Angular Uygulamalarda NGRX Mağazası Nasıl Uygulanır

Örnek bir projeyle eksiksiz bir kılavuz

Photo by Meriç Dağlı on Unsplash

NGRX Store, Angular Apps için bir durum yönetimi kitaplığıdır. Sayfa istekleri arasında bir oturum sürdürdüğümüz tek sayfalı uygulamalardan önce sunucuda oturum yönetimi yapıyorduk. Tek sayfalı uygulamalar söz konusu olduğunda, durumu korumak biraz zordur. NGRX mağazası ile daha kolay hale geliyor.

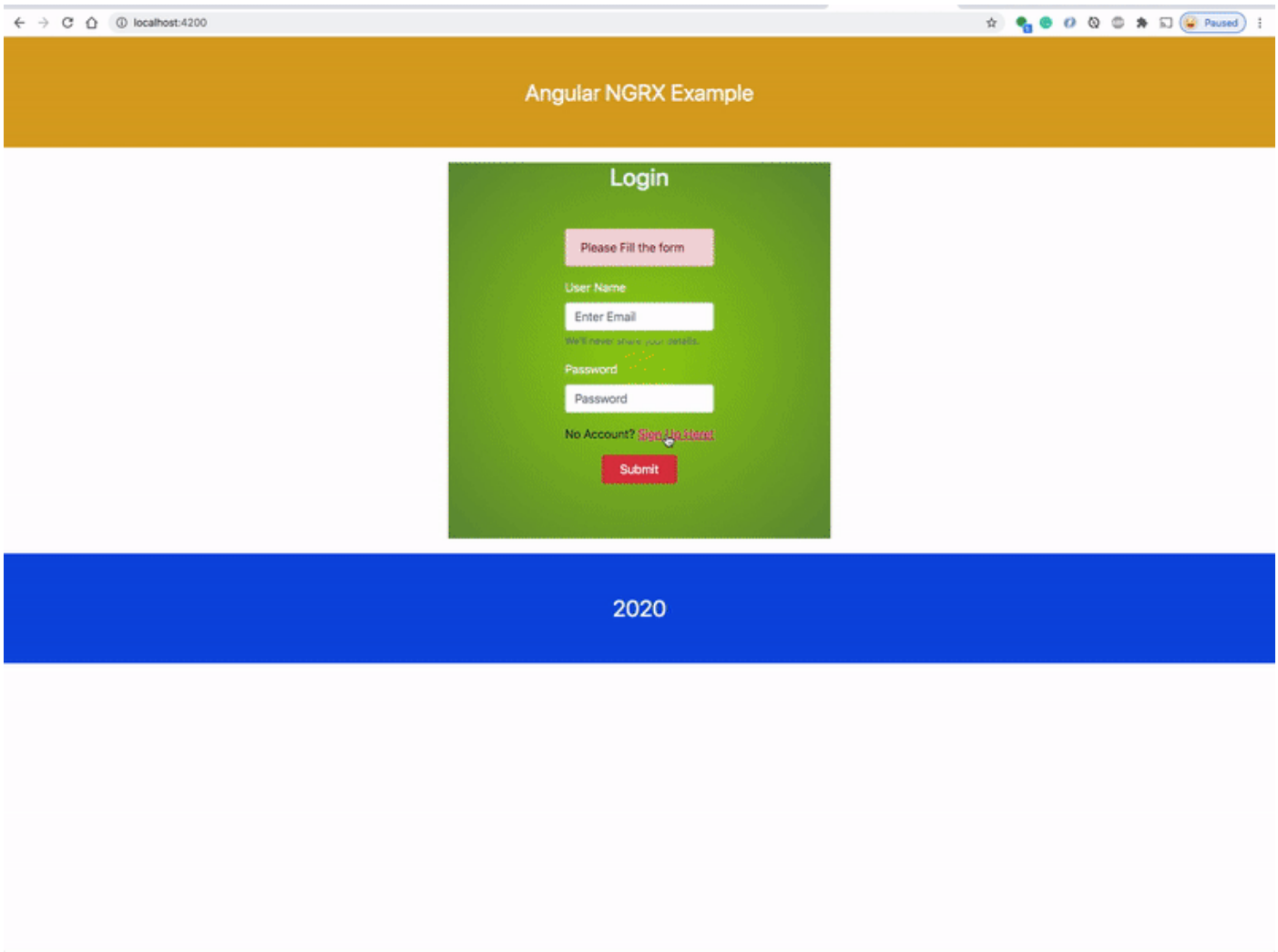
Hemen başlayalım ve NGRX Store'u kullanarak Angular ile durum yönetimini nasıl kurabileceğimize bakalım.

İşte bu makalede ele alacağımız şeyler

- ***Örnek Proje***

- ***Nasıl çalışır***
- ***NGRX Temelleri***
- ***Önkoşullar***
- ***Kurulum***
- ***API Uygulaması***
- ***NGRX Uygulaması***
- ***Açısal Uygulama***
- ***Özet***
- ***Sonuç***

Örnek projeye bir göz atalım. Oturum açabileceğimiz, kaydolabileceğimiz ve görevler ekleyebileceğimiz, görevleri silebileceğimiz ve görevleri düzenleyebileceğimiz vb. Basit bir uygulamamız var. Tüm uygulama durumu NGRX ile korunur. Eylemlerimiz, etkilerimiz, indirgeyicilerimiz vb. Var. Hepsini bu örnekte göreceğiz.



