

Angular 2+

Workshop. @NgRx.

Contents

Explanation of Colors	3
Task 01. Setup Schematics	4
Task 02. Create a State	5
Task 03. Create Actions	6
Task 04. Create a Reducer	7
Task 05. Provide Store	8
Task 06. Inject Store	9
Task 07. Reading Data From The Store	10
Task 08. Dispatching an Event To The Store	11
Task 09. Create Meta-Reducer	12
Task 10. Install Redux DevTools Extension	13
Task 11. Create Effects Class	14
Task 12. Provide Effects	15
Task 13. Get Tasks from DataBase	16
Task 14. Get Task from DataBase	19
Task 15. Update Task in DataBase	22
Task 16. Add Task to DataBase	24
Task 17. Delete Task from DataBase	26
Task 18. Replace DoneTask w/ UpdateTask Action	28
Task 19. Effects Interfaces	29
Task 20. Feature Selector	30
Task 21. State Selector	31
Task 22. Router State	33
Task 23. Compose Router and Task Selectors	35
Task 24. Router Built-in Selectors	38
Task 25. Users Store	40
Task 26. Navigation By Actions	53
Task 27. State Preloading	61
Task 28. @ngrx/entity	66

Task 29. @ngrx/data	69
Task 30. Use Facade	74

Explanation of Colors

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

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

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

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

Task 01. Setup Schematics

1. Run the following command from command line:

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

Task 02. Create a State

1. Make changes to the file `task.model.ts`. Use the following snippet of code:

```
// 1
export interface Task {
  id?: number;
  action?: string;
  priority?: number;
  estHours?: number;
  actHours?: number;
  done?: boolean;
}

// 2
export class TaskModel implements Task {...}
```

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

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

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

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)
  ]
};
```

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

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

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

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

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

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

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

Task 03. Create Actions

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

> ng g a core/@ngrx/tasks/tasks -c true

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

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

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

export const getTasks = createAction('[Tasks] GET_TASKS');

export const getTask = createAction(
  '[Tasks] GET_TASK',
  props<{ taskID: number }>()
);

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

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

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

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

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

Task 04. Create a Reducer

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

ng g r core/@ngrx/tasks/tasks --spec false

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

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

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

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

export function tasksReducer(state: TasksState | undefined, action: Action) {
  return reducer(state, action);
}
```

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

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

Task 05. Provide Store

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

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

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

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

// 2
@NgModule({
  ...
  imports: [
    CommonModule
    StoreModule.forRoot({}, {
      runtimeChecks: {
        strictStateImmutability: true,
        strictActionImmutability: true,
        strictStateSerializability: true,
        strictActionSerializability: true
      }
    })
  ],
})
export class CoreStoreModule {
  ...
}
```

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

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

// 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 '../core/@ngrx';

// 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:

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

// rxjs
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 08. Dispatching an Event To The Store

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

```
export const doneTask = createAction(
  '[Tasks] DONE_TASK',
  props<{ task: Task }>()
);
```

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

```
on(TasksActions.doneTask, (state, props) => {
  console.log('DONE_TASK action being handled!');

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

    return props.task;
  });

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

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

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

// 2
onCompleteTask(task: TaskModel): void {
  this.updateTask(task).catch(err => console.log(err));
  // task is not plain object
  // taskToComplete is a plain object
  const taskToComplete: Task = { ...task };
  this.store.dispatch(TasksActions.doneTask({ task: taskToComplete }));}

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

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

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

Task 09. Create Meta-Reducer

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

```
import { ActionReducer, MetaReducer } from '@ngrx/store';

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

    return reducer(state, action);
  };
}

export const metaReducers: MetaReducer<any>[] = [debug];
```

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

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

// 2
StoreModule.forRoot(
  {},
  {
    metaReducers,
    runtimeChecks: {
      strictStateImmutability: true,
      strictActionImmutability: true,
      strictStateSerializability: true,
      strictActionSerializability: true
    }
  }
)
```

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

Task 10. Install Redux DevTools Extension

1. If you don't have extension for Chrome **Redux DevTools Extension** installed on your machine, install it. Manual is here <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 11. Create Effects Class

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

```
ng g ef core/@ngrx/tasks/tasks -m tasks/tasks.module.ts --spec false -c true
```

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/@ngrx/tasks/index.ts**. Use the following snippet of code:

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

Task 12. 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/@ngrx';
import { TasksEffects } from '../core/@ngrx/tasks/tasks.effects';
```

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

[TASKS EFFECTS]

