Angular 2+

Workshop. Ngrx.

Contents

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. Create a State

1. Create file **app/core/+store/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/+store/app.state.ts.** Use the following snippet of code:

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

export interface AppState {
  tasks: TasksState;
}
```

3. Create file **app/core/+store/tasks/index.ts.** Use the following snippet of code:

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

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

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

Task 02. Create Actions

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

> ng g a core/+store/tasks/tasks

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

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

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

// [Tasks]- namespace
export enum TasksActionTypes {
  GET_TASKS = '[Tasks] GET_TASKS',
  GET_TASK = '[Tasks] GET_TASK',
  CREATE_TASK = '[Tasks] CREATE_TASK',
  UPDATE_TASK = '[Tasks] UPDATE_TASK',
  DELETE_TASK = '[Tasks] DELETE_TASK'
}

export class GetTasks implements Action {
  readonly type = TasksActionTypes.GET_TASKS;
}

export class GetTask implements Action {
  readonly type = TasksActionTypes.GET_TASK;
  constructor(public payload: number) { }
}

export class CreateTask implements Action {
  readonly type = TasksActionTypes.CREATE_TASK;
  constructor(public payload: TaskModel) { }
}

export class UpdateTask implements Action {
  readonly type = TasksActionTypes.UPDATE_TASK;
  constructor(public payload: TaskModel) { }
}

export class DeleteTask implements Action {
  readonly type = TasksActionTypes.DELETE_TASK;
  constructor(public payload: TaskModel) { }
}

export type TasksActions
  = GetTasks
  | GetTask
  | CreateTask
  | UpdateTask
  | DeleteTask;
```

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

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

Task 03. Create a Reducer

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

>ng g r core/+store/tasks/tasks --spec false

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

```
import { TasksActionTypes, TasksActions } from './tasks.actions';
import { TasksState, initialTasksState } from './tasks.state';

export function tasksReducer(
      state = initialTasksState,
      action: TasksActions
): TasksState {
  console.log(`Reducer: Action came in! ${action.type}`);

  switch (action.type) {
    case TasksActionTypes.GET_TASKS: {
      console.log('GET_TASKS action being handled!');
      return {...state};
    }

    case TasksActionTypes.CREATE_TASK: {
      console.log('CREATE_TASK action being handled!');
      return {...state};
    }

    case TasksActionTypes.UPDATE_TASK: {
      console.log('UPDATE_TASK action being handled!');
      return {...state};
    }

    case TasksActionTypes.DELETE_TASK: {
      console.log('DELETE_TASK action being handled!');
      return {...state};
    }

    default: {
      console.log('UNKNOWN_TASK action being handled!');
      return state;
    }
  }
}
```

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

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

Task 04. Provide Store

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

>ng g m core/+store/CoreStore --flat -m core

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

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

// 2
@NgModule({
  …
  imports: [
    CommonModule
    StoreModule.forRoot({}),
  ]
})
export class CoreStoreModule {
  …
}
```

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

```
// 1
import { StoreModule } from '@ngrx/store';
import { tasksReducer } from './../core/+store';

// 2
@NgModule({
  …
  imports: [
    …
    StoreModule.forFeature('tasks', tasksReducer)
  ]
})
export class TasksModule {}
```

Task 05. Inject Store

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

```
// 1
// @Ngrx
import { Store } from '@ngrx/store';
import { AppState } from './../../../core/+store';

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

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

    …
}
```

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

Reducer: Action came in! @ngrx/store/update-reducers

UNKNOWN_TASK action being handled!

We have a store! >Store {…}

Task 06. Reading Data From The Store

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

```
// 1
import { Store, select } from '@ngrx/store';
import { AppState, TasksState } from './../../../core/+store';
import { Observable } from 'rxjs';

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

// 2
ngOnInit() {
    console.log('We have a store! ', this.store);
    this.tasksState$ = this.store.pipe(select('tasks'));

    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 07. Dispatching an Event To The Store

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

```
// 1
export enum TasksActionTypes {
  DELETE_TASK = '[Tasks] DELETE_TASK',
  DONE_TASK = '[Tasks] DONE_TASK'
}

// 2
export class DoneTask implements Action {
  readonly type = TasksActionTypes.DONE_TASK;
  constructor(public payload: TaskModel) { }
}

// 3
export type TasksActions =
  …
  | UpdateTask
  | DeleteTask
  | DoneTask;
```

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

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

// 2
case TasksActionTypes.DONE_TASK: {
      console.log('DONE_TASK action being handled!');

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

        return task;
      });

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

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

```
// 1
import * as TasksActions from './../../../core/+store/tasks/tasks.actions';

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

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

    // do smth with updatedTask
    // ...
}
```

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

Task 08. 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 http://extension.remotedev.io/
2. Make changes to **CoreStoreModule**. 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() : [],
],
```

Task 09. Create Effects Class

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

```
>ng g ef core/+store/tasks/tasks -m tasks/tasks.module.ts --spec false
```

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

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

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

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

Task 10. Provide Effects

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

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

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

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

```
// 1
import { TasksEffects, tasksReducer } from './../core/+store';
import { TasksEffects } from '../core/+store/tasks/tasks.effects';
```

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

Reducer: Action came in! @ngrx/effects/init

UNKNOWN_TASK action being handled!

[TASKS EFFECTS]

Task 11. Get Tasks from DataBase

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

```
// 1
export interface TasksState {
  data: ReadonlyArray<TaskModel>;
  readonly loading: boolean;
  readonly loaded: boolean;
  readonly error: Error | string;
}

// 2
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:

```
// 1
export enum TasksActionTypes {
  GET_TASKS = '[Tasks] GET_TASKS',
  GET_TASKS_SUCCESS = '[Tasks] GET_TASKS_SUCCESS',
  GET_TASKS_ERROR   = '[Tasks] GET_TASKS_ERROR',
 …
}

// 2
export class GetTasksSuccess implements Action {
  readonly type = TasksActionTypes.GET_TASKS_SUCCESS;
  constructor(public payload: TaskModel[]) { }
}

export class GetTasksError implements Action {
  readonly type = TasksActionTypes.GET_TASKS_ERROR;
  constructor(public payload: Error | string) { }
}

// 3
export type TasksActions
  = GetTasks
  | GetTasksSuccess
  | GetTasksError
  …
  | DoneTask;
```

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

```
// 1
case TasksActionTypes.GET_TASKS: {
      console.log('GET_TASKS action being handled!');
      return {...state};
      return {
        ...state,
        loading: true
      };
}

case TasksActionTypes.GET_TASKS_SUCCESS: {
      console.log('GET_TASKS_SUCCESS action being handled!');
      const data = [...<Array<TaskModel>>action.payload];
      return {
        ...state,
        data,
        loading: false,
        loaded: true
      };
    }

case TasksActionTypes.GET_TASKS_ERROR: {
      console.log('GET_TASKS_ERROR action being handled!');
      const error = action.payload;
      return {
        ...state,
        loading: false,
        loaded: false,
        error
      };
    }
```

