Angular 2+

Workshop. Ngrx.

Contents

Task 01. Install @ngrx/store 2
Task 02. Create a State 3
Task 03. Create Actions 4
Task 04. Create a Reducer 6
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. Install Redux DevTools Extension 13
Task 10. Create Effects Class 14
Task 11. Provide Effects 15
Task 12. Get Tasks from DataBase 16
Task 13. Get Task from DataBase 20
Task 14. Update Task in DataBase 23
Task 15. Add Task to DataBase 25
Task 16. Delete Task from DataBase 27
Task 17. Replace DoneTask w/ UpdateTask Action 29
Task 18. Feature Selector 30
Task 19. State Selector 31
Task 20. Router State 33
Task 21. Compose Router and Task Selectors 35
Task 22. Users Store 36
Task 23. Navigation By Actions 49
Task 23. State Preloading 56
Task 24. @ngrx/entity 62

Task 01. Install @ngrx/store

1. To install **@ngrx** run the following commands from command line:

```
> npm install @ngrx/{store ,effects, entity, router-store} --save
> npm install @ngrx/store-devtools --save-dev
```

Task 02. Create a State

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

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

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

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

2. Create filr **app/+store/state/app.state.ts.** Use the following snippet of code:

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

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

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

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

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

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

Task 03. Create Actions

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

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

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

// [Tasks]- namespace
export class TasksActionTypes {
  static readonly GET_TASKS =   '[Tasks] GET_TASKS';
  static readonly GET_TASK =    '[Tasks] GET_TASK';
  static readonly CREATE_TASK = '[Tasks] CREATE_TASK';
  static readonly UPDATE_TASK = '[Tasks] UPDATE_TASK';
  static readonly DELETE_TASK = '[Tasks] DELETE_TASK';
}

export class GetTasks implements Action {
  readonly type = TasksActionTypes.GET_TASKS;
  constructor(public payload?: Task) { }
}

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

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

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

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

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

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

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

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

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

Task 04. Create a Reducer

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

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

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

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

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

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

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

```
// 1
import { ActionReducerMap } from '@ngrx/store';
import { tasksReducer } from './../reducers';

// 2
export const reducers: ActionReducerMap<AppState> = {
   tasks: tasksReducer
```

```
};
```

Task 05. Provide Store

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

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

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

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

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

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

Task 06. Inject Store

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

```
// 1
// @Ngrx
import { Store } from '@ngrx/store';
import { AppState } from './../../+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 07. Reading Data From The Store

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

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

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

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

    this.getTasks().catch(err => console.log(err));
}

// 3
private async getTasks() {
    this.tasks = await this.taskPromiseService.getTasks();
}
```

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

```
<app-task *ngFor='let task of tasks'
<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.actions.ts.** Use the following snippet of code:

```
// 1
export class TasksActionTypes {
  …
  static readonly DELETE_TASK = '[Tasks] DELETE_TASK';
  static readonly DONE_TASK =   '[Tasks] DONE_TASK';
}

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

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

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

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

// 2
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};
        }

        return task;
      });

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

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

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


// 2
completeTask(task: Task): void {
    task.done = true;
```

```
    this.taskPromiseService.updateTask(task);
    this.store.dispatch(new TasksActions.DoneTask(task));
}
```

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

Task 09. 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 **AppModule**. 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() : [],
    AppRoutingModule
],
```

Task 10. Create Effects Class

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

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

// @Ngrx
import { Actions } from '@ngrx/effects';

@Injectable()
export class TasksEffects {

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

}
```

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

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

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

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

Task 11. Provide Effects

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

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

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

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

```
// 1
import { StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import { tasksReducer } from './../+state/reducers';
import { TasksEffects } from '../+state/effects';

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

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 12. Get Tasks from DataBase

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

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

// 2
export const initialTasksState: State = {
    data: [
      new Task(1, 'Estimate', 1, 8, 8, true),
      new Task(2, 'Create', 2, 8, 4, false),
      new Task(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 class TasksActionTypes {
  static readonly GET_TASKS         = '[Tasks] GET_TASKS';
  static readonly GET_TASKS_SUCCESS = '[Tasks] GET_TASKS_SUCCESS';
  static readonly GET_TASKS_ERROR   = '[Tasks] GET_TASKS_ERROR';
  …
}