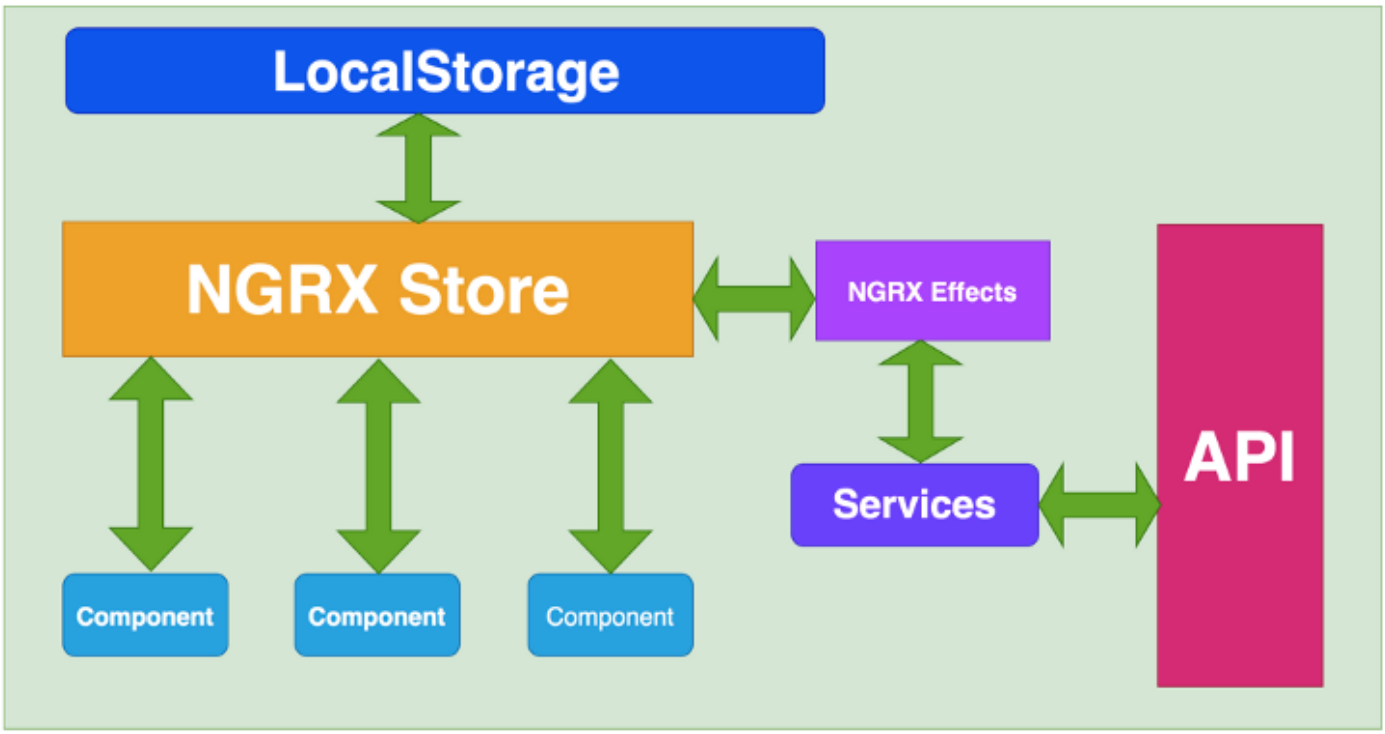
Örnek Proje

İşte GitHub'da klonlayıp yerel makinenizde çalıştırabileceğiniz örnek proje.

```
// clone the project
git clone https://github.com/bbachi/angular-ngrx-example.git
// Run the API
cd api
npm install
npm run dev
// Run the Angular App
cd ui
npm install
npm start
```

NGRX, Açısal Uygulamalar için yeniden düzenlemeden esinlenen bir durum yönetim aracıdır. Uygulamanız büyüdükçe ve büyüdüğünde iletişimin üstesinden gelmek zorlaşır. NGRX, tek yönlü veri akışı ve tüm uygulama için tek bir doğruluk kaynağı sağlar.

Mağazanın farkında olan bileşenlere akıllı bileşenler, mağazadan haberdar olmayan bileşenlere ise aptal bileşenler denir. Aşağıdaki şemaya bakarsanız, tüm akıllı bileşenler verileri mağazaya gönderir ve tüm uygulama için tek yönlü veri akışını teşvik ederek mağazadan veri alır.



NGRX Mağazası

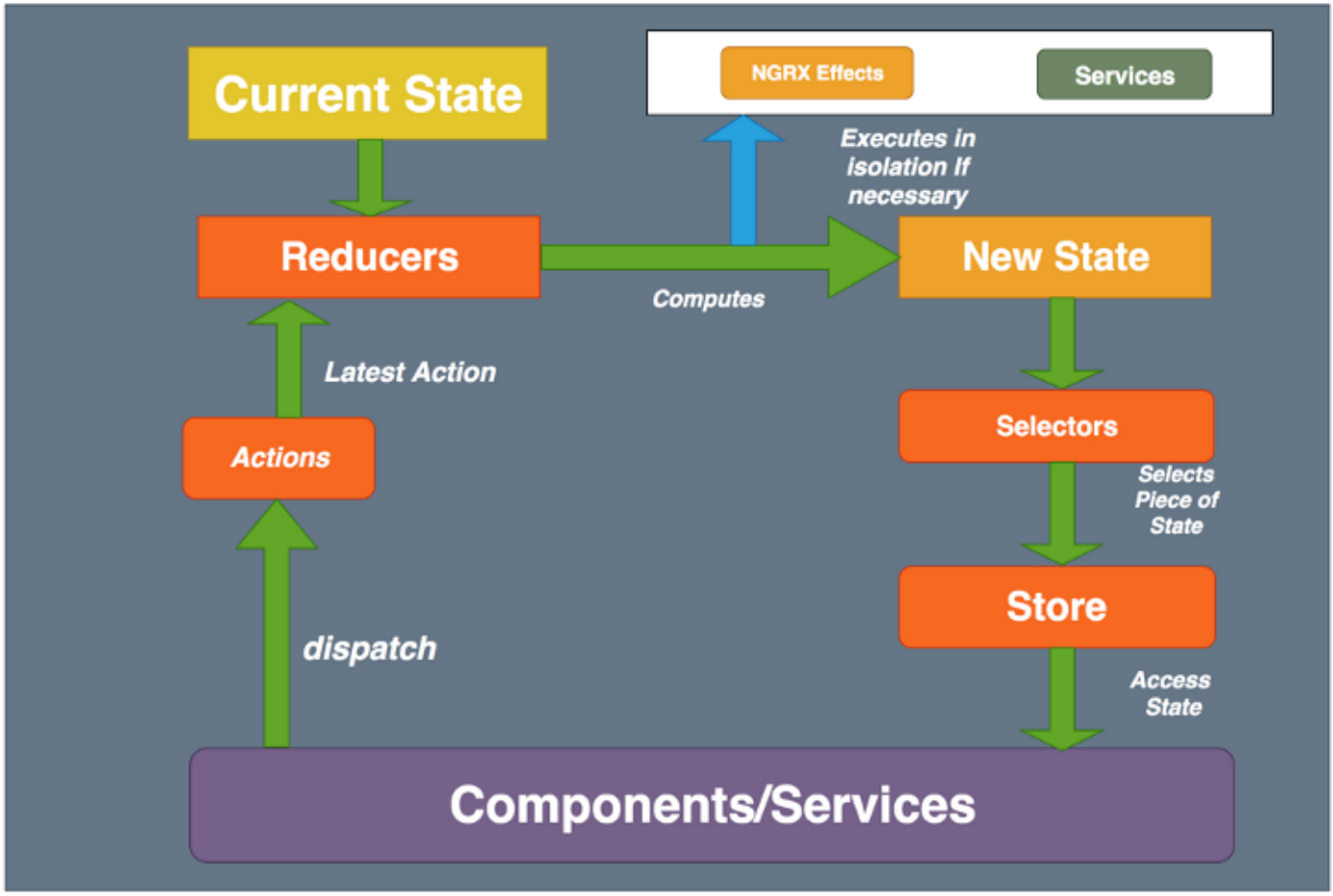
Sayfanın yenilenmesi veya yeniden yüklenmesi, uygulamanın tüm durumunu kaybedecektir. İşte o zaman yerel depolama devreye giriyor. Uygulamanın tüm durumu, sayfanın yeniden yüklenmesinden hemen önce serileştirilir ve yerel depoya kaydedilir ve tüm durum yerel depolamadan serileştirilir ve uygulamanın durumu yeniden başlatılır. Buna mağazayı yeniden nemlendirme denir.

Bazen uygulamanın verilerini almak için API çağrıları yapmamız gerekir. Mağaza, arka uç API'sinden verilere ihtiyaç duyduğunda, bir API çağrısı yapmak, verileri almak ve mağazayı güncellemek için NGRX efektlerini kullanır. Bunu ilerideki bölümlerde detaylı olarak göreceğiz.

NGRX Temelleri

NGRX Store, açısal uygulamalar için Durum yönetimi sağlayan Redux'dan esinlenmiştir ve açısal uygulamaların performansını ve tutarlılığını artıran RXJS ile güçlendirilmiştir.

Nasıl çalıştığını görelim !!.



Ayrıntılı olarak NGRX Store

- **Eylemler** , bileşenlerden ve hizmetlerden gönderilir. Bunlar, **tür** ve **yüke** sahip benzersiz olaylardır ve mağazaya gönderilebilir
- **Redüktörler** , en son eylemi ve mevcut durumu alan ve yeni durumu döndüren saf işlevlerdir.
- **Seçiciler** , durumun bir dilimini seçmemizi sağlayan saf işlevlerdir.
- **Devlet** üzerinden erişilebilir **Mağaza** bileşenleri ve hizmette gözlenebiliri
- **NGRX Efektleri** , durum için yeni verileri almak üzere çalıştırılabilen işlevlerdir. Örneğin, bileşeniniz API'den yeni verilere ihtiyaç duyarsa, bileşen bir eylem gönderir, indirgeyiciler yeni verileri almak için etkileri ve hizmetleri çağırır, indirgeyici API'den bu verilerle yeni durumu döndürür.