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

```
// 1
import { Action } from '@ngrx/store';
import { Actions, Effect, ofType } from '@ngrx/effects';
import * as TasksActions from './tasks.actions';

// rxjs
import { Observable } from 'rxjs';
import { switchMap } from 'rxjs/operators';

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

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

// 3
@Effect()
  getTasks$: Observable<Action> = this.actions$.pipe(
```

```
    // Instead of ofType<TasksActions.GetTasks>(...) you can use ofType(...)
    // It's optional.
    // Specify the action type to allow type-safe mapping to other data on the action,
    // including payload
    ofType<TasksActions.GetTasks>(TasksActions.TasksActionTypes.GET_TASKS),
    switchMap((action: TasksActions.GetTasks) =>
      this.taskPromiseService
        .getTasks()
        .then(tasks => new TasksActions.GetTasksSuccess(tasks))
        .catch(err => new TasksActions.GetTasksError(err))
    )
  );
```

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() {
    console.log('We have a store! ', this.store);
    this.tasksState$ = this.store.pipe(select('tasks'));

    this.store.dispatch(new TasksActions.GetTasks());
}

// 5
onDeleteTask(task: TaskModel) {
    // 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 12. 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>;
  …
}

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

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

```
// 1
export enum TasksActionTypes {
 …
  GET_TASK = '[Tasks] GET_TASK',
  GET_TASK_SUCCESS = '[Tasks] GET_TASK_SUCCESS',
  GET_TASK_ERROR = '[Tasks] GET_TASK_ERROR',
  …
}// 2
export class GetTaskSuccess implements Action {
  readonly type = TasksActionTypes.GET_TASK_SUCCESS;
  constructor(public payload: TaskModel) { }
}

export class GetTaskError implements Action {
  readonly type = TasksActionTypes.GET_TASK_ERROR;
  constructor(public payload: Error | string) { }
}

// 3
export type TasksActions
  =
  …
  | GetTaskSuccess
  | GetTaskError
  …
  | DoneTask;
```

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

```
// 1
import { Store, select } from '@ngrx/store';
import { AppState, TasksState } from './../../../core/+store';
import * as TasksActions from './../../../core/+store/tasks/tasks.actions';

// import { switchMap } from 'rxjs/operators';
import { Observable, Subscription } from 'rxjs';
```

```
import { AutoUnsubscribe } from './../../../core';

// 2
@AutoUnsubscribe()
export class TaskFormComponent implements OnInit {

// 3
tasksState$: Observable<TasksState>;

private sub: Subscription;

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

// 5
this.route.paramMap
      .pipe(
        switchMap((params: Params) => {
          return params.get('taskID')
          ? this.taskPromiseService.getTask(+params.get('taskID'))
          : Promise.resolve(null);
        })
      )
      .subscribe(
        // when Promise.resolve(null) => task = null => {...null} => {}
        task => this.task = {...task},
        err => console.log(err)
    );

this.tasksState$ = this.store.pipe(select('tasks'));
this.sub = this.tasksState$.subscribe(tasksState =>
      this.task = tasksState.selectedTask);

this.route.paramMap.subscribe(params => {
      const id = params.get('taskID');
      if (id) {
        this.store.dispatch(new TasksActions.GetTask(+id));
      }
    });
```

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

```
// 1
import { pluck, switchMap } from 'rxjs/operators';

// 2
@Effect()
  getTask$: Observable<Action> = this.actions$.pipe(
    ofType<TasksActions.GetTask>(TasksActions.TasksActionTypes.GET_TASK),
    pluck('payload'),
    switchMap(payload =>
      this.taskPromiseService
        .getTask(+payload)
```

```
        .then(task => new TasksActions.GetTaskSuccess(task))
        .catch(err => new TasksActions.GetTaskError(err))
    )
  );
```

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

```
case TasksActionTypes.GET_TASK: {
      console.log('GET_TASK action being handled!');
      return {
        ...state,
        loading: true
      };
}

case TasksActionTypes.GET_TASK_SUCCESS: {
      console.log('GET_TASK_SUCCESS action being handled!');
      const selectedTask = { ...<TaskModel>action.payload };
      return {
        ...state,
        loading: false,
        loaded: true,
        selectedTask
      };
    }

case TasksActionTypes.GET_TASK_ERROR: {
      console.log('GET_TASK_ERROR action being handled!');
      const error = action.payload;
      return {
        ...state,
        loading: false,
        loaded: false,
        error
      };
    }
```

Task 13. Update Task in DataBase

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

```
// 1
// [Tasks]- namespace
export enum TasksActionTypes {
…
  UPDATE_TASK = '[Tasks] UPDATE_TASK',
  UPDATE_TASK_SUCCESS = '[Tasks] UPDATE_TASK_SUCCESS',
  UPDATE_TASK_ERROR   = '[Tasks] UPDATE_TASK_ERROR',
…
}
// 2
export class UpdateTaskSuccess implements Action {
  readonly type = TasksActionTypes.UPDATE_TASK_SUCCESS;
  constructor(public payload: TaskModel) { }
}

export class UpdateTaskError implements Action {
  readonly type = TasksActionTypes.UPDATE_TASK_ERROR;
  constructor(public payload: Error | string) { }
}

// 3
export type TasksActions
  =
  …
  | UpdateTaskSuccess
  | UpdateTaskError
  …
  | DoneTask;
```

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

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

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

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

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

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

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

// 3
@Effect()
updateTask$: Observable<Action> = this.actions$.pipe(
    ofType<TasksActions.UpdateTask>(TasksActions.TasksActionTypes.UPDATE_TASK),
    pluck('payload'),
    concatMap((payload: TaskModel) =>
      this.taskPromiseService
        .updateTask(payload)
        .then(task => {
          this.router.navigate(['/home']);
          return new TasksActions.UpdateTaskSuccess(task);
        })
        .catch(err => new TasksActions.UpdateTaskError(err))
    )
);
```

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

```
case TasksActionTypes.UPDATE_TASK_SUCCESS: {
      console.log('UPDATE_TASK_SUCCESS action being handled!');
      const task = { ...<TaskModel>action.payload };
      const data = [...state.data];
      const index = data.findIndex(t => t.id === task.id);

      data[index] = task;

      return {
        ...state,
        data
      };
    }

case TasksActionTypes.UPDATE_TASK_ERROR: {
      console.log('UPDATE_TASK_ERROR action being handled!');
      const error = action.payload;
      return {
        ...state,
        error
      };
}
```

Task 14. Add Task to DataBase

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

```
// 1
export enum TasksActionTypes {
…
  CREATE_TASK = '[Tasks] CREATE_TASK',
  CREATE_TASK_SUCCESS = '[Tasks] CREATE_TASK_SUCCESS',
  CREATE_TASK_ERROR   = '[Tasks] CREATE_TASK_ERROR',
…
}
// 2
export class CreateTaskSuccess implements Action {
  readonly type = TasksActionTypes.CREATE_TASK_SUCCESS;
  constructor(public payload: TaskModel) { }
}

export class CreateTaskError implements Action {
  readonly type = TasksActionTypes.CREATE_TASK_ERROR;
  constructor(public payload: Error | string) { }
}

// 3
export type TasksActions
  =
  …
  | CreateTask
  | CreateTaskSuccess
  | CreateTaskError
  …
  | DoneTask;
```

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

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

    this.tasksState$ = this.store.pipe(select('tasks'));
    this.sub = this.tasksState$.subscribe(tasksState =>
      this.task = tasksState.selectedTask);

    this.tasksState$ = this.store.pipe(select('tasks'));
    this.sub = this.tasksState$.subscribe(tasksState => {
      if (tasksState.selectedTask) {
        this.task = tasksState.selectedTask;
      } else {
        this.task = new TaskModel();
      }
    });
…
}
```