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

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<Task>>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
      };
    }

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

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

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

```
// 1
import { Action } from '@ngrx/store';
import { Actions, Effect } from '@ngrx/effects';
import { TasksActionTypes } from './../actions';
```

```
import * as TasksActions from './../actions/tasks.actions';

import { Observable } from 'rxjs/Observable';
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$: Observable<Action> = this.actions$
    // 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>(TasksActionTypes.GET_TASKS)
    .pipe(
      switchMap(action =>
        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
tasks: Array<Task>;

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

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

// 3
deleteTask(task: Task) {
    // this.taskPromiseService.deleteTask(task)
    //   .then(() => this.tasks = this.tasks.filter(t => t !== task))
    //   .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"
```

```
    (complete)="completeTask($event)"
    (edit)="editTask($event)"
    (delete)="deleteTask($event)">
</task>
```

7. Look to the browser console.

Task 13. Get Task from DataBase

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

```
// 1
export interface TasksState {
  data: ReadonlyArray<Task>;
  selectedTask: Readonly<Task>;
  …
}

// 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 class TasksActionTypes {
  …
  static readonly GET_TASK          = '[Tasks] GET_TASK';
  static readonly GET_TASK_SUCCESS  = '[Tasks] GET_TASK_SUCCESS';
  static readonly GET_TASK_ERROR    = '[Tasks] GET_TASK_ERROR';
  static readonly CREATE_TASK       = '[Tasks] CREATE_TASK';
  …
}
// 2
export class GetTaskSuccess implements Action {
  readonly type = TasksActionTypes.GET_TASK_SUCCESS;
  constructor(public payload: Task) { }
}

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

```
// import { switchMap } from 'rxjs/operators';
import { Subscription } from 'rxjs/Subscription';
import { AutoUnsubscribe } from '../../core';

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

// 3
tasksState$: Store<TasksState>;

private sub: Subscription;

// 4
constructor(
    private taskArrayService: TaskArrayService,
    private taskPromiseService: TaskPromiseService,
    private router: Router,
    private route: ActivatedRoute,
    private store: Store<AppState>
) { }

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

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

    this.route.paramMap.subscribe(params => {
      const id = params.get('id');
      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 { Router } from '@angular/router';
import { map, switchMap } from 'rxjs/operators';

// 2
constructor(
    …
```

```
    private router: Router
) {
    console.log('[TASKS EFFECTS]');
}

// 3
@Effect() getTask$: Observable<Action> = this.actions$
    .ofType<TasksActions.GetTask>(TasksActionTypes.GET_TASK)
    .pipe(
      map((action: TasksActions.GetTask) => action.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 = { ...<Task>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 14. Update Task in DataBase

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

```
// 1
export class TasksActionTypes {
  …
  static readonly UPDATE_TASK_SUCCESS = '[Tasks] UPDATE_TASK_SUCCESS';
  static readonly UPDATE_TASK_ERROR   = '[Tasks] UPDATE_TASK_ERROR';
  …
}
// 2
export class UpdateTaskSuccess implements Action {
  readonly type = TasksActionTypes.UPDATE_TASK_SUCCESS;
  constructor(public payload: Task) { }
}

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 { TaskArrayService, TaskPromiseService } from './../services';

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

// 2
const method = task.id ? 'updateTask' : 'createTask';
this.taskPromiseService[method](task)
   .then(() => this.goBack());

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:

```
@Effect() updateTask$: Observable<Action> = this.actions$
```

```
    .ofType<TasksActions.UpdateTask>(TasksActionTypes.UPDATE_TASK)
    .pipe(
      map((action: TasksActions.UpdateTask) => action.payload),
      switchMap(payload =>
        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 = { ...<Task>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 15. Add Task to DataBase

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

```
// 1
export class TasksActionTypes {
  …
  static readonly CREATE_TASK_SUCCESS = '[Tasks] CREATE_TASK_SUCCESS';
  static readonly CREATE_TASK_ERROR   = '[Tasks] CREATE_TASK_ERROR';
  …
}

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

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 Task(null, '', null, null);

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

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

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

