

# Angular 2+

## Workshop. @NgRx.

### Contents

Explanation of Colors .....	3
Task 01. Setup Schematics .....	4
Task 02. Create a State .....	5
Task 03. Create Actions .....	6
Task 04. Create a Reducer .....	7
Task 05. Provide Store.....	8
Task 06. Inject Store .....	9
Task 07. Reading Data From The Store .....	10
Task 08. Dispatching an Event To The Store.....	11
Task 09. Create Meta-Reducer .....	13
Task 10. Install Redux DevTools Extension.....	14
Task 11. Create Effects Class .....	15
Task 12. Provide Effects .....	16
Task 13. Get Tasks from DataBase .....	17
Task 14. Get Task from DataBase .....	20
Task 15. Update Task in DataBase .....	23
Task 16. Add Task to DataBase .....	25
Task 17. Delete Task from DataBase.....	27
Task 18. Refactor completeTask Action .....	29
Task 19. Effects Interfaces.....	30
Task 20. Feature Selector .....	31
Task 21. State Selector .....	33
Task 22. Router State .....	35
Task 23. Compose Router and Task Selectors .....	37
Task 24. Router Built-in Selectors .....	41
Task 25. Users Store.....	43
Task 26. Navigation By Actions.....	55
Task 27. State Preloading .....	63
Task 28. @ngrx/entity.....	68

Task 29. @ngrx/data .....	71
Task 30. Use Facade .....	76

## Explanation of Colors

Blue color is a snippet of code that you need to fully use to create a new file.

Black color in combination with green or red, means the snippet of code that was added earlier.

Green color is the snippet of code that needs to be added.

Red color is the snippet of code that needs to be removed.

## Task 01. Setup Schematics

1. Run one of the following command from command line:

```
ng config cli.defaultCollection @ngrx/schematics
```

or

```
ng add @ngrx/schematics
```

## Task 02. Create a State

1. Create file **app/core/@ngrx/tasks/tasks.state.ts**. Use the following snippet of code:

```
import { TaskModel } from '../../../../tasks/models/task.model';

export interface TasksState {
  data: ReadonlyArray<TaskModel>;
}

export const initialTasksState: TasksState = {
  data: [
    new TaskModel(1, 'Estimate', 1, 8, 8, true),
    new TaskModel(2, 'Create', 2, 8, 4, false),
    new TaskModel(3, 'Deploy', 3, 8, 0, false)
  ]
};
```

2. Create file **app/core/@ngrx/tasks/index.ts**. Use the following snippet of code:

```
export * from './tasks.state';
```

3. Create file **app/core/@ngrx/app.state.ts**. Use the following snippet of code:

```
import type { TasksState } from './tasks';

export const tasksFeatureKey = 'tasks';
export interface AppState {
  [tasksFeatureKey]: TasksState;
}
```

4. Create file **app/core/@ngrx/index.ts**. Use the following snippet of code:

```
export * from './app.state';
export * from './tasks';
```

## Task 03. Create Actions

1. Create file **app/core/@ngrx/tasks/tasks.actions.ts**. Run the following command from command line:

**> ng g a core/@ngrx/tasks/tasks**

2. Replace the content of **tasks.actions.ts**. Use the following snippet of code:

```
import { createAction, props } from '@ngrx/store';

import type { TaskModel } from '../../../tasks/models/task.model';

export const getTasks = createAction('[Task List Page (App)] GET_TASKS');

export const getTask = createAction(
  '[Add/Edit Task Page (App)] GET_TASK',
  props<{ taskID: number }>()
);

export const createTask = createAction(
  '[Task Form Page] CREATE_TASK',
  props<{ task: TaskModel }>()
);

export const updateTask = createAction(
  '[Task Form Page] UPDATE_TASK',
  props<{ task: TaskModel }>()
);

export const completeTask = createAction(
  '[Task List Page] COMPLETE_TASK',
  props<{ task: TaskModel }>()
);

export const deleteTask = createAction(
  '[Task List Page] DELETE_TASK',
  props<{ task: TaskModel }>()
);
```

3. Make changes to file **app/core/@ngrx/tasks/index.ts**. Use the following snippet of code:

```
export * from './tasks.actions';
```

## Task 04. Create a Reducer

1. Create file **app/core/@ngrx/tasks/tasks.reducer.ts**. Run the following command from the command line:

**ng g r core/@ngrx/tasks/tasks --skip-tests true**

2. Replace the content of **tasks.reducer.ts**. Use the following snippet of code:

```
import { createReducer, on } from '@ngrx/store';

import { initialTasksState } from './tasks.state';
import * as TasksActions from './tasks.actions';

const reducer = createReducer(
  initialTasksState,
  on(TasksActions.getTasks, state => {
    console.log('GET_TASKS action being handled!');
    return { ...state };
  }),
  on(TasksActions.getTask, state => {
    console.log('GET_TASK action being handled!');
    return { ...state };
  }),
  on(TasksActions.createTask, state => {
    console.log('CREATE_TASK action being handled!');
    return { ...state };
  }),
  on(TasksActions.updateTask, state => {
    console.log('UPDATE_TASK action being handled!');
    return { ...state };
  }),
  on(TasksActions.completeTask, state => {
    console.log('COMPLETE_TASK action being handled!');
    return { ...state };
  }),
  on(TasksActions.deleteTask, state => {
    console.log('DELETE_TASK action being handled!');
    return { ...state };
  })
);
```

3. Make changes to file **app/core/@ngrx/tasks/index.ts**. Use the following snippet of code:

```
export * from './tasks.reducer';
```

## Task 05. Provide Store

1. Create **RootStoreModule**. Run the following command from the command line:

```
ng g m core/@ngrx/RootStore --flat -m app.module
```

2. Make changes to **RootStoreModule**. Use the following snippet of code:

```
// 1
// @ngrx
import { StoreModule } from '@ngrx/store';

// 2
@NgModule({
  ...
  imports: [
    CommonModule,
    StoreModule.forRoot({}, {
      // All checks will automatically be disabled in production builds
      runtimeChecks: {
        strictStateImmutability: true,      // default value is true
        strictActionImmutability: true,     // default value is true
        strictStateSerializability: true,   // default value is false
        strictActionSerializability: true,  // default value is false
        strictActionWithinNgZone: true,     // default value is false
        strictActionTypeUniqueness: true    // default value is false
      }
    })
  ]
})
export class RootStoreModule {
  ...
}
```

3. Create **TasksStoreModule**. Run the following command from the command line:

```
ng g m core/@ngrx/tasks/TasksStore --flat -m root-store.module
```

4. Make changes to **TasksStoreModule**. Use the following snippet of code:

```
// 1
import { StoreModule } from '@ngrx/store';
import { tasksReducer } from '../tasks.reducer';
import { tasksFeatureKey } from '../app.state';

// 2
@NgModule({
  ...
  imports: [
    ...
    StoreModule.forFeature(tasksFeatureKey, tasksReducer)
  ]
})
export class TasksStoreModule {}
```



## Task 06. Inject Store

1. Make changes to **TaskListComponent**. Use the following snippet of code:

```
// 1
import { Store } from '@ngrx/store';

// 2
constructor(
  ...
  private store: Store
) { }

// 3
ngOnInit(): void {
  console.log('We have a store! ', this.store);

  ...
}
```

2. Look to the browser console. You have to see the following messages:

We have a store! >Store {...}

## Task 07. Reading Data From The Store

1. Make changes to **TaskListComponent**. Use the following snippet of code:

```
// 1
import type { Observable } from 'rxjs';
import { type TasksState, type AppState, tasksFeatureKey } from '../core/@ngrx';

// 2 - add public property
tasksState$: Observable<TasksState>;

// 3
constructor(
  ...
  private store: Store<AppState>
) {}

// 4
ngOnInit(): void {
  console.log('We have a store! ', this.store);
  this.tasksState$ = this.store.select(tasksFeatureKey);

  this.tasks = this.taskPromiseService.getTasks();
}
```

2. Make changes to **TaskListComponent template**. Use the following snippet of HTML:

```
<app-task *ngFor='let task of tasks | async'
<app-task *ngFor='let task of (tasksState$ | async)?.data'
```

You have to see the list of tasks on the page.

## Task 08. Dispatching an Event To The Store

1. Make changes to file **tasks.reducer.ts**. Use the following snippet of code:

```
// 1
import type { TaskModel } from '../../../../tasks/models/task.model';

// 2
on(TasksActions.completeTask, state => {
  console.log('COMPLETE_TASK action being handled!');
  return { ...state };
}),
on(TasksActions.completeTask, (state, { task }) => {
  console.log('COMPLETE_TASK action being handled!');

  const id = task.id;
  const data = state.data.map(t => {
    if (t.id === id) {
      return { ...task, done: true } as TaskModel;
    }

    return t;
  });

  return {
    ...state,
    data
  };
})
```

2. Make changes to **TaskListComponent**. Use the following snippet of code:

```
// 1
import * as TasksActions from '../../../../core/@ngrx/tasks/tasks.actions';

// 2
onCompleteTask(task: TaskModel): void {
  this.updateTask(task).catch(err => console.log(err));
  this.store.dispatch(TasksActions.completeTask({ task }));
}

// 3
private async updateTask(task: TaskModel) {
  const updatedTask = await this.taskPromiseService.updateTask({
    ...task,
    done: true
  });

  const tasks: TaskModel[] = await this.tasks;
  const index = tasks.findIndex(t => t.id === updatedTask.id);
  tasks[index] = { ...updatedTask };
}
```

Click the button “Done”. You have to see the error in the console: “Detected unserializable action at "task"...”

3. Make changes to file **root-store.module.ts**. Use the following snippet of code:

```
runtimeChecks: {  
  strictStateImmutability: true,      // default value is true  
  strictActionImmutability: true,     // default value is true  
  strictStateSerializability: true,   // default value is false  
  // TaskModel which is used in Actions is not a plain JavaScript Object  
  strictActionSerializability: truefalse, // default value is false  
  strictActionWithinNgZone: true,     // default value is false  
  strictActionTypeUniqueness: true    // default value is false  
}
```

Click the button “Done”. You have to see changed value for the field Done.

## Task 09. Create Meta-Reducer

1. Create file **app/core/@ngrx/meta-reducers.ts**. Use the following snippet of code:

```
import type { ActionReducer, MetaReducer } from '@ngrx/store';
import { environment } from './../../../../environments/environment';

// console.log all actions and state
// export is needed for aot compilation
export function debug(reducer: ActionReducer<any>): ActionReducer<any> {
  return (state, action) => {
    console.log('state before: ', state);
    console.log('action: ', action);

    return reducer(state, action);
  };
}

export const metaReducers: MetaReducer<any>[] = !environment.production
  ? [debug]
  : [];
```

2. Make changes to **RootStoreModule**. Use the following snippet of code:

```
// 1
import { metaReducers } from './meta-reducers';

// 2
StoreModule.forRoot(
  {},
  {
    metaReducers,
    // All checks will automatically be disabled in production builds
    runtimeChecks: {
      strictStateImmutability: true,      // default value is true
      strictActionImmutability: true,     // default value is true
      strictStateSerializability: true,   // default value is false
      // TaskModel which is used in Actions is not a plain JavaScript Object
      strictActionSerializability: false, // default value is false
      strictActionWithinNgZone: true      // default value is false
      strictActionTypeUniqueness: true    // default value is false
    }
  }
)
```

You have to see the state and action objects in the console.