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

```
@Effect()
createTask$: Observable<Action> = this.actions$.pipe(
    ofType<TasksActions.CreateTask>(TasksActions.TasksActionTypes.CREATE_TASK),
    pluck('payload'),
    concatMap((payload: TaskModel) =>
      this.taskPromiseService
        .createTask(payload)
        .then(task => {
          this.router.navigate(['/home']);
          return new TasksActions.CreateTaskSuccess(task);
        })
        .catch(err => new TasksActions.CreateTaskError(err))
    )
);
```

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

```
case TasksActionTypes.CREATE_TASK_SUCCESS: {
      console.log('CREATE_TASK_SUCCESS action being handled!');
      const task = { ...<TaskModel>action.payload };
      const data = [...state.data, task];

      return {
        ...state,
        data
      };
}

case TasksActionTypes.CREATE_TASK_ERROR: {
      console.log('CREATE_TASK_ERROR action being handled!');
      const error = action.payload;
      return {
        ...state,
        error
      };
}

case TasksActionTypes.GET_TASKS_SUCCESS: {
      console.log('GET_TASKS_SUCCESS action being handled!');
      const data = [...(<Array<Task>>action.payload)];
      return {
        ...state,
        data,
        loading: false,
        loaded: true,
        selectedTask: null
      };
}
```

Task 15. Delete Task from DataBase

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

```
// 1
export enum TasksActionTypes {
…
  DELETE_TASK = '[Tasks] DELETE_TASK',
  DELETE_TASK_SUCCESS = '[Tasks] DELETE_TASK_SUCCESS',
  DELETE_TASK_ERROR = '[Tasks] DELETE_TASK_ERROR',
…
}

// 2
export class DeleteTaskSuccess implements Action {
  readonly type = TasksActionTypes.DELETE_TASK_SUCCESS;
  constructor(public payload: TaskModel) { }
}

export class DeleteTaskError implements Action {
  readonly type = TasksActionTypes.DELETE_TASK_ERROR;
  constructor(public payload: Error | string) { }
}

// 3
export type TasksActions
  =
  …
  | DeleteTask
  | DeleteTaskSuccess
  | DeleteTaskError
  | DoneTask;
```

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

```
onDeleteTask(task: TaskModel) {
    this.store.dispatch(new 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:

```
@Effect()
  deleteTask$: Observable<Action> = this.actions$.pipe(
    ofType<TasksActions.DeleteTask>(TasksActions.TasksActionTypes.DELETE_TASK),
    pluck('payload'),
    concatMap((payload: TaskModel) =>
      this.taskPromiseService
        .deleteTask(payload)
        .then(
          (/* method delete for this API returns nothing, so we will use payload */) =>
{
            return new TasksActions.DeleteTaskSuccess(payload);
```

```
          }
        )
        .catch(err => new TasksActions.DeleteTaskError(err))
    )
  );
```

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

```
case TasksActionTypes.DELETE_TASK_SUCCESS: {
      console.log('DELETE_TASK_SUCCESS action being handled!');
      const task = { ...<TaskModel>action.payload };
      const data = state.data.filter(t => t.id !== task.id);


      return {
        ...state,
        data
      };
}

case TasksActionTypes.DELETE_TASK_ERROR: {
      console.log('DELETE_TASK_ERROR action being handled!');
      const error = action.payload;
      return {
        ...state,
        error
      };
}
```

Task 16. Replace DoneTask w/ UpdateTask Action

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

```
onCompleteTask(task: TaskModel): void {
    this.store.dispatch(new TasksActions.DoneTask(task));
    const doneTask = {...task, done: true};
    this.store.dispatch(new TasksActions.UpdateTask(doneTask));
}
```

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

```
// 1
export enum TasksActionTypes {
…
  DONE_TASK = '[Tasks] DONE_TASK'
}

// 2
export class DoneTask implements Action {
  readonly type = TasksActionTypes.DONE_TASK;
  constructor(public payload: Task) { }
}

// 3
export type TasksActions
  =
  …
  | DeleteTaskSuccess
  | DeleteTaskError
  | DoneTask;
```

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

```
case TasksActionTypes.DONE_TASK: {
      console.log('DONE_TASK action being handled!');

      const id = (<Task>action.payload).id
      const data = state.data.map(task => {
        if (task.id === id) {
          return {...action.payload, done: true};
        } else {
          return task;
        }
      });
      return {
        ...state,
        data
      };
    }
```

Task 17. Feature Selector

1. Create file **app/core/+store/tasks/tasks.selectors.ts.** Use the following snippet of code:

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

import { TasksState } from './tasks.state';

export const getTasksState = createFeatureSelector<TasksState>('tasks');
```

2. Make changes to file **app/core/+store/tasks/index.ts.** Use the following snippet of code:

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

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

```
// 1
import { AppState, TasksState, getTasksState } from './../../../core/+store';

// 2
ngOnInit() {
    console.log('We have a store! ', this.store);
    this.tasksState$ = this.store.pipe(select('tasks'));
    this.tasksState$ = this.store.pipe(select(getTasksState));

    this.store.dispatch(new TasksActions.GetTasks());
}
```

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

```
// 1
import { AppState, TasksState, getTasksState } from './../../../core/+store';

// 2
ngOnInit(): void {
    this.tasksState$ = this.store.pipe(select('tasks'));
    this.tasksState$ = this.store.pipe(select(getTasksState));
…
}
```

Task 18. 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 getTasksData = createSelector(getTasksState, (state: TasksState) =>
state.data);
export const getTasksError = createSelector(getTasksState, (state: TasksState) =>
state.error);
export const getSelectedTask = createSelector(getTasksState, (state: TasksState) =>
state.selectedTask);
export const getTasksLoaded = createSelector(getTasksState, (state: TasksState) =>
state.loaded);
```

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

```
// 1
import { AppState, TasksState, getTasksState, getTasksData, getTasksError } from
'./../../../core/+store';

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

// 3
ngOnInit() {
    console.log('We have a store! ', this.store);
    this.tasksState$ = this.store.pipe(select(getTasksState));
    this.tasks$ = this.store.pipe(select(getTasksData));
    this.tasksError$ = this.store.pipe(select(getTasksError));

    this.store.dispatch(new 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 { AppState, TasksState, getTasksState, getSelectedTask }
'./../../../core/+store';
```

```
// 2
tasksState$: Observable<TasksState>;


// 3
ngOnInit(): void {
    this.tasksState$ = this.store.pipe(select(getTasksState));
    this.sub = this.tasksState$.subscribe(tasksState => {
      if (tasksState.selectedTask) {
        this.task = tasksState.selectedTask;
      } else {
        this.task = new TaskModel();
      }
    });

    this.sub = this.store.pipe(select(getSelectedTask))
     .subscribe(task => {
      if (task) {
        this.task = task;
      } else {
        this.task = new TaskModel();
      }
    });
…
}
```

Task 19. Router State

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