Bu makale için bazı ön koşullar vardır. Dizüstü bilgisayarınızda nodej'lerin kurulu olması ve http'nin nasıl çalıştığı gerekir. Pratik yapmak ve bunu dizüstü bilgisayarınızda çalıştırmak istiyorsanız, bunları dizüstü bilgisayarınızda bulundurmanız gerekir.

- NodeJS (https://nodejs.org/en/)
- Açısal CLI (https://cli.angular.io/)
- Tipler (https://www.typescriptlang.org/)
- VSCode (https://code.visualstudio.com/)
- ngx-bootstrap (https://valor-software.com/ngx-bootstrap/)
- NGRX (https://ngrx.io/)

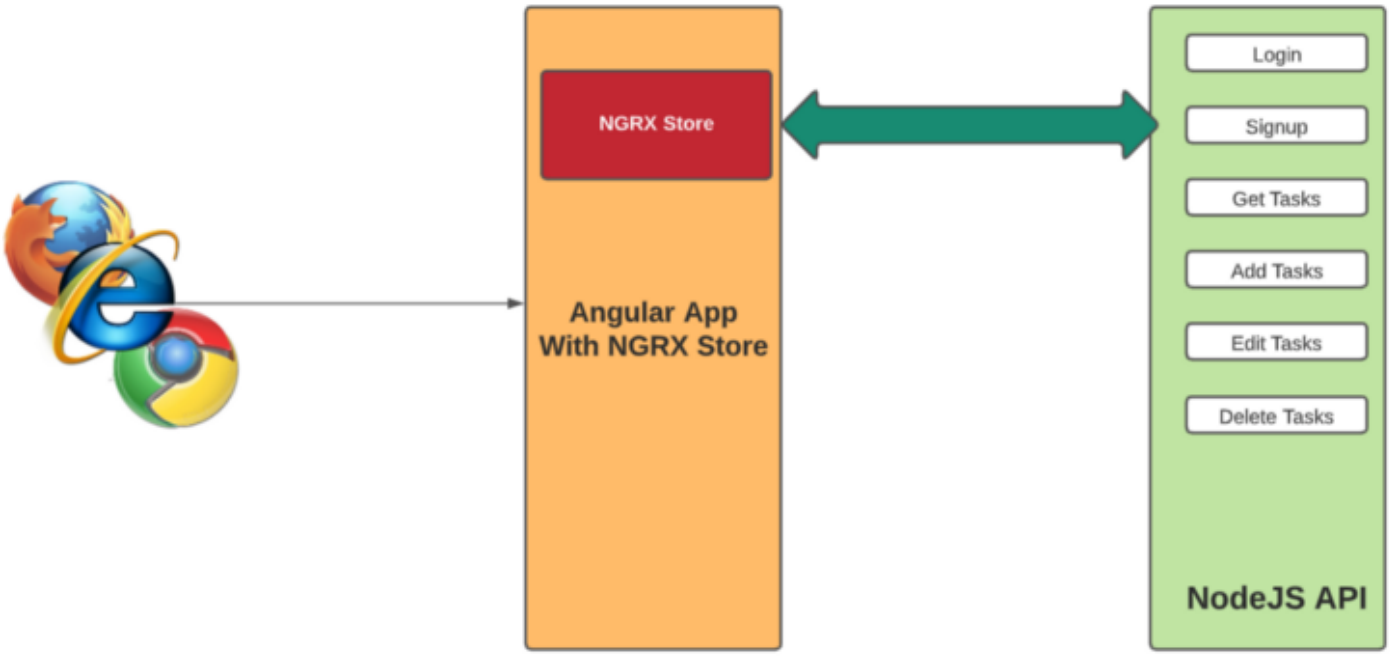
NodeJS ile Açısal Uygulama Nasıl Geliştirilir ve Oluşturulur ()

Kurulum

NGRX söz konusu olduğunda, bir sürü kütüphane kurmamız gerekiyor. NGRX için aşağıdaki tüm kütüphaneleri kuralım.

```
// install NGRX dependencies
npm install @ngrx/{effects,entity,router-store,store,store-devtools} --save
npm install ngrx-store-freeze ngrx-store-localstorage --save
```

Tüm uygulamanın API bölümünü adım adım uygulayalım. API, bir Nodejs ve ekspres çerçeve ile oluşturulmuştur. Bir oturum açma, kayıt olma, görev ekleme, görevleri silme, görevleri düzenleme ve görev işlemlerini alma hakkımız var.



Açısal arama NodeJS API

Yukarıdaki diyagrama bakarsanız, Angular with NGRX mağazası NodeJS API'yi çağırır. İşte `server.js` tüm işlemleri yapan ve **3080** portunda dinleyen dosya **.**

```
1   const express = require('express');
2   const path = require('path');
3   const app = express(),
4         randomId = require('random-id')
5         bodyParser = require("body-parser");
6         port = 3080;
7
8   const idlen = 10;
9   // place holder for the data
10  const users = [];
11  let tasks = [];
12
13  app.use(bodyParser.json());
14  app.use(express.static(path.join(__dirname, '../ui/build')));
15
16  app.get('/api/users', (req, res) => {
17    console.log('api/users called!')
18    res.json(users);
19  });
20
21  app.post('/api/user', (req, res) => {
```

```
22    const user = req.body.user;
23    console.log('Adding user:::::', user);
24    users.push(user);
25    res.json("user addedd");
26  });
27
28  app.post('/api/login', (req, res) => {
29    const user = req.body.user;
30    console.log('Adding user:::::', user);
31    const usersFound = users.filter(usr => usr.email === user.email && usr.password === user.password
32    if (usersFound.length > 0) {
33      res.json({status: true, message: 'user found', user: usersFound[0]});
34    } else {
35      res.json({status: false, message: 'user not found'});
36    }
37  });
38
39  app.post('/api/signup', (req, res) => {
40    const user = req.body.user;
41    console.log('Adding user:::::', user);
42    users.push(user);
43    res.json({status: true, numberOfUsers: users.length});
44  });
45
46  app.get('/api/tasks', (req, res) => {
47    res.json(tasks);
48  });
49
50  app.post('/api/task', (req, res) => {
51    const task = req.body.task;
52    const id = randomId(idlen);
53    task.id = id;
54    tasks.push(task);
55    res.json({status:true, taskId: task.id, message: `task ${task.id} addedd`});
56  });
57
58  app.delete('/api/task/:id', (req, res) => {
59    console.log('deleting task:::', req.params.id);
60    const id = req.params.id;
61    tasks = tasks.filter(tsk => tsk.id !== id)
62    res.json({status:true, message: `task ${id} deleted`});
63  });
64
65  app.put('/api/task', (req, res) => {
66    const task = req.body.task;
67    tasks = tasks.map(tsk => {
68        if(tsk.id === task.id) tsk = task;
```

```
69        return tsk;
70    });
71    res.json({status:true, message: `task ${task.id} edited`});
72  });
73
74  app.get('/', (req,res) => {
75    res.sendFile(path.join(__dirname, '../ui/build/index.html'));
76  });
77
78  app.listen(port, () => {
79      console.log(`Server listening on the port::${port}`);
80  });
```