```
@Effect() createTask$: Observable<Action> = this.actions$
    .ofType<TasksActions.CreateTask>(TasksActionTypes.CREATE_TASK)
    .pipe(
      map((action: TasksActions.CreateTask) => action.payload),
      switchMap(payload =>
        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 = { ...<Task>action.payload };
      const data = [...state.data];

      data.push(task);

      return {
        ...state,
        data
      };
}

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

Task 16. Delete Task from DataBase

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

```
// 1
export class TasksActionTypes {
  …
  static readonly DELETE_TASK_SUCCESS = '[Tasks] DELETE_TASK_SUCCESS';
  static readonly DELETE_TASK_ERROR   = '[Tasks] DELETE_TASK_ERROR';
  …
}

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

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:

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

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

// 3
deleteTask(task: Task) {
    this.store.dispatch(new TasksActions.DeleteTask(task));
    // this.taskPromiseService.deleteTask(task)
    //  .then(() => this.tasks = this.tasks.filter(t => t !== task))
    //  .catch(err => console.log(err));
}
```

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

```
@Effect() deleteTask$: Observable<Action> = this.actions$
    .ofType<TasksActions.DeleteTask>(TasksActionTypes.DELETE_TASK)
    .pipe(
```

```
      map((action: TasksActions.DeleteTask) => action.payload),
      switchMap(payload =>
        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 = { ...<Task>action.payload };
      const data = [...state.data];
      const index = data.findIndex(t => t.id === task.id);

      data.splice(index, 1);

      return {
        ...state,
        data
      };
}

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

Task 17. Replace DoneTask w/ UpdateTask Action

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

```
completeTask(task: Task): void {
    this.store.dispatch(new 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 class TasksActionTypes {
  …
  static readonly 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 data = state.data.map(task => {
        if (task.id === (<Task>action.payload).id) {
          return {...action.payload, done: true};
        } else {
          return task;
        }
      });
      return {
        ...state,
        data,
        error: null
      };
    }
```

Task 18. Feature Selector

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Task 19. State Selector

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

```
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
'./../../+store';

// 2
tasksState$: Store<any>;
tasks$: Store<ReadonlyArray<Task>>;
tasksError$: Store<Error | string>;

// 3
ngOnInit() {
    console.log('We have a store! ', this.store);
    this.tasksState$ = this.store.select(getTasksState);
    this.tasks$ = this.store.select(getTasksData);
    this.tasksError$ = this.store.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
<task *ngFor='let task of (tasksState$ | async).data'
<task *ngFor='let task of (tasks$ | async)'
```

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

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

// 2
tasksState$: Store<TasksState>;


// 3
ngOnInit(): void {
```

```
    this.tasksState$ = this.store.select(getTasksState);
    this.sub = this.tasksState$.subscribe(tasksState => {
      if (tasksState.selectedTask) {
        this.task = tasksState.selectedTask;
      } else {
        this.task = new Task(null, '', null, null);
      }
    });

    this.sub = this.store.select(getSelectedTask)
     .subscribe(task => {
      if (task) {
        this.task = task;
      } else {
        this.task = new Task(null, '', null, null);
      }
    });
…
}
```

Task 20. Router State

1. Create file **app/+store/state/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. Make changes to file **app/+store/state/index.ts.** Use the following snippet of code:

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

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

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

```
import { RouterStateSerializerProvider } from './+store';

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

Task 21. Compose Router and Task Selectors

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

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

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

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

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

// 2
export const getSelectedTaskByUrl = createSelector(
    getTasksData,
    getRouterState,
    (tasks, router): Task => {
        const taskID = router.state.params.id;
        if (taskID) {
            return tasks.find(task => task.id === +taskID);
        } else {
            return new Task(null, '', null, null);
        }
});
```

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

```
// 1
import { AppState, getSelectedTask, getSelectedTaskByUrl } from './../../+store';

// 2
ngOnInit(): void {
    this.sub = this.store.select(getSelectedTask)
    .subscribe(task => {
      if (task) {
        this.task = task;
      } else {
        this.task = new Task(null, '', null, null);
      }
    });
    this.sub = this.store.select(getSelectedTaskByUrl)
      .subscribe(task => this.task = task);

…
  }
```

