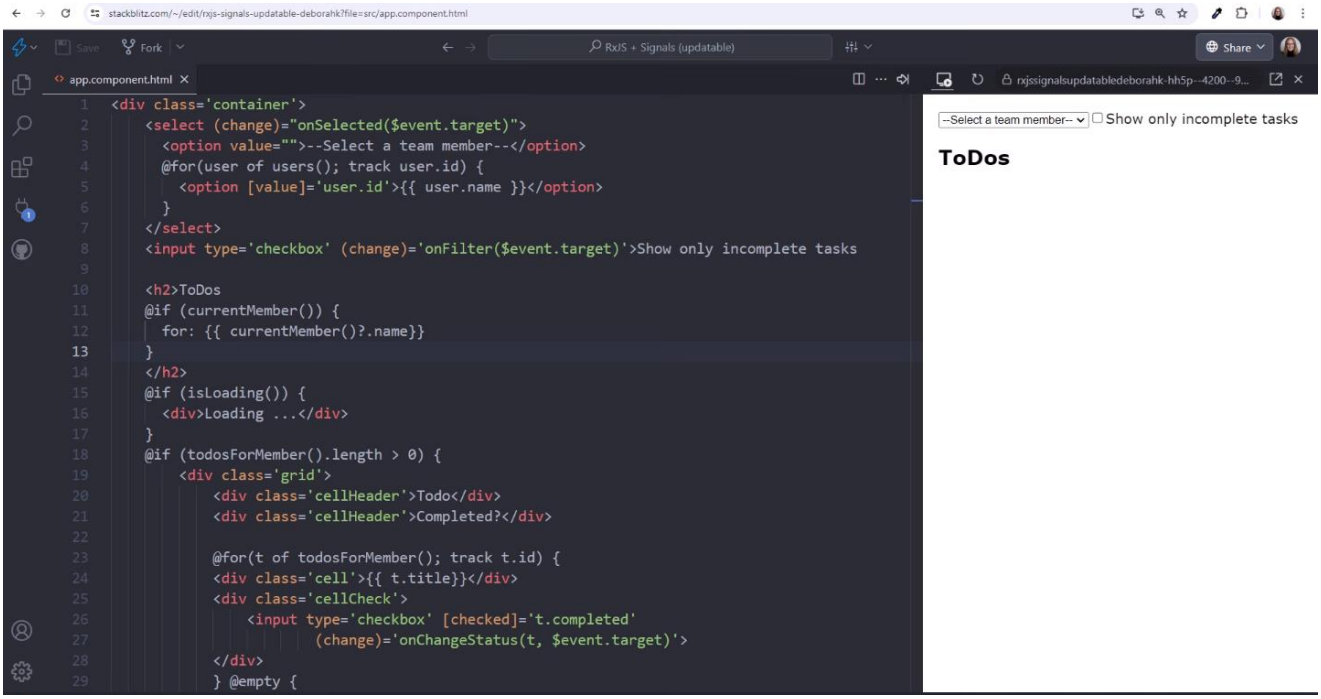
Subscribed | 11 | Share | Download

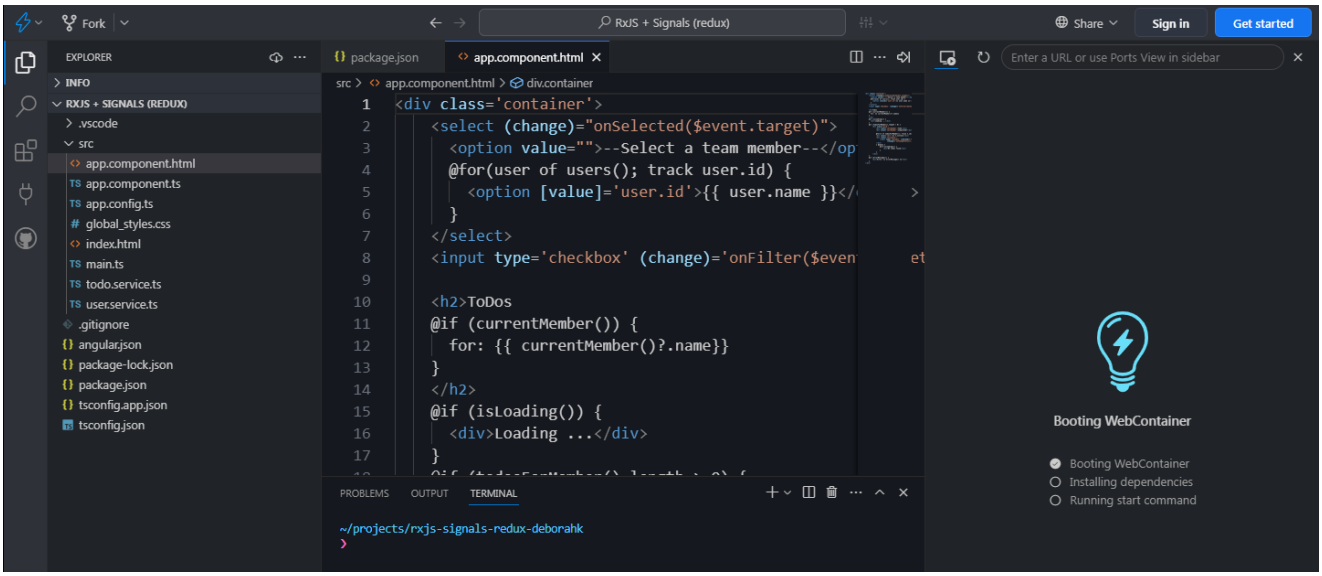
45 views Jun 19, 2024 #angular #angularsignals #angularstatemanagementwithsignals

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

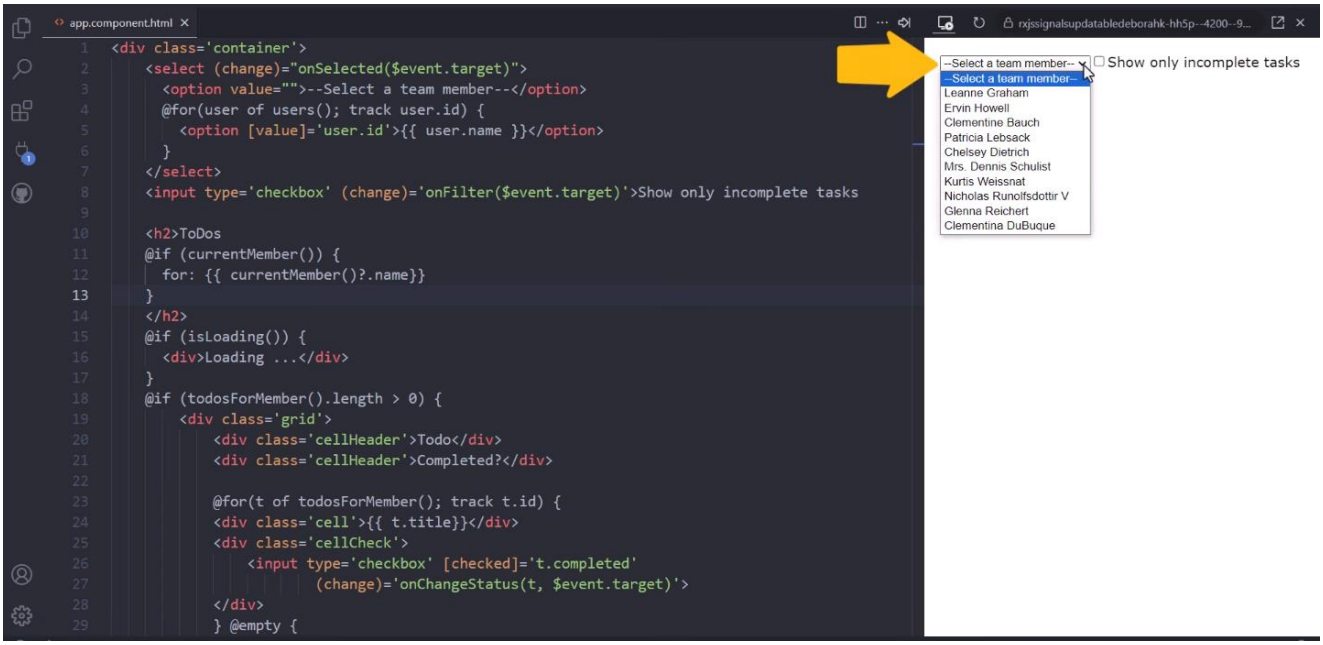
In this video, we'll walk through one such pattern for managing state with signals. This Redux-like pattern leverages the best of signals and RxJS.






Fork
RxJS + Signals (redux)
Share
Sign in
Get started
EXPLORER
INFO
RXJS + SIGNALS (REDUX)
.vscode
src
app.component.html
app.component.ts
app.config.ts
global_styles.css
index.html
main.ts
todo.service.ts
user.service.ts
.gitignore
angular.json
package-lock.json
package.json
tsconfig.app.json
tsconfig.json
package.json
app.component.html
src > app.component.html > div.container

<select (change)="onSelected($event.target)">
<option value="">--Select a team member--</op
@for(user of users(); track user.id) {
<option [value]='user.id'>{{ user.name }}</
}
</select>
<input type='checkbox' (change)='onFilter($even
et
<h2>ToDos
@if (currentMember()) {
for: {{ currentMember()?.name}}
}
</h2>
@if (isLoading()) {
Loading ...
}
PROBLEMS
OUTPUT
TERMINAL
~/projects/rxjs-signals-redux-deborahk
Enter a URL or use Ports View in sidebar
Booting WebContainer
Booting WebContainer
Installing dependencies
Running start command

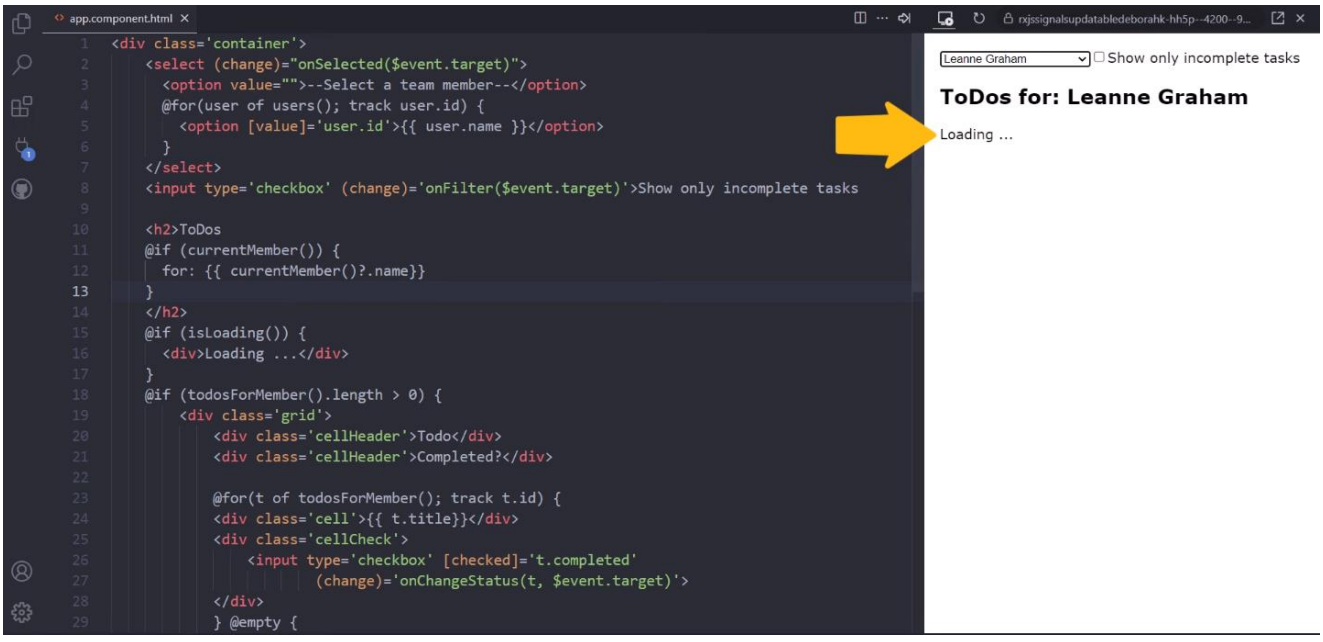

app.component.html

<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
@if (currentMember()) {
for: {{ currentMember()?.name}}
}
</h2>
@if (isLoading()) {
Loading ...
}
@if (todosForMember().length > 0) {

Todo
Completed?
@for(t of todosForMember(); track t.id) {
{{ t.title}}

<input type='checkbox' [checked]='t.completed'
(change)='onChangeStatus(t, $event.target)'>

} @empty {
--Select a team member--
Show only incomplete tasks
--Select a team member--
Leanne Graham
Ervin Howell
Clementine Bauch
Patricia Lebsack
Chelsey Dietrich
Mrs. Dennis Schulist
Kurtis Weissnat
Nicholas Runolfsdottir V
Glenna Reichert
Clementina DuBuque

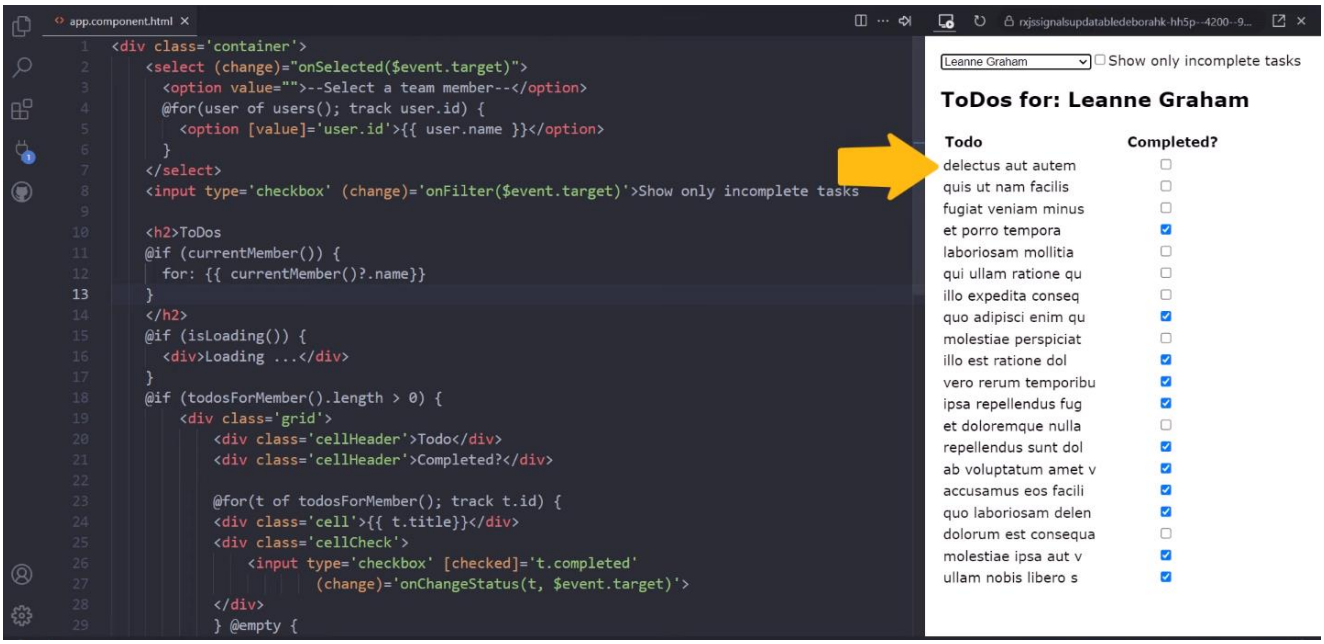

app.component.html

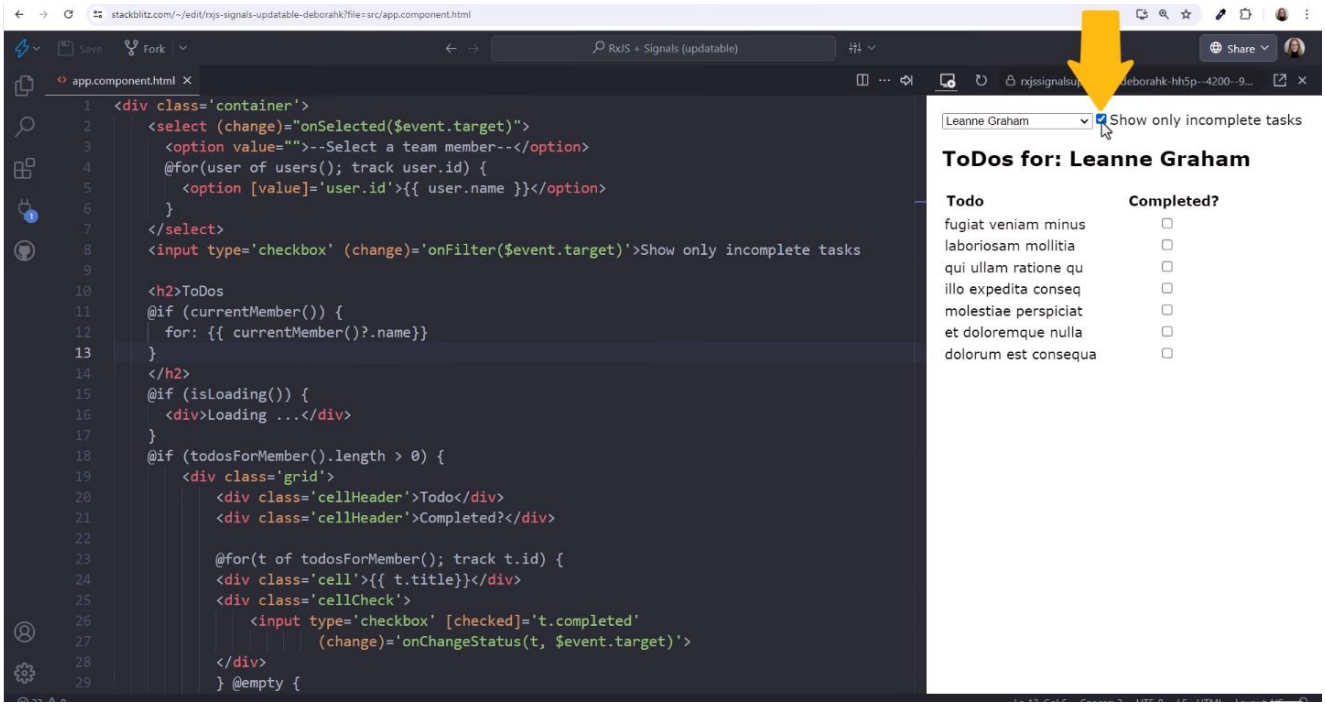
<select (change)="onSelected($event.target)">
<option value="">--Select a team member--</option>
@for(user of users(); track user.id) {
<option [value]='user.id'>{{ user.name }}</option>
}
</select>
<input type='checkbox' (change)='onFilter($event.target)'>Show only incomplete tasks
<h2>ToDos
@if (currentMember()) {
for: {{ currentMember()?.name}}
}
</h2>
@if (isLoading()) {
Loading ...
}
@if (todosForMember().length > 0) {