## Task 10. Install Redux DevTools Extension

1. If you don't have extension for Chrome **Redux DevTools Extension installed on your machine**, install it. Manual is here <https://github.com/zalmoxisus/redux-devtools-extension>
2. Make changes to **RootStoreModule**. Use the following snippet of code:

```
// 1
// NgRx
import { StoreModule } from '@ngrx/store';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { environment } from './../../../../environments/environment';

// 2
imports: [
  ...
  // Instrumentation must be imported after importing StoreModule (config is optional)
  !environment.production ? StoreDevtoolsModule.instrument({
    maxAge: 25,          // Retains last 25 states
    autoPause: true      // Pauses recording actions and state changes when the extension
                          // window is not open
  }) : [],
],
```

## Task 11. Create Effects Class

1. Create file **app/core/@ngrx/tasks/tasks.effects.ts**. Run the following command in the command line:

```
ng g ef core/@ngrx/tasks/tasks -m core/@ngrx/tasks/tasks-store.module.ts --skip-tests true
```

2. Make changes to the file **tasks.effects.ts**. Use the following snippet of code:

```
// 1
import { Actions, createEffect } from '@ngrx/effects';

// 2
constructor(
  private actions$: Actions
) {
  console.log('[TASKS EFFECTS]');
}
```

3. Make changes to file **app/core/@ngrx/tasks/index.ts**. Use the following snippet of code:

```
export * from './tasks.effects';
```

## Task 12. Provide Effects

1. Make changes to **RootStoreModule**. Use this snippet of code:

```
// 1
import { EffectsModule } from '@ngrx/effects';

// 2
imports: [
  ...,
  StoreModule.forRoot(...),
  EffectsModule.forRoot([])
]
```

Look to the browser console. You have to see the following messages:

[TASKS EFFECTS]



## Task 13. Get Tasks from DataBase

1. Make changes to file **tasks.state.ts**. Use the following snippet of code

```
// 1
import { type TaskModel } from '../tasks/models/task.model';

// 2
export interface TasksState {
  data: ReadonlyArray<TaskModel>;
  readonly loading: boolean;
  readonly loaded: boolean;
  readonly error: Error | string | null;
}

// 3
export const initialTasksState: State = {
  data: [
    new TaskModel(1, 'Estimate', 1, 8, 8, true),
    new TaskModel(2, 'Create', 2, 8, 4, false),
    new TaskModel(3, 'Deploy', 3, 8, 0, false)
  ],
  loading: false,
  loaded: false,
  error: null
};
```

2. Make changes to file **tasks.actions.ts**. Use the following snippet of code:

```
export const getTasksSuccess = createAction(
  '[Get Tasks Effect] GET_TASKS_SUCCEESS',
  props<{ tasks: TaskModel[] }>()
);
export const getTasksError = createAction(
  '[Get Tasks Effect] GET_TASKS_ERROR',
  props<{ error: Error | string | null }>()
);
```

3. Make changes to file **tasks.reducer.ts**. Use the following snippet of code:

```
// 1
on(TasksActions.getTasks, state => {
  console.log('GET_TASKS action being handled!');
  return {...state};
  return {
    ...state,
    loading: true
  };
}),

on(TasksActions.getTasksSuccess, (state, { tasks }) => {
  console.log('GET_TASKS_SUCCESS action being handled!');
  const data = [...tasks];
  return {
    ...state,
    data,
  };
}),
```

```

        loading: false,
        loaded: true
    });
  })),
  on(TasksActions.getTasksError, (state, { error }) => {
    console.log('GET_TASKS_ERROR action being handled!');
    return {
      ...state,
      loading: false,
      loaded: false,
      error
    };
  })),
});

```

4. Make changes to file **tasks.effects.ts**. Use the following snippet of code:

```

// 1
import { Actions, createEffect, ofType } from '@ngrx/effects';
import { type Action } from '@ngrx/store';
import { type Observable, switchMap } from 'rxjs';

import { TaskPromiseService } from '../../tasks/services';
import * as TasksActions from './tasks.actions';

// 2
constructor(
  private actions$: Actions,
  private taskPromiseService: TaskPromiseService
) {
  console.log('[TASKS EFFECTS]');
}

// 3
getTasks$: Observable<Action> = createEffect(() =>
  this.actions$.pipe(
    ofType(TasksActions.getTasks),
    switchMap(action =>
      // Notice!
      // If you have a connection to the Firebase,
      // the stream will be infinite - you have to unsubscribe
      // This can be performed following this pattern
      // this.taskObservableService
      //      .getTasks()
      //
    ).pipe(takeUntil(this.actions$.pipe(ofType(TasksActions.TaskListComponentIsDestroyed)))
      // If you use HttpClient, the stream is finite,
      // so you have no needs to unsubscribe
      this.taskPromiseService
        .getTasks()
        .then(tasks => TasksActions.getTasksSuccess({ tasks }))
        .catch(error => TasksActions.getTasksError({ error }))
    )
  )
);

```

5. Make changes to **TaskListComponent**. Use the following snippet of code:

```

// 1
import { TaskPromiseService } from '../services';

// 2
tasks!: Promise<Array<TaskModel>>;

// 3
constructor(
    private taskPromiseService: TaskPromiseService,
    ...
) {}

// 4
ngOnInit(): void {
    console.log('We have a store! ', this.store);
    this.tasksState$ = this.store.select('tasks');

    this.store.dispatch(TasksActions.getTasks());
}

// 5
onDeleteTask(task: TaskModel): void {
    // this.taskPromiseService
    //   .deleteTask(task)
    //   .then(() => (this.tasks = this.taskPromiseService.getTasks()))
    //   .catch(err => console.log(err));
}

```

6. Make changes to **TaskListComponent template**. Use the following snippet of HTML:

```

<p *ngIf="(tasksState$ | async)?.error as value">{{value}}</p>

<app-task *ngFor='let task of (tasksState$ | async).data'
    [task]="task"
    (completeTask)="onCompleteTask($event)"
    (editTask)="onEditTask($event)"
    (deleteTask)="onDeleteTask($event)">
</task>

```

7. Look to the browser console.

## Task 14. Get Task from DataBase

1. Make changes to file **tasks.state.ts**. Use the following snippet of code:

```
// 1
export interface TasksState {
  data: ReadonlyArray<TaskModel>;
  selectedTask: Readonly<TaskModel> | null;
  ...
}

// 2
export const initialTasksState: State = {
  tasks: {
    data: [],
    selectedTask: null,
    ...
  }
};
```

2. Make changes to file **tasks.actions.ts**. Use the following snippet of code:

```
export const getTaskSuccess = createAction(
  '[Get Task Effect] GET_TASK_SUCCESS',
  props<{ task: TaskModel }>()
);

export const getTaskError = createAction(
  '[Get Task Effect] GET_TASK_ERROR',
  props<{ error: Error | string | null }>()
);
```

3. Make changes to **TaskFormComponent**. Use the following snippet of code:

```
// 1
import { Component, type OnInit, type OnDestroy } from '@angular/core';
import { Store } from '@ngrx/store';
import { Subject, takeUntil } from 'rxjs';
import { map, switchMap } from 'rxjs';

import { tasksFeatureKey, type AppState, type TasksState } from '../core/@ngrx';
import * as TasksActions from '../core/@ngrx/tasks/tasks.actions';

// 2
export class TaskFormComponent implements OnInit, OnDestroy {

// 3
  private componentDestroyed$: Subject<void> = new Subject<void>();

// 4
  constructor(
    ...
    private store: Store<AppState>
  ) { }

// 5
  const observer = {
    next: (task: TaskModel) => (this.task = { ...task }),
```

```

        error: (err: any) => console.log(err)
    };

    this.route.paramMap
        .pipe(
            switchMap((params: ParamMap) => {
                // notes about "!"
                // params.get() returns string | null, but getTask takes string | number
                // in this case taskID is NOT a path param and can not be null
                if (params.has('taskID')) {
                    return this.taskPromiseService.getTask(params.get('taskID')!);
                } else {
                    return Promise.resolve(undefined);
                }
            }),
            // transform undefined => {}
            map(el => el ? el : {} as TaskModel)
        )
        .subscribe(observer);

    let observer: any = {
        next: (tasksState: TasksState) => {
            this.task = { ...tasksState.selectedTask } as TaskModel;
        },
        error(err: any) {
            console.log(err);
        },
        complete() {
            console.log('Stream is completed');
        }
    };

    this.store.select(tasksFeatureKey)
        .pipe(
            takeUntil(this.componentDestroyed$)
        )
        .subscribe(observer);

    observer = {
        ...observer,
        next: (params: ParamMap) => {
            const id = params.get('taskID');
            if (id) {
                this.store.dispatch(TasksActions.getTask({ taskID: +id }));
            }
        }
    };

    this.route.paramMap.subscribe(observer);

    // 6
    ngOnDestroy(): void {
        this.componentDestroyed$.next();
        this.componentDestroyed$.complete();
    }

```

4. Make changes to file **tasks.effects.ts**. Use the following snippet of code:

```
// 1
import { type Observable, map, switchMap } from 'rxjs';

// 2
getTask$: Observable<Action> = createEffect(() =>
  this.actions$.pipe(
    ofType(TasksActions.getTask),
    map(action => action.taskID),
    switchMap(taskID =>
      this.taskPromiseService
        .getTask(taskID)
        .then(task => TasksActions.getTaskSuccess({ task }))
        .catch(error => TasksActions.getTaskError({ error }))
    )
  )
);
```

5. Make changes to file **tasks.reducer.ts**. Use the following snippet of code:

```
on(TasksActions.getTask, state => {
  console.log('GET_TASK action being handled!');
  return { ...state };
  return {
    ...state,
    loading: true,
    loaded: false
  };
}),

on(TasksActions.getTaskSuccess, (state, { task }) => {
  console.log('GET_TASK action being handled!');
  const selectedTask = { ...task };
  return {
    ...state,
    loading: false,
    loaded: true,
    selectedTask
  };
}),

on(
  TasksActions.getTasksError,
  TasksActions.getTaskError,
  (state, { error }) => {
    console.log('GET_TASKS/TASK_ERROR action being handled!');
    return {
      ...state,
      loading: false,
      loaded: false,
      error
    };
  }
),
```

## Task 15. Update Task in DataBase

1. Make changes to file **tasks.actions.ts**. Use the following snippet of code:

```
export const updateTaskSuccess = createAction(
  '[Update Task Effect] UPDATE_TASK_SUCCESS',
  props<{ task: TaskModel }>()
);

export const updateTaskError = createAction(
  '[Update Task Effect] UPDATE_TASK_ERROR',
  props<{ error: Error | string | null }>()
);
```

2. Make changes to **TaskFormComponent**. Use the following snippet of code:

```
// 1
import { TaskPromiseService } from '../services';

// 2
constructor(
  private taskPromiseService: TaskPromiseService,
  ...
) { }

// 2 onSaveTask method
const method = task.id ? 'updateTask' : 'createTask';
this.taskPromiseService[method](task)
  .then(() => this.onGoBack())
  .catch(err => console.log(err));
if (task.id) {
  this.store.dispatch(TasksActions.updateTask({ task }));
} else {
  this.store.dispatch(TasksActions.createTask({ task }));
}
```

3. Make changes to file **tasks.effects.ts**. Use the following snippet of code:

```
// 1
import { Router } from '@angular/router';
import { type Observable, concatMap, map, switchMap } from 'rxjs';
import { type TaskModel } from '../../tasks/models/task.model';

// 2
constructor(
  private router: Router,
  ...
) {...}

// 3
updateTask$: Observable<Action> = createEffect(() =>
  this.actions$.pipe(
    ofType(TasksActions.updateTask),
    map(action => action.task),
    concatMap((task: TaskModel) =>
      this.taskPromiseService
        .updateTask(task)
    )
  )
);
```

```

        .then((updatedTask: TaskModel) => {
            this.router.navigate(['/home']);
            return TasksActions.updateTaskSuccess({ task: updatedTask });
        })
        .catch(error => TasksActions.updateTaskError({ error }))
    )
}
);

```

4. Make changes to file **tasks.reducer.ts**. Use the following snippet of code:

```

on(TasksActions.updateTaskSuccess, (state, { task }) => {
    console.log('UPDATE_TASK_SUCCESS action being handled!');
    const data = [...state.data];

    const index = data.findIndex(t => t.id === task.id);

    data[index] = { ...task };

    return {
        ...state,
        data
    };
}),

on(TasksActions.updateTaskError, (state, { error }) => {
    console.log('UPDATE_TASK_ERROR action being handled!');
    return {
        ...state,
        error
    };
}),

