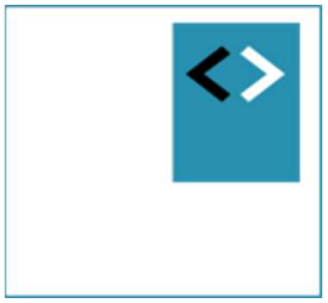
# Reactive Angular with RxJS @ngrx/store – Using Http directly

Peter Kassenaar –
info@kassenaar.com

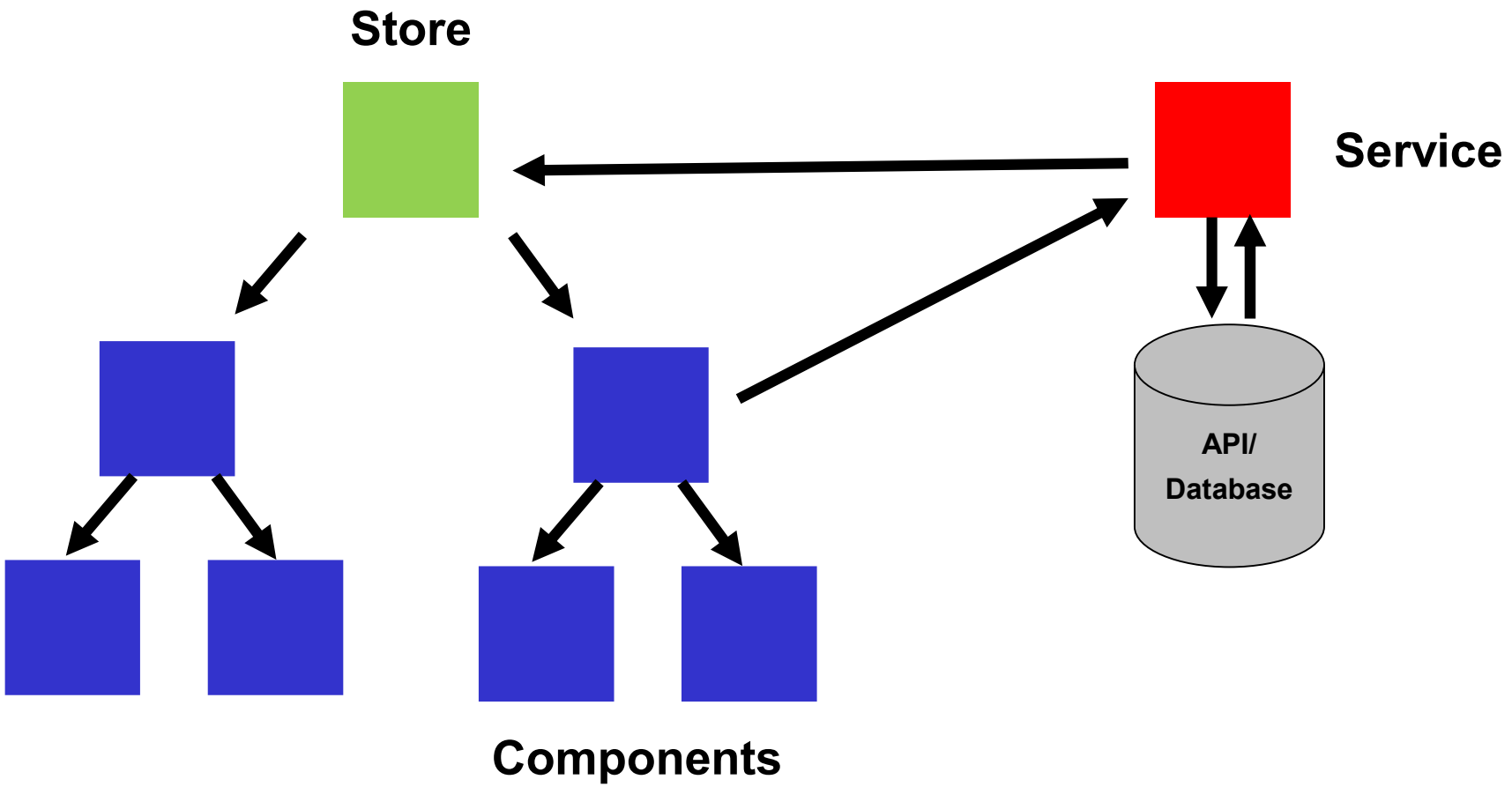


# Using `HttpClient` directly

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

# Architecture

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

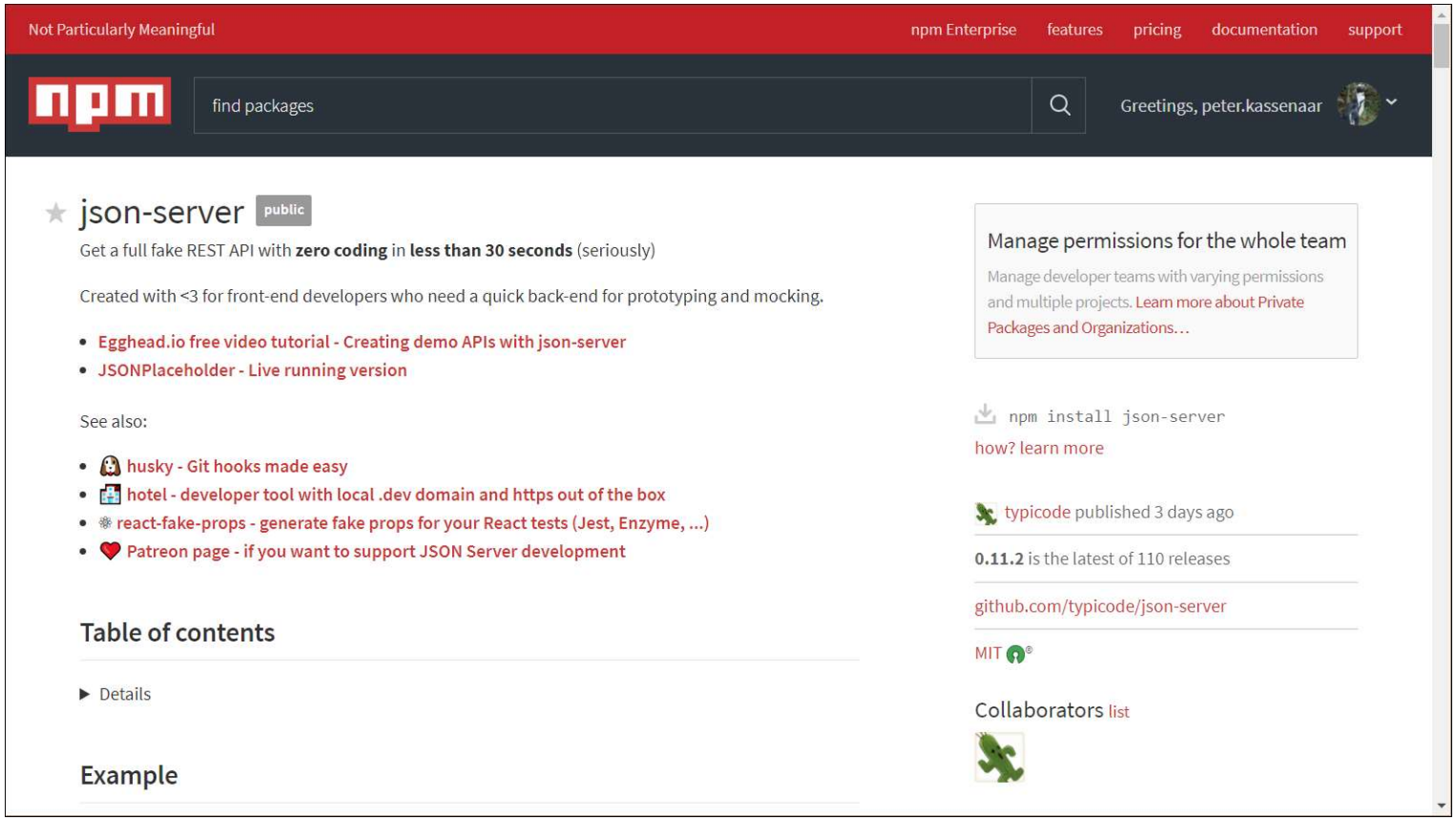