```
import { Params, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

// @NgRx
import { ActionReducerMap } from '@ngrx/store';
import { RouterReducerState, RouterStateSerializer, routerReducer } from '@ngrx/router-
store';

export interface RouterStateUrl {
    url: string;
    queryParams: Params;
    params: Params;
    fragment: string;
}

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

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

export class CustomSerializer implements RouterStateSerializer<RouterStateUrl> {
    serialize(routerState: RouterStateSnapshot): RouterStateUrl {
        const { url } = routerState;
        const { queryParams } = routerState.root;

        let state: ActivatedRouteSnapshot = routerState.root;
        while (state.firstChild) {
            state = state.firstChild;
        }
        const { params, fragment } = state;

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

export const RouterStateSerializerProvider = {
    provide: RouterStateSerializer,
    useClass: CustomSerializer
};
```

2. Create file **app/core/+store/router/index.ts.** Use the following snippet of code:

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

3. Make changes to file **app/core/+store/index.ts.** Use the following snippet of code:

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

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

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

// 2
imports: [
StoreModule.forRoot({}),
StoreModule.forRoot(routerReducers),
StoreRouterConnectingModule.forRoot(),
…
],
providers: [
      RouterStateSerializerProvider,
]
```

5. Run application. Inspect Router State in NgRx Dev Tool. Comment RouterStateSerializerProvider and inspect Router State again. Uncomment RouterStateSerializerProvider.

Task 20. Compose Router and Task Selectors

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

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

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

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

2. Make changes to file **app/core/+store/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 { getRouterState } from './../router';
import { TaskModel } from './../../../tasks/models/task.model';

// 2
export const getSelectedTaskByUrl = createSelector(
    getTasksData,
    getRouterState,
    (tasks, router): TaskModel => {
        const taskID = router.state.params.taskID;
        if (taskID) {
            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, Params, Router } from '@angular/router';
import { AppState, getSelectedTask, getSelectedTaskByUrl } from
'./../../../core/+store';
import { Observable, Subscription } from 'rxjs';

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

// 3
ngOnInit(): void {
    this.sub = this.store.pipe(select(getSelectedTask))
    .subscribe(task => {
      if (task) {
        this.task = task;
      } else {
        this.task = new TaskModel();
      }
    });
```

```
    this.route.paramMap.subscribe(params => {
      const id = params.get('taskID');
      if (id) {
        this.store.dispatch(new TasksActions.GetTask(+id));
      }
    });

  this.sub = this.store
      .pipe(select(getSelectedTaskByUrl))
      .subscribe(task => this.task = task);

…
  }
```

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 getSelectedTask = createSelector(getTasksState, (state: TasksState) =>
state.selectedTask);
```

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

```
case TasksActionTypes.GET_TASKS_SUCCESS: {
      console.log('GET_TASKS_SUCCESS action being handled!');
      const data = [...(<Array<Task>>action.payload)];
      return {
        ...state,
        data,
        loading: false,
        loaded: true,
        selectedTask: null
      };
    }

case TasksActionTypes.GET_TASK: {
      console.log('GET_TASK action being handled!');
      return {
        ...state,
        loading: true
      };
    }
```

```
case TasksActionTypes.GET_TASK_SUCCESS: {
      console.log('GET_TASK_SUCCESS action being handled!');
      const selectedTask = { ...(<TaskModel>action.payload) };
      return {
        ...state,
        loading: false,
        loaded: true,
        selectedTask
      };
    }

case TasksActionTypes.GET_TASK_ERROR: {
      console.log('GET_TASK_ERROR action being handled!');
      const error = action.payload;
      return {
        ...state,
        loading: false,
        loaded: false,
        error
      };
    }
```

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

```
@Effect()
  getTask$: Observable<Action> = this.actions$.pipe(
    ofType<TasksActions.GetTask>(TasksActions.TasksActionTypes.GET_TASK),
    pluck('payload'),
    switchMap(payload =>
      this.taskPromiseService
        .getTask(+payload)
        .then(task => new TasksActions.GetTaskSuccess(task))
        .catch(err => new TasksActions.GetTaskError(err))
    )
  );
```

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

```
// 1
GET_TASK = '[Tasks] GET_TASK',
  GET_TASK_SUCCESS = '[Tasks] GET_TASK_SUCCESS',
  GET_TASK_ERROR = '[Tasks] GET_TASK_ERROR',

// 2
export class GetTask implements Action {
  readonly type = TasksActionTypes.GET_TASK;
  constructor(public payload: number) {}
}

export class GetTaskSuccess implements Action {
  readonly type = TasksActionTypes.GET_TASK_SUCCESS;
  constructor(public payload: TaskModel) {}
}
```

```
export class GetTaskError implements Action {
  readonly type = TasksActionTypes.GET_TASK_ERROR;
  constructor(public payload: Error | string) {}
}

// 3
GetTask
  | GetTaskSuccess
  | GetTaskError
```

Task 21. Users Store

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

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

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

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

2. Create a file **app/core/+store/users/index.ts.** Use the following snippet of code:

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

3. Create file **app/core/+store/users/users.actions.ts.** Run the following command from command line:

> ng g a core/+store/users/users

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

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

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

// Actions
// [Users] - namespace
export enum UsersActionTypes {
  GET_USERS           = '[Users] GET_USERS',
  GET_USERS_SUCCESS   = '[Users] GET_USERS_SUCCESS',
  GET_USERS_ERROR     = '[Users] GET_USERS_ERROR',
  GET_USER            = '[Users] GET_USER',
  GET_USER_SUCCESS    = '[Users] GET_USER_SUCCESS',
  GET_USER_ERROR      = '[Users] GET_USER_ERROR',
  CREATE_USER         = '[Users] CREATE_USER',
  CREATE_USER_SUCCESS = '[Users] CREATE_USER_SUCCESS',
  CREATE_USER_ERROR   = '[Users] CREATE_USER_ERROR',
  UPDATE_USER         = '[Users] UPDATE_USER',
  UPDATE_USER_SUCCESS = '[Users] UPDATE_USER_SUCCESS',
  UPDATE_USER_ERROR   = '[Users] UPDATE_USER_ERROR',
  DELETE_USER         = '[Users] DELETE_USER',
  DELETE_USER_SUCCESS = '[Users] DELETE_USER_SUCCESS',
  DELETE_USER_ERROR   = '[Users] DELETE_USER_ERROR',
  SET_ORIGINAL_USER = '[Users] SET_ORIGINAL_USER'
}
```