Task 22. Users Store

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

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

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

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

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

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

3. Create file **app/+store/actions/users.actions.ts.** Use the following snippet of code:

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

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

// Actions
// [Users] - namespace
export class UsersActionTypes {
  static readonly GET_USERS           = '[Users] GET_USERS';
  static readonly GET_USERS_SUCCESS   = '[Users] GET_USERS_SUCCESS';
  static readonly GET_USERS_ERROR     = '[Users] GET_USERS_ERROR';
  static readonly GET_USER            = '[Users] GET_USER';
  static readonly GET_USER_SUCCESS    = '[Users] GET_USER_SUCCESS';
  static readonly GET_USER_ERROR      = '[Users] GET_USER_ERROR';
  static readonly CREATE_USER         = '[Users] CREATE_USER';
  static readonly CREATE_USER_SUCCESS = '[Users] CREATE_USER_SUCCESS';
  static readonly CREATE_USER_ERROR   = '[Users] CREATE_USER_ERROR';
  static readonly UPDATE_USER         = '[Users] UPDATE_USER';
  static readonly UPDATE_USER_SUCCESS = '[Users] UPDATE_USER_SUCCESS';
  static readonly UPDATE_USER_ERROR   = '[Users] UPDATE_USER_ERROR';
  static readonly DELETE_USER         = '[Users] DELETE_USER';
  static readonly DELETE_USER_SUCCESS = '[Users] DELETE_USER_SUCCESS';
  static readonly DELETE_USER_ERROR   = '[Users] DELETE_USER_ERROR';
}

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

```
  constructor(public payload?: User) {}
}

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

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: User) {}
}

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: User) {}
}

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

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: User) {}
}

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

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: User) {}
}

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

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

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

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

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

5. Create file **app/+store/reducers/users.reducer.ts.** Use the following snippet of code:

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

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

export function usersReducer (
    state = initialUsersState,
    action: fromUsers.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 = <User[]>action.payload;
  console.log(users);

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

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

case UsersActionTypes.GET_USER_SUCCESS: {
  return {
    ...state,
    loading: false,
    loaded: true
  };
}

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 = <User>action.payload;
        const entities = {
          ...state.entities,
          [user.id]: user
        };
        const originalUser = {...<User>action.payload};

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

      case UsersActionTypes.DELETE_USER_SUCCESS: {
        const user = <User>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
        };
      }

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

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

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

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

```
// 1
import { UsersState } from './users.state';
import { tasksReducer, usersReducer} from './../reducers';

// 2
export interface AppState {
  tasks: TasksState;
```

```
  users: UsersState;
}

// 3
export const reducers: ActionReducerMap<AppState> = {
  tasks: tasksReducer,
  users: usersReducer
};
```

8. Create file **app/+store/effects/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 } from '@ngrx/effects';
import { UsersActionTypes } from './../actions';
import * as UsersActions from './../actions/users.actions';

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

import { UserObservableService } from './../../users/services';

@Injectable()
export class UsersEffects {

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

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

  @Effect() updateUser$: Observable<Action> = this.actions$
```

```
    .ofType<UsersActions.UpdateUser>(UsersActionTypes.UPDATE_USER)
    .pipe(
      map((action: UsersActions.UpdateUser) => action.payload),
      switchMap(payload =>
        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$: Observable<Action> = this.actions$
    .ofType<UsersActions.CreateUser>(UsersActionTypes.CREATE_USER)
    .pipe(
      map((action: UsersActions.CreateUser) => action.payload),
      switchMap(payload =>
        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$: Observable<Action> = this.actions$
    .ofType<UsersActions.DeleteUser>(UsersActionTypes.DELETE_USER)
    .pipe(
      map((action: UsersActions.DeleteUser) => action.payload),
      switchMap(payload =>
        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)))
        )
      )
    );

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

}
```