```



## Task 16. Add Task to DataBase

1. Make changes to file **tasks.actions.ts**. Use the following snippet of code:

```
export const createTaskSuccess = createAction(
  '[Create Task Effect] CREATE_TASK_SUCCESS',
  props<{ task: TaskModel }>()
);

export const createTaskError = createAction(
  '[Create Task Effect] CREATE_TASK_ERROR',
  props<{ error: Error | string | null }>()
);
```

2. Make changes to **TaskFormComponent**. Use the following snippet of code:

```
// 3
ngOnInit(): void {
  this.task = new TaskModel();

  let observer: any = {
    next: (tasksState: TasksState) => {
      this.task = { ...tasksState.selectedTask } as TaskModel;
      if (tasksState.selectedTask) {
        this.task = { ...tasksState.selectedTask } as TaskModel;
      } else {
        this.task = new TaskModel();
      }
    },
    error(err) {
      console.log(err);
    },
    complete() {
      console.log('Stream is completed');
    }
  };
  ...
}
```

3. Make changes to file **tasks.effects.ts**. Use the following snippet of code:

```
createTask$: Observable<Action> = createEffect(() =>
  this.actions$.pipe(
    ofType(TasksActions.createTask),
    map(action => action.task),
    concatMap((task: TaskModel) =>
      this.taskPromiseService
        .createTask(task)
        .then((createdTask: TaskModel) => {
          this.router.navigate(['/home']);
          return TasksActions.createTaskSuccess({ task: createdTask });
        })
        .catch(error => TasksActions.createTaskError({ error }))
    )
  )
```

```
)  
);
```

4. Make changes to file **tasks.reducer.ts**. Use the following snippet of code:

```
on(TasksActions.createTaskSuccess, (state, { task }) => {  
  console.log('CREATE_TASK_SUCCESS action being handled!');  
  const data = [...state.data, { ...task }];  
  
  return {  
    ...state,  
    data  
  };  
}),  
  
on(  
  TasksActions.createTaskError,  
  TasksActions.updateTaskError,  
  (state, { error }) => {  
    console.log('CREATE/UPDATE_TASK_ERROR action being handled!');  
    return {  
      ...state,  
      error  
    };  
  }  
)  
,  
  
on(TasksActions.getTasksSuccess, (state, { tasks }) => {  
  console.log('GET_TASKS_SUCCESS action being handled!');  
  const data = [...tasks];  
  return {  
    ...state,  
    data,  
    loading: false,  
    loaded: true,  
    selectedTask: null  
  };  
}),
```

## Task 17. Delete Task from DataBase

1. Make changes to file **tasks.actions.ts**. Use the following snippet of code:

```
export const deleteTaskSuccess = createAction(
  '[Delete Task Effect] DELETE_TASK_SUCCESS',
  props<{ task: TaskModel }>()
);

export const deleteTaskError = createAction(
  '[Delete Task Effect] DELETE_TASK_ERROR',
  props<{ error: Error | string | null }>()
);
```

2. Make changes to **TaskListComponent**. Use the following snippet of code:

```
onDeleteTask(task: TaskModel): void {
  this.store.dispatch(TasksActions.deleteTask({ task }));
  // this.taskPromiseService
  //   .deleteTask(task)
  //   .then(() => (this.tasks = this.taskPromiseService.getTasks()))
  //   .catch(err => console.log(err));
}
```

3. Make changes to file **tasks.effects.ts**. Use the following snippet of code:

```
deleteTask$: Observable<Action> = createEffect(() =>
  this.actions$.pipe(
    ofType(TasksActions.deleteTask),
    map(action => action.task),
    concatMap((task: TaskModel) =>
      this.taskPromiseService
        .deleteTask(task)
        .then(
          /* method delete for this API returns nothing, so we will use previous task
*/ ) => {
            return TasksActions.deleteTaskSuccess({ task });
          }
        )
        .catch(error => TasksActions.deleteTaskError({ error }))
      )
    )
  );
```

4. Make changes to file **tasks.reducer.ts**. Use the following snippet of code:

```
on(TasksActions.deleteTaskSuccess, (state, { task }) => {
  console.log('DELETE_TASK_SUCCESS action being handled!');
  const data = state.data.filter(t => t.id !== task.id);

  return {
    ...state,
    data
  };
}),
on(
```

```
TasksActions.createTaskError,  
TasksActions.updateTaskError,  
TasksActions.deleteTaskError,  
(state, { error }) => {  
  console.log('CREATE/UPDATE/DELETE_TASK_ERROR action being handled!');  
  return {  
    ...state,  
    error  
  };  
}  
),
```

## Task 18. Refactor completeTask Action

1. Make changes to **TaskListComponent**. Use the following snippet of code:

```
onCompleteTask(task: TaskModel): void {  
  const taskToComplete: TaskModel = { ...task, done: true };  
  this.store.dispatch(TasksActions.completeTask({ task: taskToComplete }));  
}
```

2. Make changes to file **tasks.effects.ts**. Use the following snippet of code:

```
updateTask$: Observable<Action> = createEffect(() =>  
  this.actions$.pipe(  
    ofType(TasksActions.updateTask, TasksActions.completeTask),
```

3. Make changes to file **tasks.reducer.ts**. Use the following snippet of code:

```
// 1  
on(TasksActions.updateTaskSuccess, (state, { task }) => {  
  console.log('UPDATE_TASK_SUCCESS action being handled!');  
  const data = [...state.data];  
  
  const index = data.findIndex(t => t.id === task.id);  
  
  data[index] = { ...task };  
  
  return {  
    ...state,  
    data  
  };  
}),  
  
// 2  
on(TasksActions.completeTask, (state, { task }) => {  
  console.log('COMPLETE_TASK action being handled!');  
  
  const id = task.id;  
  const data = state.data.map(t => {  
    if (t.id === id) {  
      return { ...task, done: true };  
    }  
  
    return t;  
  });  
  
  return {  
    ...state,  
    data  
  };  
})  
);
```

## Task 19. Effects Interfaces

1. Make changes to file **tasks.effects.ts**. Use the following snippet of code:

```
// 1
import { Actions, createEffect, ofType, type OnInitEffects, type OnRunEffects, type
EffectNotification } from '@ngrx/effects';
import { type Observable, concatMap, map, switchMap, takeUntil, tap } from 'rxjs';

// 2
export class TasksEffects implements OnInitEffects, OnRunEffects {

// 3
// Implement this interface to dispatch a custom action after the effect has been added.
// You can listen to this action in the rest of the application
// to execute something after the effect is registered.
ngrxOnInitEffects(): Action {
  console.log('ngrxOnInitEffects is called');
  return { type: '[TasksEffects]: Init' };
}

// Implement the OnRunEffects interface to control the lifecycle
// of the resolved effects.
ngrxOnRunEffects(resolvedEffects$: Observable<EffectNotification>) {
  return resolvedEffects$.pipe(
    tap(val => console.log('ngrxOnRunEffects:', val)),
    // perform until create new task
    // only for demo purpose
    takeUntil(this.actions$.pipe(ofType(TasksActions.createTask)))
  );
}
```

!!! After creating the task (after TasksActions.createTask)all effects stop working. Comment the ngrxOnRunEffects and interface.

## Task 20. Feature Selector

1. Create file **app/core/@ngrx/tasks/tasks.selectors.ts**. Use the following command

**ng g se core/@ngrx/tasks/tasks --skip-tests true**

2. Replace the content of the file **app/core/@ngrx/tasks/tasks.selectors.ts** with the following snippet of code:

```
import { createFeatureSelector } from '@ngrx/store';

import { tasksFeatureKey } from '../app.state';
import { type TasksState } from '../tasks.state';

export const selectTasksState = createFeatureSelector<TasksState>(tasksFeatureKey);
```

3. Make changes to file **app/core/@ngrx/tasks/index.ts**. Use the following snippet of code:

```
export * from './tasks.selectors';
```

4. Make changes to **TaskListComponent**. Use the following snippet of code:

```
// 1
import { type TasksState, type AppState, tasksFeatureKey, selectTasksState } from
'../../../../../core/@ngrx';

// 2
constructor(private router: Router, private store: Store<AppState>) {}

// 3
ngOnInit() {
  console.log('We have a store! ', this.store);
  this.tasksState$ = this.store.select(tasksFeatureKey);
  this.tasksState$ = this.store.select(selectTasksState);

  this.store.dispatch(TasksActions.getTasks());
}
```

5. Make changes to **TaskFormComponent**. Use the following snippet of code:

```
// 1
import { tasksFeatureKey, selectTasksState, type AppState, type TasksState } from
'../../../../../core/@ngrx';

// 2
constructor(
  ...
  private store: Store<AppState>
) {}

// 3
ngOnInit(): void {
  this.store.select(tasksFeatureKeyselectTasksState)
    .pipe(
```

```
        takeUntil(this.componentDestroyed$)
    )
    .subscribe(observer);
...
}
```



## Task 21. State Selector

1. Make changes to file **tasks.selectors.ts**. Use the following snippet of code:

```
// 1
import { createFeatureSelector, createSelector } from '@ngrx/store';

// 2
export const selectTasksData = createSelector(selectTasksState, (state: TasksState) =>
state.data);
export const selectTasksError = createSelector(selectTasksState, (state: TasksState) =>
state.error);
export const selectSelectedTask = createSelector(selectTasksState, (state: TasksState)
=> state.selectedTask);
export const selectTasksLoaded = createSelector(selectTasksState, (state: TasksState) =>
state.loaded);
```

2. Make changes to **TaskListComponent**. Use the following snippet of code:

```
// 1
import {type TasksState, selectTasksState, selectTasksData, selectTasksError } from
'../../../../../core/@ngrx';

// 2
tasksState$: Observable<TasksState>;
tasks$: Observable<ReadonlyArray<TaskModel>>;
tasksError$: Observable<Error | string | null>;

// 3
ngOnInit(): void {
  this.tasksState$ = this.store.select(selectTasksState);
  this.tasks$ = this.store.select(selectTasksData);
  this.tasksError$ = this.store.select(selectTasksError);

  this.store.dispatch(TasksActions.getTasks());
}
```

3. Make changes to **TaskListComponent template**. Use the following snippet of code:

```
// 1
<p *ngIf="(tasksState$ | async).error as value">{{value}}</p>
<p *ngIf="(tasksError$ | async) as value">{{value}}</p>

// 2
<app-task *ngFor='let task of (tasksState$ | async).data'
<app-task *ngFor='let task of (tasks$ | async)'
```

4. Make changes to **TaskFormComponent**. Use the following snippet of code:

```
// 1
import {type TasksState, selectTasksState, selectSelectedTask } from '../../../../../core/@ngrx';

// 2
ngOnInit(): void {
  next: (tasksState: TasksState, task: TaskModel) => {
    if (tasksState.selectedTask) {
```

```

        this.task = { ...tasksState.selectedTask } as TaskModel;
    } else {
        this.task = new TaskModel();
    }
    if (task) {
        this.task = {...task};
    } else {
        this.task = new TaskModel();
    }
},

this.store.select(selectTasksStatesselectSelectedTask),
    .pipe(
        takeUntil(this.componentDestroyed$)
    )
    .subscribe(observer);

...
}

```

## Task 22. Router State

1. Create file **app/core/@ngrx/router/router.state.ts**. Use the following snippet of code:

```
import type { Params, Data } from '@angular/router';
import type { RouterReducerState } from '@ngrx/router-store';

export interface RouterStateUrl {
  url: string;
  queryParams: Params;
  params: Params;
  fragment: string | null;
  data: Data;
}

export interface RouterState {
  router: RouterReducerState<RouterStateUrl>;
}
```

