

ilionx



Global Knowledge®

Angular Advanced @ngrx/store – using http directly

Peter Kassenaar –
info@kassenaar.com

WORLDWIDE LOCATIONS

BELGIUM CANADA COLOMBIA DENMARK EGYPT FRANCE IRELAND JAPAN KOREA MALAYSIA MEXICO NETHERLANDS NORWAY QATAR
SAUDI ARABIA SINGAPORE SPAIN SWEDEN UNITED ARAB EMIRATES UNITED KINGDOM UNITED STATES OF AMERICA

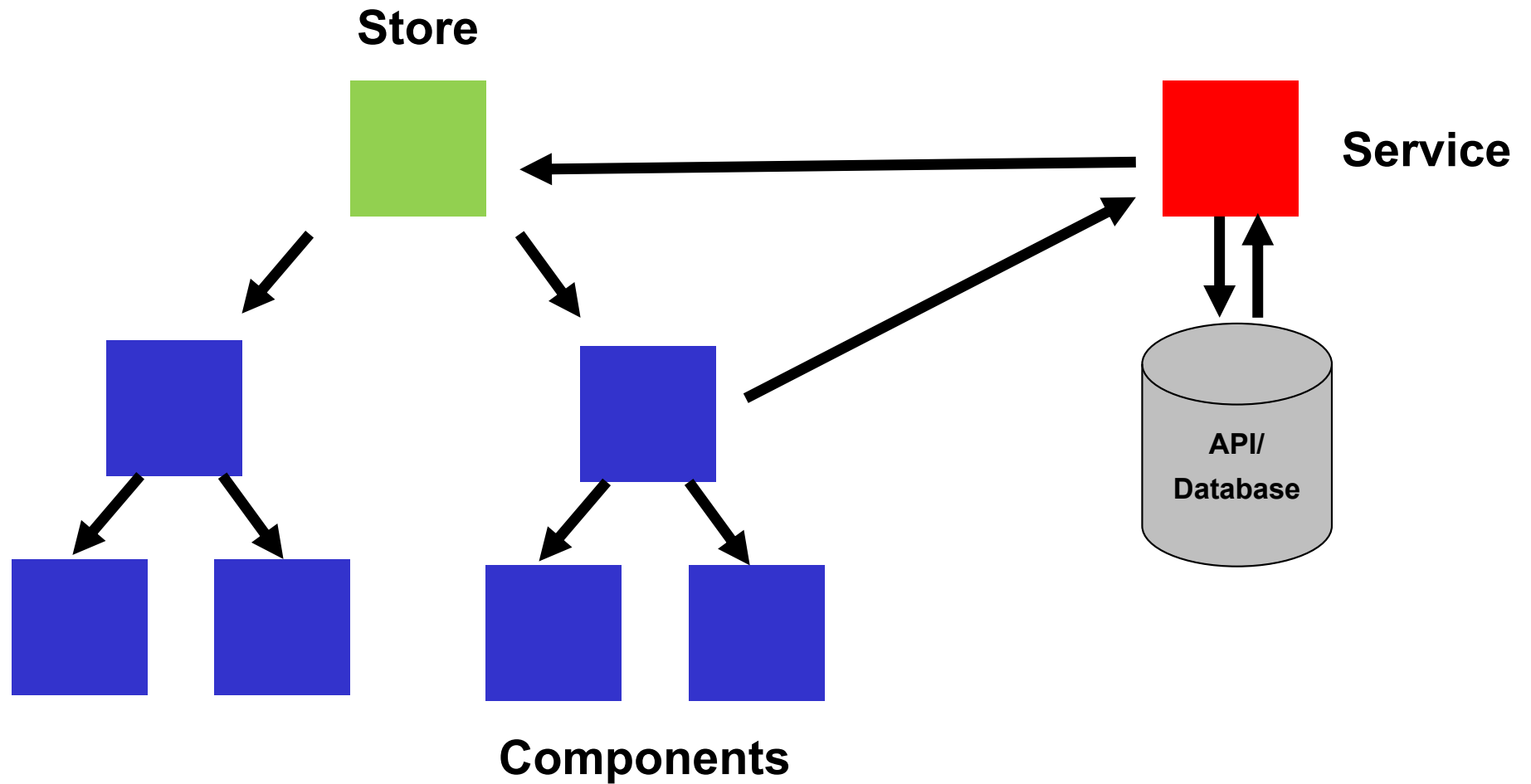


Using HttpClient directly

Talking RESTful to real API's – plain and simple!

