academeez

# @ngrx/data

## Automatic entity management

# @ngrx/data

academeez

- Many of our entities will be grabbed from a REST server
- Those entities might also have additional CRUD actions we want to perform on them while keeping our REST server in sync
- Managing those entities in the state and including sync with our REST server will require us to manage @ngrx/entity, @ngrx/effects, HttpClient requests, actions and reducers
- This will sum up to a lot of code we will have to write
- Enters @ngrx/data which can do all the above, maintain a cache of our entities and manage our state with minimal amount of code

# Without @ngrx/data

Service with HttpClient

```
@Injectable({providedIn: 'root'})
export class TodoService {
  constructor(private _http: HttpClient) {}
  ...
}
```

Reducer and adapter

```
const todoAdapter = createEntityAdapter();

const todoReducer = createReducer(
  initialState,
  on(...),
  on(...)
)
```

Effect to handle send request and deal with response

```
@Injectable({ providedIn: 'root' })
export class TodoEffects {
  todo$ = createEffect(() =>
this._actions$.pipe(...))

  constructor(private _actions$: Actions) {
}
```

Actions

```
export const getTodo = createAction(...);
export const setTodo = createAction(...);
...
```

# With @ngrx/data

- No need to create HttpClient services, effects, reducers, actions, they are created implicitly:

```
export const entityDataConfig :
EntityDataModuleConfig = {
  entityMetadata: {
    Task: {}
  }
}
```

```
imports: [
    HttpClientModule,
    EffectsModule.forRoot(),
    StoreModule.forRoot({}),
    StoreDevtoolsModule.instrument(),
    EntityDataModule.forRoot(entityDataConfig)
],
```

# EntityCollectionServiceFactory

- With a very minimal setup @ngrx/data can create a service for your server entities
- You can easily perform CRUD operation that will by synced with the server and saved in @ngrx/store state
- In you component you can create that service using **EntityCollectionServiceFactory**

```
constructor(
    serviceFactory : EntityCollectionServiceFactory
  ) {
    this._taskService = serviceFactory.create('Task');
  }
```

# EntityCollectionService

academeez

- Using EntityCollectionService we can now query the server, sync a cache in the store

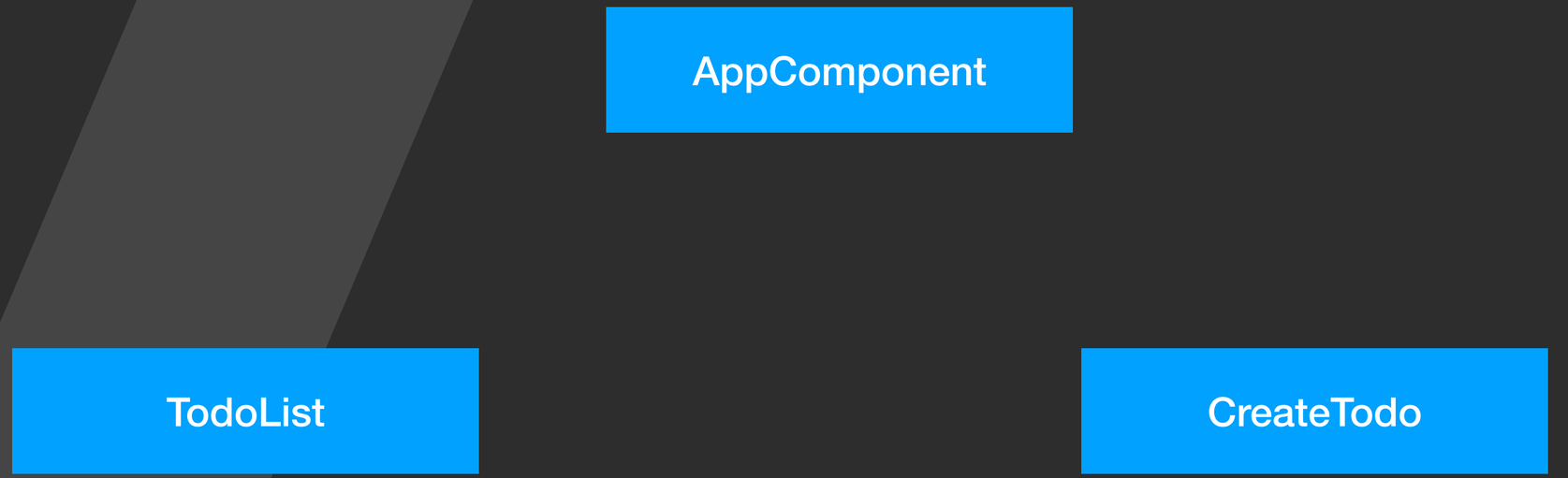
```
export interface EntityCollectionService ...{
  // inherited from EntityCommands
  add(entity: T, options ?: EntityActionOptions): Observable<T>;
  delete (key: number | string, options ?: EntityActionOptions): Observable<number | string>;
  getAll(options ?: EntityActionOptions): Observable<T[]>;
  update(entity: Partial < T >, options ?: EntityActionOptions): Observable<T>;

  // inherited from EntitySelectors
  readonly entities$: Observable<T[]> | Store<T[]>;
  readonly loading$: Observable<boolean> | Store<boolean>;

  ...
}
```

# @ngrx/data - ex

academeez

- In this ex we will learn how the power of @ngrx/data saves us a lot of code we need to write.
- We will easily create a CRUD on a todo REST server with an api
  - GET: https://nztodo.herokuapp.com/api/task
  - POST: https://nztodo.herokuapp.com/api/task
  - DELETE: https://nztodo.herokuapp.com/api/task/<id>
  - PUT: DELETE: https://nztodo.herokuapp.com/api/task/<id>

# @ngrx/data - ex - components



# TodoList

- Display a list of todo items taken from the server by using @ngrx/data

# CreateTodo

- Contains a form for creating a new todo item
- Use @ngrx/data to update the server
- Notice how the list is updated automatically

# Summary

- @ngrx/data is a shortcut library for managing entities in our state that are in sync with a server REST api
- @ngrx/data will save us creating a lot of repeating code like HttpClient services, reducers, actions, effects.
- @ngrx/data will provide us an easy to use **EntityCollectionService** which will provide us methods to query the server and read the data from a managed cache in the store

academeez

# Thank You