2. Create file **app/core/@ngrx/router/router.reducer.ts**. Use the following snippet of code:

```
import type { ActionReducerMap } from '@ngrx/store';
import { routerReducer } from '@ngrx/router-store';

import type { RouterState } from './router.state';

export const routerReducers: ActionReducerMap<RouterState> = {
  router: routerReducer
};
```

3. Create file **app/core/@ngrx/router/router.custom-serializer.ts**. Use the following snippet of code:

```
import type { ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
import type { RouterStateSerializer } from '@ngrx/router-store';

import type { RouterStateUrl } from './router.state';

export class CustomSerializer implements RouterStateSerializer<RouterStateUrl> {
  serialize(routerState: RouterStateSnapshot): RouterStateUrl {
    let route: ActivatedRouteSnapshot = routerState.root;
    while (route.firstChild) {
      route = route.firstChild;
    }

    const {
      url,
      root: { queryParams }
    } = routerState;

    const { params, fragment, data } = route;

    // Only return an object including the URL, queryParams, params, fragment and data
    // instead of the entire snapshot
    return { url, queryParams, params, fragment, data };
  }
}
```

4. Create file **app/core/@ngrx/router/index.ts**. Use the following snippet of code:

```
export * from './router.custom-serializer';
export * from './router.reducer';
export * from './router.state';
```

5. Make changes to file **app/core/@ngrx/index.ts**. Use the following snippet of code:

```
export * from './router';
```

6. Make changes to **RootStoreModule**. Use the following snippet of code:

```
// 1
import { StoreRouterConnectingModule, RouterState } from '@ngrx/router-store';
import { routerReducers, CustomSerializer } from './router';

// 2
imports: [
  StoreModule.forRoot({routerReducers, {
    metaReducers,
    // All checks will automatically be disabled in production builds
    runtimeChecks: {
      strictStateImmutability: true,           // default value is true
      strictActionImmutability: true,         // default value is true
      // router state is not serializable
      // set false if you don't use CustomSerializer
      strictStateSerializability: true,       // default value is false
      // router action is not serializable
      // set false
      strictActionSerializability: false,     // default value is false
      strictActionWithinNgZone: true,         // default value is false
      strictActionTypeUniqueness: true        // default value is false
    }
  }),
  StoreRouterConnectingModule.forRoot({
    stateKey: 'router',
    routerState: RouterState.Minimal
    // serializer: CustomSerializer // has a priority over routerState
  }),
  ...
],
```

7. Run application. Inspect Router State in NgRx Dev Tool. Uncomment serializer: CustomSerializer and inspect Router State again. Comment serializer: CustomSerializer. Play state changes in app. Uncomment serializer: CustomSerializer.

## Task 23. Compose Router and Task Selectors

1. Create file **app/core/@ngrx/router/router.selectors.ts**. Use the following snippet of code:

```
import { createFeatureSelector } from '@ngrx/store';
import type { RouterReducerState } from '@ngrx/router-store';

import type { RouterStateUrl } from './router.state';

const routerFeatureKey = 'router';
export const selectRouterState =
  createFeatureSelector<RouterReducerState<RouterStateUrl>>(routerFeatureKey);
```

2. Make changes to file **app/core/@ngrx/router/index.ts**. Use the following snippet of code:

```
export * from './router.selectors';
```

3. Make changes to file **tasks.selectors.ts**. Use the following snippet of code:

```
// 1
import { selectRouterState } from '../router';
import { TaskModel } from '../tasks/models/task.model';

// 2
export const selectSelectedTaskByUrl = createSelector(
  selectTasksData,
  selectRouterState,
  (tasks, router): TaskModel => {
    const taskID = router.state.params['taskID'];
    if (taskID && Array.isArray(tasks)) {
      return tasks.find(task => task.id === +taskID);
    } else {
      return new TaskModel();
    }
  }
);
```

4. Make changes to **TaskFormComponent**. Use the following snippet of code:

```
// 1
import { ActivatedRoute, type ParamsMap, Router } from '@angular/router';
import { selectSelectedTask, selectSelectedTaskByUrl } from '../core/+store';

// 2
constructor(
  private route: ActivatedRoute,
  ...
) {}

// 3
ngOnInit(): void {
  const let observer: any = {
    next: (task: TaskModel) => {
      if (task) {
        this.task = {...task};
      } else {
        this.task = new TaskModel();
      }
    },
  },
```

```

...}

this.store.select(selectSelectedTaskselectSelectedTaskByUrl)
  .pipe(
    takeUntil(this.componentDestroyed$)
  )
  .subscribe(observer);

observer = {
  ...observer,
  next: (params: ParamMap) => {
    const id = params.get('taskID');
    if (id) {
      this.store.dispatch(TasksActions.getTask({ taskID: +id }));
    }
  }
};

this.route.paramMap.subscribe(observer);

}

```

5. Make changes to file **tasks.state.ts**. Use the following snippet of code:

```

export interface TasksState {
  data: ReadonlyArray<TaskModel>;
  selectedTask: Readonly<TaskModel>;
  ...
}

export const initialTasksState: TasksState = {
  data: [],
  selectedTask: null,
  ...
};

```

6. Make changes to file **tasks.selectors.ts**. Use the following snippet of code:

```

export const selectSelectedTask = createSelector(getTasksState, (state: TasksState) =>
state.selectedTask);

```

7. Make changes to file **tasks.reducer.ts**. Use the following snippet of code:

```

on(TasksActions.getTasksSuccess, (state, { tasks }) => {
  console.log('GET_TASKS_SUCCESS action being handled!');
  const data = [...tasks];
  return {
    ...state,
    data,
    loading: false,
    loaded: true,
    selectedTask: null
  };
});

```

```

    })),

    on(TasksActions.getTask, state => {
      console.log('GET_TASK action being handled!');
      return {
        ...state,
        loading: true,
        loaded: false
      };
    })),

    on(TasksActions.getTaskSuccess, (state, { task }) => {
      console.log('GET_TASK action being handled!');

      const selectedTask = { ...task };
      return {
        ...state,
        loading: false,
        loaded: true,
        selectedTask
      };
    })),

    on(
      TasksActions.getTasksError,
      TasksActions.getTaskError,
      (state, { error }) => {
        console.log('GET_TASKS/TASK_ERROR action being handled!');
        return {
          ...state,
          loading: false,
          loaded: false,
          error
        };
      }
    ),
  ),
);

```

8. Make changes to file **tasks.effects.ts**. Use the following snippet of code:

```

getTask$: Observable<Action> = createEffect(() =>
  this.actions$.pipe(
    ofType(TasksActions.getTask),
    map(action => action.taskID),
    switchMap(taskID =>
      this.taskPromiseService
        .getTask(taskID)
        .then(task => TasksActions.getTaskSuccess({ task }))
        .catch(error => TasksActions.getTaskError({ error }))
    )
  )
);

```

9. Make changes to file **tasks.actions.ts**. Use the following snippet of code:

```

export const getTask = createAction(
  '[Add/Edit Task Page (App)] GET_TASK',

```

```
    props<{ taskID: number }>()
  );

export const getTaskSuccess = createAction(
  '[Get Task Effect] GET_TASK_SUCCESS',
  props<{ task: TaskModel }>()
);

export const getTaskError = createAction(
  '[Get Task Effect] GET_TASK_ERROR',
  props<{ error: Error | string | null }>()
);
```



## Task 24. Router Built-in Selectors

1. Make changes to file **router.selectors.ts**. Use the following snippet of code:

```
// 1
import type { type RouterReducerState, getSelectors } from '@ngrx/router-store';

// 2
export const {
  selectQueryParams, // select the current route query params
  selectRouteParams, // select the current route params
  selectRouteData, // select the current route data
  selectUrl // select the current url
} = getSelectors(selectRouterState);
```

2. Make changes to file **app.component.ts**. Use the following snippet of code:

```
// 1
// @ngrx
import { Store } from '@ngrx/store';
import {
  AppState,
  selectQueryParams,
  selectRouteParams,
  selectRouteData,
  selectUrl
} from './core/@ngrx';

// 2
import { type Subscription, filter, map, switchMap, merge, tap } from 'rxjs';

// 3
constructor(
  ...
  private store: Store
) {}

// 4
ngOnInit(): void {
  ...

  // Router Selectors Demo
  const url$ = this.store.select(selectUrl);
  const queryParams$ = this.store.select(selectQueryParams);
  const routeParams$ = this.store.select(selectRouteParams);
  const routeData$ = this.store.select(selectRouteData);
  const source$ = merge(url$, queryParams$, routeParams$, routeData$);
  source$.pipe(tap(val => console.log(val))).subscribe();
}
```

3. Make changes to **RootStoreModule**. Use the following snippet of code:

```
// 1
StoreRouterConnectingModule.forRoot({
  ...
  //      serializer: CustomSerializer
}
```

```
}),
```

Look to the console.

Comment Router Selectors Demo, uncomment serializer.

## Task 25. Users Store

1. Create file **app/core/@ngrx/users/users.state.ts**. Use the following snippet of code:

```
import { type UserModel } from '../../../../users/models/user.model';

export interface UsersState {
  entities: Readonly<{ [id: number]: UserModel }>;
  originalUser: Readonly<UserModel> | null;
  readonly loading: boolean;
  readonly loaded: boolean;
  readonly error: Error | string | null;
}

export const initialUsersState: UsersState = {
  entities: {},
  originalUser: null,
  loading: false,
  loaded: false,
  error: null
};
```

1. Create a file **app/core/@ngrx/users/index.ts**. Use the following snippet of code:

```
export * from './users.state';
```

2. Create file **app/core/@ngrx/users/users.actions.ts**. Run the following command from command line:

**ng g a core/@ngrx/users/users -c true --skip-tests true**

3. Replace the content of **users.actions.ts**. Use the following snippet of code:

```
import { createAction, props } from '@ngrx/store';

import { type UserModel } from '../../../../users/models/user.model';

export const getUsers = createAction('[Users Page (App)] GET_USERS');
export const getUsersSuccess = createAction(
  '[Get Users Effect] GET_USERS_SUCCEESS',
  props<{ users: UserModel[] }>()
);
export const getUsersError = createAction(
  '[Get Users Effect] GET_USERS_ERROR',
  props<{ error: Error | string | null }>()
);

export const createUser = createAction(
  '[Add/Edit User Page] CREATE_USER',
  props<{ user: UserModel }>()
);

export const createUserSuccess = createAction(
  '[Create User Effect] CREATE_USER_SUCCESS',
  props<{ user: UserModel }>()
);
```

```

export const createUserError = createAction(
  '[Create User Effect] CREATE_USER_ERROR',
  props<{ error: Error | string | null }>()
);

export const updateUser = createAction(
  '[Add/Edit User Page] UPDATE_USER',
  props<{ user: UserModel }>()
);

export const updateUserSuccess = createAction(
  '[Update User Effect] UPDATE_USER_SUCCESS',
  props<{ user: UserModel }>()
);

export const updateUserError = createAction(
  '[Update User Effect] UPDATE_USER_ERROR',
  props<{ error: Error | string | null }>()
);

export const deleteUser = createAction(
  '[User List Page] DELETE_USER',
  props<{ user: UserModel }>()
);

export const deleteUserSuccess = createAction(
  '[Delete User Effect] DELETE_USER_SUCCESS',
  props<{ user: UserModel }>()
);

export const deleteUserError = createAction(
  '[Delete User Effect] DELETE_USER_ERROR',
  props<{ error: Error | string | null }>()
);

export const setOriginalUser = createAction(
  '[Add/Edit User Page (App)] SET_ORIGINAL_USER',
  props<{ user: UserModel }>()
);

```

4. Make changes to file **app/core/@ngrx/users/index.ts**. Use the following snippet of code:

```
export * from './users.actions';
```

5. Create file **app/core/@ngrx/users/users.reducer.ts**. Run the following command from the command line:

```
ng g r core/@ngrx/users/users -c true --skip-tests true
```