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

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

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

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

import { UsersState } from './../state';
import { User } from './../../users/models/user.model';
import { getRouterState } from './../../+store/selectors/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): User => {
        const userID = router.state.params.editedUserID;
        if (userID) {
            return users[userID];
        } else {
            return null;
        }
});

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

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

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

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

```
// 1
import { StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import { UsersEffects, usersReducer } from 'app/+store';

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

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

```
// 1
import { UserArrayService, UserObservableService } from './../services';
import { Store } from '@ngrx/store';
import * as UsersActions from './../../+store/actions/users.actions';
import { AppState, getUsers, getUsersError, getEditedUser } from './../../+store';
import { Subscription } from 'rxjs/Subscription';
import { AutoUnsubscribe } from './../../core/decorators';

// 2
@AutoUnsubscribe('subscription')

// 3
users$: Observable<Array<User>>;
users$: Store<Array<User>>;
usersError$: Store<Error | string>;
private subscription: Subscription;

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

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

    // listen editedUserID from UserFormComponent
    this.route.paramMap
      .pipe(
```

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

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

// 6
deleteUser(user: User) {
    this.users$ = this.userObservableService.deleteUser(user);
}

deleteUser(user: User) {
    this.store.dispatch(new UsersActions.DeleteUser(user));
}
```

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

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

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

```
// 1
import { Subscription } from 'rxjs/Subscription';
import { AutoUnsubscribe } from './../../core';
import { ActivatedRoute, Params, Router } from '@angular/router';
// @Ngrx
import { Store } from '@ngrx/store';
import { AppState, getUsersOriginalUser } from './../../+store';
import * as UsersActions from './../../+store/actions/users.actions';
import { of } from 'rxjs/observable/of';

// 2
@AutoUnsubscribe()

// 3
originalUser: User;
```

```
private sub: Subscription;

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

// 5
ngOnInit(): void {
    this.user = new User(null, '', '');

    // data is an observable object
    // which contains custom and resolve data
    this.route.data.subscribe(data => {
      this.user = Object.assign({}, data.user);
      this.originalUser = Object.assign({}, data.user);
    });
}
ngOnInit(): void {
    this.route.data.subscribe(data => {
      this.user = {...data.user};
    });
  }

// 6
saveUser() {
    …

    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.goBack();
        },
        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.select(getUsersOriginalUser)
            .pipe(
              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?');
              })
            );
}
```

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

```
// 1
// NgRx
import { Store } from '@ngrx/store';
import { AppState, getSelectedUserByUrl } from './../../+store';
import { UserArrayService, UserObservableService } from './../services';
import { Router, Resolve, ActivatedRouteSnapshot } from '@angular/router';
import { switchMap, take } from 'rxjs/operators';

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

// 3
resolve(route: ActivatedRouteSnapshot): Observable<User> | null {
    console.log('UserResolve Guard is called');
    const id = +route.paramMap.get('userID');

    if (id) {
```

```
      return this.userObservableService.getUser(id)
      .pipe(
        catchError(() => {
          this.router.navigate(['/users']);
          return of(null);
        })
      );
    } else {
      return of(new User(null, '', ''));
    }
  }

resolve(): Observable<User> {
  return this.store.select(getSelectedUserByUrl)
      .pipe(
        switchMap(user => {
          if (user) {
            return of(user);
          } else {
            this.router.navigate(['/users']);
            return of(null);
          }
        }),
        first()
      );
}
```

Task 23. Navigation By Actions

1. Create fie **app/+store/actions/router.actions.ts.** Use the following snippet of code:

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

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

export class RouterActionTypes {
    static readonly GO      = '[Router] GO';
    static readonly BACK    = '[Router] BACK';
    static readonly 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;
```

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

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

3. Create file **app/+store/effects/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 } from '@ngrx/effects';
import { RouterActionTypes } from './../actions';
import * as RouterActions from './../actions/router.actions';

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

@Injectable()
export class RouterEffects {
```

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

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

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

    constructor(
        private actions$: Actions,
        private router: Router,
        private location: Location
    ) {}
}
```

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

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

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

```
// 1
import { RouterStateSerializerProvider, routerReducers, RouterEffects } from './+store';