Task 13. Get Tasks from DataBase

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

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

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

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

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

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

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

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

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



```

        loading: false,
        loaded: true
    });
  })),
  on(TasksActions.getTasksError, (state, props) => {
    console.log('GET_TASKS_ERROR action being handled!');
    const error = props.error;
    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, createEffect, 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
getTasks$: Observable<Action> = createEffect(() =>
  this.actions$.pipe(
    ofType(TasksActions.getTasks),
    switchMap(action =>
      this.taskPromiseService
        .getTasks()
        .then(tasks => TasksActions.getTasksSuccess({ tasks }))
        .catch(error => TasksActions.getTasksError({ error }))
    )
  )
);

```

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

```

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

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

```

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

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

  this.store.dispatch(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 14. Get Task from DataBase

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

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

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

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

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

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

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

```
// 1
// @NgRx
import { Store, select } from '@ngrx/store';
import { AppState, TasksState } from '../core/@ngrx';
import * as TasksActions from '../core/@ngrx/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: ParamMap) => {
      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} as TaskModel));

this.route.paramMap.subscribe((params: ParamMap) => {
  const id = params.get('taskID');
  if (id) {
    this.store.dispatch(TasksActions.getTask({ taskID: +id }));
  }
});

```

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

```

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

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

```

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

```

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

```

```

    });
  })),

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

  on(TasksActions.getTaskError, (state, props) => {
    console.log('GET_TASK_ERROR action being handled!');
    const error = props.error;
    return {
      ...state,
      loading: false,
      loaded: false,
      error
    };
  })),

```

Task 15. Update Task in DataBase

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

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

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

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

```
// 1
import { TaskPromiseService } from '../services';
import { TaskModel, Task } from '../models/task.model';

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

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

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

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

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

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

```

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

```

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

```

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

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

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

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

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

```

Task 16. Add Task to DataBase

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

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

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

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} as TaskModel);

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

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

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


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

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

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

on(TasksActions.createTaskError, (state, props) => {
  console.log('CREATE_TASK_ERROR action being handled!');
  const error = props.error;
  return {
    ...state,
    error
  };
}),

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

Task 17. Delete Task from DataBase

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

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

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

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

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

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

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

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

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

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

```
on(TasksActions.deleteTaskError, (state, props) => {  
  console.log('DELETE_TASK_ERROR action being handled!');  
  const error = props.error;  
  return {  
    ...state,  
    error  
  };  
}),
```

Task 18. Replace DoneTask w/ UpdateTask Action

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

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

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

```
export const doneTask = createAction('[Tasks] DONE_TASK', props<{ task: Task }>());
```

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

```
on(TasksActions.doneTask, (state, props) => {  
  console.log('DONE_TASK action being handled!');  
  
  const id = props.task.id;  
  const data = state.data.map(t => {  
    if (t.id === id) {  
      return { ...props.task, done: true };  
    }  
  
    return props.task;  
  });  
  
  return {  
    ...state,  
    data  
  };  
})  
);
```

Task 19. Effects Interfaces

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

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

// 2
export class TasksEffects implements OnInitEffects, OnRunEffects {

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

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

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

Task 20. Feature Selector

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

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

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

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

2. Make changes to file **app/core/@ngrx/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, selectTasksState } from '../../../core/@ngrx';

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

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

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

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

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

Task 21. State Selector

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

  this.sub = this.store.pipe(select(selectSelectedTask))
    .subscribe(task => {
      if (task) {
        this.task = {...task} as TaskModel;
      } else {
        this.task = new TaskModel();
      }
    });
  ...
}

```


Task 22. Router State

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

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

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

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

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

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

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

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

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

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

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

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

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

    const { params, fragment } = route;

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

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

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

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

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

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

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

// 2
imports: [
  StoreModule.forRoot({routerReducers, {
    metaReducers,
    runtimeChecks: {
      strictStateImmutability: true,
      strictActionImmutability: true,
      // router state is not serializable
      // set false if you don't use CustomSerializer
      strictStateSerializability: true,
      // router action is not serializable
      // set false
      strictActionSerializability: truefalse
    }
  })),
  StoreRouterConnectingModule.forRoot({
    serializer: CustomSerializer
  }),
  ...
],
```

7. Run application. Inspect Router State in NgRx Dev Tool. Comment `serializer: CustomSerializer` and inspect Router State again. Uncomment `serializer: CustomSerializer`.

Task 23. Compose Router and Task Selectors

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

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

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

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

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

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

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

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

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

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

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

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

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