```
// Action Creators
export class GetUsers implements Action {
  readonly type = UsersActionTypes.GET_USERS;
}

export class GetUsersSuccess implements Action {
  readonly type = UsersActionTypes.GET_USERS_SUCCESS;
  constructor(public payload: UserModel[]) {}
}

export class GetUsersError implements Action {
  readonly type = UsersActionTypes.GET_USERS_ERROR;
  constructor(public payload: Error | string) {}
}

export class GetUser implements Action {
  readonly type = UsersActionTypes.GET_USER;
  constructor(public payload: number) {}
}

export class GetUserSuccess implements Action {
  readonly type = UsersActionTypes.GET_USER_SUCCESS;
  constructor(public payload: UserModel) {}
}

export class GetUserError implements Action {
  readonly type = UsersActionTypes.GET_USER_ERROR;
  constructor(public payload: Error | string) {}
}

export class CreateUser implements Action {
  readonly type = UsersActionTypes.CREATE_USER;
  constructor(public payload: UserModel) {}
}

export class CreateUserSuccess implements Action {
  readonly type = UsersActionTypes.CREATE_USER_SUCCESS;
  constructor(public payload: UserModel) { }
}

export class CreateUserError implements Action {
  readonly type = UsersActionTypes.CREATE_USER_ERROR;
  constructor(public payload: Error | string) {}
}

export class UpdateUser implements Action {
  readonly type = UsersActionTypes.UPDATE_USER;
  constructor(public payload: UserModel) {}
}

export class UpdateUserSuccess implements Action {
  readonly type = UsersActionTypes.UPDATE_USER_SUCCESS;
  constructor(public payload: UserModel) {}
}

export class UpdateUserError implements Action {
```

```
  readonly type = UsersActionTypes.UPDATE_USER_ERROR;
  constructor(public payload: Error | string) {}
}

export class DeleteUser implements Action {
  readonly type = UsersActionTypes.DELETE_USER;
  constructor(public payload: UserModel) {}
}

export class DeleteUserSuccess implements Action {
  readonly type = UsersActionTypes.DELETE_USER_SUCCESS;
  constructor(public payload: UserModel) {}
}

export class DeleteUserError implements Action {
  readonly type = UsersActionTypes.DELETE_USER_ERROR;
  constructor(public payload: Error | string) {}
}

export class SetOriginalUser implements Action {
  readonly type = UsersActionTypes.SET_ORIGINAL_USER;
  constructor(public payload: UserModel) {}
}

export type UsersActions
  = GetUsers
  | GetUsersSuccess
  | GetUsersError
  | GetUser
  | GetUserSuccess
  | GetUserError
  | CreateUser
  | CreateUserSuccess
  | CreateUserError
  | UpdateUser
  | UpdateUserSuccess
  | UpdateUserError
  | DeleteUser
  | DeleteUserSuccess
  | DeleteUserError
  | SetOriginalUser;
```

5. Make changes to file **app/core/+store/users/index.ts.** Use the following snippet of code:

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

6. Create file **app/core/+store/users/users.reducer.ts.** Run the following command from the command line:

>ng g r core/+store/users/users --spec false

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

```
import { UsersActionTypes, UsersActions  } from './users.actions';
import { initialUsersState, UsersState } from './users.state';
```

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

export function usersReducer (
    state = initialUsersState,
    action: UsersActions
): UsersState {
    console.log(`Reducer: Action came in! ${action.type}`);

    switch (action.type) {

      case UsersActionTypes.GET_USERS:
      case UsersActionTypes.GET_USER: {
        return {
          ...state,
          loading: true
        };
      }

      case UsersActionTypes.GET_USERS_SUCCESS: {
        const users = <UserModel[]>action.payload;
        console.log(users);

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

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

      case UsersActionTypes.GET_USER_SUCCESS: {
        const originalUser = { ...(<UserModel>action.payload) };
         return {
          ...state,
          loading: false,
          loaded: true,
           originalUser
        };
      }

      case UsersActionTypes.GET_USERS_ERROR:
      case UsersActionTypes.GET_USER_ERROR: {
        const error = action.payload;

        return {
```

```
      ...state,
      loading: false,
      loaded: false,
      error
    };
  }

  case UsersActionTypes.CREATE_USER:
  case UsersActionTypes.UPDATE_USER:
  case UsersActionTypes.DELETE_USER: {
    return {
      ...state
    };
  }

  case UsersActionTypes.CREATE_USER_SUCCESS:
  case UsersActionTypes.UPDATE_USER_SUCCESS: {
    const user = <UserModel>action.payload;
    const entities = {
      ...state.entities,
      [user.id]: user
    };
    const originalUser = {...<UserModel>action.payload};

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

  case UsersActionTypes.DELETE_USER_SUCCESS: {
    const user = <UserModel>action.payload;
    const { [user.id]: removed, ...entities} = state.entities;

    return {
      ...state,
      entities
    };
  }

  case UsersActionTypes.CREATE_USER_ERROR:
  case UsersActionTypes.UPDATE_USER_ERROR:
  case UsersActionTypes.DELETE_USER_ERROR: {
    const error = action.payload;
    return {
      ...state,
      error
    };
  }

 case UsersActionTypes.SET_ORIGINAL_USER: {
  const originalUser = { ...(<UserModel>action.payload) };

  return {
    ...state,
    originalUser
```

```
      };
     }

     default: {
       console.log('UNKNOWN_USER action being handled!');
       return state;
     }
    }
  }
```

8. Make changes to file **app/core/+store/users/index.ts.** Use the following snippet of code:

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

9. Make changes to file **app/core/+store/app.state.ts.** Use the following snippet of code:

```
// 1
import { UsersState } from './users';
// 2
export interface AppState {
  tasks: TasksState;
  users: UsersState;
}
```

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

```
>ng g ef core/+store/users/users -m users/users.module.ts --spec false
```

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

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';

// @Ngrx
import { Action } from '@ngrx/store';
import { Actions, Effect, ofType } from '@ngrx/effects';
import * as UsersActions from './users.actions';

// Rxjs
import { Observable, of } from 'rxjs';
import { switchMap, map, catchError, concatMap, pluck } from 'rxjs/operators';

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

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

```
@Effect()
getUsers$: Observable<Action> = this.actions$.pipe(
  ofType<UsersActions.GetUsers>(UsersActions.UsersActionTypes.GET_USERS),
  switchMap(action =>
    this.userObservableService
      .getUsers()
      .pipe(
        map(users => new UsersActions.GetUsersSuccess(users)),
        catchError(err => of(new UsersActions.GetUsersError(err)))
      )
  )
);

@Effect()
getUser$: Observable<Action> = this.actions$.pipe(
  ofType<UsersActions.GetUser>(UsersActions.UsersActionTypes.GET_USER),
  pluck('payload'),
  switchMap((payload: number) =>
    this.userObservableService
      .getUser(payload)
      .pipe(
        map(user => new UsersActions.GetUserSuccess(user)),
        catchError(err => of(new UsersActions.GetUserError(err)))
      )
  )
);

@Effect()
updateUser$: Observable<Action> = this.actions$.pipe(
  ofType<UsersActions.UpdateUser>(UsersActions.UsersActionTypes.UPDATE_USER),
  pluck('payload'),
  concatMap((payload: UserModel ) =>
    this.userObservableService.updateUser(payload).pipe(
      map(user => {
        this.router.navigate(['/users', { editedUserID: user.id }]);
        return new UsersActions.UpdateUserSuccess(user);
      }),
      catchError(err => of(new UsersActions.UpdateUserError(err)))
    )
  )
);

@Effect()
createUser$: Observable<Action> = this.actions$.pipe(
  ofType<UsersActions.CreateUser>(UsersActions.UsersActionTypes.CREATE_USER),
  pluck('payload'),
  concatMap((payload: UserModel) =>
    this.userObservableService.createUser(payload).pipe(
      map(user => {
        this.router.navigate(['/users']);
        return new UsersActions.CreateUserSuccess(user);
      }),
      catchError(err => of(new UsersActions.CreateUserError(err)))
    )
  )
);
```

```
  @Effect()
  deleteUser$: Observable<Action> = this.actions$.pipe(
    ofType<UsersActions.DeleteUser>(UsersActions.UsersActionTypes.DELETE_USER),
    pluck('payload'),
    concatMap((payload: UserModel) =>
      this.userObservableService.deleteUser(payload).pipe(
        // Note: json-server doesn't return deleted user
        // so we use payload
        map(() => new UsersActions.DeleteUserSuccess(payload)),
        catchError(err => of(new UsersActions.DeleteUserError(err)))
      )
    )
  );
}
```

