**ANGULAR PROJECT – HOW TO**

Version: 1

Reference: <https://angular.io/tutorial/toh-pt6>

**Progress Checklist:**

Basic input form

Basic table

Mock data displayed in table via property binding

Replace property binding with data fetching service

Implement Observable in Service

Subscribe Observable in data display Component

Setup independent route navigation for input form and table

Make the input form functional so that it populates the table

Implement Reactive form

Do I make the table a form too?

Edit – in table, click edit for item, redirect to form and populate form.

Edit – make changes to item, save and redirect to table page.

Delete Person

Add Form Validation

Implement Error Handling

**Create A New Project**

1. Open Terminal and set desired location:
   1. cd <location>
2. Create new project
   1. ng new <name-of-project>
      1. “Would you like to add Angular routing?”: Yes.
      2. “Which stylesheet format would you like to use?” CSS

**Build A Basic Front End**

1. Open project with VS Code
2. Create data model (class/interface)
3. Create an input form
   1. ng generate component my-form
4. Create a table
   1. ng generate component my-table
5. Add the following to app.component.html
   1. <app-my-form>
   2. <app-my-table>

**Introduce Reactive Form**

1. In app.module.ts, wire-up/Register reactive forms elements
   1. Import ReactiveFormsModule
   2. Register ReactiveFormsModule in imports array.
2. In my-form.component.ts, wire-up desired form with reactive controls
   1. Import FormControl
   2. Create form control instances for each field.
3. In, my-table.component.ts, wire-up table with reactive controls
   1. *Pending*
   2. *Pending*

**Implement Mock Data**

1. In the app folder, create a mock data class.
   1. mock-people.ts.
2. In my-table.components.ts, import mock-people.
3. Define a property for binding of people array.
   1. people = PEOPLE;
4. In my-table.component.html, add a block an ngFor block to spit display data from the mock class.

<li \*ngFor="let person of people">

  <span>{{ person.firstName }} {{ person.lastName }}</span>

</li>

**Implement Service**

1. To create the service, run the following command:
   1. ng generate service hero
2. Import data model
   1. import { Person } from 'src/person';
3. Import mock data
   1. import { PEOPLE } from './mock-people';
4. Add a get method

  getPeople() : Person[]{

    return PEOPLE

  }

1. To set up the get method, go to the component that will display the data – my-table – and import the service.
2. Replace the …
   1. old people definition …
      1. people = PEOPLE;
   2. With the a declaration
      1. people: Person[] = [];
3. Add the service to the constructor as a parameter.
   1. constructor(private personService: PersonService) {}
4. *Now* create the get method in the my-table component. Note that this will not work yet. We’re just setting it up.

  getPeople(): void {

    this.people = this.PersonService.getPeople();

  }

1. In the same my-table.components, call it from the ngOnInit method.

  ngOnInit(): void {

    this.getPeople();

  }

1. You should have successfully replaced the direct implementation of PEOPLE with the service. Check the browser. Data still populates list.

**Implement Observable**

1. In person.service, import Observable
2. Replace the current getPeople method from Implement Service Line 8 above with…

  getPeoples(): Observable<Person[]> {

    const people = of(PEOPLE);

    return people;

  }

* 1. *Note: an error is thrown. Data displays but new mock data does not show up.*

1. Back in my-table.components, rewrite the current getPeople method to include subscription to the observable.

  getPeople(): void {

    this.personService.getPeople().subscribe((people) => (this.people = people));

  }

* 1. *This should not fix the error from Line 2. New mock data will display.*

**Update the Routing Module**

1. Import the MyTableComponent (aka People Component).
2. Add path objects to the routes const

  { path: 'my-form', component: MyFormComponent },

  { path: 'my-table', component: MyTableComponent },

1. In the app.component.html page add the following links and router tag

<nav>

  <a routerLink="/my-form">Person Form</a>

  &nbsp;

  <a routerLink="/my-table">People Table</a>

</nav>

<router-outlet></router-outlet>

1. If desired, add a default path:

 { path: '', redirectTo: '/my-table', pathMatch: 'full' },

**Implement HTTP Services**

1. In app.module.ts, add the following
   1. import { HttpClientModule } from '@angular/common/http';
2. Still in app.module.ts, in the imports array, add HttpClientModule

**Implement In-Memory-Data Service**

1. Install the package
   1. npm install angular-in-memory-web-api –save
2. In the app.module.ts, add the following
   1. import { HttpClientInMemoryWebApiModule } from 'angular-in-memory-web-api';
   2. import { InMemoryDataService } from './in-memory-data.service';
      1. note: this may be error “not found”.
3. In app.module.ts, in the imports array, after HttpClientModule add the following code
   1. HttpClientInMemoryWebApiModule.forRoot( InMemoryDataService, { dataEncapsulation: false } )
4. Generate the in-memory-data.service by running the following command:
   1. ng generate service InMemoryData
      1. *note: the error mentioned above goes away, but, now InMemoryDataService throws an error. “property createDb is missing”*
5. In in-memory-data-service.ts, add the following import
   1. import { InMemoryDbService } from 'angular-in-memory-web-api';
      1. *note: error is still present. To get rid of error keep building IMDS*
   2. Import your data model,
      1. import { Person } from 'src/person';
   3. To the export class, implement InMemoryDbService
      1. export class InMemoryDataService implements InMemoryDbService {…
   4. In the export class body, add createDb method along with an array of data based on your data model and return it.