```

    this.route.paramMap.subscribe((params: ParamMap) => {
      const id = params.get('taskId');
      if (id) {
        this.store.dispatch(TasksActions.getTask({ taskId: +id}));
      }
    });

    this.sub = this.store
      .pipe(select(selectSelectedTaskByUrl))
      .subscribe(task => this.task = {...task});

    ...
  }

```

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

```

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

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

```

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

```

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

```

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

```

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

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

```

```

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

    on(TasksActions.getTaskError, (state, props) => {
      console.log('GET_TASK_ERROR action being handled!');
      const error = props.error;
      return {
        ...state,
        loading: false,
        loaded: false,
        error
      };
    })),
  ));

```

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

```

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

```

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

```

export const getTask = createAction(
  '[Tasks] GET_TASK',
  props<{ taskID: number }>()
);

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

export const getTaskError = createAction(
  '[Tasks] GET_TASK_ERROR',
  props<{ error: Error | string }>()
);

```

Task 24. Router Built-in Selectors

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

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

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

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

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

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

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

// 4
ngOnInit() {
  ...

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

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

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

```
}),  
  
// 2  
strictStateSerializability: true false,
```

Look to the console.

Comment Router Selectors Demo, uncomment serializer and change strictStateSerializability value to true.

Task 25. Users Store

1. Make changes to the file **user.model.ts**. Use the following snippet of code:

```
// 1
export interface User {
  id: number;
  firstName: string;
  lastName: string;
}

// 2
export class UserModel implements User {...}
```

2. Create file **app/core/@ngrx/users/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
};
```

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

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

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

```
ng g a core/@ngrx/users/users -c true
```

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

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

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

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



```

export const createUser = createAction(
  '[Users] CREATE_USER',
  props<{ user: User }>()
);

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

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

export const updateUser = createAction(
  '[Users] UPDATE_USER',
  props<{ user: User }>()
);

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

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

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

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

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

export const setOriginalUser = createAction(
  '[Users] SET_ORIGINAL_USER',
  props<{ user: User }>()
);

```

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

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

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

ng g r core/@ngrx/users/users --spec false -c true

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

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

import { UsersState, initialUsersState } from './users.state';
import * as UsersActions from './users.actions';
import { User } from '../../../users/models/user.model';

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

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

  on(UsersActions.getUsersError, (state, props) => {
    const error = props.error;

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

  on(UsersActions.createUserSuccess, (state, props) => {
    const createdUser = { ...props.user };
    const entities = {
      ...state.entities,
```

```

    [createdUser.id]: createdUser
  };
  const originalUser = { ...createdUser };

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

on(UsersActions.createUserError, (state, props) => {
  const error = props.error;

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

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

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

on(UsersActions.updateUserError, (state, props) => {
  const error = props.error;

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

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

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

on(UsersActions.deleteUserError, (state, props) => {
  const error = props.error;

  return {

```

```

        ...state,
        error
    });
    }),

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

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

export function usersReducer(state: UsersState | undefined, action: Action) {
    return reducer(state, action);
}

```

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

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

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

```

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

```

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

```
ng g ef core/@ngrx/users/users -m users/users.module.ts --spec false -c true
```

10. 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, createEffect, 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';

```

```

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

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

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

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

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

```

```

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

```

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

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

12. Create file **app/core/@ngrx/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 { selectRouterState } from '../../../router/router.selectors';

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

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

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

/**
 * transform object to array
 */

```

```

export const selectUsers = createSelector(
  selectUsersEntitites,
  entities => {
    return Object.keys(entities).map(id => entities[+id]);
  }
);

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

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

```

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

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

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

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

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

```

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

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

```

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

```
// 1
import { ActivatedRoute, ParamMap, Router } from '@angular/router';
import { UserObservableService } from '../services';
import { switchMap } from 'rxjs/operators';
import { Store, select } from '@ngrx/store';
import * as UsersActions from '../core/@ngrx/users/users.actions';
import { AppState, selectUsers, selectUsersError, selectEditedUser } from
  '../core/@ngrx';
import { Observable, Subscription, of } from 'rxjs';
import { AutoUnsubscribe } from '../core/decorators';
import { UserModel, User } from '../models/user.model';

// 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: ParamMap) => {
        return params.get('editedUserID')
          ? this.userObservableService.getUser(+params.get('editedUserID'))
          : of(null);
      })
    )
    .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(selectUsers));
  this.usersError$ = this.store.pipe(select(selectUsersError));
  this.store.dispatch(UsersActions.getUsers());
}
```