6. Replace the content of **users.reducer.ts**. Use the following snippet of code:

```

import { createReducer, on } from '@ngrx/store';

import { initialUsersState } from './users.state';
import { type UserModel } from './../../../../users/models/user.model';
import * as UsersActions from './users.actions';

```

```

export const usersReducer = createReducer(
  initialUsersState,
  on(UsersActions.getUsers, state => {
    return {
      ...state,
      loading: true
    };
  }),

  on(UsersActions.getUsersSuccess, (state, { users }) => {
    const data = [...users];

    const entities = data.reduce(
      (result: { [id: number]: UserModel }, user: UserModel) => {
        return {
          ...result,
          [user.id!]: user
        };
      },
      {
        ...state.entities
      }
    );

    return {
      ...state,
      loading: false,
      loaded: true,
      entities
    };
  }),

  on(UsersActions.getUsersError, (state, { error }) => {
    return {
      ...state,
      loading: false,
      loaded: false,
      error
    };
  }),

  on(
    UsersActions.createUserSuccess,
    UsersActions.updateUserSuccess,
    (state, { user }) => {
      const createdUpdatedUser = { ...user };
      const entities = {
        ...state.entities,
        [createdUpdatedUser.id!]: createdUpdatedUser
      };
      const originalUser = { ...createdUpdatedUser };

      return {
        ...state,
        entities,
        originalUser
      };
    }
  )
);

```

```

    }),
    on(
      UsersActions.createUserError,
      UsersActions.updateUserError,
      UsersActions.deleteUserError,
      (state, { error }) => {
        return {
          ...state,
          error
        };
      }
    ),

    on(UsersActions.deleteUserSuccess, (state, { user }) => {
      const { [user.id!]: removed, ...entities } = state.entities;

      return {
        ...state,
        entities
      };
    }),

    on(UsersActions.setOriginalUser, (state, { user }) => {
      const originalUser = { ...user };

      return {
        ...state,
        originalUser
      };
    })
  );

```

7. Make changes to file **app/core/@ngrx/users/index.ts**. Use the following snippet of code:

```
export * from './users.reducer';
```

8. Make changes to file **app/core/@ngrx/app.state.ts**. Use the following snippet of code:

```

// 1
import type { UsersState } from './users';

// 2
export const usersFeatureKey = 'users';

// 3
export interface AppState {
  [tasksFeatureKey]: TasksState;
  [usersFeatureKey]: UsersState;
}

```

9. Create file **app/core/@ngrx/users/users-store.module.ts**. Run the following command in the command line:

```
ng g m core/@ngrx/users/UsersStore --flat -m root-store.module
```

10. Create file **app/core/@ngrx/users/users.effects.ts**. Run the following command in the command line:

```
ng g ef core/@ngrx/users/users -m core/@ngrx/users/users-store.module.ts -c true --skip-tests true
```

11. Replace the content of **users.effects.ts**. Use the following snippet of code:

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { Actions, createEffect, ofType } from '@ngrx/effects';
import { type Action } from '@ngrx/store';
import { type Observable, of, switchMap, map, catchError, concatMap } from 'rxjs';

import { UserObservableService } from '../../../users/services';
import { type UserModel } from '../../../users/models/user.model';
import * as UsersActions from './users.actions';

@Injectable()
export class UsersEffects {
  constructor(
    private actions$: Actions,
    private userObservableService: UserObservableService,
    private router: Router
  ) {
    console.log('[USERS EFFECTS]');
  }

  getUsers$: Observable<Action> = createEffect(() =>
    this.actions$.pipe(
      ofType(UsersActions.getUsers),
      switchMap(action =>
        this.userObservableService.getUsers().pipe(
          map(users => UsersActions.getUsersSuccess({ users })),
          catchError(error => of(UsersActions.getUsersError({ error })))
        )
      )
    )
  );

  updateUser$: Observable<Action> = createEffect(() =>
    this.actions$.pipe(
      ofType(UsersActions.updateUser),
      map(action => action.user),
      concatMap((user: UserModel) =>
        this.userObservableService.updateUser(user).pipe(
          map(updatedUser => {
            this.router.navigate(['/users', { editedUserID: updatedUser.id }]);
            return UsersActions.updateUserSuccess({ user: updatedUser });
          }),
          catchError(error => of(UsersActions.updateUserError({ error })))
        )
      )
    )
  );
};
```

```

createUser$: Observable<Action> = createEffect(() =>
  this.actions$.pipe(
    ofType(UsersActions.createUser),
    map(action => action.user),
    concatMap((user: UserModel) =>
      this.userObservableService.createUser(user).pipe(
        map(createdUser => {
          this.router.navigate(['/users']);
          return UsersActions.createUserSuccess({ user: createdUser });
        }),
        catchError(error => of(UsersActions.createUserError({ error })))
      )
    )
  )
);

deleteUser$: Observable<Action> = createEffect(() =>
  this.actions$.pipe(
    ofType(UsersActions.deleteUser),
    map(action => action.user),
    concatMap((user: UserModel) =>
      this.userObservableService.deleteUser(user).pipe(
        // Note: json-server doesn't return deleted user
        // so we use user
        map(() => UsersActions.deleteUserSuccess({ user })),
        catchError(error => of(UsersActions.deleteUserError({ error })))
      )
    )
  )
);
}

```

12. Make changes to file **app/core/@ngrx/users/index.ts**. Use the following snippet of code:

```
export * from './users.effects';
```

13. Create file **app/core/@ngrx/users/users.selectors.ts**. Use the following snippet of code:

```

import { createFeatureSelector, createSelector } from '@ngrx/store';

import { usersFeatureKey } from '../app.state';
import { UserModel } from '../../users/models/user.model';
import { selectRouterState } from '../../router/router.selectors';
import { type UsersState } from './users.state';

const selectEntities = (state: UsersState) => state.entities;
const selectOriginalUser = (state: UsersState) => state.originalUser;
const selectLoaded = (state: UsersState) => state.loaded;
const selectLoading = (state: UsersState) => state.loading;
const selectError = (state: UsersState) => state.error;

export const selectUsersState = createFeatureSelector<UsersState>(usersFeatureKey);

const selectUsersEntitites = createSelector(
  selectUsersState,
  selectEntities
);

```



```

export const selectUsersOriginalUser = createSelector(
  selectUsersState,
  selectOriginalUser
);
export const selectUsersLoaded = createSelector(
  selectUsersState,
  selectLoaded
);
export const selectUsersLoading = createSelector(
  selectUsersState,
  selectLoading
);
export const selectUsersError = createSelector(
  selectUsersState,
  selectError
);

/**
 * transform object to array
 */
export const selectUsers = createSelector(
  selectUsersEntitites,
  entities => {
    return Object.keys(entities).map(id => entities[+id]);
  }
);

export const selectEditedUser = createSelector(
  selectUsersEntitites,
  selectRouterState,
  (users, router): UserModel | null => {
    const userID = router.state.params['editedUserID'];
    if (userID && users) {
      return users[userID];
    } else {
      return null;
    }
  }
);

export const selectSelectedUserByUrl = createSelector(
  selectUsersEntitites,
  selectRouterState,
  (users, router): UserModel => {
    const userID = router.state.params['userID'];
    if (userID && users) {
      return users[userID];
    } else {
      return new UserModel(null, '', '');
    }
  }
);

```

14. Make changes to file **app/core/@ngrx/users/index.ts**. Use the following snippet of code

```
export * from './users.selectors';
```

15. Make changes to file **app/core/@ngrx/index.ts**. Use the following snippet of code

```
export * from './users';
```

16. Make changes to **UsersStoreModule**. Use the following snippet of code

```
// 1
import { StoreModule } from '@ngrx/store';

import { usersFeatureKey } from '../app.state';
import { usersReducer } from './users.reducer';

// 2
@NgModule({
  imports: [
    ...
    StoreModule.forFeature(usersFeatureKey, usersReducer),
  ]
})
```

17. Make changes to **UserListComponent**. Use the following snippet of code:

```
// 1
import { ActivatedRoute, type ParamMap, Router } from '@angular/router';
import { UserObservableService } from '../../../services';
import { type Observable, EMPTY, switchMap } from 'rxjs';
import { Store } from '@ngrx/store';
import * as UsersActions from '../../../core/@ngrx/users/users.actions';
import { selectUsers, selectUsersError, selectEditedUser } from '../../../core/@ngrx';
import { type Observable, type Subscription } from 'rxjs';
import { AutoUnsubscribe } from '../../../core/decorators';

// 2
@AutoUnsubscribe('subscription')

// 3
usersError$: Observable<Error | string | null>;
private subscription!: Subscription;
private editedUser!: UserModel | null;

// 4
constructor(
  private userObservableService: UserObservableService,
  private route: ActivatedRoute,
  private store: Store
) { }

// 5
ngOnInit(): void {
  this.users$ = this.userObservableService.getUsers();

  // listen editedUserID from UserFormComponent
  const observer = {
    next: (user: UserModel) => {
```

```

        this.editedUser = { ...user };
        console.log(
            `Last time you edited user ${JSON.stringify(this.editedUser)}`
        );
    },
    error: (err: any) => console.log(err)
});
this.route.paramMap
    .pipe(
        switchMap((params: ParamMap) => {
            return params.get('editedUserID')
                ? this.userObservableService.getUser(+params.get('editedUserID'))
                : of(null);
        })
    )
    .subscribe(observer);
}
ngOnInit(): void {
    this.users$ = this.store.select(selectUsers);
    this.usersError$ = this.store.select(selectUsersError);
    this.store.dispatch(UsersActions.getUsers());

    this.subscription = this.store.select(selectEditedUser).subscribe({
        next: (user: UserModel | null) => {
            this.editedUser = { ...user } as UserModel;
            console.log(
                `Last time you edited user ${JSON.stringify(this.editedUser)}`
            );
        },
        error: err => console.log(err)
    });
}
// 6
onDeleteUser(user: User): void {
    this.users$ = this.userObservableService.deleteUser(user);
    this.store.dispatch(UsersActions.deleteUser({ user }));
}

```

1. Make changes to **UserListComponent template**. Use the following snippet of HTML

```
<p *ngIf="(usersError$ | async) as errorMessage">{{errorMessage}}</p>
```

2. Make changes to **UserFormComponent**. Use the following snippet of code:

```

// 1
import { type Observable, type Subscription, map, of, switchMap } from 'rxjs';
import { AutoUnsubscribe, DialogService, type CanComponentDeactivate } from
    '../core';
import { UserObservableService } from '../services';
import { ActivatedRoute, Router, type Data, type UrlTree } from '@angular/router';
// @Ngrx
import { Store } from '@ngrx/store';
import { selectUsersOriginalUser } from '../core/@ngrx';
import * as UsersActions from '../core/@ngrx/users/users.actions';

// 2

```

```

@AutoUnsubscribe()

// 3
originalUser!: User;
private sub!: Subscription;

// 4
constructor(
    ...
    private store: Store
    private router: Router,
    private userObservableService: UserObservableService
) { }

// 5
ngOnInit(): void {
    this.route.data.pipe(map((data: Data) => data['user'])).subscribe((user: UserModel)
=> {
        this.user = { ...user };
        this.originalUser = { ...user };
    })
}

// 6
onSaveUser(): void {
    ...

    const method = user.id ? 'updateUser' : 'createUser';
    const observer = {
        next: (savedUser: UserModel) => {
            this.originalUser = { ...savedUser };
            user.id
                ? // optional parameter: http://localhost:4200/users;editedUserID=2
                  this.router.navigate(['users', { editedUserID: user.id }])
                : this.onGoBack();
        },
        error: (err: any) => console.log(err)
    };
    this.sub = this.userObservableService[method](user).subscribe(observer);
    if (user.id) {
        this.store.dispatch(UsersActions.updateUser({ user }));
    } else {
        this.store.dispatch(UsersActions.createUser({ user }));
    }
}