Todo
Completed?
@for(t of todosForMember(); track t.id) {
{{ t.title}}

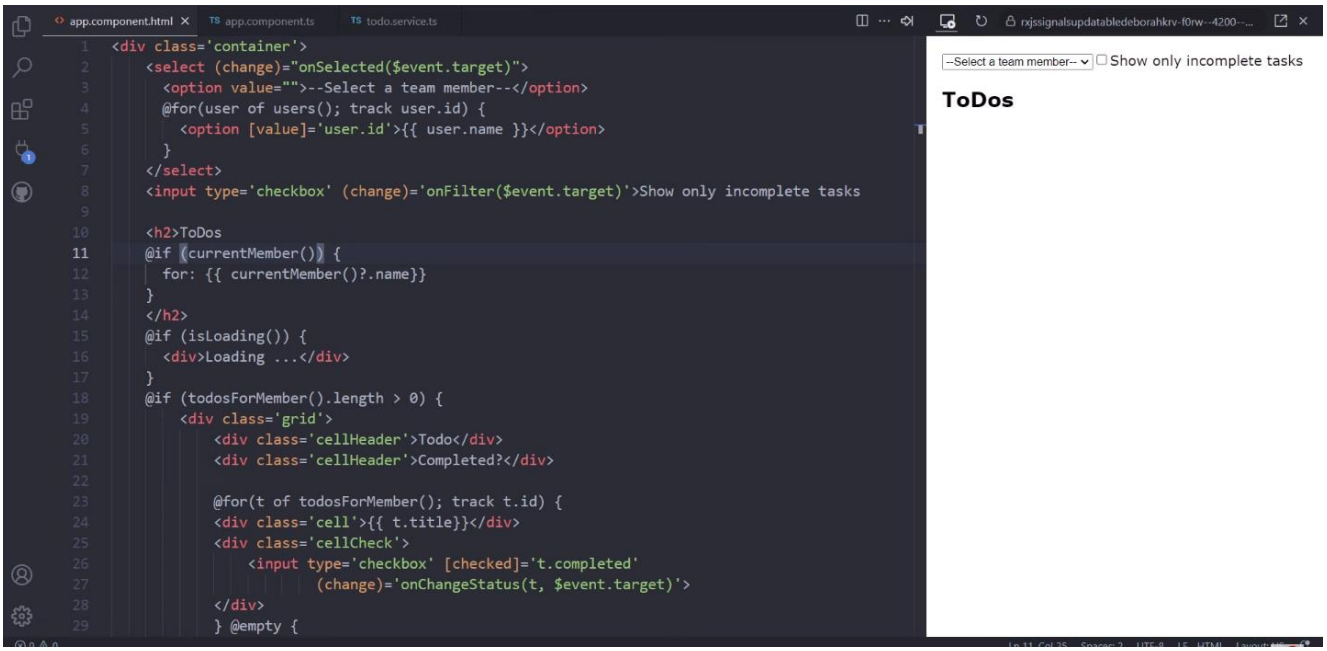
<input type='checkbox' [checked]='t.completed'
(change)='onChangeStatus(t, $event.target)'>

} @empty {
Leanne Graham
Show only incomplete tasks
ToDos for: Leanne Graham
Loading ...





Now let us start from the beginning and code this app

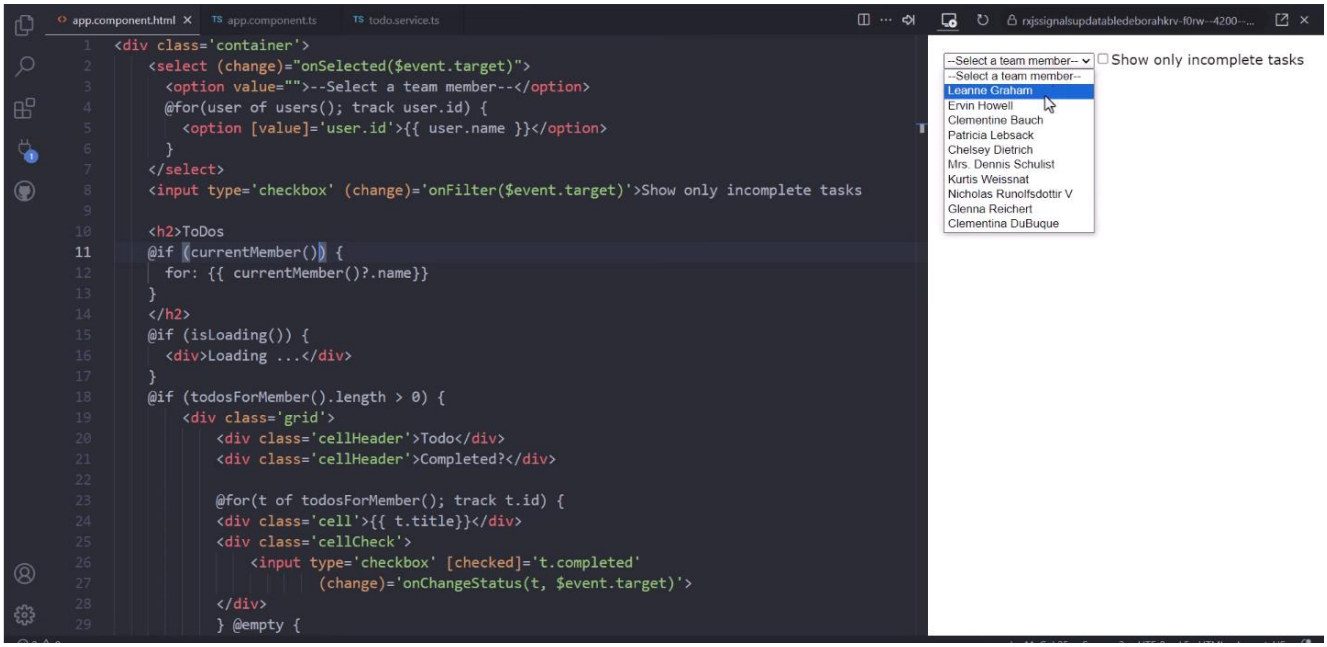

app.component.html
app.component.ts
todo.service.ts

<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
@if (currentMember()) {
for: {{ currentMember()?.name}}
}
</h2>
@if (isLoading()) {
Loading ...
}
@if (todosForMember().length > 0) {

Todo
Completed?
@for(t of todosForMember(); track t.id) {
{{ t.title}}

<input type='checkbox' [checked]='t.completed'
(change)='onChangeStatus(t, $event.target)'>

} @empty {
--Select a team member--
Show only incomplete tasks
ToDos

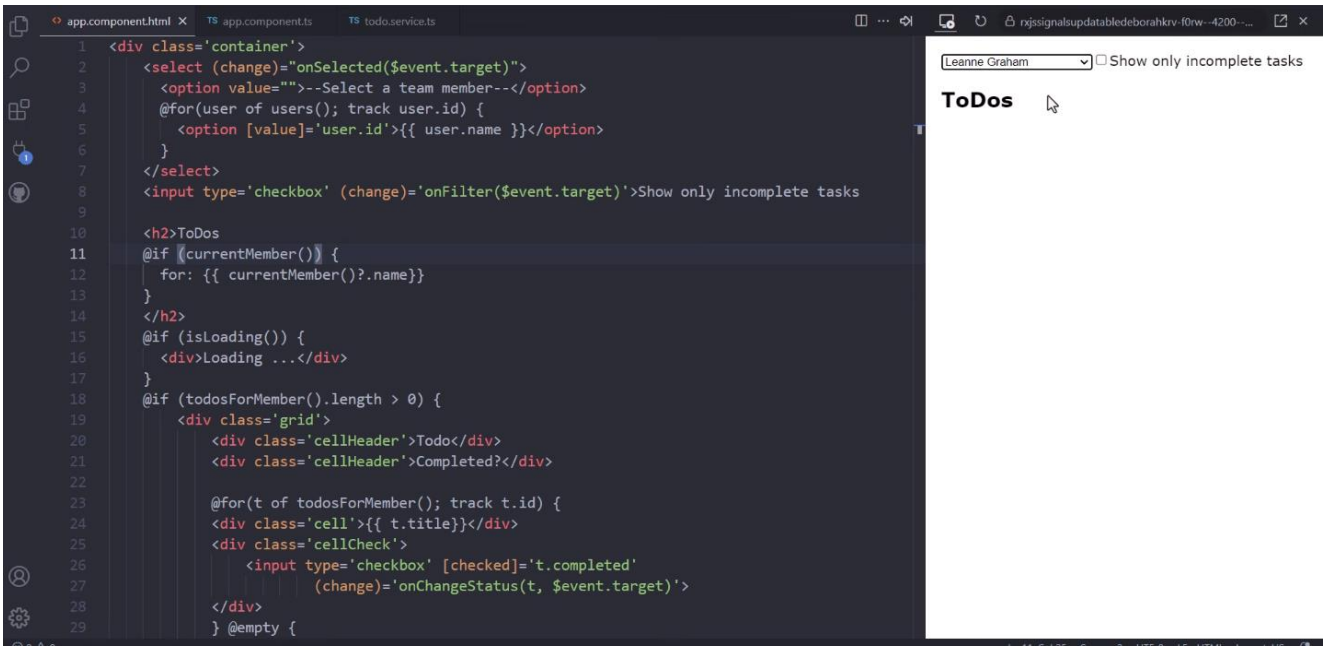

app.component.html
app.component.ts
todo.service.ts

<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
@if (currentMember()) {
for: {{ currentMember()?.name}}
}
</h2>
@if (isLoading()) {
Loading ...
}
@if (todosForMember().length > 0) {

Todo
Completed?
@for(t of todosForMember(); track t.id) {
{{ t.title}}

<input type='checkbox' [checked]='t.completed'
(change)='onChangeStatus(t, $event.target)'>

} @empty {
--Select a team member--
Show only incomplete tasks
--Select a team member--
Leanne Graham
Ervin Howell
Clementine Bauch
Patricia Lebsack
Chelsey Dietrich
Mrs. Dennis Schulist
Kurtis Weissnat
Nicholas Runolfsdottir V
Glenna Reichert
Clementina DuBuque

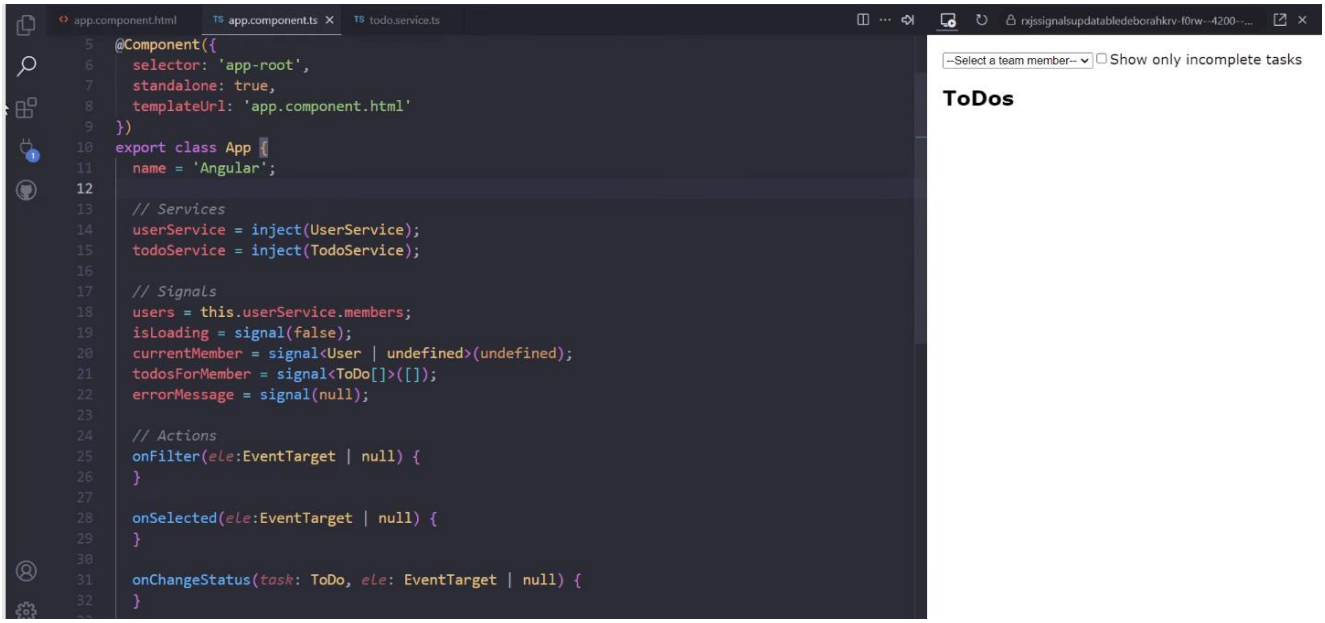

app.component.html
app.component.ts
todo.service.ts

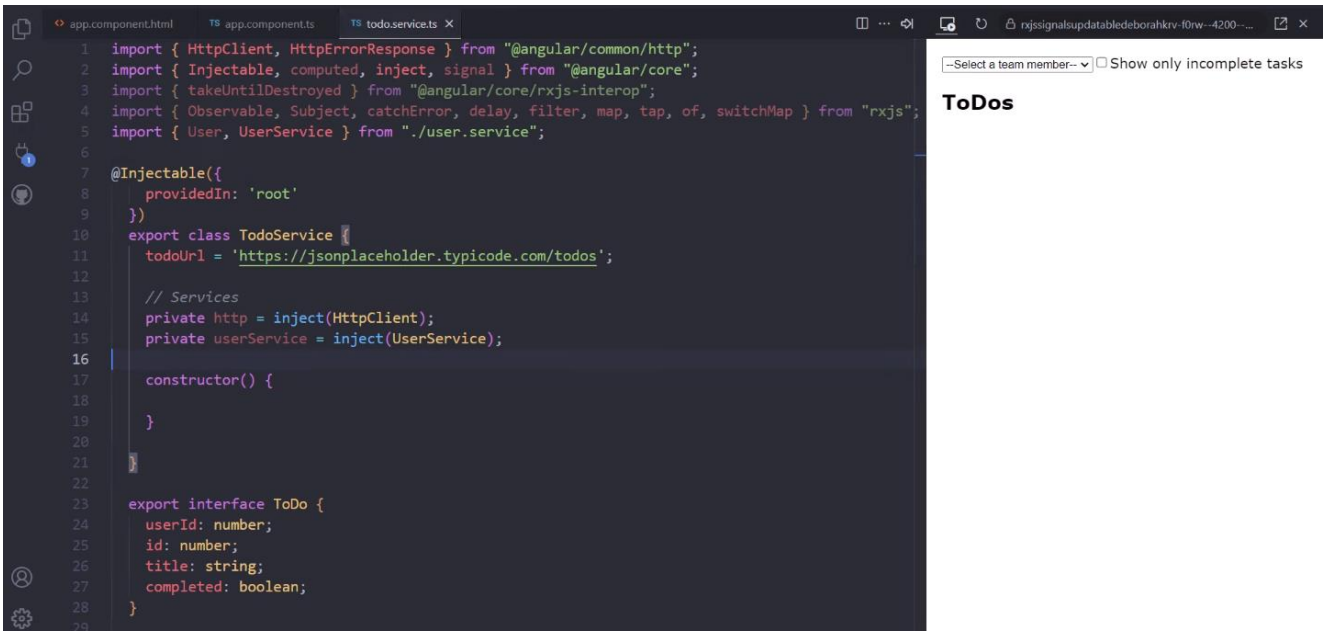
<select (change)="onSelected($event.target)">
<option value="">--Select a team member--</option>
@for(user of users(); track user.id) {
<option [value]='user.id'>{{ user.name }}</option>
}
</select>
<input type='checkbox' (change)='onFilter($event.target)'>Show only incomplete tasks
<h2>ToDos
@if (currentMember()) {
for: {{ currentMember()?.name}}
}
</h2>
@if (isLoading()) {
Loading ...
}
@if (todosForMember().length > 0) {