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

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

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

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

import { UsersState } from './users.state';
import { UserModel } from './../../../users/models/user.model';
import { getRouterState } from './../router/router.selectors';

const getEntities = (state: UsersState) => state.entities;
const getOriginalUser = (state: UsersState) => state.originalUser;
const getLoaded = (state: UsersState) => state.loaded;
const getLoading = (state: UsersState) => state.loading;
const getError = (state: UsersState) => state.error;

export const getUsersState = createFeatureSelector<UsersState>('users');

const getUsersEntitites = createSelector(getUsersState, getEntities);
export const getUsersOriginalUser = createSelector(getUsersState, getOriginalUser);
export const getUsersLoaded = createSelector(getUsersState, getLoaded);
export const getUsersLoading = createSelector(getUsersState, getLoading);
export const getUsersError = createSelector(getUsersState, getError);

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

export const getEditedUser = createSelector(
    getUsersEntitites,
    getRouterState,
    (users, router): UserModel => {
        const userID = router.state.params.editedUserID;
        if (userID) {
            return users[userID];
```

```
        } else {
            return null;
        }
});

export const getSelectedUserByUrl = createSelector(
    getUsersEntitites,
    getRouterState,
    (users, router): UserModel => {
        const userID = router.state.params.userID;
        if (userID) {
            return users[userID];
        } else {
            return new UserModel(null, '', '');
        }
});
```

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

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

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

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

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

```
// 1
import { UsersEffects } from '../core/+store/users/users.effects';
import { StoreModule } from '@ngrx/store';
import { UsersEffects, usersReducer } from './../core/+store';

// 2
@NgModule({
  imports: [
    …
    UsersRoutingModule,
    StoreModule.forFeature('users', usersReducer),
  ],
…
})
```

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

```
// 1
import { ActivatedRoute, Params, Router } from '@angular/router';
import { UserObservableService } from './../../services';
import { switchMap } from 'rxjs/operators';
import { Store, select } from '@ngrx/store';
import * as UsersActions from './../../../core/+store/users/users.actions';
import { AppState, getUsers, getUsersError, getEditedUser } from
'./../../../core/+store';
import { Observable, Subscription, of } from 'rxjs';
import { AutoUnsubscribe } from './../../../core/decorators';
```

```
// 2
@AutoUnsubscribe('subscription')

// 3
usersError$: Observable<Error | string>;
private subscription: Subscription;

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

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

    // listen editedUserID from UserFormComponent
    this.route.paramMap
      .pipe(
        switchMap((params: Params) =>
             this.userObservableService.getUser(+params.get('editedUserID')))
      )
      .subscribe(
        (user: UserModel) => {
          this.editedUser = {...user};
          console.log(`Last time you edited user ${JSON.stringify(this.editedUser)}`);
        },
        err => console.log(err)
      );
  }
ngOnInit() {
    this.users$ = this.store.pipe(select(getUsers));
    this.usersError$ = this.store.pipe(select(getUsersError));
    this.store.dispatch(new UsersActions.GetUsers());

    // listen editedUserID from UserFormComponent
    this.subscription = this.store.pipe(select(getEditedUser))
    .subscribe(
      user => {
        this.editedUser = user;
        console.log(`Last time you edited user ${JSON.stringify(this.editedUser)}`);
      },
      err => console.log(err)
    );
  }

// 6
onDeleteUser(user: User) {
    this.users$ = this.userObservableService.deleteUser(user);
    this.store.dispatch(new UsersActions.DeleteUser(user));
}
```

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

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

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

```
// 1
import { Observable, Subscription, of } from 'rxjs';
import { pluck, switchMap } from 'rxjs/operators';
import { AutoUnsubscribe, DialogService, CanComponentDeactivate } from
'./../../../core';
import { UserObservableService } from './../../services';
import { ActivatedRoute, Router } from '@angular/router';
// @Ngrx
import { Store, select } from '@ngrx/store';
import { AppState, getUsersOriginalUser } from './../../../core/+store';
import * as UsersActions from './../../../core/+store/users/users.actions';

// 2
@AutoUnsubscribe()

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

// 4
constructor(
    …
    private store: Store<AppState>
    private router: Router,
    private userObservableService: UserObservableService
  ) { }

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

// 6
onSaveUser() {
    …

    const method = user.id ? 'updateUser' : 'createUser';
    const sub = this.userObservableService[method](user)
      .subscribe(
        () => {
          this.originalUser = {...this.user};
          user.id
            // optional parameter: http://localhost:4200/users;id=2
            ? this.router.navigate(['users', { editedUserID: user.id }])
            : this.onGoBack();
        },
        error => console.log(error)
      );
    this.sub.push(sub);
```

```
    if (user.id) {
      this.store.dispatch(new UsersActions.UpdateUser(user));
    } else {
      this.store.dispatch(new UsersActions.CreateUser(user));
    }
}

// 7
canDeactivate(): Observable<boolean> | Promise<boolean> | boolean {
    const flags = Object.keys(this.originalUser).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 = [];

    return this.store.pipe(
      select(getUsersOriginalUser),
      switchMap(originalUser => {
        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?');
      })
    );
}
```

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

```
// 1
// NgRx
import { Store, select } from '@ngrx/store';
import { AppState, getSelectedUserByUrl } from './../../core/+store';
import * as UsersActions from './../../core/+store/users/users.actions';
import { UserObservableService } from './../services';
import { Router, Resolve, ActivatedRouteSnapshot } from '@angular/router';
```

```
import { delay, map, catchError, finalize, tap, take } from 'rxjs/operators';

// 2
constructor(
    private userObservableService: UserObservableService,
    private store: Store<AppState>,
    …
  ) {}

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

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

    this.spinner.show();
    const id = +route.paramMap.get('userID');

    return this.userObservableService.getUser(id).pipe(
      delay(2000),
      map(user => {
        if (user) {
          return user;
        } else {
          this.router.navigate(['/users']);
          return of(null);
        }
      }),
      take(1),
      catchError(() => {
        this.router.navigate(['/users']);
        return of(null);
      }),
      finalize(() => this.spinner.hide())
    );
  }
}


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

    return this.store.pipe(
      select(getSelectedUserByUrl),
      tap(user => this.store.dispatch(new UsersActions.SetOriginalUser(user))),
      delay(2000),
      map(user => {
        if (user) {
          return user;
        } else {
          this.router.navigate(['/users']);
          return null;
        }
      }),
```