// 7
canDeactivate(): Observable<Boolean | UrlTree> | Promise<Boolean | UrlTree> | boolean |
UrlTree {
    const flags = (Object.keys(this.originalUser) as (keyof UserModel)[[]]).map(key => {
        if (this.originalUser[key] === this.user[key]) {
            return true;
        }
        return false;
    });

    if (flags.every(el => el)) {

```

```

        return true;
    }

    // Otherwise ask the user with the dialog service and return its
    // promise which resolves to true or false when the user decides
    return this.dialogService.confirm('Discard changes?');

    const flags: boolean[] = [];

    return this.store.select(selectUsersOriginalUser).pipe(
        switchMap((originalUser: UserModel | null) => {
            (Object.keys(originalUser!) as (keyof UserModel)[]).map(key => {
                if (originalUser![key] === this.user[key]) {
                    flags.push(true);
                } else {
                    flags.push(false);
                }
            });

            if (flags.every(el => el)) {
                return of(true);
            }

            // Otherwise ask the user with the dialog service and return its
            // promise which resolves to true or false when the user decides
            return this.dialogService.confirm('Discard changes?');
        })
    );
}

```

3. Make changes to file **users/guards/user-resolve.guard.ts**. Use the following snippet of code:

```

// 1
// NgRx
import { Store } from '@ngrx/store';
import { selectSelectedUserByUrl } from '../../../core/@ngrx';
import * as UsersActions from '../../../core/@ngrx/users/users.actions';
import { UserObservableService } from '../../../services';
import { Router, type Resolve, type ActivatedRouteSnapshot } from '@angular/router';
import { type Observable, of, EMPTY, catchError, take, switchMap, delay, finalize, tap }
from 'rxjs';

// 2
constructor(
    private userObservableService: UserObservableService,
    private store: Store,
    ...
) {}

// 3
resolve(route: ActivatedRouteSnapshot): Observable<UserModel> {
    console.log('UserResolve Guard is called');

    if (!route.paramMap.has('userID')) {
        return of(new UserModel(null, '', ''));
    }
}

```

```

const id = route.paramMap.get('userID')!;

return this.userArrayService.getUser(id).pipe(
  switchMap((user: UserModel) => {
    if (user) {
      return of(user);
    } else {
      this.router.navigate(['/users']);
      return EMPTY;
    }
  }),
  take(1),
  catchError(() => {
    this.router.navigate(['/users']);
    // catchError MUST return observable
    return EMPTY;
  })
);

resolve(): Observable<UserModel> {
  console.log('UserResolve Guard is called');
  this.spinner.show();

  return this.store.select(selectSelectedUserByUrl).pipe(
    tap(user => this.store.dispatch(UsersActions.setOriginalUser({ user }))),
    delay(2000),
    switchMap((user: UserModel) => {
      if (user) {
        return of(user);
      } else {
        this.router.navigate(['/users']);
        return EMPTY;
      }
    }),
    take(1),
    catchError(() => {
      this.router.navigate(['/users']);
      // catchError MUST return observable
      return EMPTY;
    }),
    finalize(() => this.spinner.hide())
  );
}

```

## Task 26. Navigation By Actions

1. Create file **app/core/@ngrx/router/router.actions.ts**. Run the following command from command line:

```
ng g a core/@ngrx/router/router -c true --skip-tests true
```

2. Replace the content of **router.actions.ts**. Use the following snippet of code:

```
import { createAction, props } from '@ngrx/store';
import { type NavigationExtras } from '@angular/router';

export const forward = createAction('[Router] FORWARD');
export const back = createAction('[Router] BACK');
export const go = createAction(
  '[Router] GO',
  props<{ path: any[]; queryParams?: object; extras?: NavigationExtras }>()
);
```

3. Make changes to file **app/core/@ngrx/router/index.ts**. Use the following snippet of code:

```
export * from './router.actions';
```

4. Create file **app/core/@ngrx/router/router.effects.ts**. Run the following command from command line:

```
ng g ef core/@ngrx/router/router --root true -m core/@ngrx/root-store.module.ts -c true --skip-tests true
```

5. Replace the content of **router.effects.ts**. Use the following snippet of code:

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { Location } from '@angular/common';
import { Actions, createEffect, EffectConfig, ofType } from '@ngrx/effects';
import { tap } from 'rxjs';

import * as RouterActions from './router.actions';

const effectConfig: EffectConfig = {
  dispatch: false
};

@Injectable()
export class RouterEffects {
  constructor(
    private actions$: Actions,
    private router: Router,
    private location: Location
  ) {}

  navigate$ = createEffect(
    () =>
      this.actions$.pipe(
        ofType(RouterActions.go),
        tap(action => {
          const { type: deleted, path, queryParams, extras } = { ...action };
          this.router.navigate(path, { queryParams, ...extras });
        })
      )
  );
```

```

        })
      ),
      effectConfig
    );

    navigateBack$ = createEffect(
      () =>
        this.actions$.pipe(
          ofType(RouterActions.back),
          tap(() => this.location.back())
        ),
      effectConfig
    );

    navigateForward$ = createEffect(
      () =>
        this.actions$.pipe(
          ofType(RouterActions.forward),
          tap(() => this.location.forward())
        ),
      effectConfig
    );
  }
}

```

6. Make changes to file **app/core/@ngrx/router/index.ts**. Use the following snippet of code:

```
export * from './router.effects';
```

7. Make changes to **RootStoreModule**. Use the following snippet of code:

```

// 1
import { RouterEffects } from './router/router.effects';
import { CustomSerializer, routerReducers, RouterEffects } from './router';

```

8. Make changes to file **app/core/@ngrx/tasks/tasks.effects.ts**. Use the following snippet of code:

```

// 1
import * as RouterActions from './../router/router.actions';
import { Router } from '@angular/router';

```

```

// 2
constructor(
  private actions$: Actions,
  private router: Router,
  private taskPromiseService: TaskPromiseService
) {
  console.log('[TASKS EFFECTS]');
}

```

```

// 3
updateTask$: Observable<Action> = createEffect(() =>
  this.actions$.pipe(
    ofType(TasksActions.updateTask),
    map(action => action.task),
    concatMap((task: TaskModel) =>
      this.taskPromiseService
        .updateTask(task)
        .then((updatedTask: TaskModel) => {

```



```

        this.router.navigate(['/home']);
        return TasksActions.updateTaskSuccess({ task: updatedTask });
    })
    .catch(error => TasksActions.updateTaskError({ error })))
    )
    );

// 3
createTask$: Observable<Action> = createEffect(() =>
    this.actions$.pipe(
        ofType(TasksActions.createTask),
        map(action => action.task),
        concatMap((task: TaskModel) =>
            this.taskPromiseService
                .createTask(task)
                .then((createdTask: TaskModel) => {
                    this.router.navigate(['/home']);
                    return TasksActions.createTaskSuccess({ task: createdTask });
                })
                .catch(error => TasksActions.createTaskError({ error })))
        )
    );

// 4
createUpdateTaskSuccess$: Observable<Action> = createEffect(() => {
    return this.actions$.pipe(
        ofType(TasksActions.createTaskSuccess, TasksActions.updateTaskSuccess),
        map(action =>
            RouterActions.go({
                path: ['/home']
            })
        )
    );
});

```

9. Make changes to file **app/@ngrx/effects/users.effects.ts**. Use the following snippet of code:

```

// 1
import { Router } from '@angular/router';
import * as RouterActions from '../router/router.actions';

// 2
constructor(
    private actions$: Actions,
    private router: Router,
    private userObservableService: UserObservableService
) {
    console.log('[USERS EFFECTS]');
}

// 3
updateUser$: Observable<Action> = createEffect(() =>
    this.actions$.pipe(
        ofType(UsersActions.updateUser),

```

```

        map(action => action.user),
        concatMap((user: UserModel) =>
            this.userObservableService.updateUser(user).pipe(
                map(updatedUser => {
                    this.router.navigate(['/users', { editedUserID: updatedUser.id }]);
                    return UsersActions.updateUserSuccess({ user: updatedUser });
                }),
                catchError(error => of(UsersActions.updateUserError({ error })))
            )
        )
    );

// 3
createUser$: Observable<Action> = createEffect(() =>
    this.actions$.pipe(
        ofType(UsersActions.createUser),
        map(action => action.user),
        concatMap((user: UserModel) =>
            this.userObservableService.createUser(user).pipe(
                map(createdUser => {
                    this.router.navigate(['/users']);
                    return UsersActions.createUserSuccess({ user: createdUser });
                }),
                catchError(error => of(UsersActions.createUserError({ error })))
            )
        )
    );

// 4
createUserSuccess$: Observable<Action> = createEffect(() =>
    this.actions$.pipe(
        ofType(UsersActions.createUserSuccess),
        map(action => {
            const path = ['/users'];
            return RouterActions.go({ path });
        })
    );

updateUserSuccess$: Observable<Action> = createEffect(() =>
    this.actions$.pipe(
        ofType(UsersActions.updateUserSuccess),
        map(action => {
            const { user: { id: userID } } = action;
            const path = ['/users', { editedUserID: userID }];
            return RouterActions.go({ path });
        })
    );

```

10. Make changes to **AuthGuard**. Use the following snippet of code:

```

// 1
import { Store } from '@ngrx/store';

```

```

import * as RouterActions from '../@ngrx/router/router.actions';
import { Router } from '@angular/router';
// 2
constructor(
  ...
  private router: Router,
  private store: Store
) { }

// 3
private checkLogin(url: string): boolean | UrlTree {
  ...
  this.router.navigate(['/login'], navigationExtras);
  this.store.dispatch(RouterActions.go({
    path: ['/login'],
    extras: navigationExtras
  }));
}

```

11. Make changes to **TaskListComponent**. Use the following snippet of code:

```

// 1
import { Router } from '@angular/router';
import * as RouterActions from '../@ngrx/router/router.actions';

// 2
constructor(
  private router: Router,
  ...
) { }

// 3
onCreateTask(): void {
  const link = ['/add'];
  this.router.navigate(link);
  this.store.dispatch(RouterActions.go({
    path: ['/add']
  }));
}

// 4
onEditTask(task: TaskModel): void {
  const link = ['/edit', task.id];
  this.router.navigate(link);
  this.store.dispatch(RouterActions.go({
    path: link
  }));
}

```

12. Make changes to **TaskFormComponent**. Use the following snippet of code:

```

// 1
import { Router } from '@angular/router';
import * as RouterActions from '../@ngrx/router/router.actions';

// 2

```

```

constructor(
    private store: Store
    private router: Router
) {}

// 3
onGoBack(): void {
    this.router.navigate(['/home']);
    this.store.dispatch(RouterActions.go({
        path: ['/home']
    }));
}

```

10. Make changes to **UserListComponent**. Use the following snippet of code:

```

// 1
import { Router } from '@angular/router';
import * as RouterActions from '../core/@ngrx/router/router.actions';

// 2
constructor(
    ...
    private router: Router
) { }

// 3
onEditUser(user: UserModel): void {
    const link = ['/users/edit', user.id];
    this.router.navigate(link);
    this.store.dispatch(RouterActions.go({
        path: link
    }));
}

```

11. Make changes to **UserFormComponent**. Use the following snippet of code:

```

// 1
import { Location } from '@angular/common';
import * as RouterActions from '../core/@ngrx/router/router.actions';

// 2
constructor(
    ...
    private location: Location
) { }

// 3
onGoBack(): void {
    this.location.back();
    this.store.dispatch(RouterActions.back());
}

```

12. Make changes to **UserResolveGuard**. Use the following snippet of code:

```

// 1
import { Router, type Resolve } from '@angular/router';
import * as RouterActions from '../core/@ngrx/router/router.actions';