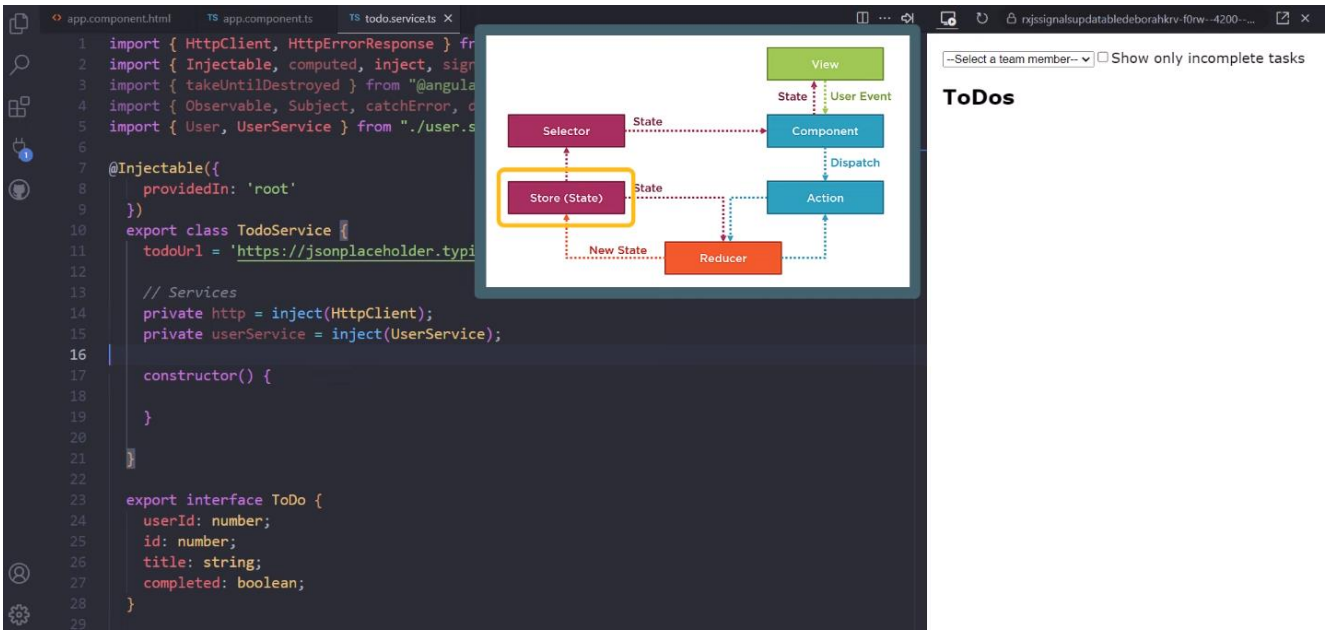
Todo
Completed?
@for(t of todosForMember(); track t.id) {
{{ t.title}}

<input type='checkbox' [checked]='t.completed'
(change)='onChangeStatus(t, $event.target)'>

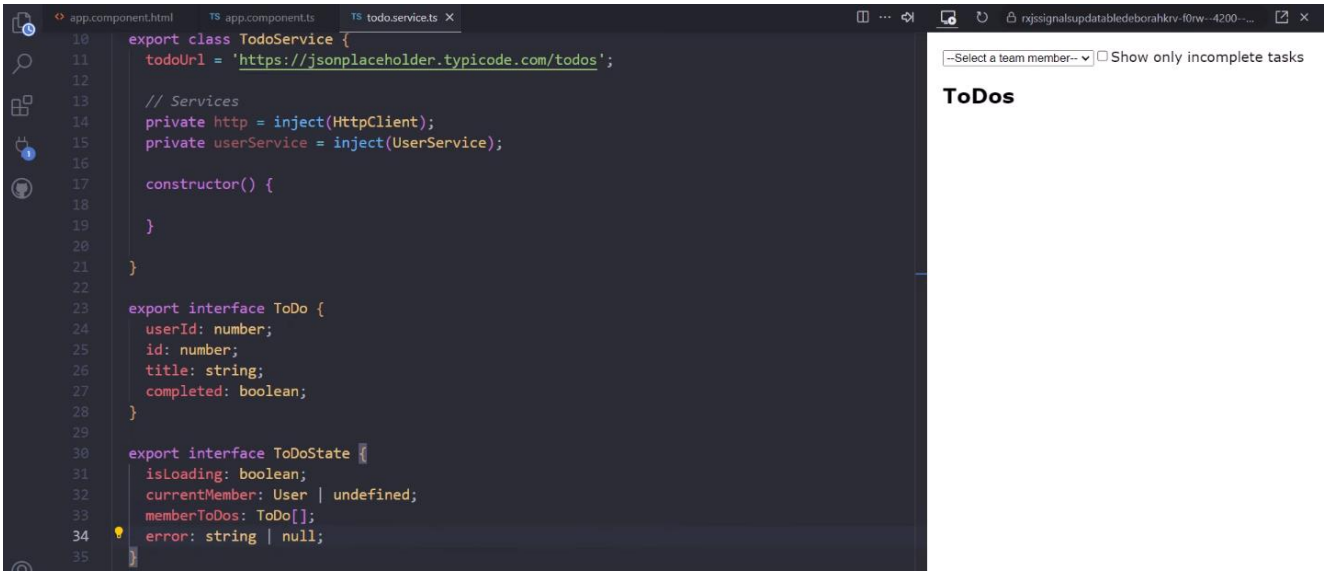
} @empty {
Leanne Graham
Show only incomplete tasks
ToDos

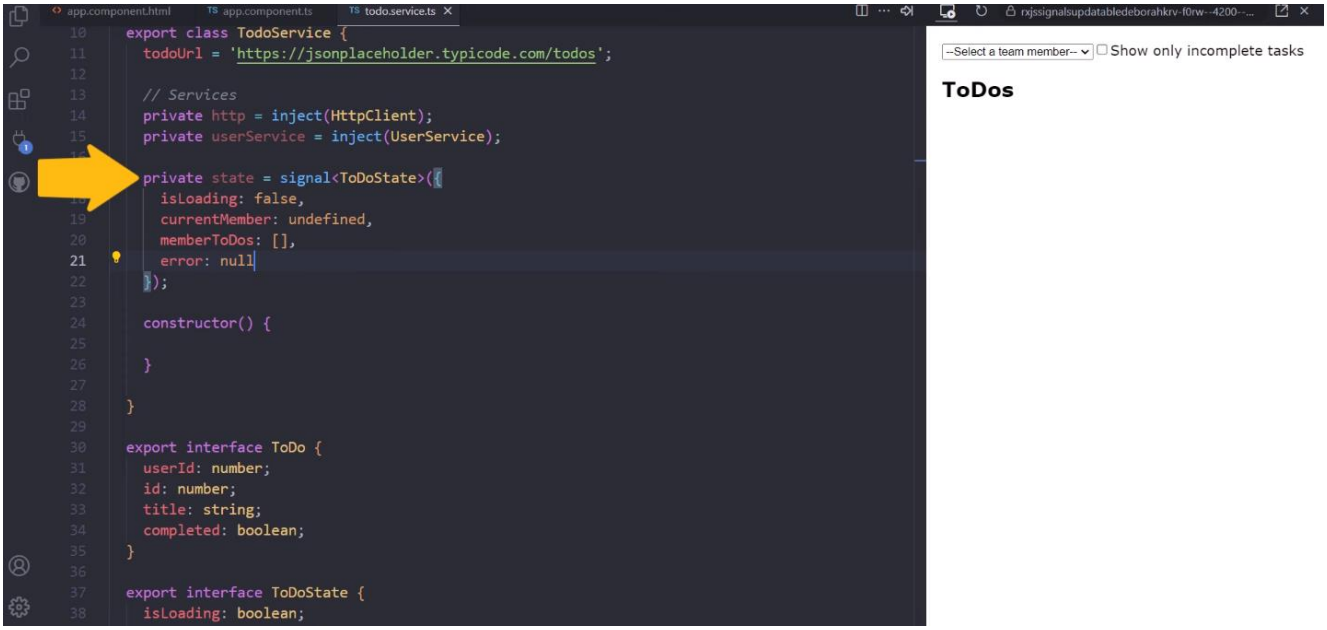

app.component.html
app.component.ts
todo.service.ts
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = signal(false);
currentMember = signal<User | undefined>(undefined);
todosForMember = signal<ToDo[]>([]);
errorMessage = signal(null);
// Actions
onFilter(ele:EventTarget | null) {
}
onSelected(ele:EventTarget | null) {
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
}
--Select a team member--
Show only incomplete tasks
ToDos

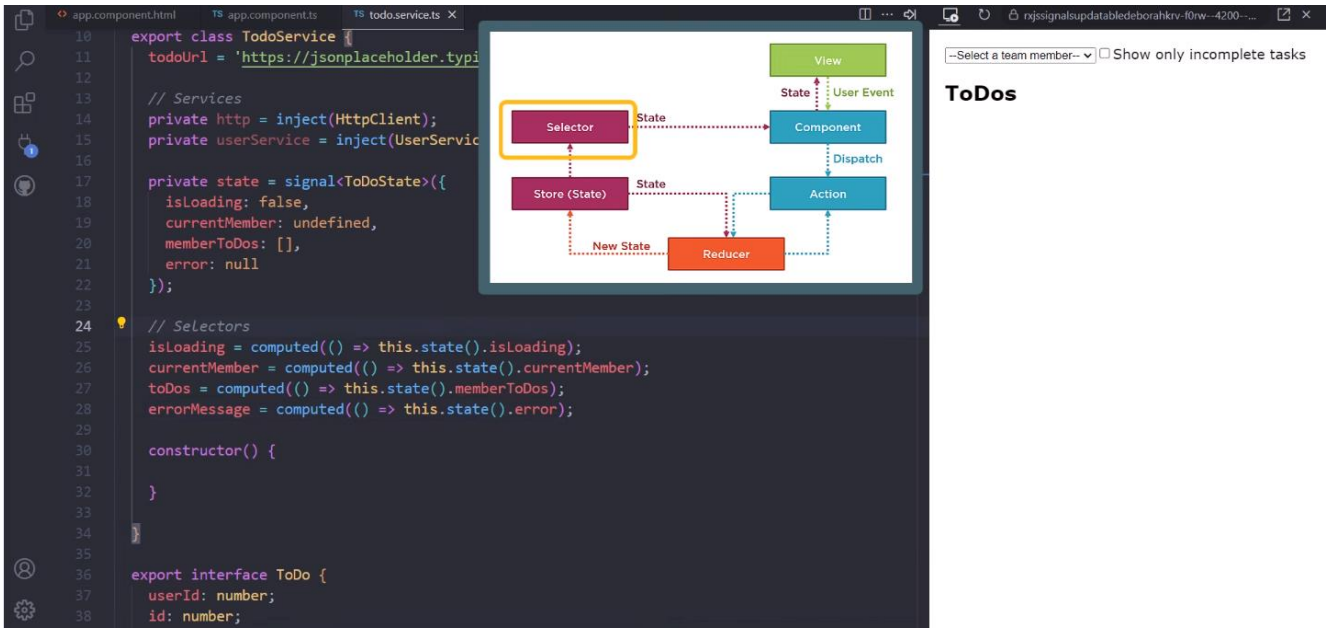




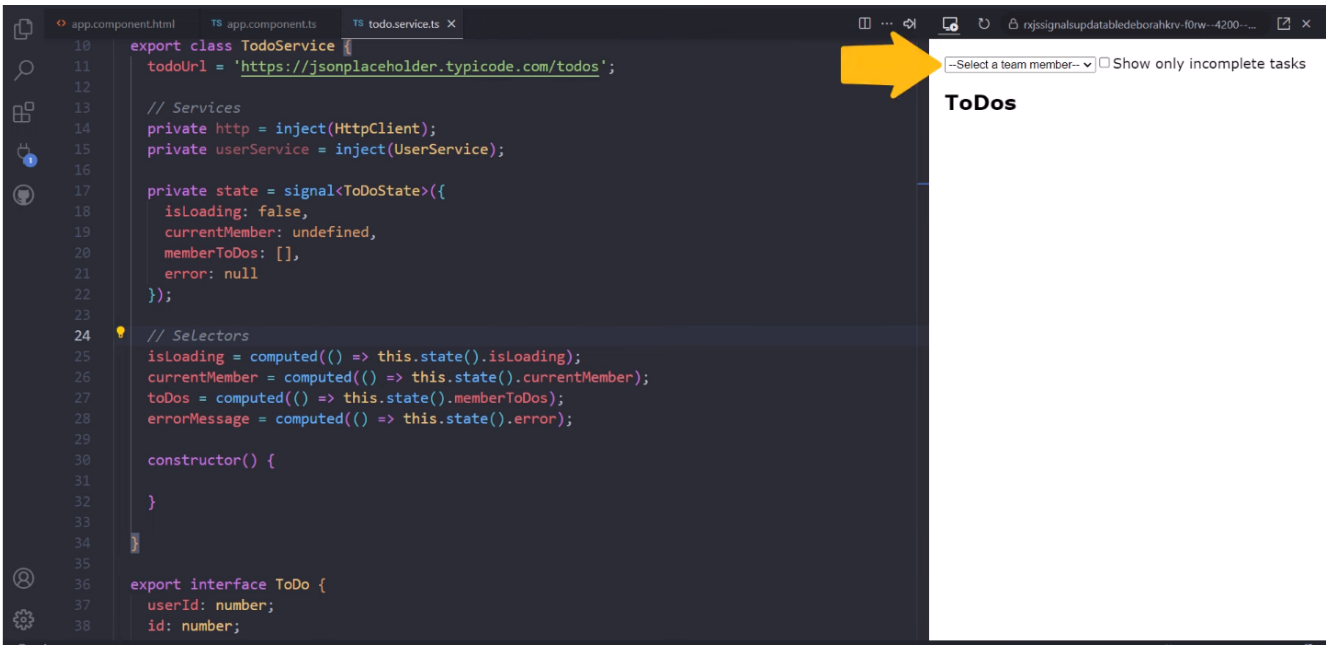
What state or data does our ToDo service need to manage?



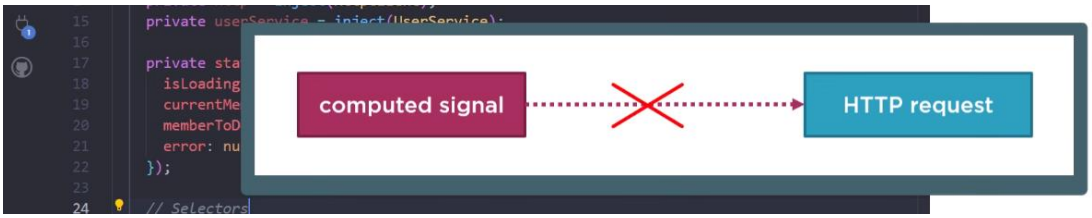




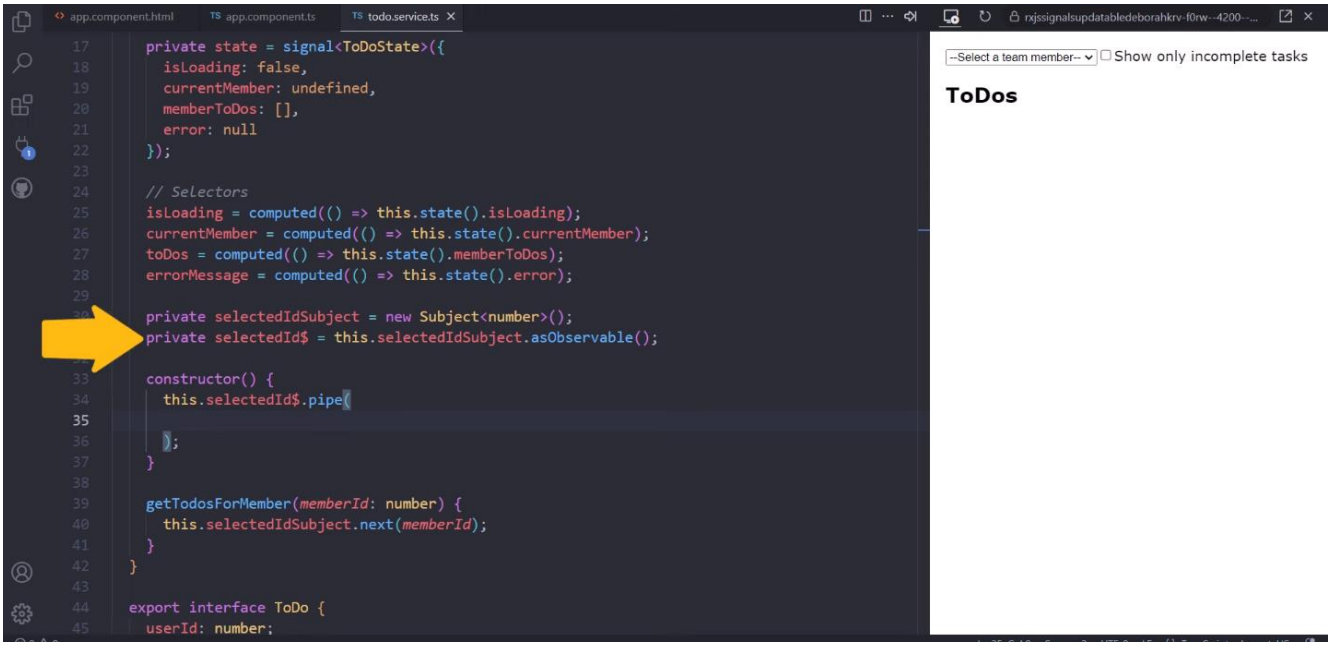
A Selector selects a specific piece or slice of our state as a computed signal as seen above.