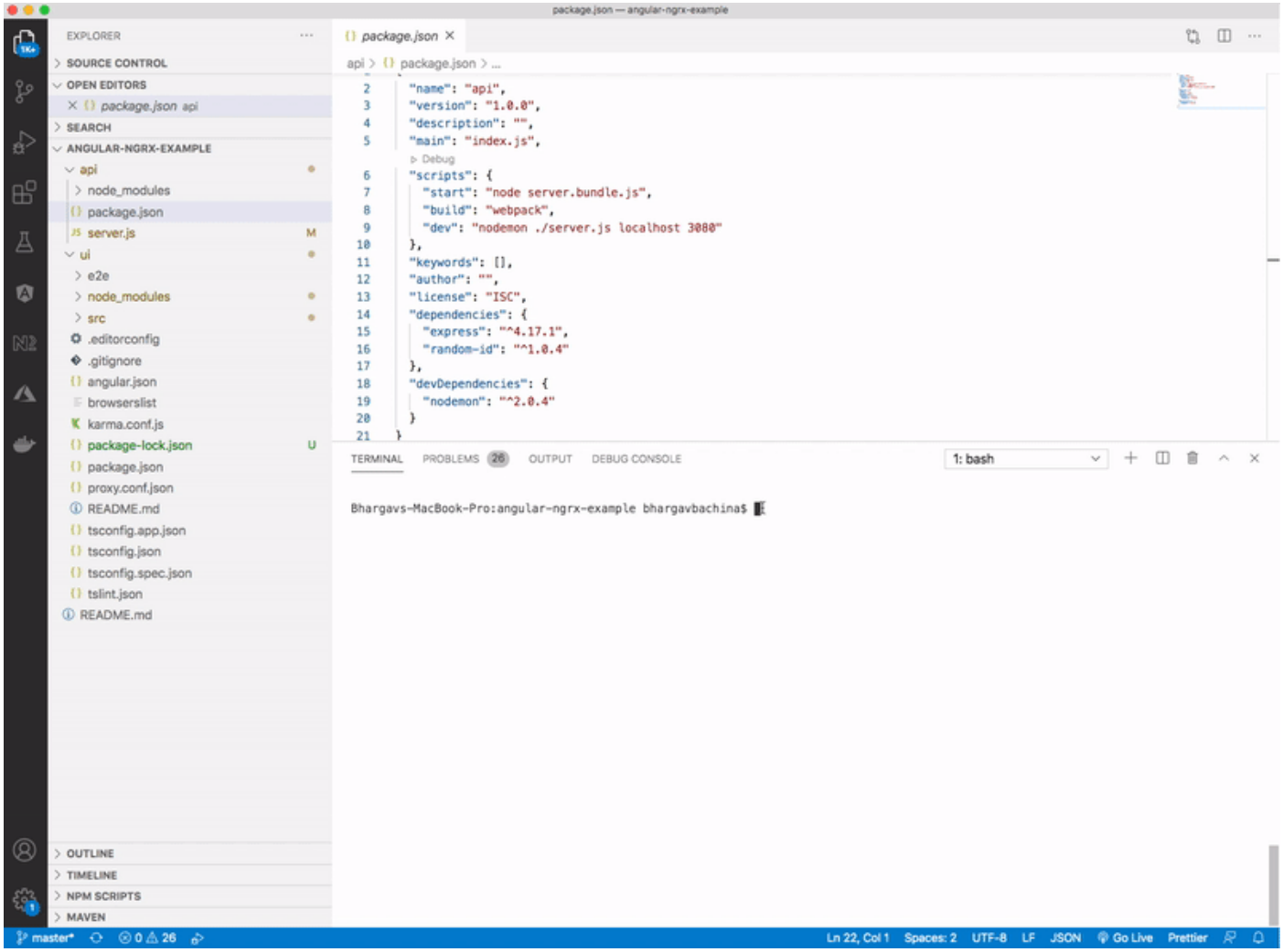
raw
s://gist.github.com/bbachi/69e54dcdfdfc54c291efe4961c671730/raw/3704db3b785a45983fa67d1b5ef4f89ce25efa4b/server.js)
server.js (https://gist.github.com/bbachi/69e54dcdfdfc54c291efe4961c671730#file-server-js) hosted with ♡ by GitHub (https://github.com)

Herhangi bir dosya değişikliğini izlemek ve sunucuyu yeniden başlatmak için geliştirme ortamında nodemon kullanıyoruz. `npm run dev` Nodejs sunucusunu geliştirme modunda başlatmak için bu komutu çalıştırmanız gereken tek şey.

```
{
  "name": "api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node server.bundle.js",
    "build": "webpack",
    "dev": "nodemon ./server.js localhost 3080"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "random-id": "^1.0.4"
  },
  "devDependencies": {
    "nodemon": "^2.0.4"
  }
}
```

t.github.com/bbachi/078a14bcde596d64ae4bf814a1a630b1/raw/2712040940dcbdbc03d1ccd59198d466e17d8af2/package.json)
package.json (https://gist.github.com/bbachi/078a14bcde596d64ae4bf814a1a630b1#file-package-json) hosted with ♡ by GitHub (https://github.com)

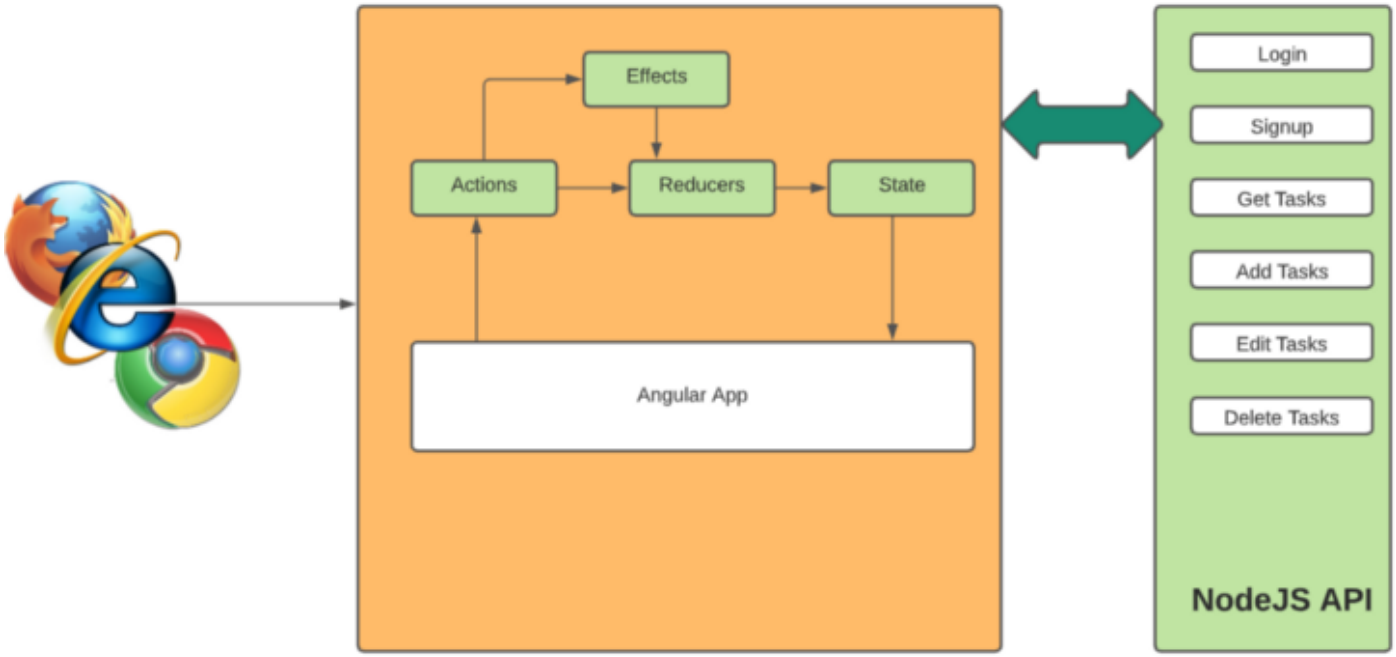
Sunucuyu nodemon ile başlattığınız demo burada.