```

```

// 2
constructor(
    private router: Router,
    ...
) {}

// 3
return this.store.select(selectSelectedUserByUrl).pipe(
    tap(user => this.store.dispatch(UsersActions.setOriginalUser({ user }))),
    delay(2000),
    switchMap(user => {
        if (user) {
            return of(user);
        } else {
            this.router.navigate(['/users']);
            this.store.dispatch(RouterActions.go({
                path: ['/users']
            }));
            return EMPTY;
        }
    }),
    take(1),
    catchError(() => {
        this.router.navigate(['/users']);
        this.store.dispatch(RouterActions.go({
            path: ['/users']
        }));
        return EMPTY;
    }),
    finalize(() => this.spinner.hide())
);

```

13. Make changes to **MessagesComponent**. Use the following snippet of code:

```

// 1
import { Router } from '@angular/router';
import { Store } from '@ngrx/store';
import * as RouterActions from '../core/@ngrx/router/router.actions';

// 2
constructor(
    public messagesService: MessagesService,
    private router: Router,
    private store: Store
) { }

// 3
onClose(): void {
    this.router.navigate([{ outlets: { messages: null } }]);
    this.store.dispatch(RouterActions.go({
        path: [{ outlets: { messages: null } }]
    }));
    this.messagesService.isDisplayed = false;
}

```

14. Make changes to **AppComponent**. Use the following snippet of code:

```
// 1
import * as RouterActions from './core/@ngrx/router/router.actions';

// 2
onDisplayMessages(): void {
  this.router.navigate([{ outlets: { messages: ['messages'] } }]);
  this.store.dispatch(RouterActions.go({
    path: [{ outlets: { messages: ['messages'] } } ]
  }));
  this.messagesService.isDisplayed = true;
}
```

15. Make changes to **LoginComponent**. Use the following snippet of code:

```
// 1
import { Router, type NavigationExtras } from '@angular/router';
import { Store } from '@ngrx/store';
import * as RouterActions from './../../core/@ngrx/router/router.actions';

// 2
constructor(
  public authService: AuthService,
  private router: Router,
  private store: Store
) {}

// 3
this.router.navigate([redirect], navigationExtras);
this.store.dispatch(RouterActions.go({
  path: [redirect],
  extras: navigationExtras
}));
```

## Task 27. State Preloading

1. Create file **app/tasks/guards/tasks-state-preloading.guard.ts**. Run the following command from the command line:

**ng g g tasks/guards/tasks-state-preloading --skip-tests true --implements CanActivate**

2. Create a function **app/tasks/guards/check-store.function.ts**. Use the following snippet of code:

```
import { Store } from '@ngrx/store';
import { type Observable, tap, filter, take } from 'rxjs';

import { selectTasksLoaded } from '../../../core/@ngrx';
import * as TasksActions from '../../../core/@ngrx/tasks/tasks.actions';

export function checkStore(store: Store): Observable<boolean> {
  return store.select(selectTasksLoaded).pipe(
    // make a side effect
    tap((loaded: boolean) => {
      if (!loaded) {
        store.dispatch(TasksActions.getTasks());
      }
    }),

    // wait, while loaded = true
    filter((loaded: boolean) => loaded),

    // automatically unsubscribe
    take(1)
  );
}
```

3. Replace the content of the file **app/tasks/guards/tasks-state-preloading.guard.ts** with the following snippet of code:

```
import { Injectable } from '@angular/core';
import { Store } from '@ngrx/store';
import { type CanActivate } from '@angular/router';
import { type Observable, of, catchError } from 'rxjs';

import { checkStore } from './check-store.function';

@Injectable({
  providedIn: 'root'
})
export class TasksStatePreloadingGuard implements CanActivate {
  constructor(private store: Store) {}

  canActivate(): Observable<boolean> {
    return checkStore(this.store).pipe(
      catchError(() => of(false))
    );
  }
}
```

4. Create file **app/tasks/guards/task-exists.guard.ts**. Run the following command from the command line:

**ng g g tasks/guards/task-exists --skip-tests true --implements CanActivate**

5. Replace the content of the file with the following snippet of code:

```
import { Injectable } from '@angular/core';
import { Store } from '@ngrx/store';
import type { CanActivate, ActivatedRouteSnapshot } from '@angular/router';
import { type Observable, map, switchMap, take, tap } from 'rxjs';

import { selectTasksData } from '../../../core/@ngrx';
import { checkStore } from './check-store.function';
import { type TaskModel } from '../models/task.model';
import * as RouterActions from '../../../core/@ngrx/router/router.actions';

@Injectable({
  providedIn: 'root'
})
export class TaskExistsGuard implements CanActivate {
  constructor(private store: Store) {}

  canActivate(route: ActivatedRouteSnapshot): Observable<boolean> {
    return checkStore(this.store).pipe(
      switchMap(() => {
        const id = +route.paramMap.get('taskID')!;
        return this.hasTask(id);
      })
    );
  }

  private hasTask(id: number): Observable<boolean> {
    return this.store.select(selectTasksData).pipe(

      // check if task with id exists
      map((tasks: readonly TaskModel[]) => !!tasks.find(task => task.id === id)),

      // make a side effect
      tap(result => {
        if (!result) {
          this.store.dispatch(RouterActions.go({ path: ['/home'] }));
        }
      }),

      // automatically unsubscribe
      take(1)
    );
  }
}
```

1. Create file **app/tasks/guards/index.ts**. Use the following snippet of code:

```
export * from './task-exists.guard';
export * from './tasks-state-preloading.guard';
```

2. Make changes to **TasksRoutingModule**. Use the following snippet of code:



```
// 1
import { TasksStatePreloadingGuard, TaskExistsGuard } from './guards';

// 2
{
  path: 'home',
  component: TaskListComponent,
  canActivate: [TasksStatePreloadingGuard],
  ...
},
{
  path: 'edit/:taskID',
  component: TaskFormComponent,
  canActivate: [TaskExistsGuard]
}
}
```

3. Make changes to **TaskListComponent**. Use the following snippet of code:

```
ngOnInit(): void {
  ...
  this.store.dispatch(TasksActions.getTasks());
}
```

4. Create file **app/users/guards/users-state-preloading.guard.ts**. Run the following command from the command line:

```
ng g g users/guards/users-state-preloading --skip-tests true --implements CanActivate
```

5. Replace the content of the file with the following snippet of code:

```
import { Injectable } from '@angular/core';
import { Store } from '@ngrx/store';
import { type CanActivate } from '@angular/router';
import { type Observable, of, catchError, take, tap } from 'rxjs';

import { selectUsersLoaded } from '../../../core/@ngrx';
import * as UsersActions from '../../../core/@ngrx/users/users.actions';

@Injectable({
  providedIn: 'any'
})
export class UsersStatePreloadingGuard implements CanActivate {
  constructor(private store: Store) {}

  canActivate(): Observable<boolean> {
    return this.checkStore().pipe(
      catchError(() => of(false))
    );
  }

  private checkStore(): Observable<boolean> {
    return this.store.select(selectUsersLoaded).pipe(
      tap((loaded: boolean) => {
        if (!loaded) {
          this.store.dispatch(UsersActions.getUsers());
        }
      }),
    );
  }
}
```

```

        take(1)
    );
}
}

```

6. Make changes to **users/guards/index.ts**. Use the following snippet of code:

```
export * from './users-state-preloading.guard';
```

7. Make changes to **UsersRoutingModule**. Use the following snippet of code:

```
// 1
import { UserResolveGuard, UsersStatePreloadingGuard } from './guards';

// 2
{
    path: 'edit/:userID',
    component: UserFormComponent,
    canActivate: [CanDeactivateGuard],
    resolve: {
        user: UserResolveGuard
    }
},
{
    path: '',
    component: UserListComponent,
    canActivate: [UsersStatePreloadingGuard]
}

```

8. Make changes to **UserListComponent**. Use the following snippet of code:

```
ngOnInit() {
    this.users$ = this.store.select(getUsers);
    this.usersError$ = this.store.select(getUsersError);
    this.store.dispatch(UsersActions.getUsers());
    ...
}

```

9. Make changes to **UserFormComponent**. Use the following snippet of code:

```
// 1
import { selectUsersOriginalUser, selectSelectedUserByUrl } from
'../../../../../core/@ngrx';
import { type Observable, map, of, switchMap, type Subscription } from 'rxjs';
import { ActivatedRoute, type Data, type UrlTree } from '@angular/router';
import { AutoUnsubscribe, DialogService, type CanComponentDeactivate } from
'../../../../../core';

// 2
@AutoUnsubscribe()

// 3
private sub!: Subscription;

// 4
constructor(
    private route: ActivatedRoute,

```

```

    ...
  ) { }

// 5
ngOnInit(): void {
  this.route.data.pipe(map((data: Data) => data['user'])).subscribe((user: UserModel)
=> {
    this.user = { ...user };
  });
  this.sub = this.store.select(selectSelectedUserByUrl)
    .subscribe((user: UserModel) => {
      this.user = { ...user };
      this.store.dispatch(UsersActions.setOriginalUser({ user }));
    });
}

```

10. Make changes to file **app/users/guards/index.ts**. Use the following snippet of code:

```
export * from './user-resolve.guard';
```

11. Delete **UserResolveGuard**.

## Task 28. @ngrx/entity

1. Make changes to file **tasks.state.ts**. Use the following snippet of code:

```
// 1
import { createEntityAdapter, type EntityState, type EntityAdapter } from
 '@ngrx/entity';

// 2
export interface TasksState extends EntityState<TaskModel> {
  data: ReadonlyArray<TaskModel>;
  readonly loading: boolean;
  readonly loaded: boolean;
  readonly error: Error | string | null;
}

// 3
function selectTaskId(task: TaskModel): number {
  // In this case this would be optional since primary key is id
  return task.id!;
}

function sortTasksByAction(task1: TaskModel, task2: TaskModel): number {
  return task1.action.localeCompare(task2.action);
}

// 4
export const adapter: EntityAdapter<TaskModel> = createEntityAdapter<TaskModel>({
  selectId: selectTaskId,
  sortComparer: sortTasksByAction
});

// 5
export const initialTasksState: TasksState = adapter.getInitialState({
  data: [],
  loading: false,
  loaded: false,
  error: null
});
```

2. Make changes to file **tasks.reducer.ts**. Use the following snippet of code:

```
// 1
import { adapter, initialTasksState } from './tasks.state';

on(TasksActions.getTasksSuccess, (state, { tasks }) => {
  console.log('GET_TASKS_SUCCESS action being handled!');
  const data = [...tasks];
  return {
    ...state,
    data,
    loading: false,
    loaded: true
  };
});
return adapter.setAll(tasks, {...state, loading: false, loaded: true });
```

```

    })),
    on(TasksActions.createTaskSuccess, (state, { task }) => {
      console.log('CREATE_TASK_SUCCESS action being handled!');
      const data = [...state.data, { ...task }];
      return {
        ...state,
        data
      };
      return adapter.addOne(task, state);
    })),
    on(TasksActions.updateTaskSuccess, (state, { task }) => {
      console.log('UPDATE_TASK_SUCCESS action being handled!');
      const data = [...state.data];
      const index = data.findIndex(t => t.id === task.id);

      data[index] = { ...task };

      return {
        ...state,
        data
      };
      return adapter.updateOne({ id: task.id!, changes: task }, state);
    })),
    on(TasksActions.deleteTaskSuccess, (state, { task }) => {
      console.log('DELETE_TASK_SUCCESS action being handled!');
      const data = state.data.filter(t => t.id !== props.task.id);

      return {
        ...state,
        data
      };
      return adapter.removeOne(task.id!, state);
    })),
  },

```

3. Make changes to file **tasks.selectors.ts**. Use the following snippet of code:

```

// 1
import { adapter, type TasksState } from './tasks.state';

// 2
export const selectTasksData = createSelector(getTasksState, (state: TasksState) =>
state.data);
export const {
  selectEntities: selectTasksEntities,
  selectAll: selectTasksData
} = adapter.getSelectors(selectTasksState);

// 3
export const selectSelectedTaskByUrl = createSelector(
  selectTasksData,
  selectTasksEntities
  getRouterState,
  (tasks, router): TaskModel => {