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



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

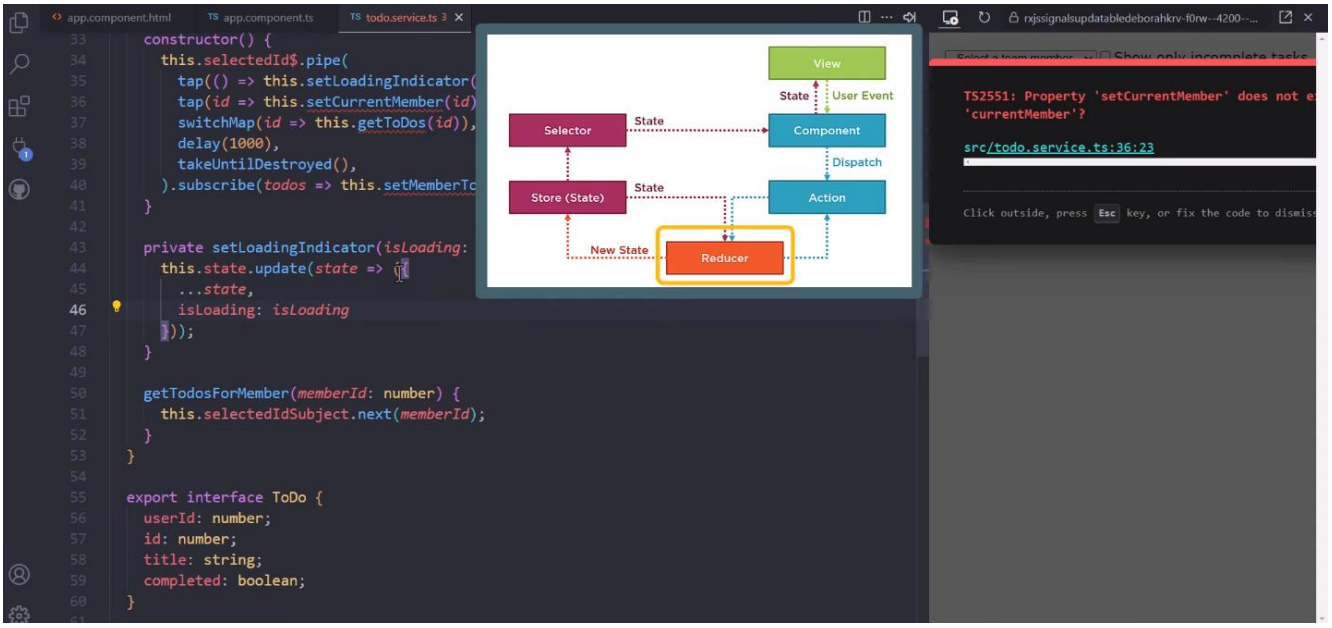
```ts
  private state = signal<ToDoState>({
    isLoading: false,
    currentMember: undefined,
    memberToDos: [],
    error: null
  });

  // Selectors
  isLoading = computed(() => this.state().isLoading);
  currentMember = computed(() => this.state().currentMember);
  toDos = computed(() => this.state().memberToDos);
  errorMessage = computed(() => this.state().error);

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

  constructor() {
    this.selectedId$.pipe(
      tap(() => this.setLoadingIndicator(true)),
      tap(id => this.setCurrentMember(id))
    );
  }

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

export interface ToDo {
```

```
TS2339: Property 'setLoadingIndicator' does not

src/todo.service.ts:35:23

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

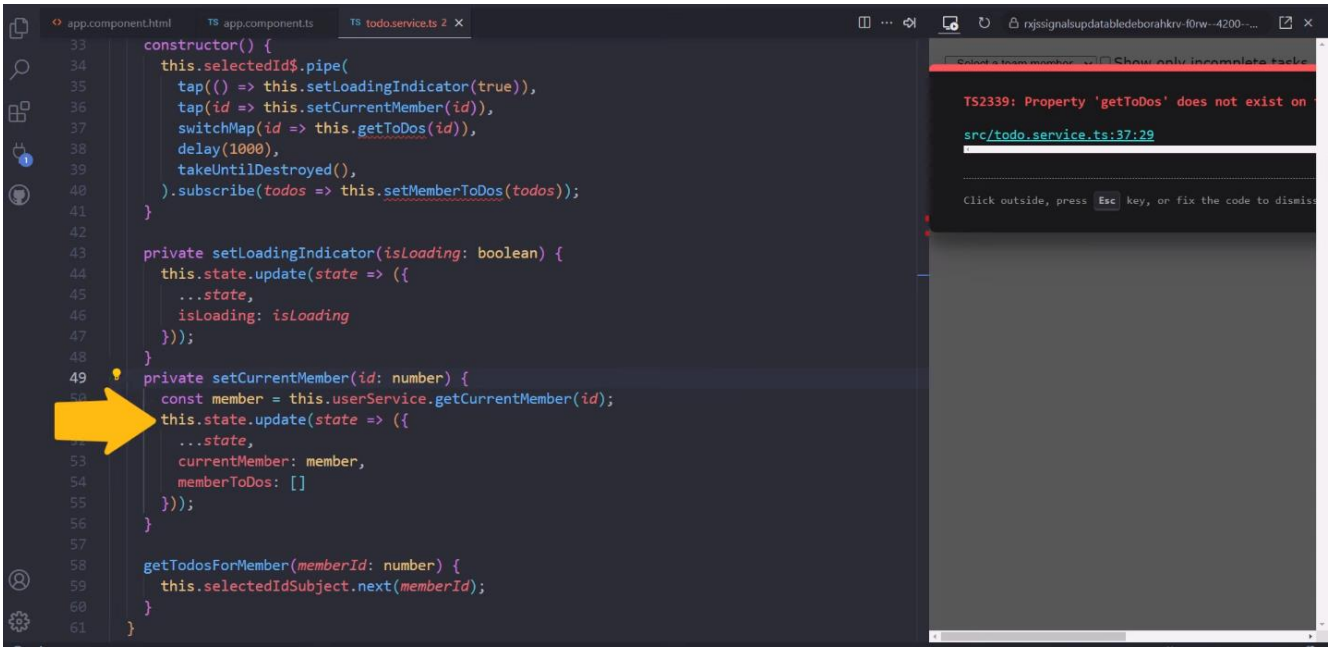
app.component.html | app.component.ts | todo.service.ts 4

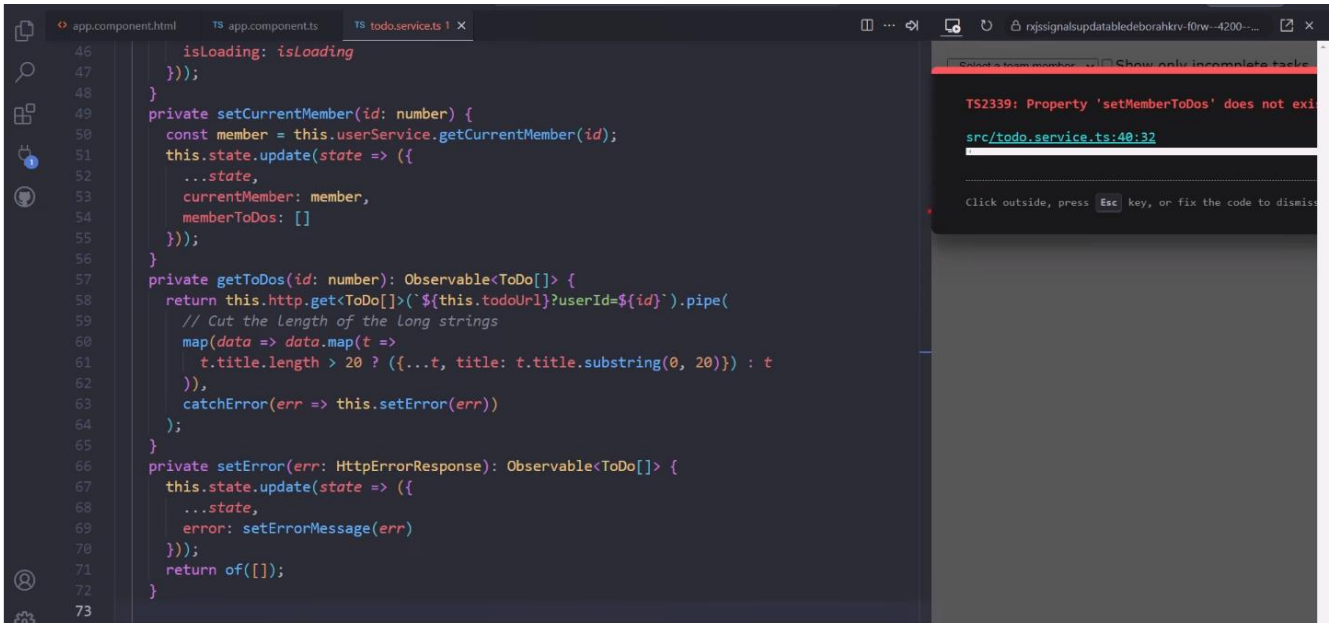
```ts
  private state = signal<ToDoState>({
    isLoading: false,
    currentMember: undefined,
    memberToDos: [],
    error: null
  });

  // Selectors
  isLoading = computed(() => this.state().isLoading);
  currentMember = computed(() => this.state().currentMember);
  toDos = computed(() => this.state().memberToDos);
  errorMessage = computed(() => this.state().error);