```
    take(1),
    catchError(() => {
      this.router.navigate(['/users']);
      // catchError MUST return observable
      return of(null);
    }),
    finalize(() => this.spinner.hide())
  );
}
```

Task 22. Navigation By Actions

1. Create fie **app/core/+store/router/router.actions.ts. .** Run the following command from command line:

> ng g a core/+store/router/router

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

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

import { NavigationExtras } from '@angular/router';

export enum RouterActionTypes {
  GO = '[Router] GO',
  BACK = '[Router] BACK',
  FORWARD = '[Router] FORWARD'
}

export class Go implements Action {
    readonly type = RouterActionTypes.GO;
    constructor(
        public payload: {
            path: any[],
            queryParams?: object,
            extras?: NavigationExtras
    }) { }
}

export class Back implements Action {
    readonly type = RouterActionTypes.BACK;
}

export class Forward implements Action {
    readonly type = RouterActionTypes.FORWARD;
}

export type RouterActions
 = Go
 | Back
 | Forward;
```

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

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

1. Create file **app/core/+store/router/router.effects.ts.** Run the following command from command line:

> ng g ef core/+store/router/router --root true -m core/+store/core-store.module.ts --spec false

2. 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 { Effect, Actions, ofType } from '@ngrx/effects';
import * as RouterActions from './router.actions';

import { tap, pluck } from 'rxjs/operators';

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

  @Effect({ dispatch: false })
  navigate$ = this.actions$.pipe(
    ofType<RouterActions.Go>(RouterActions.RouterActionTypes.GO),
    pluck('payload'),
    tap(({ path, queryParams, extras }) => {
      this.router.navigate(path, { queryParams, ...extras });
    })
  );

  @Effect({ dispatch: false })
  navigateBack$ = this.actions$.pipe(
    ofType<RouterActions.Back>(RouterActions.RouterActionTypes.BACK),
    tap(() => this.location.back())
  );

  @Effect({ dispatch: false })
  navigateForward$ = this.actions$.pipe(
    ofType<RouterActions.Forward>(RouterActions.RouterActionTypes.FORWARD),
    tap(() => this.location.forward())
  );
}
```

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

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

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

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

5. Make changes to file **app/core/+store/tasks/tasks.effects.ts.** Use the following snippet of code:

```
// 1
import * as RouterActions from './../router/router.actions';
import { Router } from '@angular/router';
import { pluck, concatMap, switchMap, map } from 'rxjs/operators';

// 2
constructor(
    private actions$: Actions,
    private router: Router,
    private taskPromiseService: TaskPromiseService
```

```
  ) {
    console.log('[TASKS EFFECTS]');
  }

// 3
@Effect()
  updateTask$: Observable<Action> = this.actions$.pipe(
    ofType<TasksActions.UpdateTask>(TasksActions.TasksActionTypes.UPDATE_TASK),
    pluck('payload'),
    concatMap((payload: TaskModel) =>
      this.taskPromiseService
        .updateTask(payload)
        .then(task => {
          this.router.navigate(['/home']);
          return new TasksActions.UpdateTaskSuccess(task);
        })
        .then(task => new TasksActions.UpdateTaskSuccess(task))
        .catch(err => new TasksActions.UpdateTaskError(err))
    )
  );

// 3
@Effect()
  createTask$: Observable<Action> = this.actions$.pipe(
    ofType<TasksActions.CreateTask>(TasksActions.TasksActionTypes.CREATE_TASK),
    pluck('payload'),
    concatMap((payload: TaskModel) =>
      this.taskPromiseService
        .createTask(payload)
        .then(task => {
          this.router.navigate(['/home']);
          return new TasksActions.CreateTaskSuccess(task);
        })
        .then(task => new TasksActions.CreateTaskSuccess(task))
        .catch(err => new TasksActions.CreateTaskError(err))
    )
  );

// 4
@Effect()
  createUpdateTaskSuccess$: Observable<Action> = this.actions$.pipe(
    ofType<TasksActions.CreateTask | TasksActions.UpdateTask>(
      TasksActions.TasksActionTypes.CREATE_TASK_SUCCESS,
      TasksActions.TasksActionTypes.UPDATE_TASK_SUCCESS
    ),
    map(
      action =>
        new RouterActions.Go({
          path: ['/home']
        })
    )
  );
```

10. Make changes to file **app/+store/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
@Effect()
  updateUser$: Observable<Action> = this.actions$.pipe(
    ofType<UsersActions.UpdateUser>(UsersActions.UsersActionTypes.UPDATE_USER),
    pluck('payload'),
    concatMap((payload: UserModel ) =>
      this.userObservableService.updateUser(payload).pipe(
        map(user => {
          this.router.navigate(['/users', { editedUserID: user.id }]);
          return new UsersActions.UpdateUserSuccess(user);
        }),
        map(user => new UsersActions.UpdateUserSuccess(user)),
        catchError(err => of(new UsersActions.UpdateUserError(err)))
      )
    )
  );

// 3
@Effect()
  createUser$: Observable<Action> = this.actions$.pipe(
    ofType<UsersActions.CreateUser>(UsersActions.UsersActionTypes.CREATE_USER),
    pluck('payload'),
    concatMap((payload: UserModel) =>
      this.userObservableService.createUser(payload).pipe(
        map(user => {
          this.router.navigate(['/users']);
          return new UsersActions.CreateUserSuccess(user);
        }),
        map(user => new UsersActions.CreateUserSuccess(user)),
        catchError(err => of(new UsersActions.CreateUserError(err)))
      )
    )
  );

// 4
@Effect()
  createUpdateUserSuccess$: Observable<Action> = this.actions$.pipe(
    ofType<UsersActions.CreateUser | UsersActions.UpdateUser>(
      UsersActions.UsersActionTypes.CREATE_USER_SUCCESS,
      UsersActions.UsersActionTypes.UPDATE_USER_SUCCESS
    ),
    pluck('payload'),
    map((user: UserModel) => {
      // in this case we always pass created and edited user
      const path = ['/users', { editedUserID: user.id }];
```

```
      return new RouterActions.Go({ path });
    })
  );
```

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

```
// 1
// @Ngrx
import { Store } from '@ngrx/store';
import { AppState } from './../+store';
import * as RouterActions from './../+store/router/router.actions';
import {
  CanActivate, CanActivateChild, CanLoad, Router, Route,
  ActivatedRouteSnapshot, RouterStateSnapshot, NavigationExtras
} from '@angular/router';

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

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

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

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

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

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

// 4
onEditTask(task: TaskModel) {
    const link = ['/edit', task.id];
```

```
    this.router.navigate(link);
    this.store.dispatch(new RouterActions.Go({
      path: link
    }));
}
```

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

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

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

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

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

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

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

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

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

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

// 2
constructor(
    …
```

```
    private location: Location
  ) { }

// 3
onGoBack() {
    this.location.back();
    this.store.dispatch(new RouterActions.Back());
  }
```

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

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

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

// 3
return this.store.pipe(
      select(getSelectedUserByUrl),
      tap(user => this.store.dispatch(new UsersActions.SetOriginalUser(user))),
      delay(2000),
      map(user => {
        if (user) {
          return user;
        } else {
          this.router.navigate(['/users']);
          this.store.dispatch(new RouterActions.Go({
              path: ['/users']
            }));
          return null;
        }
      }),
      take(1),
      catchError(() => {
        this.spinner.hide();
        this.router.navigate(['/users']);
        this.store.dispatch(new RouterActions.Go({
              path: ['/users']
            }));
        return of(null);
      }),
      finalize(() => this.spinner.hide())
    );
```