// 2
EffectsModule.forRoot([RouterEffects]),
```

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

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

// 2
@Effect() updateTask$: Observable<Action> = this.actions$
    .ofType(TasksActionTypes.UPDATE_TASK)
    .pipe(
      map((action: TasksActions.UpdateTask) => action.payload),
      switchMap(payload =>
        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$: Observable<Action> = this.actions$
    .ofType(TasksActionTypes.CREATE_TASK)
    .pipe(
      map((action: TasksActions.CreateTask) => action.payload),
      switchMap(payload =>
        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$: Observable<Action> = this.actions$
    .ofType<TasksActions.CreateTask | TasksActions.UpdateTask>(
      TasksActionTypes.CREATE_TASK_SUCCESS,
      TasksActionTypes.UPDATE_TASK_SUCCESS
    )
    .pipe(
      map(action => new RouterActions.Go({
        path: ['/home']
      }))
);
```

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

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

// 2
@Effect() updateUser$: Observable<Action> = this.actions$
    .ofType(UsersActionTypes.UPDATE_USER)
    .pipe(
      map((action: UsersActions.UpdateUser) => action.payload),
      switchMap(payload =>
        this.userObservableService.updateUser(payload)
        .pipe(
            map(user => {
                this.router.navigate(['/users']);
                return new UsersActions.UpdateUserSuccess(user);
            }),
            map(user => new UsersActions.UpdateUserSuccess(user)),
            catchError(err => of(new UsersActions.UpdateUserError(err)))
        )
      )
    );

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

```
    .ofType(UsersActionTypes.CREATE_USER)
    .pipe(
      map((action: UsersActions.CreateUser) => action.payload),
      switchMap(payload =>
        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$: Observable<Action> = this.actions$
    .ofType<UsersActions.CreateUser | UsersActions.UpdateUser>(
      UsersActionTypes.CREATE_USER_SUCCESS,
      UsersActionTypes.UPDATE_USER_SUCCESS
    )
    .pipe(
      map(action => (<any>action).payload),
      map(user => {
        const path = user.id
          ? ['/users', { editedUserId: user.id }]
          : ['/users'];
        return new RouterActions.Go({ path });
      })
);
```

6. 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/actions/router.actions';

// 2
constructor(
    …
    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
}));
}
```

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

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

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

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

// 4
editTask(task: Task) {
    const link = ['/edit', task.id];
    this.router.navigate(link);
    this.store.dispatch(new RouterActions.Go({
      path: link
    }));
}
```

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

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

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

// 3
goBack(): void {
    this.location.back();
    this.store.dispatch(new RouterActions.Go({
      path: ['/home']
    }));
}
```

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

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

// 2
```

```
  constructor(
    …
    private router: Router
  ) { }

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

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

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

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

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

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

```
// 1
import * as RouterActions from './../../+store/actions/router.actions';

// 2
return this.store.select(getSelectedUserByUrl)
      .pipe(
        switchMap(user => {
          if (user) {
            return of(user);
          } else {
            this.router.navigate(['/users']);
            this.store.dispatch(new RouterActions.Go({
              path: ['/users']
            }));
            return of(null);
          }
        }),
        take(1)
      );
```

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

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

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

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

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

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

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

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

Task 24. State Preloading

1. Create file **app/tasks/guards/tasks-state-preloading.guard.ts.** Use the following snippet of code:

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

import { Store } from '@ngrx/store';
import { AppState, getTasksLoaded } from './../../+store';
import * as TasksActions from './../../+store/actions/tasks.actions';

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

@Injectable()
export class TasksStateLoadingGuard implements CanActivate {

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

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

    private checkStore(): Observable<boolean> {
        return this.store.select(getTasksLoaded)
            .pipe(
                tap(loaded => {
                    if (!loaded) {
                        this.store.dispatch(new TasksActions.GetTasks);
                    }
                }),
                take(1)
            );
    }
}
```