  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));
  }

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

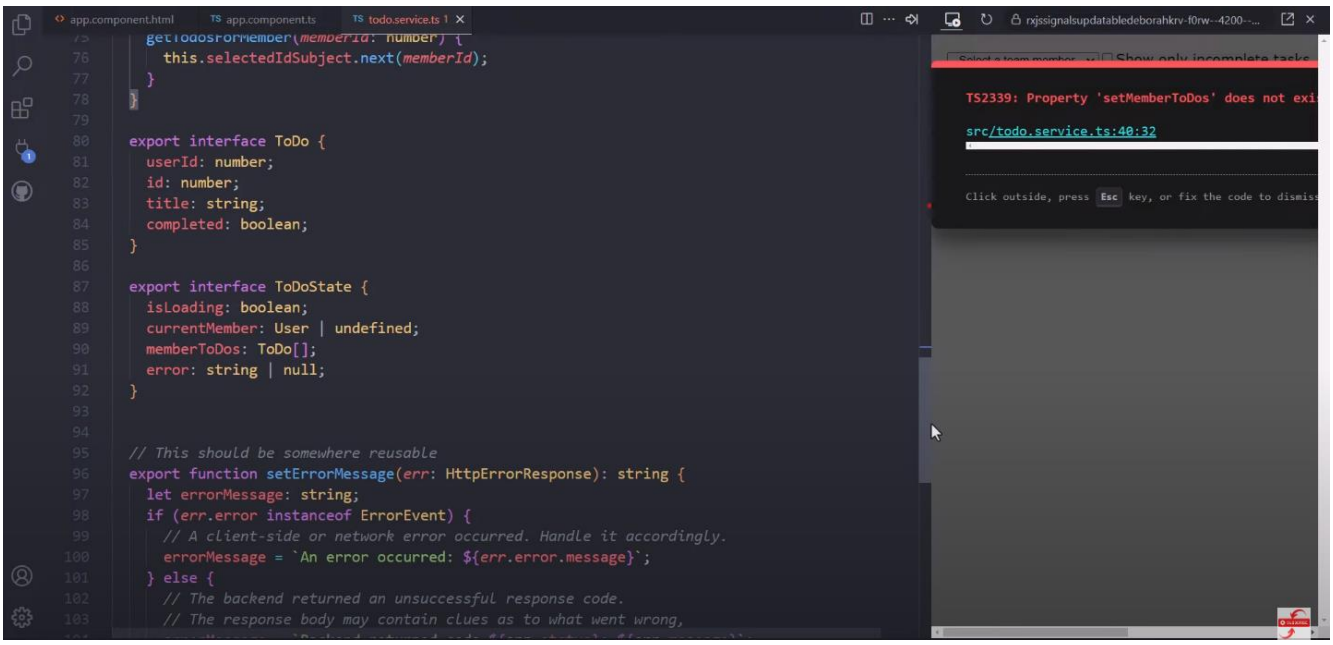
```
TS2339: Property 'setLoadingIndicator' does not

src/todo.service.ts:35:23

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

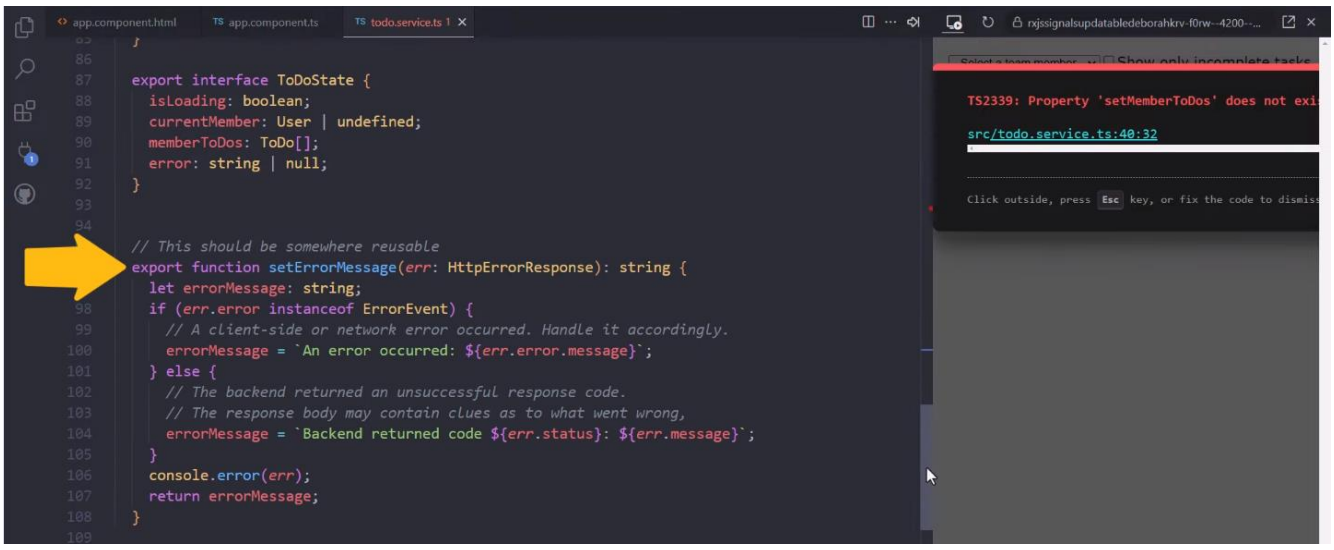


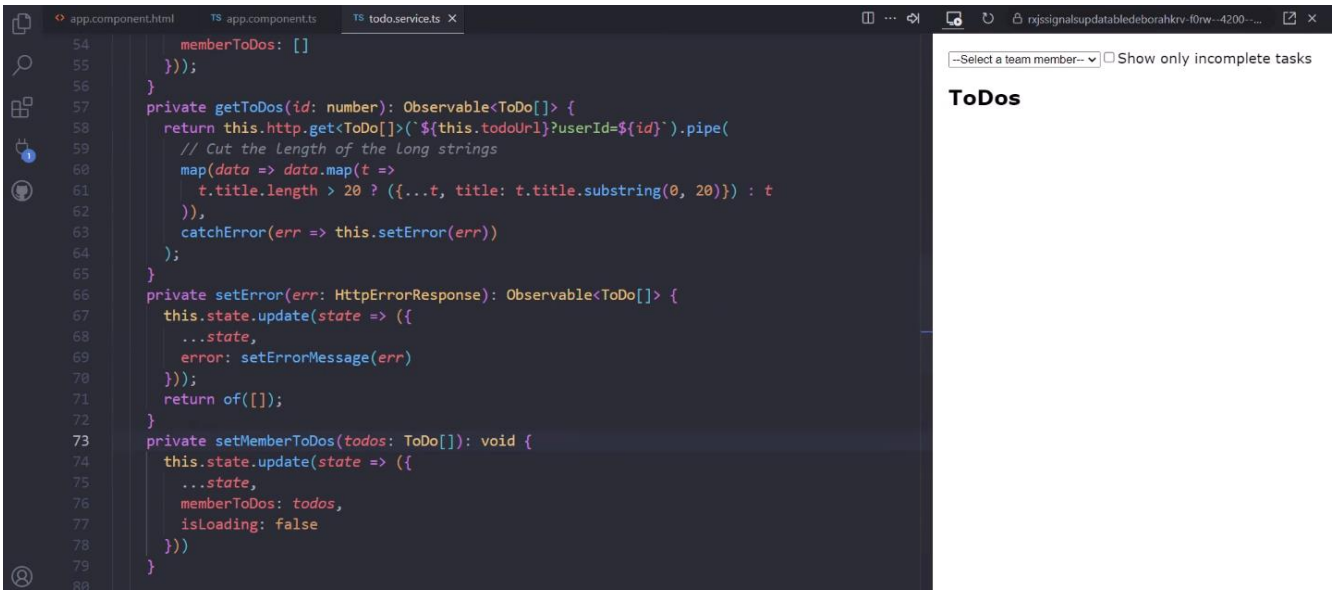
A Reducer defines how an action should update state

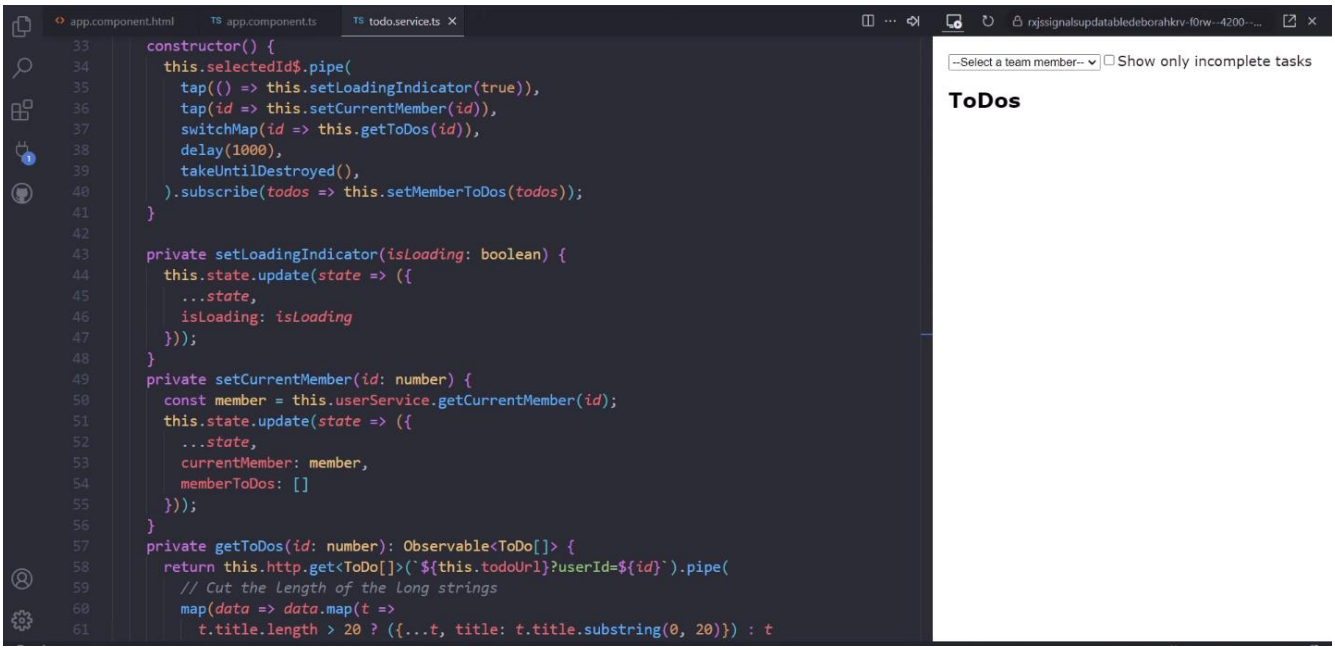



app.component.html
app.component.ts
todo.service.ts 1
isLoading: isLoading
}));
}
private setCurrentMember(id: number) {
const member = this.userService.getCurrentMember(id);
this.state.update(state => ({
...state,
currentMember: member,
memberToDos: []
}));
}
private getToDos(id: number): Observable<ToDo[]> {
return this.http.get<ToDo[]>(`${this.todoUrl}?userId=${id}`).pipe(
// Cut the length of the long strings
map(data => data.map(t =>
t.title.length > 20 ? ({...t, title: t.title.substring(0, 20)}) : t
)),
catchError(err => this.setError(err))
);
}
private setError(err: HttpErrorResponse): Observable<ToDo[]> {
this.state.update(state => ({
...state,
error: setErrorMessage(err)
}));
return of([]);
}
73
TS2339: Property 'setMemberToDos' does not exi
src/todo.service.ts:40:32
Click outside, press Esc key, or fix the code to dismiss

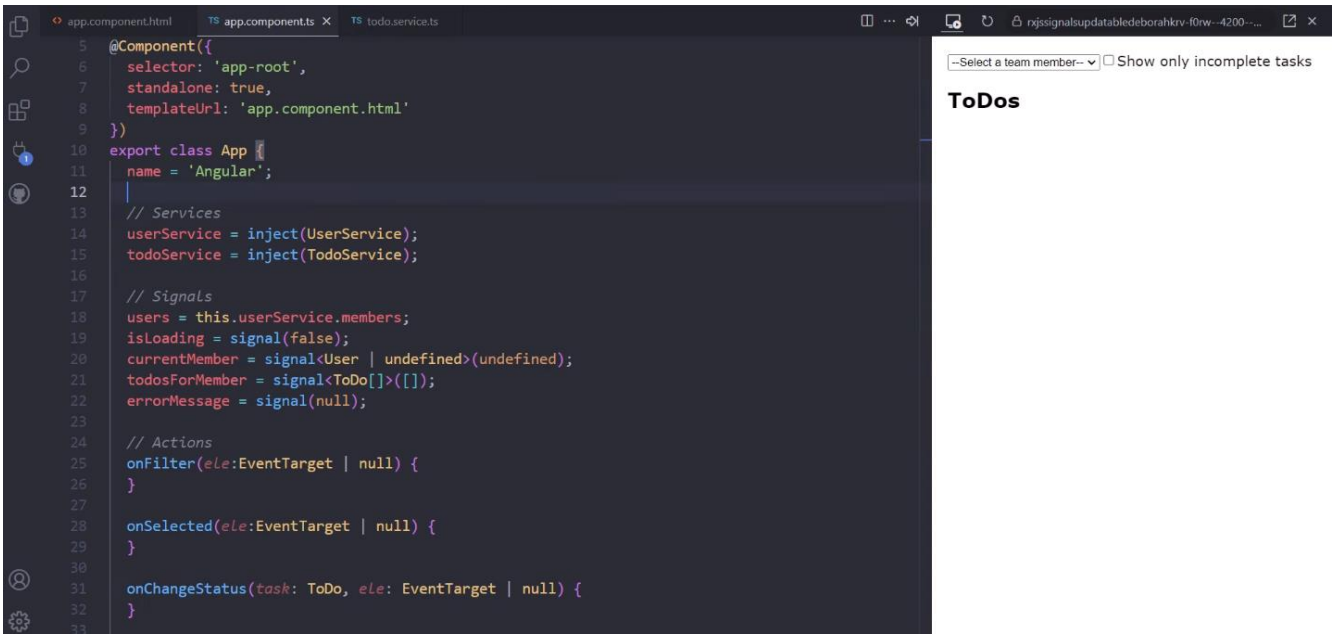

app.component.html
app.component.ts
todo.service.ts 1
this.selectedIdSubject.next(memberId);
}
}
export interface ToDo {
userId: number;
id: number;
title: string;
completed: boolean;
}
export interface ToDoState {
isLoading: boolean;
currentMember: User | undefined;
memberToDos: ToDo[];
error: string | null;
}
// This should be somewhere reusable
export function setErrorMessage(err: HttpErrorResponse): string {
let errorMessage: string;
if (err.error instanceof ErrorEvent) {
// A client-side or network error occurred. Handle it accordingly.
errorMessage = `An error occurred: ${err.error.message}`;
} else {
// The backend returned an unsuccessful response code.
// The response body may contain clues as to what went wrong,
TS2339: Property 'setMemberToDos' does not exi
src/todo.service.ts:40:32
Click outside, press Esc key, or fix the code to dismiss

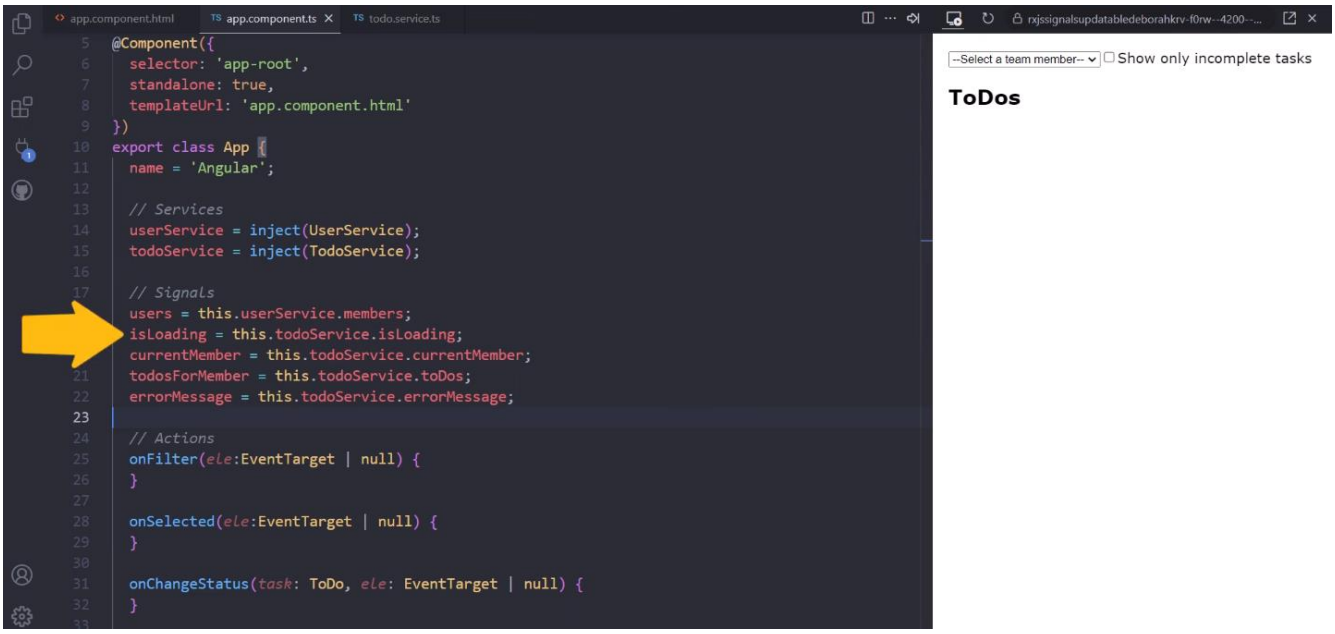

app.component.html
app.component.ts
todo.service.ts 1
export interface ToDoState {
isLoading: boolean;
currentMember: User | undefined;
memberToDos: ToDo[];
error: string | null;
}
// This should be somewhere reusable
export function setErrorMessage(err: HttpErrorResponse): string {
let errorMessage: string;
if (err.error instanceof ErrorEvent) {
// A client-side or network error occurred. Handle it accordingly.
errorMessage = `An error occurred: ${err.error.message}`;
} else {
// The backend returned an unsuccessful response code.
// The response body may contain clues as to what went wrong,
errorMessage = `Backend returned code ${err.status}: ${err.message}`;
}
console.error(err);
return errorMessage;
}
TS2339: Property 'setMemberToDos' does not exi
src/todo.service.ts:40:32
Click outside, press Esc key, or fix the code to dismiss

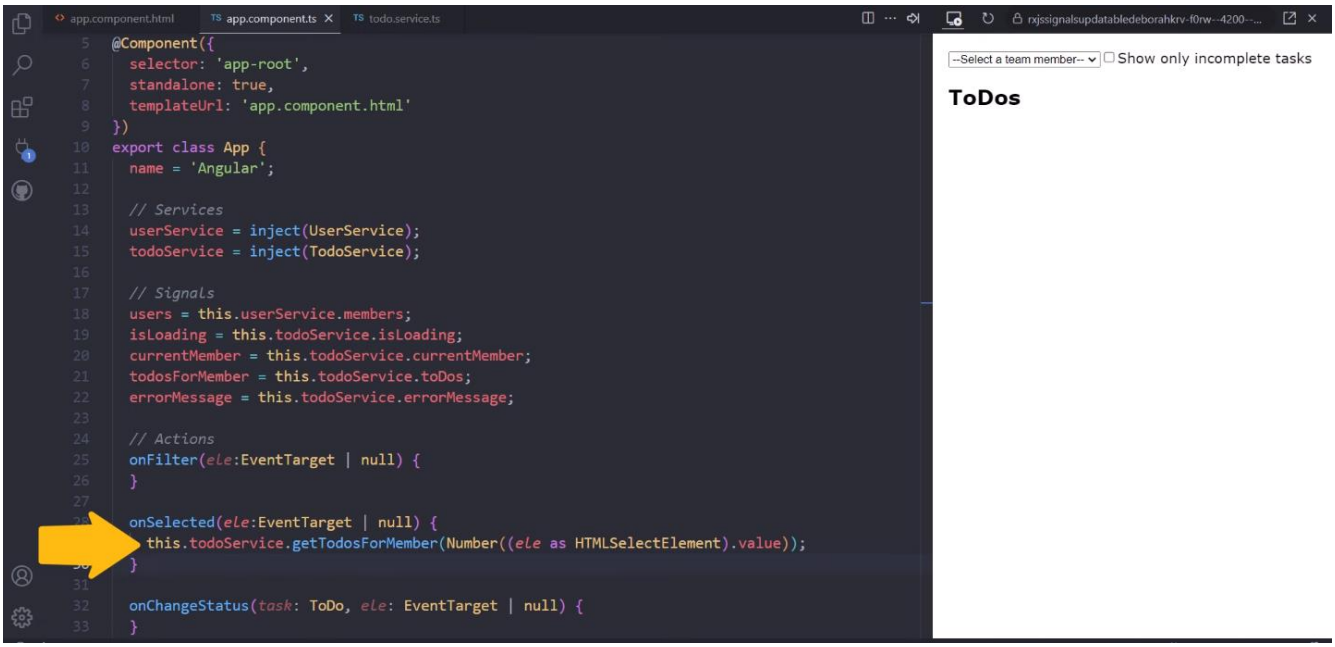


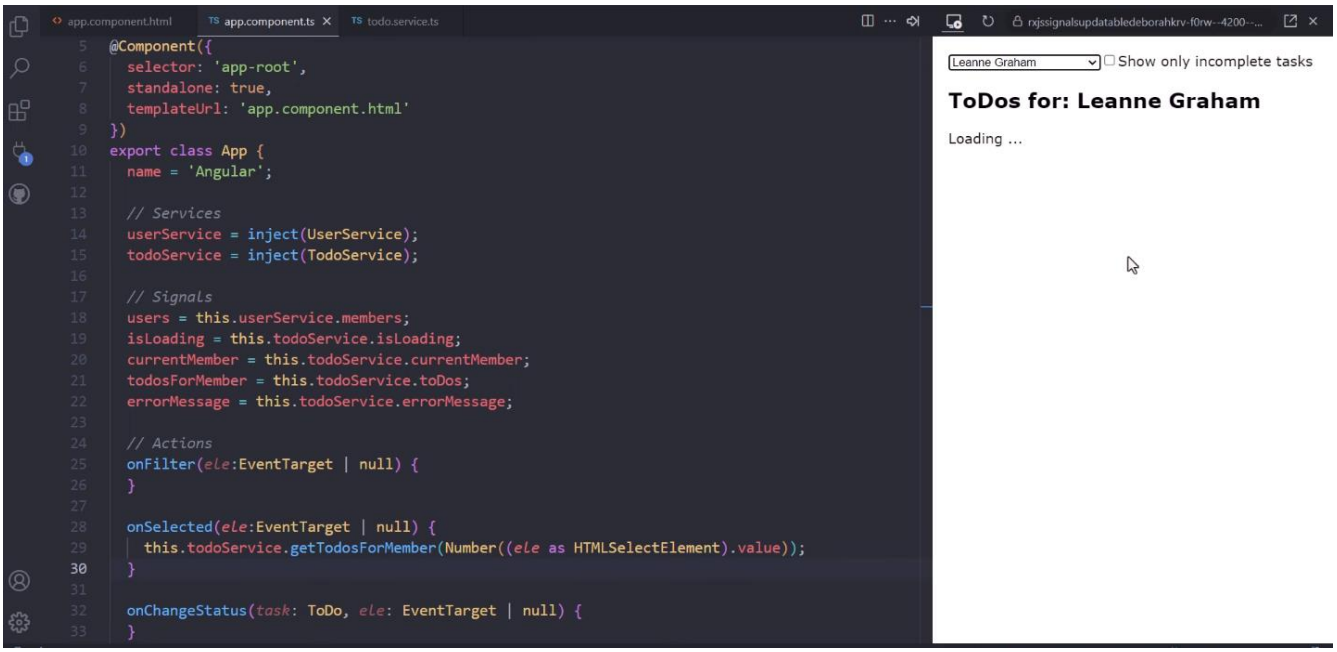


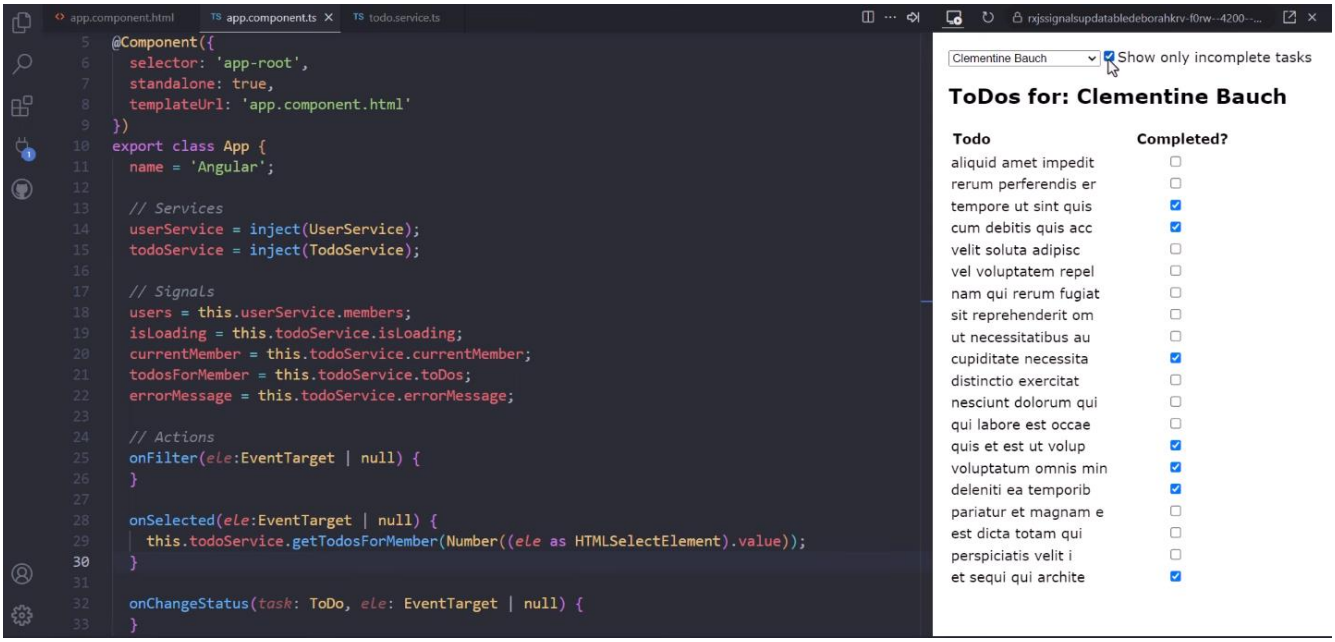
Next, let us modify the component to use our ToDo service

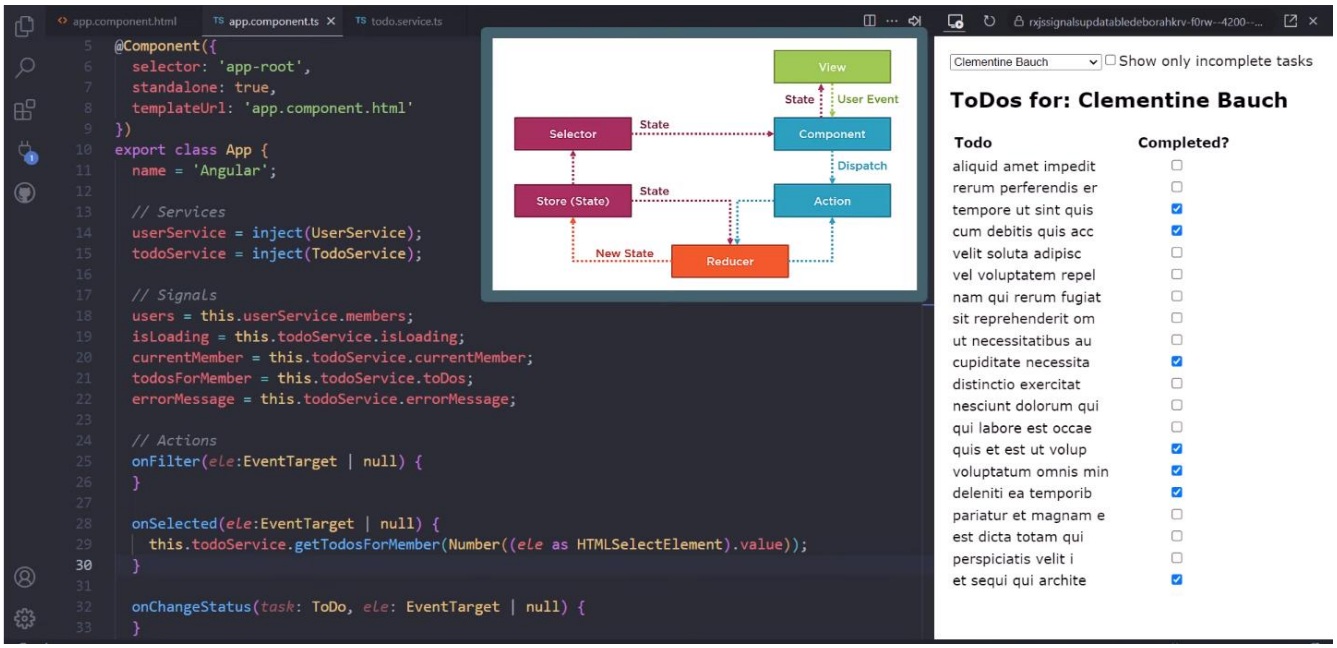

app.component.html
app.component.ts
todo.service.ts
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = signal(false);
currentMember = signal<User | undefined>(undefined);
todosForMember = signal<ToDo[]>([]);
errorMessage = signal(null);
// Actions
onFilter(ele:EventTarget | null) {
}
onSelected(ele:EventTarget | null) {
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
}
--Select a team member--
Show only incomplete tasks
ToDos

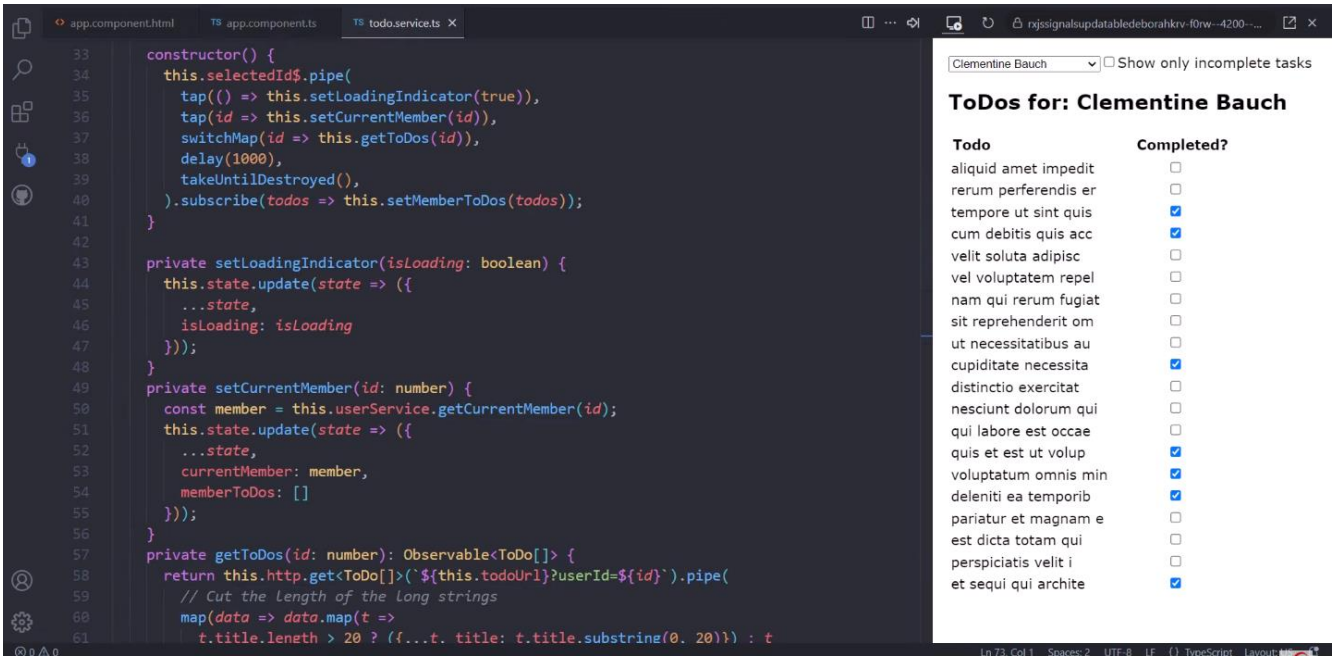

app.component.html
app.component.ts
todo.service.ts
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = this.todoService.isLoading;
currentMember = this.todoService.currentMember;
todosForMember = this.todoService.toDos;
errorMessage = this.todoService.errorMessage;
// Actions
onFilter(ele:EventTarget | null) {
}
onSelected(ele:EventTarget | null) {
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
}
--Select a team member--
Show only incomplete tasks
ToDos

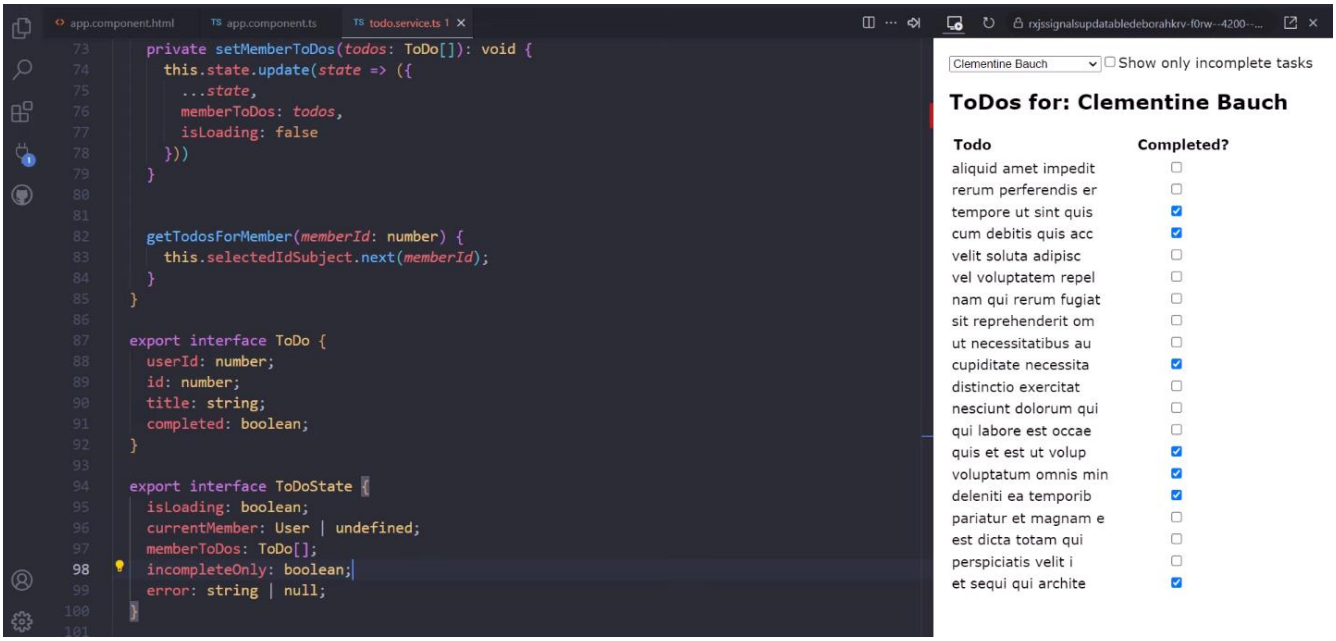

app.component.html
app.component.ts
todo.service.ts
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = this.todoService.isLoading;
currentMember = this.todoService.currentMember;
todosForMember = this.todoService.toDos;
errorMessage = this.todoService.errorMessage;
// Actions
onFilter(ele:EventTarget | null) {
}
onSelected(ele:EventTarget | null) {
this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
}
--Select a team member--