Architecture

Call API in Service, dispatch to Store, subscribe in Components

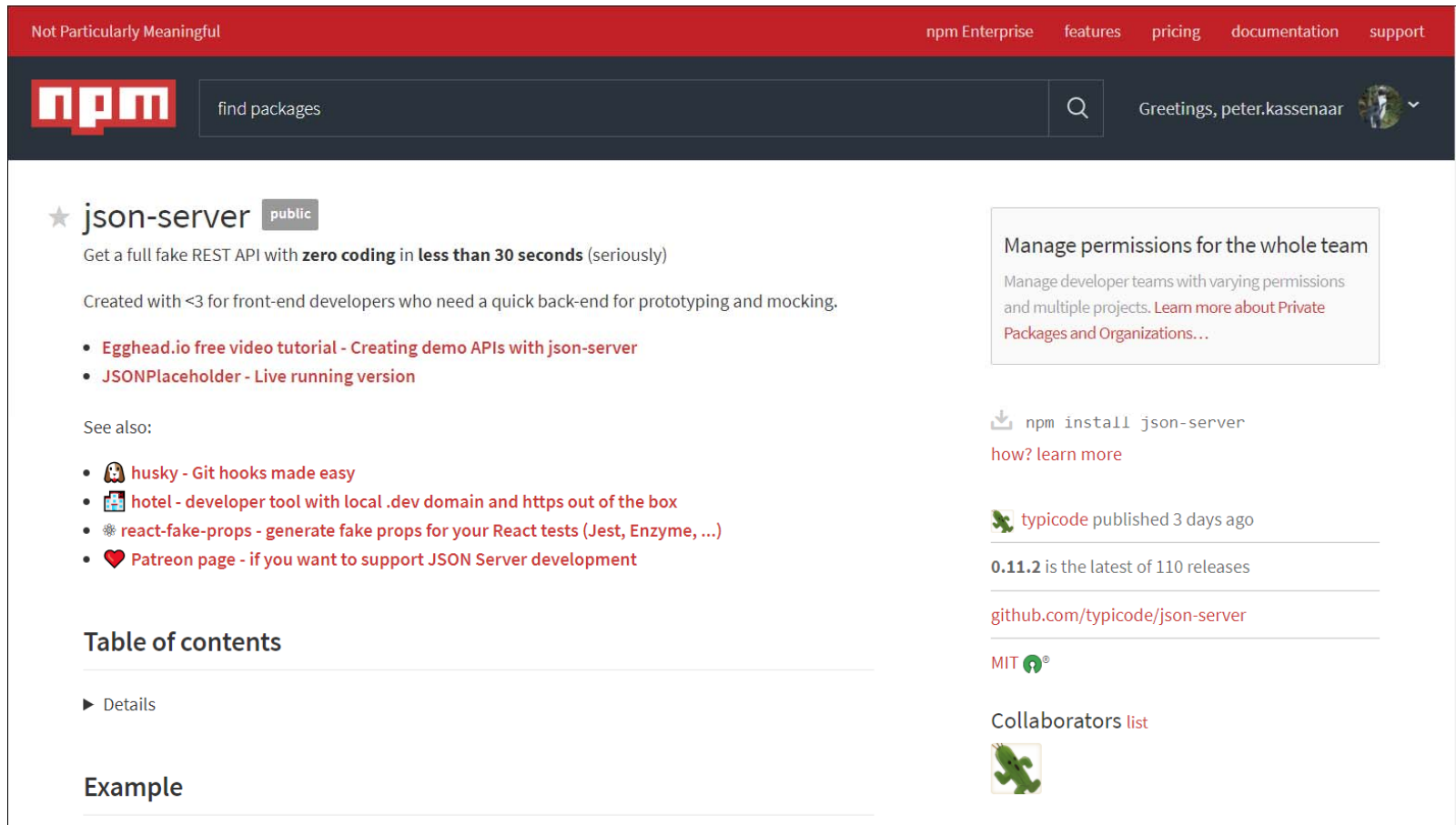


Actions and Reducers

- No changes on Actions and reducers.
- Add a service (if you haven't done so already) that talks to the outside world
- When a result comes back, dispatch the result to the store.