2. Create file **app/tasks/guards/task-exists.guard.ts.** Use the following snippet of code:

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

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

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

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

```
@Injectable()
export class TaskExistGuard implements CanActivate {

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

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

    private hasTask(id: number): Observable<boolean> {
        return this.store.select(getTasksData)
        .pipe(
           map(tasks => !!tasks.find(task => task.id === id)),
           tap(result => {
               if (!result) {
                   this.store.dispatch(new RouterActions.Go({path: ['/home']}));
               }
           }),
           take(1)
        );
    }

    private checkStore(): Observable<boolean> {
        return this.store.select(getTasksLoaded)
            .pipe(
                tap(loaded => {
                    if (!loaded) {
                        this.store.dispatch(new TasksActions.GetTasks);
                    }
                }),
                take(1)
            );
    }
}
```

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

```
import { TaskExistGuard } from './task-exists.guard';
import { TasksStateLoadingGuard } from './tasks-state-loading.guard';

export const allGuards: any[] = [TaskExistGuard, TasksStateLoadingGuard];

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

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

```
// 1
import * as Guards from './guards';
```

```
// 2
{
    path: 'home',
    component: TaskListComponent,
    canActivate: [Guards.TasksStateLoadingGuard],
    …
},
{
    path: 'edit/:id',
    component: TaskFormComponent,
    canActivate: [Guards.TaskExistGuard]
  }

// 3
providers: [
    ...Guards.allGuards
],
```

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

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

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

```
// 1
import { ActivatedRoute } from '@angular/router';

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

// 3
ngOnInit(): void {
    …

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

7. Create file **app/users/guards/users-state-preloading.guard.ts.** Use the following snippet of code:

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

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

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

@Injectable()
export class UsersStateLoadingGuard implements CanActivate {

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

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

    private checkStore(): Observable<boolean> {
        return this.store.select(getUsersLoaded)
            .pipe(
                tap(loaded => {
                    if (!loaded) {
                        this.store.dispatch(new UsersActions.GetUsers);
                    }
                }),
                take(1)
            );
    }
}
```

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

```
// 1
import { UsersStateLoadingGuard } from './guards/users-state-loading.guard';
import { UserResolveGuard } from './../guards/user-resolve.guard';

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

// 3
providers: [
    CanDeactivateGuard,
```

```
    UsersStateLoadingGuard
  ],
```

9. 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());

    …
  }
```

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

```
// 1
import { Router } from '@angular/router';
import { AppState, getUsersOriginalUser, getSelectedUserByUrl } from './../../+store';
import { Subscription } from 'rxjs/Subscription';
import { UserObservableService } from './../services/user-observable.service';
import { AutoUnsubscribe } from '../../core';

// 2
@AutoUnsubscribe()

// 3
private sub: Subscription;

// 4
constructor(
    private userObservableService: UserObservableService,
    private route: ActivatedRoute,
    private router: Router,
    …
  ) { }

// 5
ngOnInit(): void {
    this.route.data.subscribe(data => {
      this.user = {...data.user};
    });
    this.sub = this.store.select(getSelectedUserByUrl)
      .subscribe(user => this.user = user);
  }
```

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

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

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

```
// 1
import { UserComponent, UserArrayService, UserObservableService, UserResolveGuard } from
'.';

// 2
providers: [
```

```
  …
  UserResolveGuard
]
```

13. Delete **UserResolveGuard.**

Task 25. @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<Task> {
  data: ReadonlyArray<Task>;
  selectedTask: Readonly<Task>;
  readonly loading: boolean;
  readonly loaded: boolean;
  readonly error: Error | string;
}

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

// 4
export const intitialTasksState: TasksState = taskAdapter.getInitialState({
  data: [],
  selectedTask: null,
  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 './../state/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:
    case TasksActionTypes.GET_TASK: {
      return {
        ...state,
        loading: true
      };
    }

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

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


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

case TasksActionTypes.GET_TASK_SUCCESS: {
  const selectedTask = { ...<Task>action.payload };
  return {
    ...state,
    loading: false,
    loaded: true,
    selectedTask
  };
}

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

  return taskAdapter.addOne(task, state);
}

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

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

case TasksActionTypes.DELETE_TASK_SUCCESS: {
  const task = { ...<Task>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 './../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.id;
        if (taskID) {
            return tasks.find(task => task.id === +router.state.params.id);
            return tasks[taskID];
        } else {
            return new Task(null, '', null, null);
        }
});
```