Show only incomplete tasks
ToDos

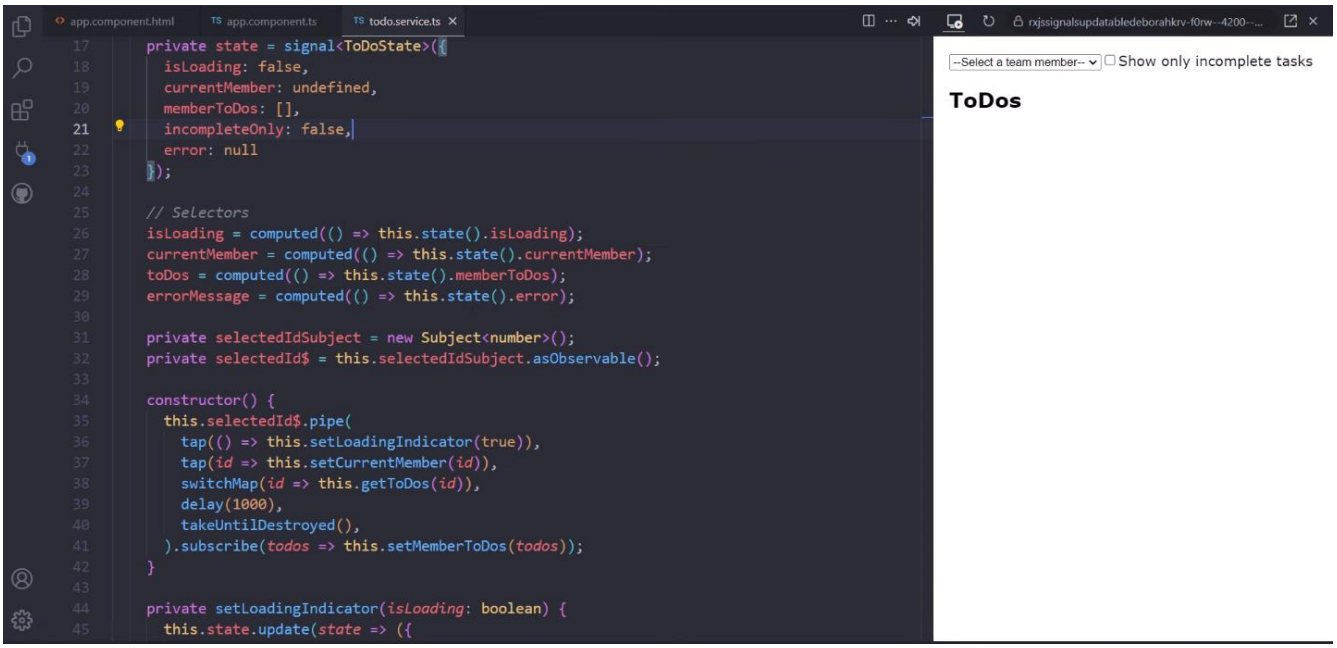

app.component.html
app.component.ts
todo.service.ts
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = this.todoService.isLoading;
currentMember = this.todoService.currentMember;
todosForMember = this.todoService.toDos;
errorMessage = this.todoService.errorMessage;
// Actions
onFilter(ele:EventTarget | null) {
}
onSelected(ele:EventTarget | null) {
this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
}
Leanne Graham
Show only incomplete tasks
ToDos for: Leanne Graham
Loading ...

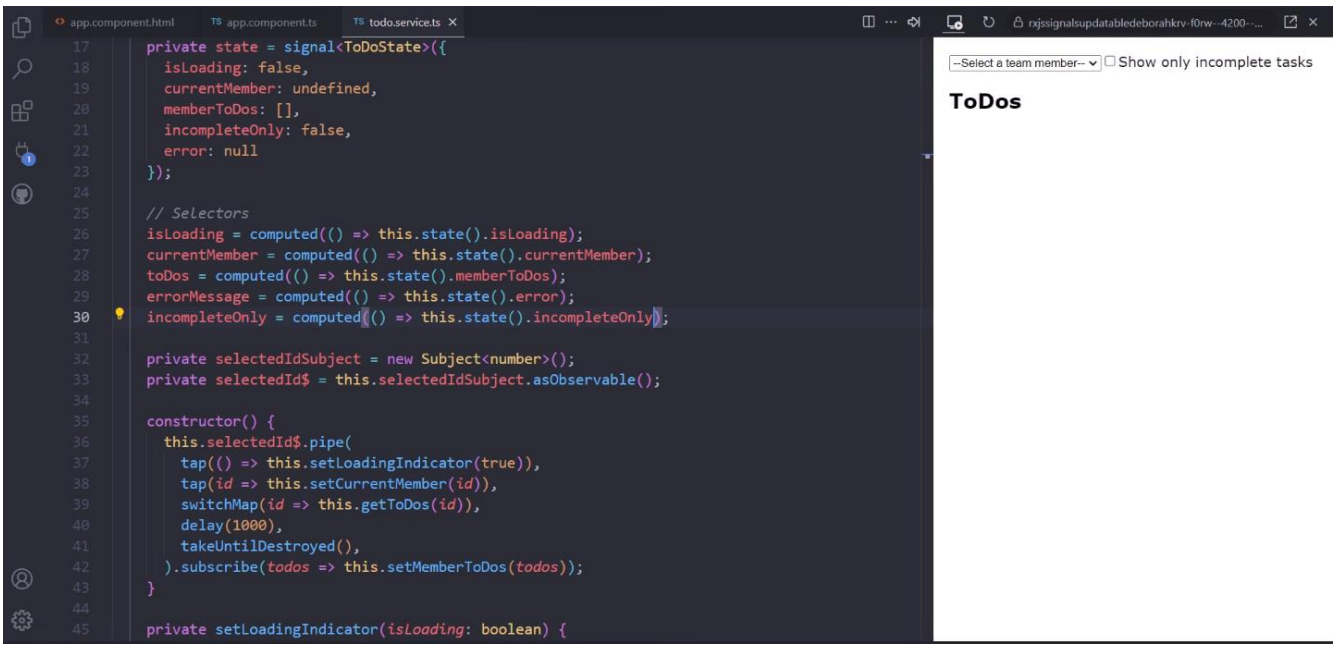

app.component.html
app.component.ts
todo.service.ts
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = this.todoService.isLoading;
currentMember = this.todoService.currentMember;
todosForMember = this.todoService.toDos;
errorMessage = this.todoService.errorMessage;
// Actions
onFilter(ele:EventTarget | null) {
}
onSelected(ele:EventTarget | null) {
this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
}
Clementine Bauch
Show only incomplete tasks
ToDos for: Clementine Bauch
Todo
Completed?
aliquid amet impedit
rerum perferendis er
tempore ut sint quis
cum debitis quis acc
velit soluta adipisc
vel voluptatem repel
nam qui rerum fugiat
sit reprehenderit om
ut necessitatibus au
cupiditate necessita
distinctio exercitat
nesciunt dolorum qui
qui labore est occae
quis et est ut volup
voluptatum omnis min
deleniti ea temporib
pariatur et magnam e
est dicta totam qui
perspiciatis velit i
et sequi qui archite

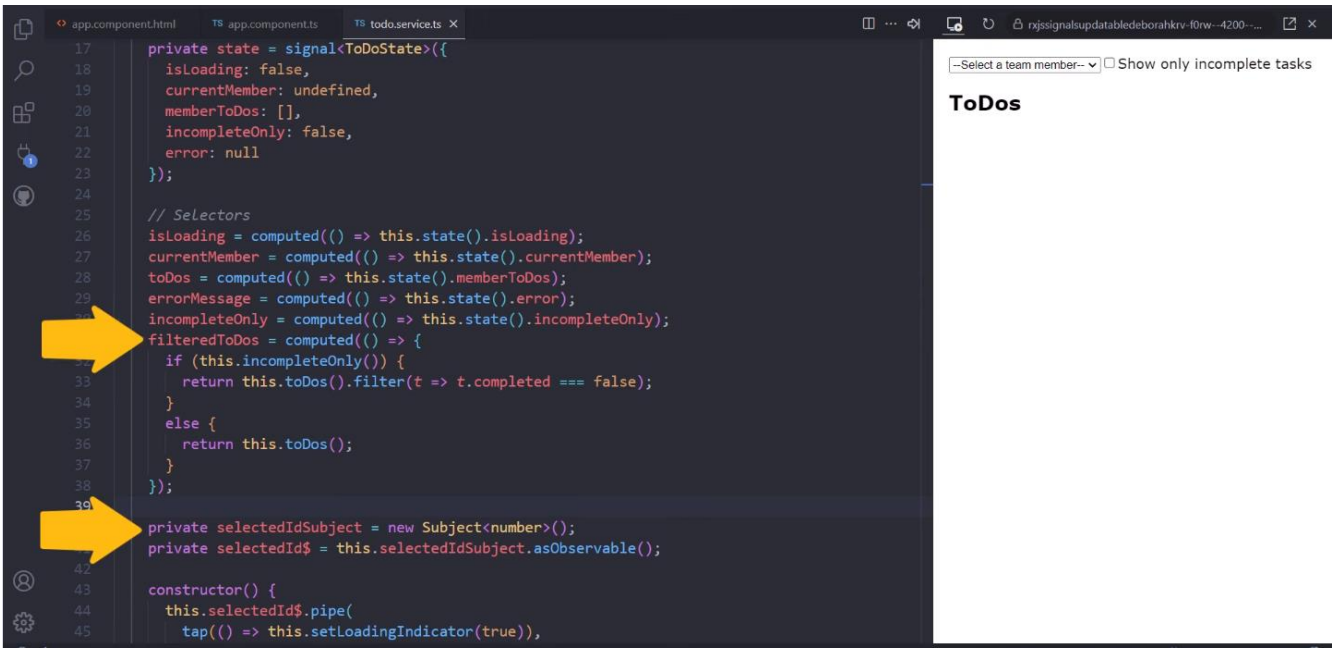

app.component.html
app.component.ts
todo.service.ts
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = this.todoService.isLoading;
currentMember = this.todoService.currentMember;
todosForMember = this.todoService.toDos;
errorMessage = this.todoService.errorMessage;
// Actions
onFilter(ele:EventTarget | null) {
}
onSelected(ele:EventTarget | null) {
this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
}
View
State
User Event
Selector
State
Component
Dispatch
Store (State)
State
Action
New State
Reducer
Clementine Bauch
Show only incomplete tasks
ToDos for: Clementine Bauch
Todo
Completed?
aliquid amet impedit
rerum perferendis er
tempore ut sint quis
cum debitis quis acc
velit soluta adipisc
vel voluptatem repel
nam qui rerum fugiat
sit reprehenderit om
ut necessitatibus au
cupiditate necessita
distinctio exercitat
nesciunt dolorum qui
qui labore est occae
quis et est ut volup
voluptatum omnis min
deleniti ea temporib
pariatur et magnam e
est dicta totam qui
perspiciatis velit i
et sequi qui archite

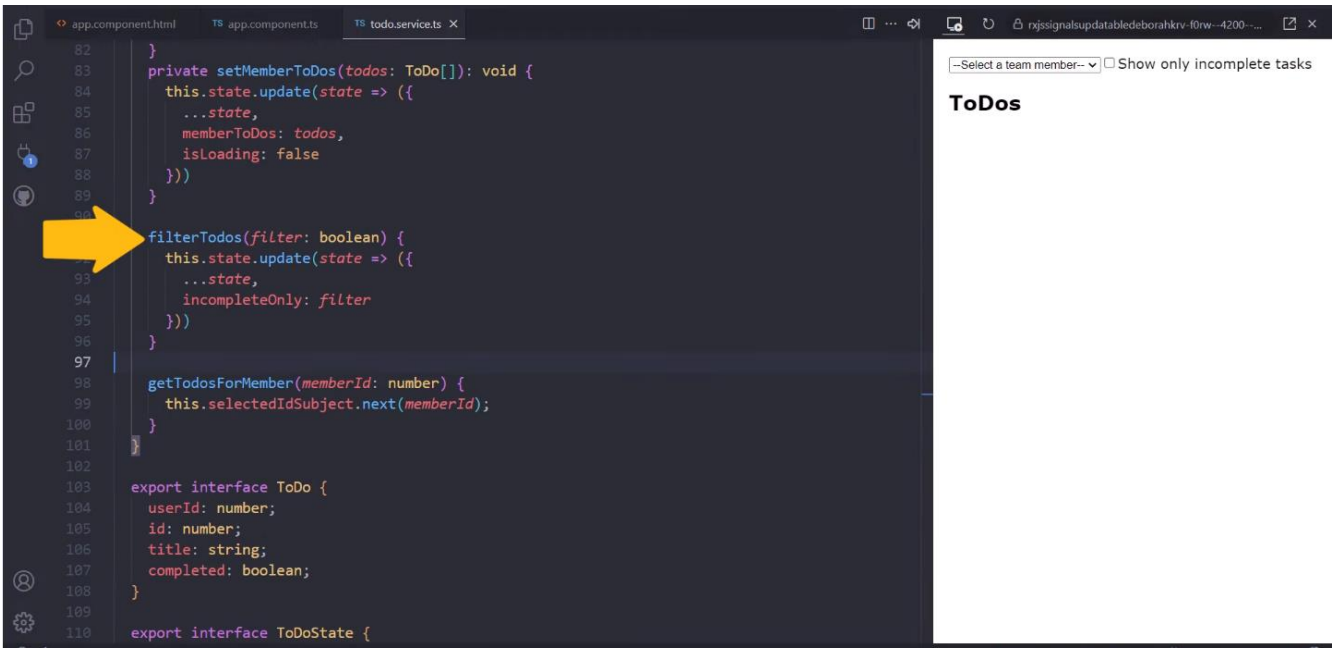

app.component.html
app.component.ts
todo.service.ts
rxjssignalsupdatabledeborahkrv-f0rw--4200--...
Clementine Bauch
Show only incomplete tasks
ToDos for: Clementine Bauch
Todo
Completed?
aliquid amet impedit
rerum perferendis er
tempore ut sint quis
cum debitis quis acc
velit soluta adipisc
vel voluptatem repel
nam qui rerum fugiat
sit reprehenderit om
ut necessitatibus au
cupiditate necessita
distinctio exercitat
nesciunt dolorum qui
qui labore est occae
quis et est ut volup
voluptatum omnis min
deleniti ea temporib
pariatur et magnam e
est dicta totam qui
perspiciatis velit i
et sequi qui archite

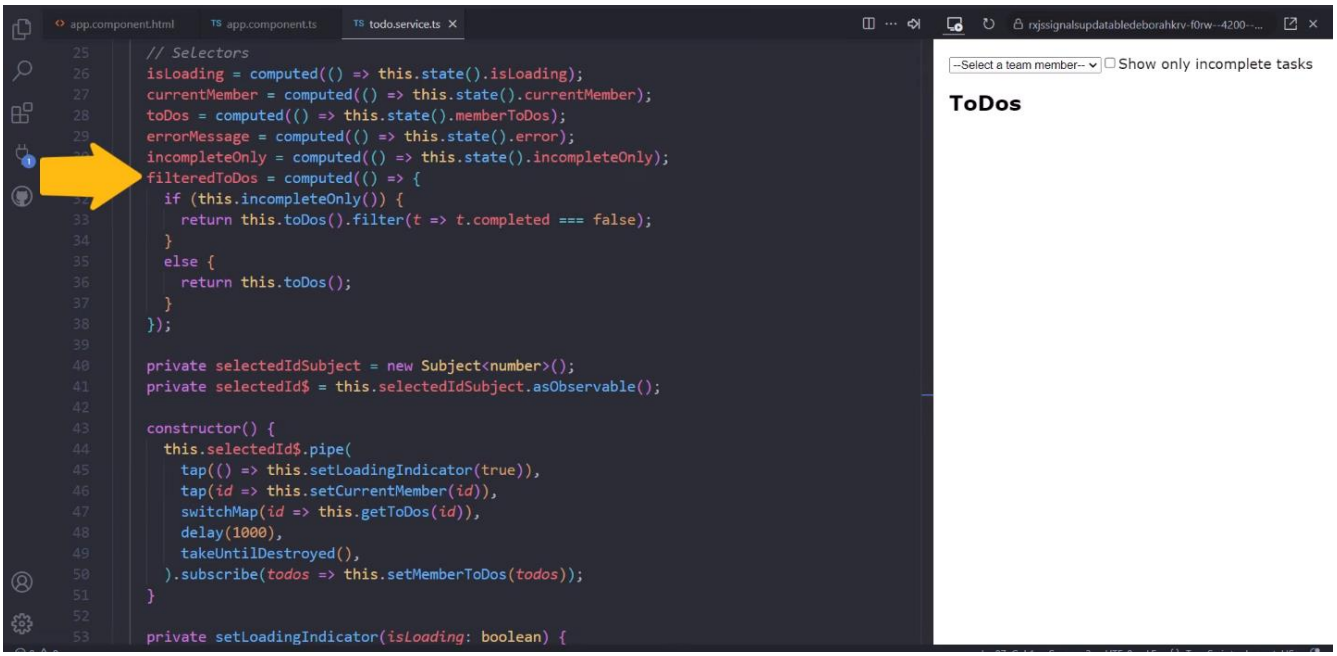

app.component.html
app.component.ts
todo.service.ts 1
rxjssignalsupdatabledeborahkrv-f0rw--4200--...
Clementine Bauch
Show only incomplete tasks
ToDos for: Clementine Bauch
Todo
Completed?
aliquid amet impedit
rerum perferendis er
tempore ut sint quis
cum debitis quis acc
velit soluta adipisc
vel voluptatem repel
nam qui rerum fugiat
sit reprehenderit om
ut necessitatibus au
cupiditate necessita
distinctio exercitat
nesciunt dolorum qui
qui labore est occae
quis et est ut volup
voluptatum omnis min
deleniti ea temporib
pariatur et magnam e
est dicta totam qui
perspiciatis velit i
et sequi qui archite

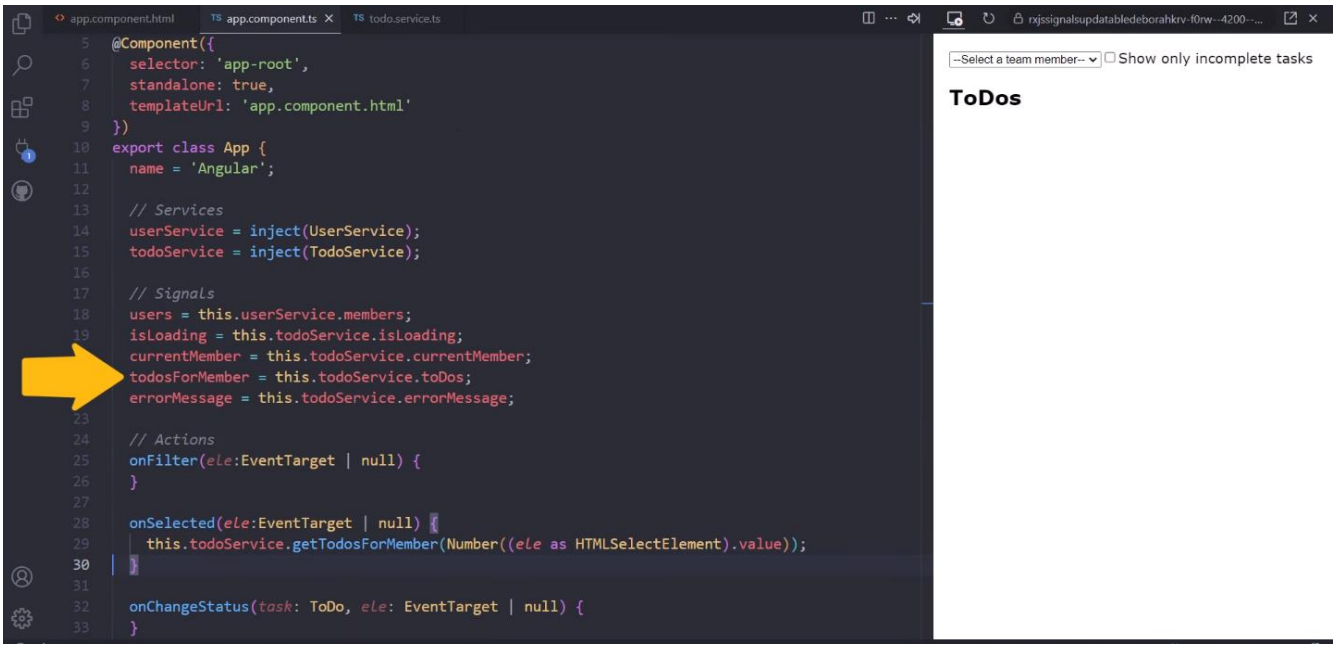

app.component.html
app.component.ts
todo.service.ts
rxjssignalsupdatabledeborahkrv-f0rw---4200--...
--Select a team member--
Show only incomplete tasks
ToDos
private state = signal<ToDoState>({
isLoading: false,
currentMember: undefined,
memberToDos: [],
incompleteOnly: false,
error: null
});
// Selectors
isLoading = computed(() => this.state().isLoading);
currentMember = computed(() => this.state().currentMember);
toDos = computed(() => this.state().memberToDos);
errorMessage = computed(() => this.state().error);
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 => ({

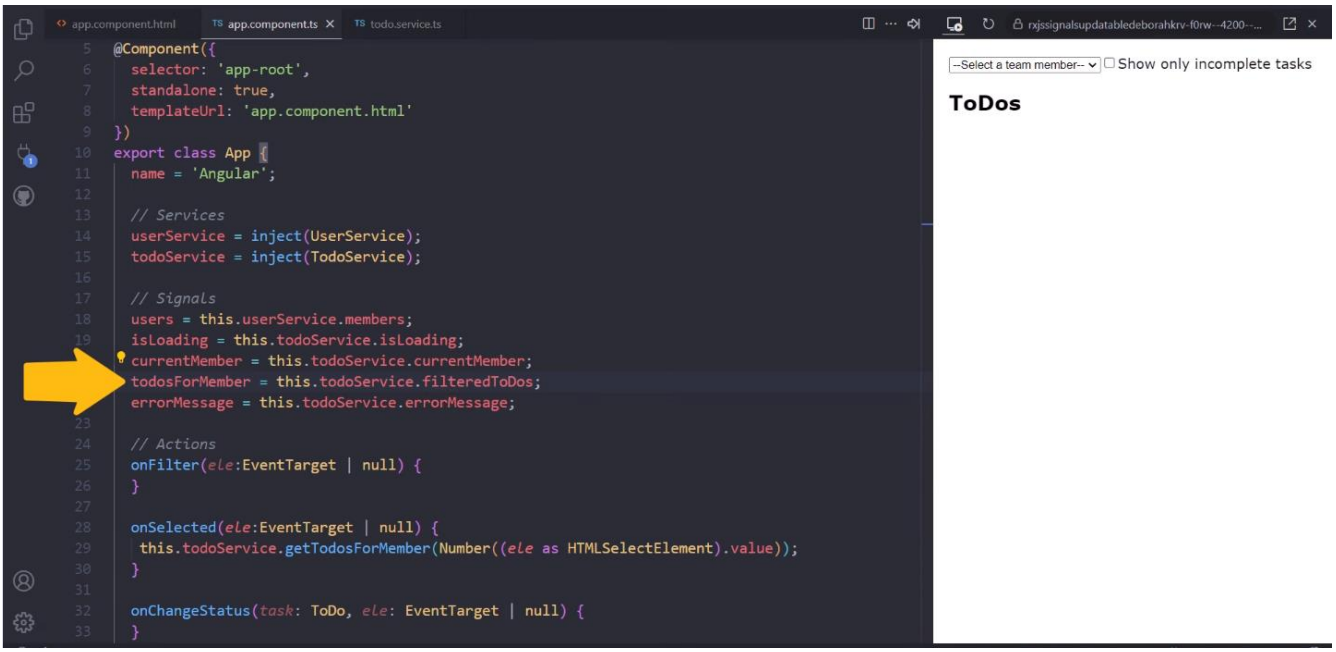

app.component.html
app.component.ts
todo.service.ts
rxjssignalsupdatabledeborahkrv-f0rw---4200--...
--Select a team member--
Show only incomplete tasks
ToDos
private state = signal<ToDoState>({
isLoading: false,