First – add a server

- We're using json-server here
- Provides a simple RESTful API, based on .json-file in webroot



The screenshot shows the npm website interface for the 'json-server' package. At the top, there's a red navigation bar with links for 'npm Enterprise', 'features', 'pricing', 'documentation', and 'support'. Below this is a dark blue header with the 'npm' logo, a search bar containing 'find packages', and a user profile for 'Greetings, peter.kassenaar'. The main content area features the 'json-server' package name with a 'public' badge. A description states it's a 'full fake REST API with zero coding in less than 30 seconds (seriously)'. It mentions it was created with '<3 for front-end developers who need a quick back-end for prototyping and mocking.' There are two links: 'Egghead.io free video tutorial - Creating demo APIs with json-server' and 'JSONPlaceholder - Live running version'. A 'See also:' section lists related projects like 'husky', 'hotel', 'react-fake-props', and a 'Patreon page'. On the right, there's a 'Manage permissions for the whole team' section, a command 'npm install json-server', and information about the latest release '0.11.2'. The bottom left has sections for 'Table of contents' (with 'Details' expanded) and 'Example'.

Not Particularly Meaningful

npm Enterprise features pricing documentation support

npm find packages

Greetings, peter.kassenaar

★ json-server public

Get a full fake REST API with **zero coding** in **less than 30 seconds** (seriously)

Created with <3 for front-end developers who need a quick back-end for prototyping and mocking.

- [Egghead.io free video tutorial - Creating demo APIs with json-server](#)
- [JSONPlaceholder - Live running version](#)

See also:

- [husky - Git hooks made easy](#)
- [hotel - developer tool with local .dev domain and https out of the box](#)
- [react-fake-props - generate fake props for your React tests \(Jest, Enzyme, ...\)](#)
- [Patreon page - if you want to support JSON Server development](#)

Table of contents

- ▶ Details

Example

Manage permissions for the whole team

Manage developer teams with varying permissions and multiple projects. [Learn more about Private Packages and Organizations...](#)

npm install json-server

[how?](#) [learn more](#)


[typicode](#) published 3 days ago

0.11.2 is the latest of 110 releases

[github.com/typicode/json-server](#)

MIT

Collaborators [list](#)

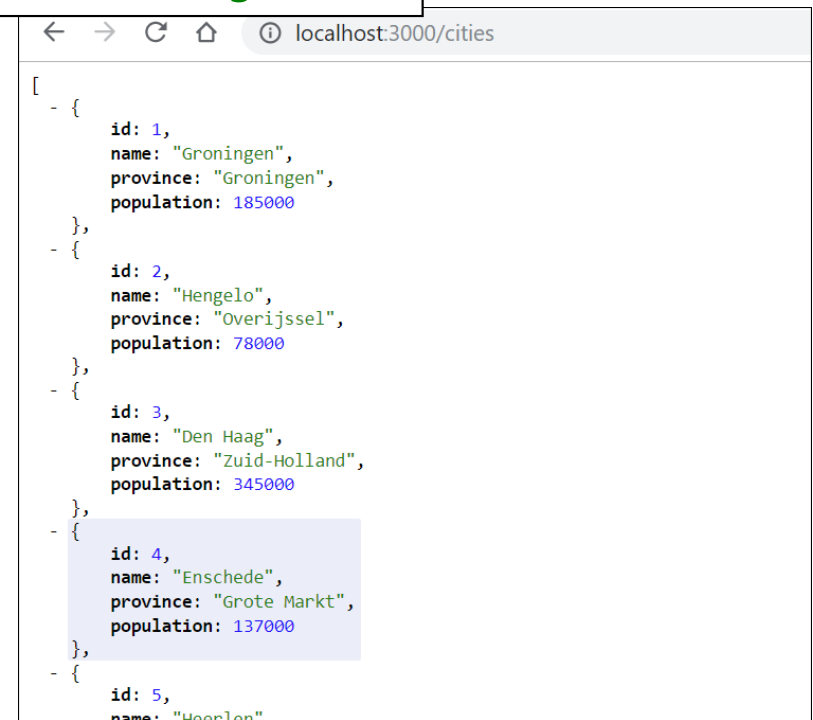


<https://www.npmjs.com/package/json-server>

Add a script to start json-server

- NOT necessary if you talk to a 'real' endpoint.
- But we're using `cities.json` and `json-server` here. So add to `package.json`

```
"json-server": "json-server --watch cities.json"
```