API Demosu

NGRX Uygulaması

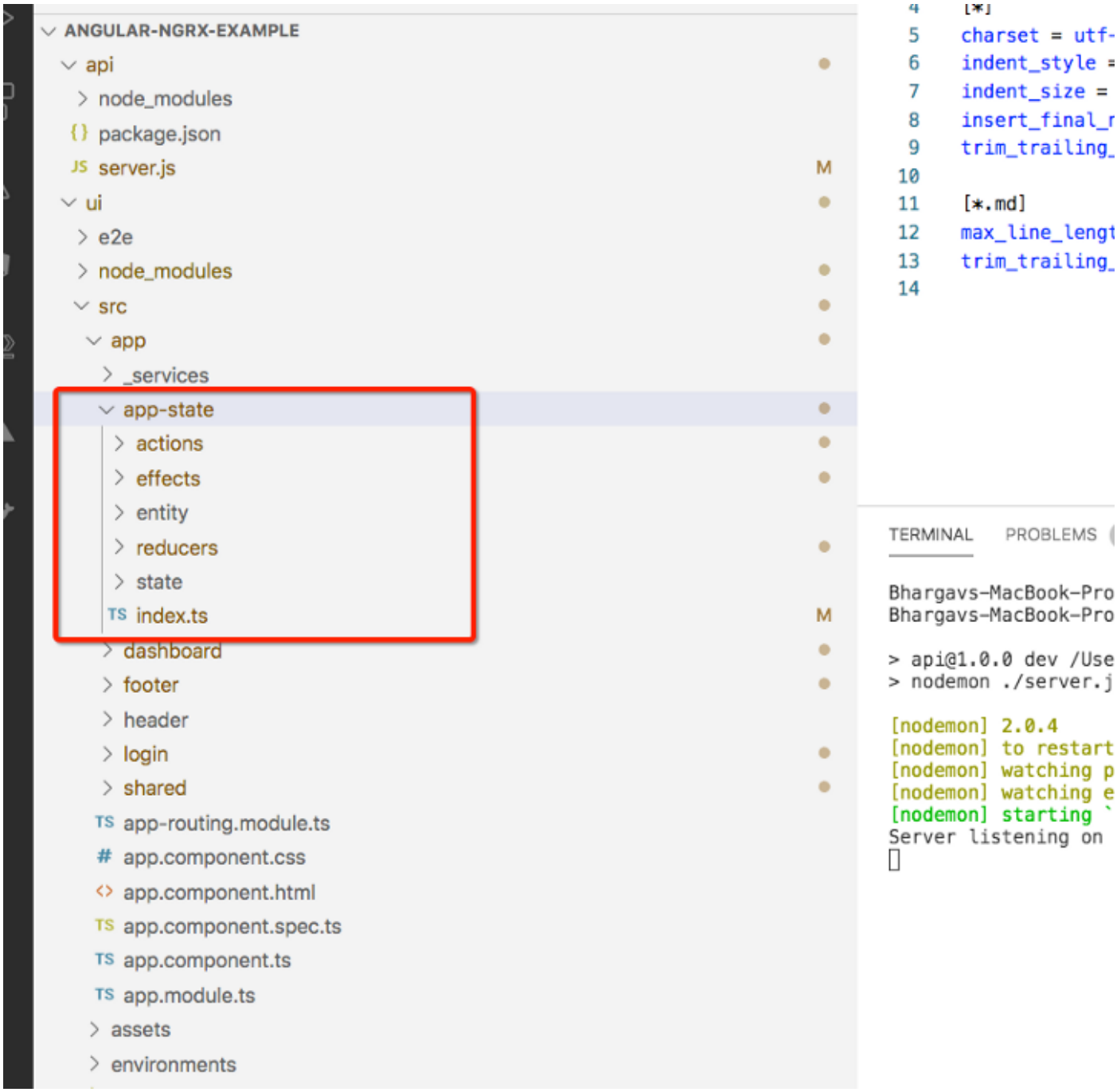
NGRX'i adım adım uygulayalım. İşte uygulama için NGRX yapısının şeması. Eylemlerimiz, azaltıcılarımız, efektlerimiz vb. Var.



Angular With NGRX, NodeJS API'yi çağırıyor

Yukarıdaki diyagrama bakarsak, Angular Uygulamasındaki bileşenler, Eylemler göndererek NGRX Mağazasını çağırır. Eylemlerin API'yi çağırmak gibi yan etkileri varsa, etkiler aracılığıyla API'yi çağırır. API'den yanıtı aldığımızda, redüktörler aracılığıyla uygulamanın durumunu değiştiriyoruz. Buradaki redüktörler saf fonksiyonlardır, yani bunlar mevcut durumu alır ve mevcut durumu değiştirmeden yeni durumu verir.

Öncelikle Angular klasör yapısında durumu tanımlamamız gerekiyor. Genellikle uygulamanın durumu için ayrı bir klasör tutuyorum.



Başvurunun durumu

Giriş / Kayıt Akışı

Giriş ve kayıt akışına ait tüm eylemler aşağıda tanımlanmıştır. Eylemler, türü ve yükü olan nesnelerdir. Yük burada isteğe bağlıdır. CreateAction ve props'un ngrx / store'dan içe aktarıldığına dikkat edin.

```
1   import { createAction, props } from '@ngrx/store';
2   import { User } from '../entity';
3
4   export const USER_LOGIN = '[Login Page] Login';
```

```
export const USER_LOGIN_SUCCESS = '[Login Page] Login Success';
export const USER_LOGIN_FAILURE = '[Login Page] Login Failure';

export const login = createAction(
  USER_LOGIN,
  props<{user: User}>()
);

export const loginSuccess = createAction(
  USER_LOGIN_SUCCESS,
  props<any>()
)

export const loginFailure = createAction(
  USER_LOGIN_FAILURE,
  props<{message: string}>()
)
```

ithub.com/bbachi/b295c293abc8436b6c522f1483dd0789/raw/06e0c73c1e26e00b63470279fd613ed416c9c4fd/login.actions.ts) login.actions.ts (https://gist.github.com/bbachi/b295c293abc8436b6c522f1483dd0789#file-login-actions-ts) hosted with ♡ by **GitHub (https://github.com)**

Bu, kayıt akışı için Eylemler dosyasıdır.

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

export const USER_SIGNUP = '[SignUp Page] Signup';
export const USER_SIGNUP_SUCCESS = '[SignUp Page] Signup Success';
export const USER_SIGNUP_FAILURE = '[SignUp Page] Signup Failure';

export const signup = createAction(
  USER_SIGNUP,
  props<{user: User}>()
);

export const signupSuccess = createAction(
  USER_SIGNUP_SUCCESS,
  props<any>()
)

export const signupFailure = createAction(
  USER_SIGNUP_FAILURE,
  props<{message: string}>()
)
```

github.com/bbachi/1f0328659c876f6eaf2d279fedd8d66a/raw/b68fe5ea24204c0cfc2fbeca60355d188ec5f178/signup.actions.ts) signup.actions.ts (https://gist.github.com/bbachi/1f0328659c876f6eaf2d279fedd8d66a#file-signup-actions-ts) hosted with ♡ by **GitHub (https://github.com)**

Oturum açma ve kayıt işlemleri, kullanıcıların kimliğini doğrulamak ve kaydetmek için API'yi çağırmalıdır. Yan etkileri işleyen etkileri tanımlamanız gerekir.