currentMember: undefined,
memberToDos: [],
incompleteOnly: false,
error: null
});
// Selectors
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);
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) {

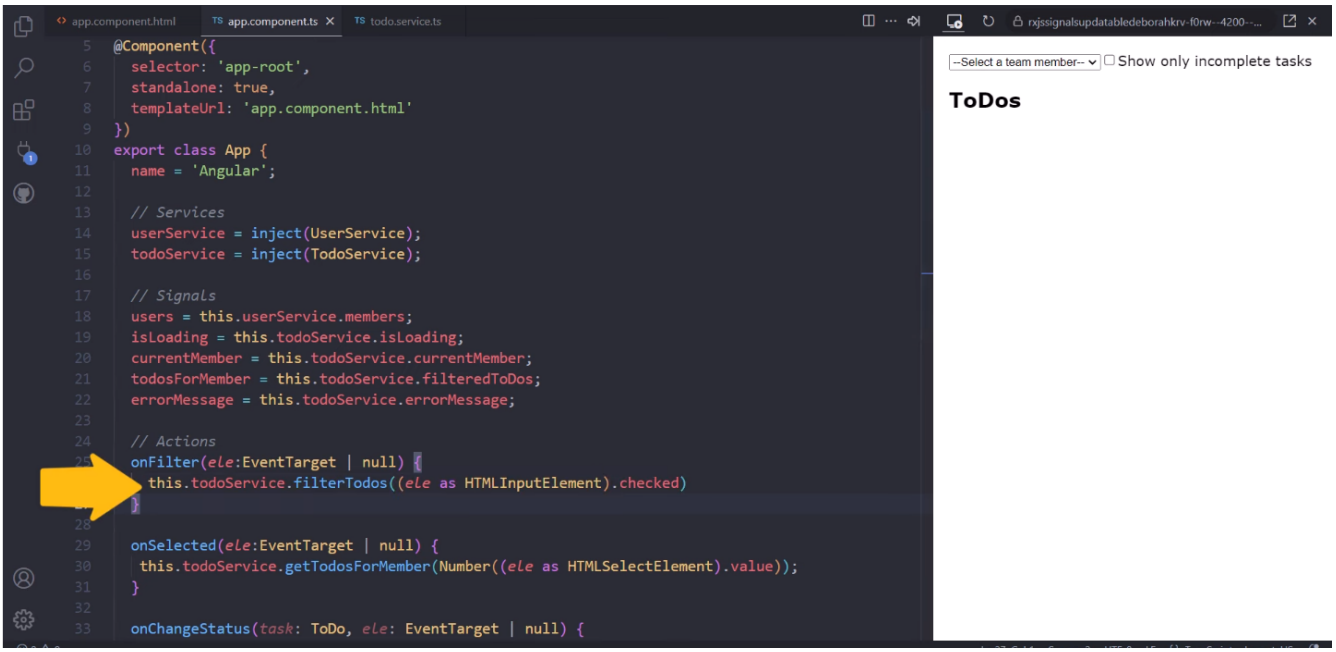

app.component.html
app.component.ts
todo.service.ts
rxjssignalsupdatabledeborahkrv-f0rw---4200-...
--Select a team member--
Show only incomplete tasks
ToDos
private state = signal<ToDoState>({
isLoading: false,
currentMember: undefined,
memberToDos: [],
incompleteOnly: false,
error: null
});
// Selectors
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)),

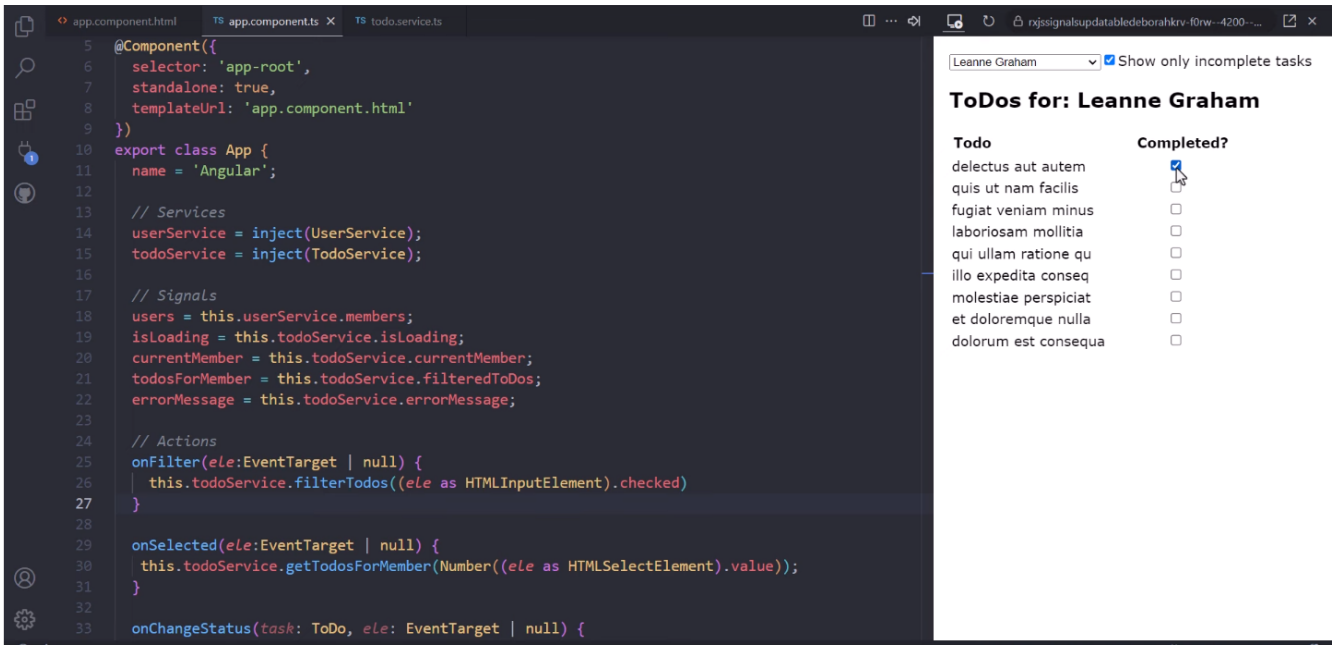

app.component.html
app.component.ts
todo.service.ts
rxjssignalsupdatabledeborahkrv-f0rw---4200-...
--Select a team member--
Show only incomplete tasks
ToDos
}
private setMemberToDos(todos: ToDo[]): void {
this.state.update(state => ({
...state,
memberToDos: todos,
isLoading: false
}))
}
filterTodos(filter: 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 {

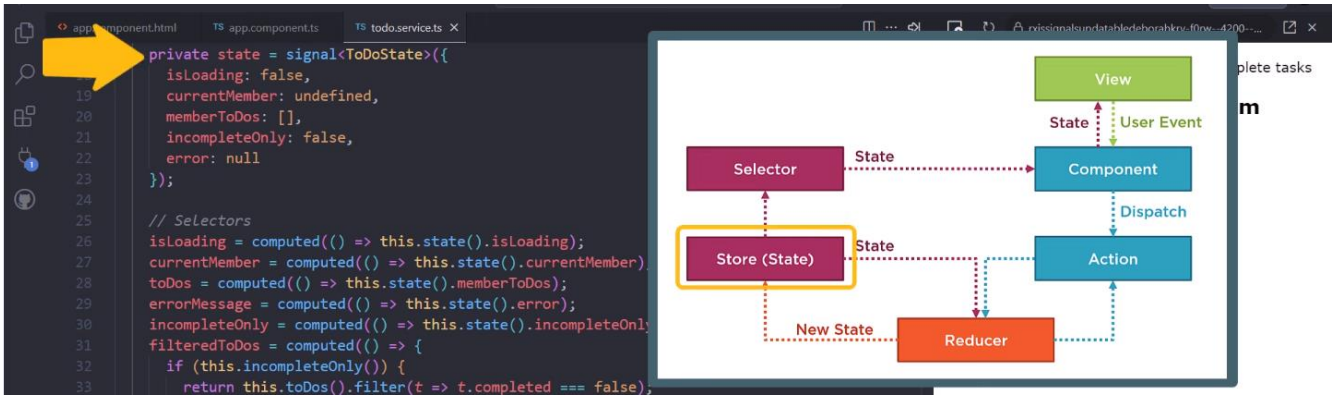

app.component.html
app.component.ts
todo.service.ts
// Selectors
isLoading = computed(() => this.state().isLoading);
currentMember = computed(() => this.state().currentMember);
toDos = computed(() => this.state().memberToDos);
errorMessage = computed(() => this.state().error);
incompleteOnly = computed(() => 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)),
tap(id => this.setCurrentMember(id)),
switchMap(id => this.getToDos(id)),
delay(1000),
takeUntilDestroyed(),
).subscribe(todos => this.setMemberToDos(todos));
}
private setLoadingIndicator(isLoading: boolean) {
--Select a team member--
Show only incomplete tasks
ToDos

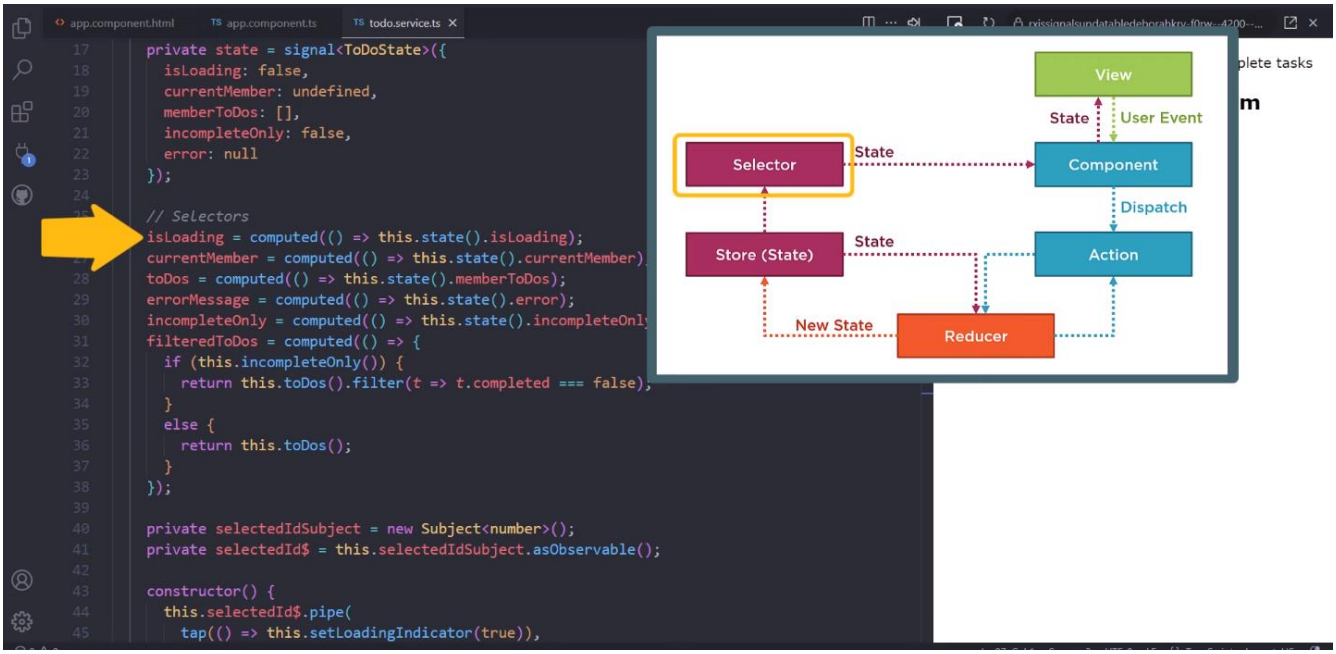

app.component.html
app.component.ts
todo.service.ts
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = this.todoService.isLoading;
currentMember = this.todoService.currentMember;
todosForMember = this.todoService.toDos;
errorMessage = this.todoService.errorMessage;
// Actions
onFilter(ele:EventTarget | null) {
}
onSelected(ele:EventTarget | null) {
this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
}
--Select a team member--
Show only incomplete tasks
ToDos


app.component.html
app.component.ts
todo.service.ts
rxjssignalsupdatabledeborahkrv-f0rw--4200--...
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = this.todoService.isLoading;
currentMember = this.todoService.currentMember;
todosForMember = this.todoService.filteredToDos;
errorMessage = this.todoService.errorMessage;
// Actions
onFilter(ele:EventTarget | null) {
}
onSelected(ele:EventTarget | null) {
this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
}
--Select a team member--
Show only incomplete tasks
ToDos


app.component.html
app.component.ts
todo.service.ts
rxjssignalsupdatabledeborahkrv-f0rw--4200--...
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = this.todoService.isLoading;
currentMember = this.todoService.currentMember;
todosForMember = this.todoService.filteredToDos;
errorMessage = this.todoService.errorMessage;
// Actions
onFilter(ele:EventTarget | null) {
this.todoService.filterTodos((ele as HTMLInputElement).checked)
}
onSelected(ele:EventTarget | null) {
this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
--Select a team member--
Show only incomplete tasks
ToDos

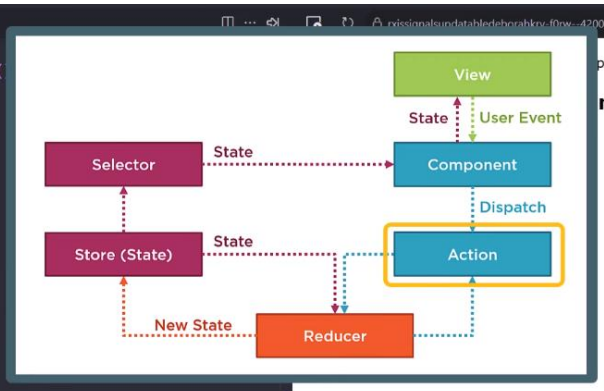

app.component.html
app.component.ts
todo.service.ts
@Component({
selector: 'app-root',
standalone: true,
templateUrl: 'app.component.html'
})
export class App {
name = 'Angular';
// Services
userService = inject(UserService);
todoService = inject(TodoService);
// Signals
users = this.userService.members;
isLoading = this.todoService.isLoading;
currentMember = this.todoService.currentMember;
todosForMember = this.todoService.filteredToDos;
errorMessage = this.todoService.errorMessage;
// Actions
onFilter(ele:EventTarget | null) {
this.todoService.filterTodos((ele as HTMLInputElement).checked)
}
onSelected(ele:EventTarget | null) {
this.todoService.getTodosForMember(Number((ele as HTMLSelectElement).value));
}
onChangeStatus(task: ToDo, ele: EventTarget | null) {
Leanne Graham
Show only incomplete tasks
ToDos for: Leanne Graham
Todo
Completed?
delectus aut autem
quis ut nam facilis
fugiat veniam minus
laboriosam mollitia
qui ullam ratione qu
illo expedita conseq
molestiae perspiciat
et doloremque nulla
dolorum est consequa


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);
View
State
User Event
Selector
State
Component
Dispatch
Store (State)
State
Action
New State
Reducer


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)),
View
State
User Event
Selector
State
Component
Dispatch
Store (State)
State
Action
New State
Reducer

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

