Recently the official NgRX docs (https://ngrx.io/) have completely changed. Now they don't mention the traditional syntax used by Max any longer. They have completely switched to NgRX's new `createAction()`/`createReducer()`/`createEffect()` syntax, even though the "old" syntax is still fully working and not deprecated (you will find the "old" syntax here: https://v7.ngrx.io/docs).

This change came very surprisingly, and with a delay after the Angular 8 release which has been the basis for Max' course updates.

With the new syntax we can avoid some of the traditional boilerplate code, and the actions can be easier tracked in the whole application (with the traditional approach we have to use the actions' string types in the reducers and the effects' `ofType()`, but the TS types of the actions themselves in the other parts of an app).

In spite of this change the structure of the code remains exactly the same (since it's the Redux pattern in the end), but it might be very confusing at first sight when you want to compare the course code with what you will find in the new docs.

Here is a description how you can transform the final code of this section into the new syntax, exactly preserving the app's functionality. More features (like `createSelector()`, `createFeatureSelector()`, `createEntityAdapter()` etc.) can be found in the NgRX docs, but these are beyond the scope of this course which is about Angular, not NgRX in depth.

____

If you want to switch to the new syntax ...

● **... replace the action files by these ones:**

auth.actions.ts

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

export const loginStart = createAction(
  '[Auth] Login Start',
  props<{
    email: string;
    password: string
  }>()
);


export const signupStart = createAction(
  '[Auth] Signup Start',
  props<{
    email: string;
    password: string
  }>()
);
```

```typescript


export const authenticateSuccess = createAction(
  '[Auth] Authenticate Success',
  props<{
    email: string;
    userId: string;
    token: string;
    expirationDate: Date;
    redirect: boolean
  }>()
);


export const authenticateFail = createAction(
  '[Auth] Authenticate Fail',
  props<{
    errorMessage: string
  }>()
);


export const clearError = createAction(
  '[Auth] Clear Error'
);


export const autoLogin = createAction(
  '[Auth] Auto Login'
);


export const logout = createAction(
  '[Auth] Logout'
);
```

recipe.actions.ts

```typescript
import { createAction, props } from '@ngrx/store';
import { Recipe } from '../recipe.model';


```

```
export const addRecipe = createAction(
  '[Recipe] Add Recipe',
  props<{
    recipe: Recipe
  }>()
);


export const updateRecipe = createAction(
  '[Recipe] Update Recipe',
  props<{
    index: number,
    recipe: Recipe
  }>()
);


export const deleteRecipe = createAction(
  '[Recipe] Delete Recipe',
  props<{
    index: number
  }>()
);


export const setRecipes = createAction(
  '[Recipe] Set Recipes',
  props<{
    recipes: Recipe[]
  }>()
);


export const fetchRecipes = createAction(
  '[Recipe] Fetch Recipes'
);


export const storeRecipes = createAction(
  '[Recipe] Store Recipes'
);
```

shopping-list.actions.ts

```
import { createAction, props } from '@ngrx/store';
import { Ingredient } from '../../shared/ingredient.model';


export const addIngredient = createAction(
  '[Shopping List] Add Ingredient',
  props<{
    ingredient: Ingredient
  }>()
);


export const addIngredients = createAction(
  '[Shopping List] Add Ingredients',
  props<{
    ingredients: Ingredient[]
  }>()
);


export const updateIngredient = createAction(
  '[Shopping List] Update Ingredient',
  props<{
    ingredient: Ingredient
  }>()
);


export const deleteIngredient = createAction(
  '[Shopping List] Delete Ingredient'
);


export const startEdit = createAction(
  '[Shopping List] Start Edit',
  props<{
    index: number
  }>()
);


export const stopEdit = createAction(
  '[Shopping List] Stop Edit'
);
```

- **… replace the reducer files by these ones:**

auth.reducer.ts

```
import { Action, createReducer, on } from '@ngrx/store';
import { User } from '../user.model';
import * as AuthActions from './auth.actions';


export interface State {
  user: User;
  authError: string;
  loading: boolean;
}


const initialState: State = {
  user: null,
  authError: null,
  loading: false
};


const _authReducer = createReducer(

  initialState,

  on(
    AuthActions.loginStart,
    AuthActions.signupStart,
    (state) => ({
      ...state,
      authError: null,
      loading: true
    })
  ),

  on(
    AuthActions.authenticateSuccess,
    (state, action) => ({
      ...state,
      authError: null,
      loading: false,
      user: new User(
        action.email,
        action.userId,
        action.token,
        action.expirationDate
      )
    })
  ),

  on(
    AuthActions.authenticateFail,
    (state, action) => ({
      ...state,
      user: null,
      authError: action.errorMessage,
      loading: false
    })
  ),

  on(
    AuthActions.logout,
    (state) => ({
      ...state,
      user: null
    })
  ),

  on(
    AuthActions.clearError,
    (state) => ({
      ...state,
      authError: null
    })
  ),

);


export function authReducer(state: State, action: Action) {
  return _authReducer(state, action);
```