```
import { Injectable } from '@angular/core';
import { Actions, createEffect, ofType } from '@ngrx/effects';
import { map, exhaustMap, catchError } from 'rxjs/operators';
import { of } from 'rxjs';
import { AppService } from '../../_services/app.service';
import * as userActions from '../actions';

@Injectable()
export class UserEffects {

  constructor(
    private actions$: Actions,
    private appService: AppService
  ) {}

  userLogin$ = createEffect(() =>
    this.actions$.pipe(
      ofType(userActions.login),
      exhaustMap(action =>
        this.appService.login(action.user).pipe(
          map(response => userActions.loginSuccess(response)),
          catchError((error: any) => of(userActions.loginFailure(error))))
      )
    )
  );

  userSignup$ = createEffect(() =>
    this.actions$.pipe(
      ofType(userActions.signup),
      exhaustMap(action =>
        this.appService.signup(action.user).pipe(
          map(response => userActions.signupSuccess(response)),
          catchError((error: any) => of(userActions.signupFailure(error))))
      )
    )
  );

}
```

t.github.com/bbachi/561ae28fb7390baf39fa6a45389d0373/raw/da6e95d509f8431a4459cf650908dd4c65fdfe58/user.effects.ts) **user.effects.ts (https://gist.github.com/bbachi/561ae28fb7390baf39fa6a45389d0373#file-user-effects-ts)** hosted with ♡ by **GitHub (https://github.com)**

Bu efektler dosyasında iki fonksiyon tanımladık. Birincisi, oturum açma türündeki Eylem için, diğeri ise kayıt türündeki Eylem içindir. API çağrılarını çağırdığımızı ve yanıtı uygun Eylem yüküne eşlediğimizi görebilirsiniz.

İşte kullanıcı girişi ve kayıt akışı için azaltıcı. Aşağıdaki dosyayı fark ederseniz, başlangıç durumunu tanımlamışızdır. Her eylem için durumu buna göre değiştiriyoruz. Bileşenler için yararlı olan bazı verileri veren bazı işlevleri de dışa aktarıyoruz.

```
1   import { Action, createReducer, on } from '@ngrx/store';
2   import { User } from '../entity';
3   import * as userActions from '../actions';
4   import * as storage from '../state/storage';
5
6   export interface State {
7     user: User;
8     result: any;
9     isLoading: boolean;
10    isLoadingSuccess: boolean;
11    isLoadingFailure: boolean;
12  }
13
14  export const initialState: State = {
15    user: storage.getItem('user').user,
16    result: '',
17    isLoading: false,
18    isLoadingSuccess: false,
19    isLoadingFailure: false
20  };
21
22  const loginReducer = createReducer(
23    initialState,
24    on(userActions.login, (state, {user}) => ({user, isLoading: true})),
25    on(userActions.loginSuccess, (state, result) => ({user: result.user, result, isLoading: false, is
26    on(userActions.signup, (state, {user}) => ({user, isLoading: true})),
27    on(userActions.signupSuccess, (state, result) => ({user: state.user, result, isLoading: false, is
28  );
29
30  export function reducer(state: State | undefined, action: Action): any {
31    return loginReducer(state, action);
32  }
33
34  export const getLoggedInUser = (state: State) => {
35    return {
36      user: state.user,
37      isLoadingSuccess: state.isLoadingSuccess
38    }
39  };
40
41  export const userLogin = (state: State) => {
42    return {
```

```
    user: state.user,
    result: state.result,
    isLoading: state.isLoading,
    isLoadingSuccess: state.isLoadingSuccess
  }
};

export const userSignup = (state: State) => {
  return {
    user: state.user,
    result: state.result,
    isLoading: state.isLoading,
    isLoadingSuccess: state.isLoadingSuccess
  }
};
```

github.com/bbachi/8c95c5024fbddfd5a0d37867fa1753d2/raw/5474c37626560a56a23bb37fa5241215cf059cba/user.reducer.ts) user.reducer.ts (https://gist.github.com/bbachi/8c95c5024fbddfd5a0d37867fa1753d2#file-user-reducer-ts) hosted with ♡ by GitHub (https://github.com)

Yapılacak Görevler Akışı

ToDo akışına ait tüm eylemler aşağıda tanımlanmıştır. Eylemler, türü ve yükü olan nesnelerdir. Yük burada isteğe bağlıdır. CreateAction ve props'un ngrx / store'dan içe aktarıldığına dikkat edin.

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

export const GET_TASKS = '[Task] Get Tasks';
export const GET_TASKS_SUCCESS = '[Task] Get Tasks Success';
export const GET_TASKS_FAILURE = '[Task] Get Tasks Failure';

export const CREATE_TASK = '[Task] Create Task';
export const CREATE_TASK_SUCCESS = '[Task] Create Task Success';
export const CREATE_TASK_FAILURE = '[Task] Create Task Failure';

export const DELETE_TASK = '[Task] Delete Task';
export const DELETE_TASK_SUCCESS = '[Task] Delete Task Success';
export const DELETE_TASK_FAILURE = '[Task] Delete Task Failure';

export const EDIT_TASK = '[Task] Edit Task';
export const EDIT_TASK_SUCCESS = '[Task] Edit Task Success';
export const EDIT_TASK_FAILURE = '[Task] Edit Task Failure';


export const getTasks = createAction(
  GET_TASKS
```

```
23  );
24
25  export const getTasksSuccess = createAction(
26    GET_TASKS_SUCCESS,
27    props<any>()
28  );
29
30  export const getTasksFailure = createAction(
31    GET_TASKS_FAILURE,
32    props<{any}>()
33  );
34
35  export const createTask = createAction(
36    CREATE_TASK,
37    props<{task: any}>()
38  );
39
40  export const createTaskSuccess = createAction(
41    CREATE_TASK_SUCCESS,
42    props<any>()
43  );
44
45  export const createTaskFailure = createAction(
46    CREATE_TASK_FAILURE,
47    props<{any}>()
48  );
49
50  export const deleteTask = createAction(
51    DELETE_TASK,
52    props<{taskid}>()
53  );
54
55  export const deleteTaskSuccess = createAction(
56    DELETE_TASK_SUCCESS,
57    props<any>()
58  );
59
60  export const deleteTaskFailure = createAction(
61    DELETE_TASK_FAILURE,
62    props<{any}>()
63  );
64
65  export const editTask = createAction(
66    EDIT_TASK,
67    props<{task: any}>()
68  );
69
```

```
70  export const editTaskSuccess = createAction(
71    EDIT_TASK_SUCCESS,
72    props<any>()
73  );
74
75  export const editTaskFailure = createAction(
76    EDIT_TASK_FAILURE,
77    props<{any}>()
78  );
```

ithub.com/bbachi/355bcbcfd209161bca86208dc74d17cc/raw/9e989cf64be24d6bb43b4994aab30e6df7ad48fc/todo.actions.ts) todo.actions.ts (https://gist.github.com/bbachi/355bcbcfd209161bca86208dc74d17cc#file-todo-actions-ts) hosted with ♡ by **GitHub (https://github.com)**

Yapılacak işlemlerin, API'den görevleri oluşturmak, silmek, düzenlemek ve almak için API'yi çağırması gerektiğinden. Yan etkileri işleyen etkileri tanımlamanız gerekir.

```
1   import { Injectable } from '@angular/core';
2   import { Actions, createEffect, ofType } from '@ngrx/effects';
3   import { of } from 'rxjs';
4   import { map, exhaustMap, catchError } from 'rxjs/operators';
5   import { TodoService } from '../../_services';
6   import * as todoActions from '../actions';
7
8   @Injectable()
9   export class TodoEffects {
10
11    constructor(
12      private actions$: Actions,
13      private todoService: TodoService
14    ) {}
15
16    getTasks$ = createEffect(() =>
17      this.actions$.pipe(
18        ofType(todoActions.getTasks),
19        exhaustMap(action =>
20          this.todoService.getTasks().pipe(
21            map(response => {
22              console.log("response:::", response)
23              return todoActions.getTasksSuccess({response})
24            }),
25            catchError((error: any) => of(todoActions.getTasksFailure(error))))
26        )
27      )
28    );
29
30    createTask$ = createEffect(() =>
31      this.actions$.pipe(
```

```
        ofType(todoActions.createTask),
        exhaustMap(action =>
          this.todoService.addTask(action.task).pipe(
            map(response => todoActions.createTaskSuccess(response)),
            catchError((error: any) => of(todoActions.createTaskFailure(error))))
        )
      )
    );


    deleteTask$ = createEffect(() =>
      this.actions$.pipe(
        ofType(todoActions.deleteTask),
        exhaustMap(action => this.todoService.deleteTask(action.taskid).pipe(
            map(response => todoActions.deleteTaskSuccess(response)),
            catchError((error: any) => of(todoActions.deleteTaskFailure(error))))
        )
      )
    );

    editTask$ = createEffect(() =>
      this.actions$.pipe(
        ofType(todoActions.editTask),
        exhaustMap(action =>
          this.todoService.editTask(action.task).pipe(
            map(response => todoActions.editTaskSuccess(response)),
            catchError((error: any) => of(todoActions.editTaskFailure(error))))
        )
      )
    );

  }
```

.github.com/bbachi/9c48287d7b49841ba3e768f5ea1424dc/raw/80840f133fbcb6f3ef9ddb544fb2fde9629ad81b/todo.effects.ts)
todo.effects.ts (https://gist.github.com/bbachi/9c48287d7b49841ba3e768f5ea1424dc#file-todo-effects-ts) hosted with ♡ by **GitHub (https://github.com)**

Bu efekt dosyasında dört fonksiyon tanımladık. Her CRUD işlemi için bir tane. API çağrılarını çağırdığımızı ve yanıtı uygun Eylem yüküne eşlediğimizi görebilirsiniz.