```
40  private selectedIdSubject = new Subject<number>();
41  private selectedId$ = this.selectedIdSubject.asObservable(
42
43  constructor() {
44    this.selectedId$.pipe(
45      tap(() => this.setLoadingIndicator(true)),
46      tap(id => this.setCurrentMember(id)),
47      switchMap(id => this.getToDos(id)),
48      delay(1000),
49      takeUntilDestroyed(),
50    ).subscribe(todos => this.setMemberToDos(todos));
51  }
52
53  private setLoadingIndicator(isLoading: boolean) {
54    this.state.update(state => ({
55      ...state,
56      isLoading: isLoading
57    }));
58  }
59  private setCurrentMember(id: number) {
60    const member = this.userService.getCurrentMember(id);
61    this.state.update(state => ({
62      ...state,
63      currentMember: member,
64      memberToDos: []
65    }));
66  }
67  private getToDos(id: number): Observable<ToDo[]> {
68    return this.http.get<ToDo[]>(`${this.todoUrl}?userId=${id}`).pipe(
```

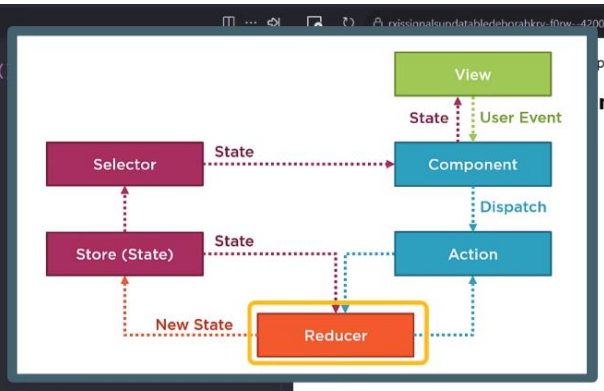


plete tasks

m

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

```
40  private selectedIdSubject = new Subject<number>();
41  private selectedId$ = this.selectedIdSubject.asObservable(
42
43  constructor() {
44    this.selectedId$.pipe(
45      tap(() => this.setLoadingIndicator(true)),
46      tap(id => this.setCurrentMember(id)),
47      switchMap(id => this.getToDos(id)),
48      delay(1000),
49      takeUntilDestroyed(),
50    ).subscribe(todos => this.setMemberToDos(todos));
51  }
52
53  private setLoadingIndicator(isLoading: boolean) {
54    this.state.update(state => ({
55      ...state,
56      isLoading: isLoading
57    }));
58  }
59  private setCurrentMember(id: number) {
60    const member = this.userService.getCurrentMember(id);
61    this.state.update(state => ({
62      ...state,
63      currentMember: member,
64      memberToDos: []
65    }));
66  }
67  private getToDos(id: number): Observable<ToDo[]> {
68    return this.http.get<ToDo[]>(`${this.todoUrl}?userId=${id}`).pipe(
```



plete tasks

m