  createDb() {

    const people = [

      { id: 1, firstName: 'John', lastName: 'Doe' },

      { id: 2, firstName: 'Jane', lastName: 'Doe' },

    ];

    return people;

  }

* + 1. Error is now gone from app.module.ts
  1. To ensure that you always generate an ID add the following code

  genId(people: Person[]): number {

    return people.length > 0

      ? Math.max(...people.map((person) => person.id)) + 1

      : 11;

  }

**Implement HTTP in the Service**

1. In person.service.ts, import the following
   1. import { [HttpClient](https://angular.io/api/common/http/HttpClient), [HttpHeaders](https://angular.io/api/common/http/HttpHeaders) } from '@angular/common/[http](https://angular.io/api/common/http)';
2. Now inject HttpClient into the constructor:
   1. constructor( private [http](https://angular.io/api/common/http): [HttpClient](https://angular.io/api/common/http/HttpClient)) { }
3. Add a url property:
   1. private peopleUrl = 'api/people';
4. Refactor the get method by replacing RxJs style of(PEOPLE) with HTTP style this.http.get…
   1. Old fetched mock data directly

  getPeople(): Observable<Person[]> {

    const people = of(PEOPLE);

    return people;

  }

* 1. New fetches data from server

  getPeople(): Observable<Person[]> {

    return this.http.get<Person[]>(this.peopleUrl)

  }

1. *Now go to* the in-memory-data.service.ts file and update the return to an object. *Note: If not, data will fail to populate. Another thing, the RxJs method works fine with or without the curly braces. But, Http throws an error if the curly braces aren’t in there.*
   1. Old
      1. return people;
   2. new
      1. return { people };
2. Refresh the screen to confirm data still loads.
3. Back in the service file, feel free to remove the PEOPLE import.

**Implement Add/Create Person**

1. In person.service, import

import { HttpClient, HttpHeaders } from '@angular/common/http';

1. Still in person.servce create the following property

  httpOptions = {

    headers: new HttpHeaders({ 'Content-Type': 'application/json' }),

  };

1. Still in person.service, add a method that creates a new person entry

  createPerson(person: Person): Observable<Person> {

    return this.http.post<Person>(this.peopleUrl, person, this.httpOptions);

  }

1. Go to my-form.component.ts, import PersonService
2. Then add the Person service to the constructor parameter

constructor(private fb: FormBuilder, private personService: PersonService) {}

1. Just below the personForm property, add person data model property

  person!: Person;

1. Now add a method that will fire off when the submit button is pushed

  addPerson(): void {

    const p = { ...this.person, ...this.personForm.value };

    this.personService.createPerson(p).subscribe();

  }

* 1. *Note: the subscribe method ensures that the data is added to the current list of people.*

1. Finally, in my-form.component.html, enable the addPerson function

<form (ngSubmit)="addPerson()"

1. Also, ensure that the button type is “submit”

  <button type="submit">Submit</button>

**Reroute to Person List upon Save**

1. Import route library

import { ActivatedRoute, Router } from '@angular/router';

1. Add the router class into the constructor parameter

  constructor(private fb: FormBuilder, **private router: Router**, private personService: PersonService) {}

1. Add the code as the last line of the addPerson method. Note: this can be moved if you want.

  this.router.navigate(['/my-table']);

**Enable Person Update/Edit – Get Person – Main technique**

I have two ways of getting a person. I think I’ll document the simpler one first and then update if I have to.

1. In app-routing.module, in the routes array, add a path to enable getting a single existing person entry to the form

{ path: 'my-form/:id', component: MyFormComponent }

1. Go to person.service, create a method to get person.

  getPerson(id: number): Observable<Person> {

    const url = `${this.peopleUrl}/${id}`;

    return this.http.get<Person>(url));

  }

* 1. *Optional: adding the following pipe to the end of the http.get() will allow you to write the fetched data to the console log.*

      .pipe(

        tap((data) => console.log('getPersonService: ' + JSON.stringify(data)))

1. Go to my-form.component.ts and create a method to fetch a person

  getPerson(): void {

    const id = parseInt(this.route.snapshot.paramMap.get('id')!, 10);

    this.personService

      .getPerson(id)

      .subscribe({ next: (person: Person) => this.displayPerson(person) });

  }

* 1. *Note that the method gets the id, subscribes and calls to a method to display the data.*