İşte yapılacak iş akışı için azaltıcı. Aşağıdaki dosyayı fark ederseniz, başlangıç durumunu tanımlamışızdır. Her eylem için durumu buna göre değiştiriyoruz. Bileşenler için yararlı olan bazı verileri veren bazı işlevleri de dışa aktarıyoruz.

```
import { Action, createReducer, on } from '@ngrx/store';
import { Task } from '../entity';
import * as todoActions from '../actions';
import * as _ from 'lodash'
import * as storage from '../state/storage';
```

```
 6
 7  export interface State {
 8    tasks?: Task[];
 9    currentTask?: Task;
10    deleteTaskId?: any;
11    result?: any;
12    isLoading?: boolean;
13    isLoadingSuccess?: boolean;
14    isLoadingFailure?: boolean;
15  }
16
17  export const initialState: State = {
18    tasks: storage.getItem('todo').tasks,
19    currentTask: {},
20    deleteTaskId: '',
21    result: '',
22    isLoading: false,
23    isLoadingSuccess: false,
24    isLoadingFailure: false
25  };
26
27  const todoReducer = createReducer(
28    initialState,
29
30    // GeTasks
31    on(todoActions.getTasks, (state) => ({...state, isLoading: true})),
32    on(todoActions.getTasksSuccess, (state, result) => ({tasks: result.response, isLoading: false, is
33
34    // Create Task Reducers
35    on(todoActions.createTask, (state, {task}) => ({...state, isLoading: true, currentTask: task})),
36    on(todoActions.createTaskSuccess, (state, result) => {
37      const tasks = undefined !== state.tasks ? _.cloneDeep(state.tasks) : [];
38      const currentTask = undefined !== state.currentTask ? _.cloneDeep(state.currentTask) : {};
39      currentTask.id = result.taskId;
40      tasks.push(currentTask);
41      return {
42        tasks,
43        isLoading: false,
44        isLoadingSuccess: true
45      };
46    }),
47
48    // Delete Task Reducers
49    on(todoActions.deleteTask, (state, {taskid}) => ({...state, isLoading: true, deleteTaskId: taskid
50    on(todoActions.deleteTaskSuccess, (state, result) => {
51      let tasks = undefined !== state.tasks ? _.cloneDeep(state.tasks) : [];
52      if (result.status) {
```

```
        tasks = tasks.filter(task => task.id !== state.deleteTaskId);
      }
      return {
        tasks,
        isLoading: false,
        isLoadingSuccess: true
      };
    }),

     // Edit Task Reducers
     on(todoActions.editTask, (state, {task}) => ({...state, isLoading: true, currentTask: task})),
     on(todoActions.editTaskSuccess, (state, result) => {
      let tasks = undefined !== state.tasks ? _.cloneDeep(state.tasks) : [];
      const currentTask = undefined !== state.currentTask ? _.cloneDeep(state.currentTask) : {};
      tasks = tasks.map(tsk => {
        if (tsk.id === currentTask.id) {
          tsk = currentTask;
        }
        return tsk;
      });
      return {
        tasks,
        isLoading: false,
        isLoadingSuccess: true
      };
    })
  );

  export function reducer(state: State | undefined, action: Action): any {
    return todoReducer(state, action);
  }

  export const getTasks = (state: State) => {
    return {
      tasks: state.tasks,
      isLoading: state.isLoading,
      isLoadingSuccess: state.isLoadingSuccess
    };
  };
```

:hub.com/bbachi/a608a7c4dce5dc1d38e3d7bd84640a9b/raw/99e01a093f7ac625a05e154765d669ddb787acd5/todo.reducer.ts) **todo.reducer.ts (https://gist.github.com/bbachi/a608a7c4dce5dc1d38e3d7bd84640a9b#file-todo-reducer-ts)** hosted with ♡ by **GitHub (https://github.com)**

İndirgeyicilerinizi ve dışa aktarma işlevlerinizi seçiciler yardımıyla tanımladığınız tam index.ts dosyası burada.

```
import {
```

```
  ActionReducer,
  ActionReducerMap,
  createFeatureSelector,
  createSelector,
  MetaReducer
} from '@ngrx/store';
import { localStorageSync } from 'ngrx-store-localstorage';
import { environment } from '../../environments/environment';
import * as fromUser from './reducers/user.reducer';
import * as fromTodo from './reducers/todo.reducer';

export interface State {
  user: fromUser.State;
  todo: fromTodo.State;
}

export const reducers: ActionReducerMap<State> = {
  user: fromUser.reducer,
  todo: fromTodo.reducer,
};

const reducerKeys = ['user', 'todo'];
export function localStorageSyncReducer(reducer: ActionReducer<any>): ActionReducer<any> {
  return localStorageSync({keys: reducerKeys})(reducer);
}

// console.log all actions
export function debug(reducer: ActionReducer<any>): ActionReducer<any> {
  return function(state, action) {
    console.log('state', state);
    console.log('action', action);

    return reducer(state, action);
  };
}


export const metaReducers: MetaReducer<State>[] = !environment.production ? [debug, localStorageSyn

export const getLoginState = createFeatureSelector<fromUser.State>('user');

export const getLoggedInUser = createSelector(
  getLoginState,
  fromUser.getLoggedInUser
);

export const userLogin = createSelector(
```

```
49    getLoginState,
50    fromUser.userLogin
51  );
52
53  export const userSignup = createSelector(
54    getLoginState,
55    fromUser.userSignup
56  );
57
58
59  // Todo reducers Begin
60
61  export const geTodoState = createFeatureSelector<fromTodo.State>('todo');
62
63  export const getTasks = createSelector(
64    geTodoState,
65    fromTodo.getTasks
66  );
```

raw
s://gist.github.com/bbachi/11ef6889ee5a25c35e1f534ac7f37cbd/raw/659360699d7f584855621b27f9b829f2a332a2b3/index.ts)
index.ts (https://gist.github.com/bbachi/11ef6889ee5a25c35e1f534ac7f37cbd#file-index-ts) hosted with ♡ by GitHub
(https://github.com)

Açısal Uygulama

API ve NGRX uygulamalarını gördük. NGRX mağazasını Angular uygulamasına nasıl entegre edebileceğimizi görmenin zamanı geldi. Yapmamız gereken ilk şey, tüm NGRX ile ilgili kodu aşağıdaki gibi Uygulama modülüne veya özellik modülüne aktarmaktır.

```
1   import { BrowserModule } from '@angular/platform-browser';
2   import { NgModule } from '@angular/core';
3   import { FormsModule } from '@angular/forms';
4   import { HttpClientModule } from '@angular/common/http';
5   import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
6
7
8   import { AppRoutingModule } from './app-routing.module';
9   import { DashboardModule } from './dashboard/dashboard.module';
10  import { SharedModule } from './shared/shared.module';
11  import { LoginModule } from './login/login.module';
12  import { ModalModule } from 'ngx-bootstrap/modal';
13  import { AppComponent } from './app.component';
14
15  // ngrx related imports
16  import { StoreModule } from '@ngrx/store';
17  import { reducers, metaReducers } from './app-state';
18  import { UserEffects, TodoEffects } from './app-state/effects';
```