```
}
```

recipe.reducer.ts

```
import { Action, createReducer, on } from '@ngrx/store';
import { Recipe } from '../recipe.model';
import * as RecipesActions from '../store/recipe.actions';


export interface State {
  recipes: Recipe[];
}


const initialState: State = {
  recipes: []
};


const _recipeReducer = createReducer(

  initialState,

  on(
    RecipesActions.addRecipe,
    (state, action) => ({
      ...state,
      recipes: state.recipes.concat({ ...action.recipe })
    })
  ),

  on(
    RecipesActions.updateRecipe,
    (state, action) => ({
      ...state,
      recipes: state.recipes.map(
        (recipe, index) => index === action.index ? { ...action.recipe } : recipe
      )
    })
  ),

  on(
    RecipesActions.deleteRecipe,
    (state, action) => ({
      ...state,
      recipes: state.recipes.filter(
        (_, index) => index !== action.index
      )
    })
  ),

  on(
    RecipesActions.setRecipes,
    (state, action) => ({
      ...state,
      recipes: [ ...action.recipes ]
    })
  )

);


export function recipeReducer(state: State, action: Action) {
  return _recipeReducer(state, action);
}
```

shopping-list.reducer.ts

```
import { Action, createReducer, on } from '@ngrx/store';
import { Ingredient } from '../../shared/ingredient.model';
import * as ShoppingListActions from './shopping-list.actions';


export interface State {
  ingredients: Ingredient[];
  editIndex: number;
}


const initialState: State = {
  ingredients: [
    new Ingredient('Apples', 5),
    new Ingredient('Tomatoes', 10)
  ],
  editIndex: -1
};


const _shoppingListReducer = createReducer(

  initialState,

  on(
    ShoppingListActions.addIngredient,
    (state, action) => ({
      ...state,
      ingredients: state.ingredients.concat(action.ingredient)
    })
  ),

  on(
    ShoppingListActions.addIngredients,
    (state, action) => ({
      ...state,
      ingredients: state.ingredients.concat(...action.ingredients)
    })
  ),

  on(
    ShoppingListActions.updateIngredient,
    (state, action) => ({
      ...state,
      editIndex: -1,
      ingredients: state.ingredients.map(
        (ingredient, index) => index === state.editIndex ? { ...action.ingredient } : ingredient
      )
    })
  ),

  on(
    ShoppingListActions.deleteIngredient,
    (state) => ({
      ...state,
      editIndex: -1,
      ingredients: state.ingredients.filter(
        (_, index) => index !== state.editIndex
      )
    })
  ),

  on(
    ShoppingListActions.startEdit,
    (state, action) => ({
      ...state, editIndex:
      action.index
    })
  ),

  on(
    ShoppingListActions.stopEdit,
    (state) => ({
      ...state, editIndex: -1
    })
  )

);


export function shoppingListReducer(state: State, action: Action) {
  return _shoppingListReducer(state, action);
}
```

- **... change these places in the effect files:**

auth.effects.ts

```
  authSignup$ = createEffect(() =>
    this.actions$.pipe(
      ofType(AuthActions.signupStart),
      switchMap(action => {
              ...
              email: action.email,
              password: action.password,
              ...
  );

  authLogin$ = createEffect(() =>
    this.actions$.pipe(
      ofType(AuthActions.loginStart),
      switchMap(action => {
              ...
              email: action.email,
              password: action.password,
              ...
  );

  authRedirect$ = createEffect(() =>
    this.actions$.pipe(
      ofType(AuthActions.authenticateSuccess),
      tap(action =>  action.redirect && this.router.navigate(['/']))
    ), { dispatch: false }
  );

  autoLogin$ = createEffect(() =>
    this.actions$.pipe(
    ofType(AuthActions.autoLogin),
    ...
  );

  authLogout$ = createEffect(() =>
    this.actions$.pipe(
      ofType(AuthActions.logout),
      ...
    { dispatch: false }
  );
```

recipe.effects.ts

```
fetchRecipes$ = createEffect(() =>
  this.actions$.pipe(
    ofType(RecipesActions.fetchRecipes),
          ...
);

storeRecipes$ = createEffect(() =>
  this.actions$.pipe(
    ofType(RecipesActions.storeRecipes),
          ...
  { dispatch: false }
);
```

● **... make a global search (Ctrl + Shift + F) for `dispatch` and change the related action syntax in those places:**

1. remove the `new` keyword

2. change the action name to lowerCamelCase

3. change the passed parameter to an object

E.g.:

`this.store.dispatch(new ShoppingListActions.StartEdit(index));`

... becomes ...

`this.store.dispatch(ShoppingListActions.startEdit({index}));`

... and ...

`this.store.dispatch(new RecipesActions.AddRecipe(this.recipeForm.value));`

... becomes ...

`this.store.dispatch(RecipesActions.addRecipe({recipe: this.recipeForm.value}));`

Important: Don't forget to apply all changes to the remaining actions in the two effect files as well, even though you won't find them via a search for `dispatch` there!

● **... and please note:**

1.Inside the reducers I replaced Max' transformation logic with a more condensed syntax, using `concat` / `map` / `filter`. This change is not related to the new NgRX syntax. You can also use Max' transformations with the new NgRX syntax, or my transformations with the old one.

2. In my code I removed the `editedItem` property from the shopping list state, since it's kind of redundant. It would only be used in one place of the app (`shopping-edit.component.ts`), and there it can be easily accessed via its index:

```
...
.subscribe(stateData => {
  const index = stateData.editIndex;
  if (index > -1) {
    this.editedItem = stateData.ingredients[index];
    ...
```

3. In the `Resolver` class don't forget to change the generic type of the `Resolve` interface from `Resolve<Recipe[]>` to `Resolve<{recipes: Recipe[]}>`, and the return value in the else case from `recipes` to `{recipes}`.