## Actions and Reducers

- 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

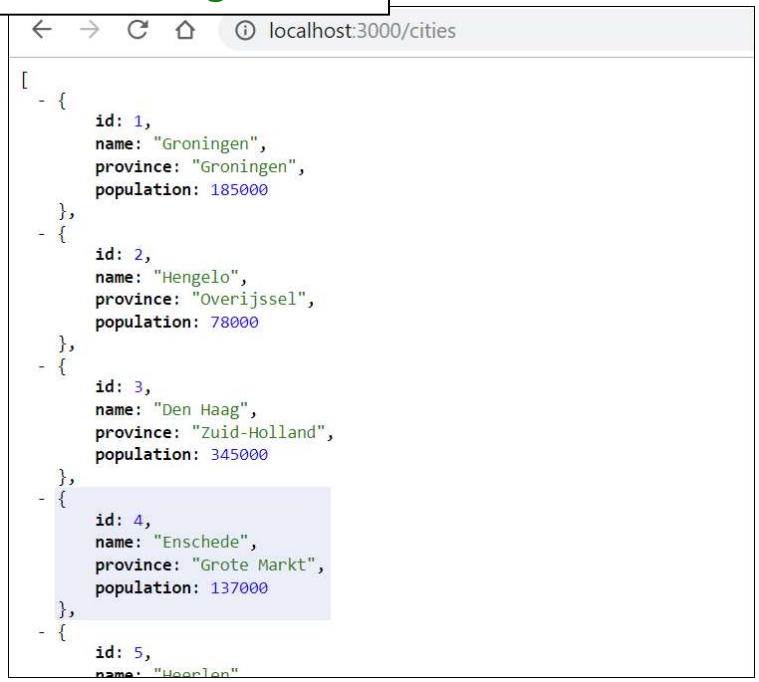



https://www.npmjs.com/package/json-server

# 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"
```



# Add HttpModule 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}));
      });

  }
```

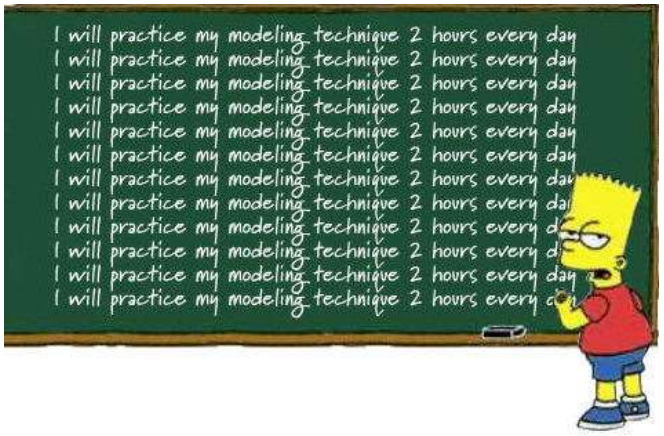
# Adding and deleting cities

- 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