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

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    BrowserAnimationsModule,
    HttpClientModule,
    DashboardModule,
    SharedModule,
    LoginModule,
    ModalModule.forRoot(),
    // ngrx related imports
    StoreModule.forRoot(reducers, {
      metaReducers
    }),
    EffectsModule.forRoot([UserEffects, TodoEffects])
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

github.com/bbachi/c2b580829671f189b6e15eb7b896c135/raw/c354da3366849ca1f14bbae92108bd255edd5d7a/app.module.ts)
app.module.ts (https://gist.github.com/bbachi/c2b580829671f189b6e15eb7b896c135#file-app-module-ts) hosted with ♡ by **GitHub (https://github.com)**

Yukarıda görebileceğiniz gibi, efektleri içe aktarmamız gerekiyor ve NGRX Store'dan EffectsModule'a kaydolmamız gerekiyor. Eylemleri nasıl gönderebileceğimize ve mağaza değişikliklerini nasıl dinleyebileceğimize bir örnek görelim.

İşte ngrx deposunu içe aktardığımız giriş bileşeni ve yukarıda tanımladığımız eylemler.

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { Router } from '@angular/router';
import { NgForm } from '@angular/forms';
import { Store } from '@ngrx/store';
import * as userActions from '../app-state/actions';
import * as fromRoot from '../app-state';
import { Subject } from 'rxjs';
import { takeUntil } from 'rxjs/operators';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
```

```
13    styleUrls: ['./login.component.css']
14  })
15  export class LoginComponent implements OnInit, OnDestroy {
16
17    constructor(private router: Router, private readonly store: Store) {
18      this.store.select(fromRoot.userLogin).pipe(
19        takeUntil(this.destroy$)
20      ).subscribe(data => {
21        console.log('data::::', data);
22        if (data.isLoadingSuccess && data.result.status) {
23          this.router.navigate(['/dashboard']);
24        }
25      });
26    }
27
28    model: User = new User();
29    destroy$: Subject<boolean> = new Subject<boolean>();
30
31    ngOnInit() {
32    }
33
34    onSubmit(loginForm: NgForm) {
35      console.log(this.model)
36      this.store.dispatch(userActions.login({user: { email: this.model.email, password: this.model.pa
37    }
38
39    ngOnDestroy(){
40      this.destroy$.next(true);
41      this.destroy$.unsubscribe();
42  }
43
44  }
45
46  export class User {
47
48    constructor(
49
50    ) {  }
51
52    public email: string;
53    public password: string;
54
55  }
```

ub.com/bbachi/b5b689040d191fd586161e7b82704017/raw/908afa4a354b3436c3f32fd2f087557b0d51c841/login.component.ts) login.component.ts (https://gist.github.com/bbachi/b5b689040d191fd586161e7b82704017#file-login-component-ts) hosted with ♡ by **GitHub (https://github.com)**

Yukarıdaki bileşende olduğu gibi ***this.store.dispatch*** ile eylemler ***gönderiyoruz*** ve ***this.store.select*** gibi selektörler yardımıyla mağazadan ***okuyoruz.***

İşte NGRX efektleri tarafından çağrılan API'ler

```
import { Injectable } from '@angular/core';
import { Subject, Observable, of } from 'rxjs';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { map, catchError } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class AppService {

  private userLoggedIn = new Subject<boolean>();
  loginUrl = '/api/login';
  signupUrl = '/api/signup';

  constructor(private http: HttpClient) {
    this.userLoggedIn.next(false);
  }

  setUserLoggedIn(userLoggedIn: boolean) {
    this.userLoggedIn.next(userLoggedIn);
  }

  getUserLoggedIn(): Observable<boolean> {
    return this.userLoggedIn.asObservable();
  }

  login(user: any) {
    const headers = new HttpHeaders({'Content-Type' : 'application/json'});
    const options = {headers};
    return this.http.post(this.loginUrl,  {user}, options).pipe(
      map((response: Response) => response),
      catchError(err => {
        console.log(err);
        return of([]);
      })
  );
  }

  signup(user: any) {
    const headers = new HttpHeaders({'Content-Type' : 'application/json'});
    const options = {headers};
    return this.http.post(this.signupUrl,  {user}, options).pipe(
      map((response: Response) => response),
```

```
      catchError(err => {
        console.log(err);
        return of([]);
      })
    );
  }
}
```

github.com/bbachi/1480590751d76f7ee3ed37ec66e4f34a/raw/0e8db032e030a9d299e478afd59b2929ec1e6d92/app.service.ts) app.service.ts (https://gist.github.com/bbachi/1480590751d76f7ee3ed37ec66e4f34a#file-app-service-ts) hosted with ♡ by GitHub (https://github.com)

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

@Injectable({
  providedIn: 'root'
})
export class TodoService {

  constructor(private http: HttpClient) { }

  rootURL = '/api';

  getTasks() {
    return this.http.get(this.rootURL + '/tasks');
  }

  addTask(task: any) {
    return this.http.post(this.rootURL + '/task', {task});
  }

  editTask(task: any) {
    return this.http.put(this.rootURL + '/task', {task});
  }

  deleteTask(taskId: any) {
    console.log('deleting task:::', taskId);
    return this.http.delete(`${this.rootURL}/task/${taskId}`);
  }
}
```

t.github.com/bbachi/bf9a095b2901f8d7d0517afa24833159/raw/3a946d7f91f1f9ee1fd1405391255d8d26948679/todo.service.ts) todo.service.ts (https://gist.github.com/bbachi/bf9a095b2901f8d7d0517afa24833159#file-todo-service-ts) hosted with ♡ by GitHub (https://github.com)

Yukarıda verilen Github bağlantısında örneğin geri kalanını kontrol edebilirsiniz.

Özet

- NGRX, Açısal Uygulamalar için yeniden düzenlemeden esinlenen bir durum yönetim aracıdır.
- Uygulamanız büyüdükçe ve büyüdüğünde iletişimin üstesinden gelmek zorlaşır. NGRX, tek yönlü veri akışı ve tüm uygulama için tek bir doğruluk kaynağı sağlar.
- Mağazanın farkında olan bileşenlere akıllı bileşenler, mağazadan haberdar olmayan bileşenlere ise aptal bileşenler denir.
- Bazen uygulamanın verilerini almak için API çağrıları yapmamız gerekir. Mağaza, arka uç API'sinden verilere ihtiyaç duyduğunda, bir API çağrısı yapmak, verileri almak ve mağazayı güncellemek için NGRX efektlerini kullanır.
- **Eylemler** , bileşenlerden ve hizmetlerden gönderilir. Bunlar, **tür** ve **yüke** sahip benzersiz olaylardır ve mağazaya gönderilebilir
- **Redüktörler** , en son eylemi ve mevcut durumu alan ve yeni durumu döndüren saf işlevlerdir.
- **Seçiciler** , durumun bir dilimini seçmemizi sağlayan saf işlevlerdir.
- **Devlet** üzerinden erişilebilir **Mağaza** bileşenleri ve hizmette gözlenebiliri
- **NGRX Efektleri** , durum için yeni verileri almak üzere çalıştırılabilen işlevlerdir. Örneğin, bileşeniniz API'den yeni verilere ihtiyaç duyarsa, bileşen bir eylem gönderir, indirgeyiciler yeni verileri almak için etkileri ve hizmetleri çağırır, indirgeyici API'den bu verilerle yeni durumu döndürür.

Çok fazla standart kod olduğundan NGRX küçük Angular uygulamalar için değildir. Bu aynı zamanda herhangi bir Angular uygulamasında NGRX deposunu uygulamak için gereken çok fazla öğrenme eğrisidir. Bu yüzden akıllıca seçin.