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

```
// 1
import { Router } from '@angular/router';
// @Ngrx
import { Store } from '@ngrx/store';
import { AppState } from './../../../core/+store';
```

```
import * as RouterActions from './../../../core/+store/router/router.actions';

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

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

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

```
// 1
// @Ngrx
import { Store } from '@ngrx/store';
import { AppState } from './core/+store';
import * as RouterActions from './core/+store/router/router.actions';

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

// 3
onDisplayMessages(): void {
    this.router.navigate([{ outlets: { messages: ['messages'] } }]);
    this.store.dispatch(new RouterActions.Go({
      path: [{ outlets: { messages: ['messages'] } }]
    }));
    this.messagesService.isDisplayed = true;
  }
```

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

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


// 2
constructor(
       public authService: AuthService,
       private router: Router,
```

```
      private store: Store<AppState>
) {}

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

Task 23. 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 --spec false**

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

```
import { select } from '@ngrx/store';
import { getTasksLoaded } from './../../core/+store';
import * as TasksActions from './../../core/+store/tasks/tasks.actions';

import { Observable } from 'rxjs';
import { tap, filter, take } from 'rxjs/operators';

export function checkStore(store): Observable<boolean> {
  return store.pipe(
    select(getTasksLoaded),

    // make a side effect
    tap((loaded: boolean) => {
      if (!loaded) {
        store.dispatch(new 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 { CanActivate } from '@angular/router';

// ngrx
import { Store } from '@ngrx/store';
import { AppState } from './../../core/+store';

// rxjs
import { Observable, of } from 'rxjs';
import { catchError, switchMap } from 'rxjs/operators';

import { TasksServicesModule } from '../tasks-services.module';
import { checkStore } from './check-store.function';

@Injectable({
  providedIn: TasksServicesModule
})
export class TasksStatePreloadingGuard implements CanActivate {
  constructor(private store: Store<AppState>) {}
```

```
  canActivate(): Observable<boolean> {
    return checkStore(this.store).pipe(
      switchMap(() => of(true)),
      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 --spec false**

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

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot } from '@angular/router';

// ngrx
import { Store, select } from '@ngrx/store';
import { AppState, getTasksData } from './../../core/+store';
import * as RouterActions from './../../core/+store/router/router.actions';

// rxjs
import { Observable } from 'rxjs';
import { map, switchMap, take, tap } from 'rxjs/operators';

import { TasksServicesModule } from '../tasks-services.module';
import { checkStore } from './check-store.function';

@Injectable({
  providedIn: TasksServicesModule
})
export class TaskExistGuard implements CanActivate {
  constructor(private store: Store<AppState>) {}

  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.pipe(
      select(getTasksData),

      // check if task with id exists
      map(tasks => !!tasks.find(task => task.id === id)),

      // make a side effect
      tap(result => {
        if (!result) {
```

```
          this.store.dispatch(new 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, TaskExistGuard } from './guards';

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

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

```
ngOnInit() {
    …
    this.store.dispatch(new 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 --spec false**

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

```
import { Injectable } from '@angular/core';
import { CanActivate } from '@angular/router';

import { Store, select } from '@ngrx/store';
import { AppState, getUsersLoaded } from './../../core/+store';
import * as UsersActions from './../../core/+store/users/users.actions';

import { Observable, of } from 'rxjs';
import { catchError, switchMap, take, tap } from 'rxjs/operators';

import { UsersServicesModule } from '../users-services.module';
```

```
@Injectable({
  providedIn: UsersServicesModule
})
export class UsersStatePreloadingGuard implements CanActivate {

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

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

  private checkStore(): Observable<boolean> {
    return this.store.pipe(
      select(getUsersLoaded),
      tap(loaded => {
        if (!loaded) {
          this.store.dispatch(new 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,
        canDeactivate: [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(new UsersActions.GetUsers());

    …
  }
```

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

```
// 1
import { AppState, getUsersOriginalUser, getSelectedUserByUrl } from
'./../../../core/+store';
import { Observable, of, Subscription } from 'rxjs';
import { ActivatedRoute } from '@angular/router';
import { AutoUnsubscribe, DialogService, CanComponentDeactivate } from
'./../../../core';

// 2
@AutoUnsubscribe()

// 3
private sub: Subscription;

// 4
constructor(
    private route: ActivatedRoute,

    …
  ) { }

// 5
ngOnInit(): void {
    this.route.data.subscribe(data => {
      this.user = {...data.user};
    });
    this.sub = this.store.pipe(select(getSelectedUserByUrl))
      .subscribe(user => this.user = 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 24. @ngrx/entity

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

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

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

// 3
export const taskAdapter: EntityAdapter<TaskModel> = createEntityAdapter<TaskModel>();

// 4
export const intitialTasksState: TasksState = taskAdapter.getInitialState({
  data: [],
  loading: false,
  loaded: false,
  error: null
});
```

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

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

// 2
Удалите функцию tasksReducer

// 3
export function tasksReducer(
  state = initialTasksState,
  action: TasksActions
): TasksState {
  console.log(`Reducer: Action came in! ${action.type}`);

  switch (action.type) {

    case TasksActionTypes.GET_TASKS: {
      return {
        ...state,
        loading: true
      };
    }

    case TasksActionTypes.GET_TASKS_SUCCESS: {
      const tasks = [...<Array<TaskModel>>action.payload];

      return taskAdapter.addAll(tasks, {...state, loading: false, loaded: true});
    }
```

```
    case TasksActionTypes.GET_TASKS_ERROR: {
      const error = action.payload;
      return {
        ...state,
        loading: false,
        loaded: false,
        error
      };
    }

    case TasksActionTypes.CREATE_TASK_SUCCESS: {
      const task = { ...<TaskModel>action.payload };

      return taskAdapter.addOne(task, state);
    }

    case TasksActionTypes.UPDATE_TASK_SUCCESS: {
      const task = { ...<TaskModel>action.payload };

      return taskAdapter.updateOne({
          id: task.id,
          changes: task
      }, state);
    }

    case TasksActionTypes.DELETE_TASK_SUCCESS: {
      const task = { ...<TaskModel>action.payload };

      return taskAdapter.removeOne(task.id, state);
    }

    case TasksActionTypes.CREATE_TASK_ERROR:
    case TasksActionTypes.UPDATE_TASK_ERROR:
    case TasksActionTypes.DELETE_TASK_ERROR: {
      const error = action.payload;
      return {
        ...state,
        error
      };
    }

    default: {
      return state;
    }
  }
}
```

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

```
// 1
import { taskAdapter, TasksState } from './tasks.state';

// 2
export const getTasksData = createSelector(getTasksState, (state: TasksState) =>
state.data);
export const {
```

```
    selectEntities: getTasksEntities,
    selectAll: getTasksData
} = taskAdapter.getSelectors(getTasksState);

// 3
export const getSelectedTaskByUrl = createSelector(
    getTasksData,
    getTasksEntities
    getRouterState,
    (tasks, router): Task => {
        const taskID = router.state.params.taskID;
        if (taskID) {
            return tasks.find(task => task.id === +taskID);
            return tasks[taskID];
        } else {
            return new TaskModel();
        }
});
```