1. Create a method to display the data

  displayPerson(person: Person): void {

    if (this.personForm) {

      this.personForm.reset();

    }

    this.person = person;

    if (this.person.id === 0) {

      this.pageTitle = 'Add Person';

    } else {

      this.pageTitle = `Edit Person: ${this.person.firstName}`;

    }

    this.personForm.patchValue({

      firstName: this.person.firstName,

      lastName: this.person.lastName,

    });

  }

1. In the ngOnInit method, call getPerson. This will run as soon as the page loads.

this.getPerson();

1. Still in my-form.component.ts, add the following property that will allow you to display conditional data from the displayPerson method.

  pageTitle = 'My Form';

1. Go to my-form.component.html and add the following tag. This will enable you to see what mode the form is in: Add or Edit.

<p>{{ pageTitle }}</p>

**Enable Person Update/Edit – Get Person – Altered technique**

This only slightly alters the above technique by pulling the code that gets the person ID out of the getPerson method and tossing it in the ngOnInit.

1. Add the following property

  private sub!: Subscription;

1. Inside ngOnInit add the following block to read the person Id from the route parameter.

    this.sub = this.route.paramMap.subscribe((params) => {

      const id = Number(this.route.snapshot.paramMap.get('id'));

      this.getPerson(id);

    });

1. No in the getPerson method add pass in the ID as a parameter and remove the line that reads the person ID from the route.

  getPerson(id: number): void {

    this.personService

      .getPerson(id)

      .subscribe({ next: (person: Person) => this.displayPerson(person) });

  }

**Enable Person Update/Edit – Continued – Update Person**

1. In person.ts, make sure to add type “any” to the id property.

id: number | any;

* 1. *Note: this is so that the service can take a null value for person.id.*

1. In app.component.html, make sure the routerLink contains a default number to pass in as the person id

<a [routerLink]="['/my-form', '0']">Person Form</a>

1. In person.service, add the following method to initialize the form.

  private initializePerson(): Person {

    return {

      id: 0,

      firstName: '',

      lastName: '',

    };

  }

1. Still in person.service, add a conditional based on person.id to either initialized the form or get the person.id.

  getPerson(id: number): Observable<Person> {

**if (id === 0) {**

**return of(this.initializePerson());**

**}**

    const url = `${this.peopleUrl}/${id}`;

    return this.http

      .get<Person>(url)

      .pipe(

        tap((data) => console.log('getPersonService: ' + JSON.stringify(data)))

      );

  }

1. Go to my-form.component and create a new method that will replace the existing one being called by the submit button. It will check the person.id and flow to the appropriate Create or Update methods.

  savePerson(): void {

    const p = { ...this.person, ...this.personForm.value };

    if (p.id === 0) {

      this.personService.createPerson(p).subscribe();

      this.router.navigate(['/my-table']);

    } else {

      this.personService.updatePerson(p).subscribe();

      this.router.navigate(['/my-table']);

    }

  }

1. Go back to person.service, add a line of code to set id to null

  createPerson(person: Person): Observable<Person> {

**person.id = null;**

    return this.http.post<Person>(this.peopleUrl, person, this.httpOptions);

  }

* 1. *Note: this ensure that data model’s id is set to receive a number. Otherwise a new the id will be NaN.*

1. In person.service, create an updatePerson method

  updatePerson(person: Person): Observable<Person> {

    return this.http.put<Person>(this.peopleUrl, person, this.httpOptions);

  }

1. Finally go to my-form.component.html and change the ngSubmit call to savePerson()

<form (ngSubmit)="savePerson()" [formGroup]="personForm">

**Implement Delete via Form – Basic Setup**

1. In person.service, add the following method

  deletePerson(id: number): Observable<{}> {

    const url = `${this.peopleUrl}/${id}`;

    return this.http.delete<Person>(url, this.httpOptions);

  }

* 1. Note: you can use the following for the url, still works. Looks messy though.
     1. const url = this.peopleUrl + '/' + id;

1. In my-form.component.ts, add the following method

  deletePerson(): void {

    this.personService.deletePerson(this.person.id).subscribe();

    this.onSaveComplete();

  }

1. Finally, in my-form.component.html add a delete button

  <button type="button" (click)="deletePerson()">Delete</button>

1. *Note: I added an onSaveComplete method that contains the reroute to table page.*

  onSaveComplete(): void {

    this.router.navigate(['/my-table']);

  }

**Implement Delete via Form – Add checks**

1. In my-form.components.ts, add the two IF statements will ensure that delete will only work when ID is present and user will be prompted before deleting.

  deletePerson(): void {

    if (this.person.id === 0) {

      this.onSaveComplete();

    } else if (this.person.id) {

      if (

        confirm(

          `Are you sure you want to delete: ${this.person.firstName} ${this.person.lastName}?`

        )

      )

        this.personService.deletePerson(this.person.id).subscribe();

      this.onSaveComplete();

    }

  }