```

    // listen editedUserID from UserFormComponent
    this.subscription = this.store.pipe(select(selectEditedUser)).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);
  const userToDelete: User = { ...user };
  this.store.dispatch(UsersActions.deleteUser({ user: userToDelete }));
}

```

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

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

18. 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, UrlTree } from '@angular/router';
// @Ngrx
import { Store, select } from '@ngrx/store';
import { AppState, selectUsersOriginalUser } from './../../core/@ngrx';
import * as UsersActions from './../../core/@ngrx/users/users.actions';

// 2
@AutoUnsubscribe()

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

// 4
constructor(
  ...
  private store: Store<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(
        (savedUser) => {
          this.originalUser = {...savedUser};
          user.id
            // optional parameter: http://localhost:4200/users?id=2
            ? this.router.navigate(['users', { editedUserID: user.id }])
            : this.onGoBack();
        },
        error => console.log(error)
      );
    const user = { ...this.user } as User;
    if (user.id) {
      this.store.dispatch(UsersActions.updateUser({ user }));
    } else {
      this.store.dispatch(UsersActions.createUser({ user }));
    }
  }

  // 7
  canDeactivate(): Observable<Boolean | UrlTree> | Promise<Boolean | UrlTree> | boolean |
  UrlTree {
    const flags = Object.keys(this.originalUser).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(selectUsersOriginalUser),
      switchMap(originalUser => {
        for (const key in originalUser) {
          if (originalUser[key] === this.user[key]) {
            flags.push(true);
          } else {
            flags.push(false);
          }
        }
      })
    );
  }

```

```

        if (flags.every(e1 => e1)) {
            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?');
    })
    );
}

```

19. 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, selectSelectedUserByUrl } from '../../../core/@ngrx';
import * as UsersActions from '../../../core/@ngrx/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: UserModel) => {
            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(selectSelectedUserByUrl),
    tap(user => this.store.dispatch(UsersActions.setOriginalUser({ user }))),
    delay(2000),
    map((user: UserModel) => {
      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 26. Navigation By Actions

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

ng g a core/@ngrx/router/router -c true

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

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

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

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

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

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

ng g ef core/@ngrx/router/router --root true -m core/@ngrx/core-store.module.ts --spec false -c true

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

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

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

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

```

        })
      ),
      { dispatch: false }
    );

    navigateBack$ = createEffect(
      () =>
        this.actions$.pipe(
          ofType(RouterActions.back),
          tap(() => this.location.back())
        ),
      { dispatch: false }
    );

    navigateForward$ = createEffect(
      () =>
        this.actions$.pipe(
          ofType(RouterActions.forward),
          tap(() => this.location.forward())
        ),
      { dispatch: false }
    );
  }
}

```

3. Make changes to file **app/core/@ngrx/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 { CustomSerializer, routerReducers, RouterEffects } from './router';

```

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

```

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

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

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

```

```

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

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

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

```

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

```

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

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

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

```

```

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

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

// 4
createUpdateUserSuccess$: Observable<Action> = createEffect(() =>
    this.actions$.pipe(
        ofType(UsersActions.createUser, UsersActions.updateUser),
        map(action => {
            const userID = action.user.id;
            const actionType = action.type;
            let path: any[];

            if (actionType === '[Users] UPDATE_USER') {
                path = ['/users', { editedUserID: userID }];
            } else {
                path = ['/users'];
            }

            return RouterActions.go({ path });
        })
    )
);

```

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

```

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

```



```

import * as RouterActions from '../@ngrx/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(RouterActions.go({
    path: ['/login'],
    extras: navigationExtras
  }));
}

```

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

```

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

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

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

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

```

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

```

// 1

```

```

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

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

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

```

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

```

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

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

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

```

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

```

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

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

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

```

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

```

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

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

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

```

8. 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/@ngrx';
import * as RouterActions from '../core/@ngrx/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(RouterActions.go({
      path: [{ outlets: { messages: null } }]
    }));
    this.messagesService.isDisplayed = false;
  }

```

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

```

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

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

```

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

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

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

```

Task 27. State Preloading

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

ng g g tasks/guards/tasks-state-preloading --skipTests true --implements CanActivate

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

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

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

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

    // make a side effect
    tap((loaded: boolean) => {
      if (!loaded) {
        store.dispatch(TasksActions.getTasks());
      }
    }),

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

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

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

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

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

// 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 --skipTests true --implements CanActivate

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, selectTasksData } from '../core/@ngrx';
import * as RouterActions from '../core/@ngrx/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 TaskExistsGuard 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(selectTasksData),

      // 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(RouterActions.go({ path: ['/home'] }));
    }
  })),

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

```

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

```

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

```

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

```

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

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

```

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

```

ngOnInit() {
  ...
  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 --skipTests true --implements CanActivate