A screenshot of a web browser window showing a JSON array of city data. The browser's address bar displays 'localhost:3000/cities'. The JSON data is as follows:

```
[  
  - {  
    id: 1,  
    name: "Groningen",  
    province: "Groningen",  
    population: 185000  
  },  
  - {  
    id: 2,  
    name: "Hengelo",  
    province: "Overijssel",  
    population: 78000  
  },  
  - {  
    id: 3,  
    name: "Den Haag",  
    province: "Zuid-Holland",  
    population: 345000  
  },  
  - {  
    id: 4,  
    name: "Enschede",  
    province: "Grote Markt",  
    population: 137000  
  },  
  - {  
    id: 5,  
    name: "Haarlem"
```

Add HttpClientModule to application

- Update `app.module.ts` and `city.service.ts`
- Since we're using services, the HTML and Component are unaltered
- Use `HttpClientModule` in Module and Service

```
import {http[Client]Module} from '@angular/common/http';  
...  
  
@NgModule({  
  ...  
  imports      : [  
    HttpClientModule,  
  ],  
  ...  
})
```

Edit city.service.ts



Add `Http` and call API in `loadCities()`.

Upon subscription, dispatch data to the store

```
// Some stuff that our server (json-server) needs:
const BASE_URL = 'http://localhost:3000/cities';
const HEADERS = {
  headers: new HttpHeaders().set('Content-Type', 'application/json')
};

@Injectable({ providedIn: 'root' })
export class CityService {
  constructor(private store: Store<CitiesState>,
              private http: HttpClient) {
    this.loadCities(); // load cities once the service is started
  }

  loadCities() {
    this.http.get(BASE_URL)
      .pipe(
        tap(res => console.log('We talked to json-server and received: ', res))
      )
      .subscribe((response: City[]) => {
        return this.store.dispatch(loadCities({cities: response}));
      });
  }
}
```



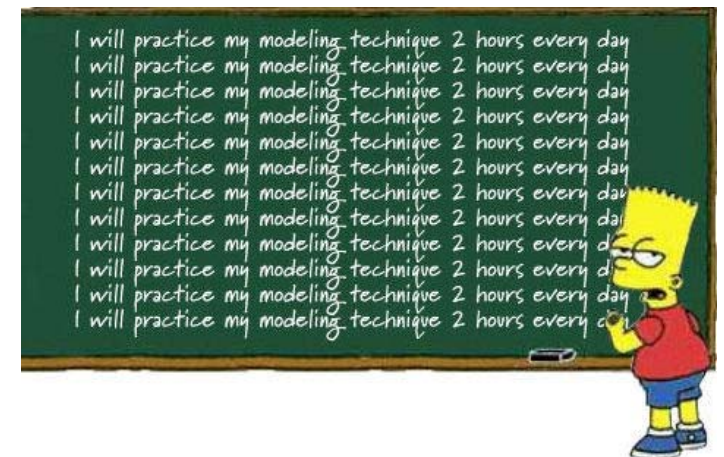
Adding and deleting cities

- Same procedure...

```
removeCity(city: City) {  
  this.http.delete(BASE_URL + `/${city.id}`, HEADERS)  
    .subscribe(() => {  
      console.log('City removed', city);  
      // optimistic delete - assume everything went fine in the backend,  
      this.store.dispatch(removeCity({city}));  
    });  
}  
addCity(city: City){  
  ...  
}
```

Workshop

- Use your own app, add a service and call HTTP to load .json-data
- OR: Start from `../215-ngrx-store-http`
- Make yourself familiar with the store concepts and http-flow. Study the example code.
- Add the `addCity()` method on the service, that adds a city to the .json file via json-server
- Add the `updateCity()` method on the service, to edit an existing city



Next Steps

- [@ngrx/effects](#) - Side Effect model for @ngrx/store to model event sources as actions.
- [@ngrx/router-store](#) - Bindings to connect the Angular Router to @ngrx/store
- [@ngrx/store-devtools](#) - Store instrumentation that enables a powerful time-travelling debugger
- [@ngrx/entity](#) - Entity State adapter for managing record collections.

<https://ngrx.io/docs>