```

```
const taskID = router.state.params.params['taskID'];
if (taskID && Array.isArray(tasks)) {
  return tasks.find(task => task.id === +taskID);
  return tasks[taskID] as TaskModel;
} else {
  return new TaskModel();
}
});
```

## Task 29. @ngrx/data

1. Create file **@ngrx/data/entity-store.module.ts**. Use the following snippet of code:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { EntityDataModule, DefaultDataServiceConfig, type EntityMetadataMap } from '@ngrx/data';
import { createFeatureSelector, createSelector } from '@ngrx/store';

import { selectRouterState } from '../router/router.selectors';
import { UserModel } from 'src/app/users/models/user.model';

// specify config
const defaultDataServiceConfig: DefaultDataServiceConfig = {
  root: 'http://localhost:3000/'
};

// specify rules for the plural
// the rule is specific for json-server
const pluralNames = {
  User: 'User'
};

// add only one collection User
export const entityMetadata: EntityMetadataMap = {
  User: {}
};

// custom feature selector
export const selectEntityCacheState = createFeatureSelector('entityCache');

// кастомный селектор
export const selectUsersEntitites = createSelector(
  selectEntityCacheState,
  (entityState: any) => {
    return entityState.User.entities;
  }
);

// custom selector
export const selectEditedUser = createSelector(
  selectUsersEntitites,
  selectRouterState,
  (users, router): UserModel | null => {
    const userID = router.state.params['editedUserID'];
    if (userID && users) {
      return users[userID];
    } else {
      return null;
    }
  }
);

// custom selector
export const selectSelectedUserByUrl = createSelector(
  selectUsersEntitites,
```

```

selectRouterState,
(users, router): UserModel => {
  const userID = router.state.params['userID'];
  if (userID && users) {
    return users[userID];
  } else {
    return new UserModel(null, '', '');
  }
}
);

@NgModule({
  imports: [
    CommonModule,
    EntityDataModule.forRoot({ entityMetadata, pluralNames })
  ],
  providers: [
    { provide: DefaultDataServiceConfig, useValue: defaultDataServiceConfig }
  ]
})
export class EntityStoreModule {}

```

2. Make changes to file **db.json**. Rename key **users** to **user**
3. Make changes to **RootStoreModule**. Use the following snippet of code:

```

// 1
import { EntityStoreModule } from './data/entity-store.module';

// 2
imports: [
  ...
  EntityStoreModule
]

```

4. Make changes to **UserListComponent**. Use the following snippet of code:

```

// 1
import { EntityServices, EntityCollectionService } from '@ngrx/data';
import { selectEditedUser } from './../../../../core/@ngrx/data/entity-store.module';
import { type Observable, type Subscription, map } from 'rxjs';
import * as UsersActions from './../../../../core/@ngrx/users/users.actions';
import { selectUsers, selectUsersError, selectEditedUser } from './../../../../core/@ngrx';

// 2
private userService!: EntityCollectionService<UserModel>;

// 3
constructor(private store: Store, entitytServices: EntityServices) {
  // get service for the entity User
  this.userService = entitytServices.getEntityCollectionService('User');
}

// 4
ngOnInit() {
  this.users$ = this.store.select(selectUsers);
  this.usersError$ = this.store.select(selectUsersError);
}

```



```

    // use built-in selector
    this.users$ = this.userService.entities$;

    // use built-in selector with transformation
    // error is in EntityAction
    this.usersError$ = this.userService.errors$.pipe(
      map(action => action.payload.error!)
    );
  ...
}

// 5
onDeleteUser(user: UserModel) {
  const userToDelete: User = { ...user };
  this.store.dispatch(UsersActions.deleteUser({ user: userToDelete }));
  // use service to dispatch EntitytAction
  this.userService.delete(user.id!);
}

```

5. Make changes to **UserFormComponent**. Use the following snippet of code:

```

// 1
import { type Observable, type Subscription, map, of, switchMap } from 'rxjs';
import { selectUsersOriginalUser, selectSelectedUserByUrl } from
'../../../../../core/@ngrx';
import * as UsersActions from '../../../../../core/@ngrx/users/users.actions';
import { type Data, type UrlTree } from '@angular/router';
import { Component, type OnInit, ViewChild } from '@angular/core';
import { selectSelectedUserByUrl } from 'src/app/core/@ngrx/data/entity-store.module';
import { EntityCollectionService, EntityServices } from '@ngrx/data';
import { NgForm } from '@angular/forms';

// 2
@ViewChild('form', { static: false })
userForm!: NgForm;
private userService!: EntityCollectionService<UserModel>;
private isSubmitClick = false;

// 3
constructor(
  private dialogService: DialogService,
  private store: Store,
  entitytServices: EntityServices
) {
  // get service for the entity User
  this.userService = entitytServices.getEntityCollectionService('User');
}

// 4
onSaveUser() {
  const user = { ...this.user } as User;
  this.isSubmitClick = true;

  if (user.id) {
    this.store.dispatch(UsersActions.updateUser({ user }));
  }
}

```

```

        this.userService.update(user);
    } else {
        this.store.dispatch(UsersActions.createUser({ user }));
        this.userService.add(user);
    }

    this.onGoBack();
}

// 5
canDeactivate():
    | Observable<boolean | UrlTree>
    | Promise<boolean | UrlTree>
    | boolean
    | UrlTree {
    const flags = [];

    return this.store.select(selectUsersOriginalUser).pipe(
        switchMap((originalUser: UserModel) => {
            for (const key in originalUser) {
                if (originalUser[key] === this.user[key]) {
                    flags.push(true);
                } else {
                    flags.push(false);
                }
            }

            if (flags.every(el => el)) {
                return of(true);
            }

            // Otherwise ask the user with the dialog service and return its
            // promise which resolves to true or false when the user decides
            return this.dialogService.confirm('Discard changes?');
        })
    );
    if (this.isSubmitClick) {
        return true;
    }

    if (this.userForm.pristine) {
        return true;
    }

    // Otherwise ask the user with the dialog service and return its
    // promise which resolves to true or false when the user decides
    return this.dialogService.confirm('Discard changes?');
}

```

6. Make changes to **UsersStatePreloadingGuard**. Use the following snippet of code:

```

// 1
import { Store } from '@ngrx/store';
import { selectUsersLoaded } from '../../../core/@ngrx';
import * as UsersActions from '../../../core/@ngrx/users/users.actions';
import { EntityServices, EntityCollectionService } from '@ngrx/data';

```

```

import { UserModel } from '../models/user.model';

// 2
private userService!: EntityCollectionService<UserModel>;

// 3
constructor(
    private store: Store
    entitytServices: EntityServices
) {
    this.userService = entitytServices.getEntityCollectionService('User');
}

// 4
private checkStore(): Observable<boolean> {
    return this.store.select(selectUsersLoaded).pipe(
        return this.userService.loaded$.pipe(
            tap(loaded => {
                if (!loaded) {
                    this.userService.getAll();
                }
            })),
            take(1)
        );
}

```

## Task 30. Use Facade

1. Create the file **core/@ngrx/tasks/tasks.facade.ts**. Use the following snippet of code:

```
import { Injectable } from '@angular/core';
import { Store } from '@ngrx/store';
import { type Observable } from 'rxjs';
import { type NavigationExtras } from '@angular/router';

import { selectTasksData, selectTasksError, selectSelectedTaskByUrl } from
'./tasks.selectors';
import { type TaskModel } from 'src/app/tasks/models/task.model';
import * as TasksActions from './../../../../core/@ngrx/tasks/tasks.actions';
import * as RouterActions from './../../../../core/@ngrx/router/router.actions';

@Injectable({
  providedIn: 'root'
})
export class TasksFacade {
  tasks$: Observable<ReadonlyArray<TaskModel>>;
  tasksError$: Observable<Error | string | null>;
  selectedTaskByUrl$: Observable<TaskModel>;

  constructor(private store: Store) {
    this.tasks$ = this.store.select(selectTasksData);
    this.tasksError$ = this.store.select(selectTasksError);
    this.selectedTaskByUrl$ = this.store.select(selectSelectedTaskByUrl);
  }

  createTask(props: { task: TaskModel }) {
    this.store.dispatch(TasksActions.createTask(props));
  }

  updateTask(props: { task: TaskModel }) {
    this.store.dispatch(TasksActions.updateTask(props));
  }

  deleteTask(props: { task: TaskModel }) {
    this.store.dispatch(TasksActions.deleteTask(props));
  }

  // TODO: should it be moved to RouterFacade?
  goTo(props: {
    path: any[];
    queryParams?: object;
    extras?: NavigationExtras;
  }) {
    this.store.dispatch(RouterActions.go(props));
  }
}
```

2. Make changes to the file **core/@ngrx/tasks/index.ts**. Use the following snippet of code:

```
export * from './tasks.facade';
```

3. Make changes to the **TaskListComponent**. Use the following snippet of code:

```
// 1
import { TasksFacade } from './../../../../core/@ngrx';
import { Store } from '@ngrx/store';
import { selectTasksData, selectTasksError } from './../../../../core/@ngrx';
import * as TasksActions from './../../../../core/@ngrx/tasks/tasks.actions';
import * as RouterActions from './../../../../core/@ngrx/router/router.actions';

// 2
constructor(private store: Store, tasksFacade: TasksFacade) {}

// 3
ngOnInit(): void {
  this.tasks$ = this.store.select(selectTasksData);
  this.tasksError$ = this.store.select(selectTasksError);
  this.tasks$ = this.tasksFacade.tasks$;
  this.tasksError$ = this.tasksFacade.tasksError$;
}

// 4
onCreateTask(): void {
  this.store.dispatch(
    RouterActions.go({
      path: ['/add']
    })
  );
  this.tasksFacade.goTo({ path: ['/add'] });
}

// 5
onCompleteTask(task: TaskModel): void {
  // task is not plain object
  // taskToComplete is a plain object
  const taskToComplete: Task = { ...task, done: true };
  this.store.dispatch(TasksActions.updateTask({ task: taskToComplete }));
  this.tasksFacade.updateTask({ task: taskToComplete });
}

// 6
onEditTask(task: TaskModel): void {
  const link = ['/edit', task.id];
  this.store.dispatch(
    RouterActions.go({
      path: link
    })
  );
  this.tasksFacade.goTo({ path: link });
}

// 7
onDeleteTask(task: TaskModel): void {
  const taskToDelete: Task = { ...task };
  this.store.dispatch(TasksActions.deleteTask({ task: taskToDelete }));
  this.tasksFacade.deleteTask({ task: taskToDelete });
}
```

4. Make changes to the **TaskFormComponent**. Use the following snippet of code:

```
// 1
import { Store } from '@ngrx/store';
import { selectSelectedTaskByUrl } from '../../../core/@ngrx';
import * as TasksActions from '../../../core/@ngrx/tasks/tasks.actions';
import * as RouterActions from '../../../core/@ngrx/router/router.actions';
import { TasksFacade } from 'src/app/core/@ngrx/tasks/tasks.facade';

// 2
constructor(private store: Store, tasksFacade: TasksFacade) {}

// 3
ngOnInit(): void {
  this.store.select(selectSelectedTaskByUrl)
    .pipe(
      takeUntil(this.componentDestroyed$)
    )
    .subscribe(observer);
}

// 4
onSaveTask(): void {
  const task = { ...this.task } as TaskModel;

  if (task.id) {
    this.store.dispatch(TasksActions.updateTask({ task }));
  } else {
    this.store.dispatch(TasksActions.createTask({ task }));
  }

  const method = task.id ? 'updateTask' : 'createTask';
  this.tasksFacade[method]({ task });
}

// 5
onGoBack(): void {
  this.store.dispatch(
    RouterActions.go({
      path: ['/home']
    })
  );
  this.tasksFacade.goTo({ path: ['/home'] });
}
```