```

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

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

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

```

```

@Injectables({
  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(selectUsersLoaded),
      tap(loaded => {
        if (!loaded) {
          this.store.dispatch(UsersActions.getUsers());
        }
      }),
      take(1)
    );
  }
}

```

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

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

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

```

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

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

```

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

```

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

```



```
}
```

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

```
// 1
import { AppState, selectUsersOriginalUser, selectSelectedUserByUrl } from
'../../../../../core/@ngrx';
import { Observable, of, Subscription } from 'rxjs';
import { ActivatedRoute, UrlTree } from '@angular/router';
import { AutoUnsubscribe, DialogService, CanComponentDeactivate } from
'../../../../../core';
import { TaskModel, Task } from '../../../../../models/task.model';

// 2
@AutoUnsubscribe()

// 3
private sub: Subscription;

// 4
constructor(
  private route: ActivatedRoute,
  ...
) { }

// 5
ngOnInit(): void {
  this.route.data.pipe(pluck('user')).subscribe((user: UserModel) => {
    this.user = { ...user };
  });
  this.sub = this.store.pipe(select(selectSelectedUserByUrl))
    .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 28. @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>;
  readonly loading: boolean;
  readonly loaded: boolean;
  readonly error: Error | string;
}

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

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

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

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

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

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

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

```

return adapter.addAll(tasks, {...state, loading: false, loaded: true,
selectedTask: null });
}),

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

  return {
    ...state,
    data
  };
  return adapter.addOne(task, state);
}),

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

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

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

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

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

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

```

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

```

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

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

```

```
// 3
export const selectSelectedTaskByUrl = createSelector(
  selectTasksData,
  selectTasksEntities
  getRouterState,
  (tasks, router): Task => {
    const taskID = router.state.params.taskID;
    if (taskID && Array.isArray(tasks)) {
      return tasks.find(task => task.id === +taskID);
      return tasks[taskID] as TaskModel;
    } else {
      return new TaskModel();
    }
  });
```

Task 29. @ngrx/data

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

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import {
  EntityMetadataMap,
  EntityDataModule,
  DefaultDataServiceConfig
} from '@ngrx/data';
import { createFeatureSelector, createSelector } from '@ngrx/store';
import { selectRouterState } from '../router/router.selectors';
import { UserModel, User } from 'src/app/users/models/user.model';

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

// rule for json-server
const pluralNames = {
  User: 'User'
};

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

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

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

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

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

```

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

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

```

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

```

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

// 2
imports: [
  ...
  EntityStoreModule
]

```

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

```

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

// 2
private userService: EntityCollectionService<User>;

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

// 4
ngOnInit() {

```

```

    this.users$ = this.store.pipe(select(selectUsers));
    this.usersError$ = this.store.pipe(select(selectUsersError));

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

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

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

```

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

```

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

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

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

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

```

```

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

    this.onGoBack();
  }

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

  return this.store.pipe(
    select(selectUsersOriginalUser),
    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?');
    })
  );
  if (this.isSubmitClick) {
    return true;
  }

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

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

```

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

```

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

```



```

import { AppState, selectUsersLoaded } from '../core/@ngrx';
import * as UsersActions from '../core/@ngrx/users/users.actions';
import { EntityServices, EntityCollectionService } from '@ngrx/data';
import { User } from '../models/user.model';

// 2
private userService: EntityCollectionService<User>;

// 3
constructor(
  private store: Store<AppState>
  entitytServices: EntityServices
) {
  // получить сервис для entity User
  this.userService = entitytServices.getEntityCollectionService('User');
}

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

```

Task 30. Use Facade

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

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

// @ngrx
import { Store, select } from '@ngrx/store';
import { AppState } from '../app.state';
import {
  selectTasksData,
  selectTasksError,
  selectSelectedTaskByUrl
} from './tasks.selectors';
import * as TasksActions from '../../core/@ngrx/tasks/tasks.actions';
import * as RouterActions from '../../core/@ngrx/router/router.actions';

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

import { Task, TaskModel } from 'src/app/tasks/models/task.model';
import { NavigationExtras } from '@angular/router';

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

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

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

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

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

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

```
}
```

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

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

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

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

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

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

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

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

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

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

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

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

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

// 3
ngOnInit(): void {
  this.sub = this.store.pipe(select(selectSelectedTaskByUrl))
    this.tasksFacade.selectedTaskByUrl$.subscribe(
      task => (this.task = { ...task })
    );
}

// 4
onSaveTask() {
  const task = { ...this.task } as Task;

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

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

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