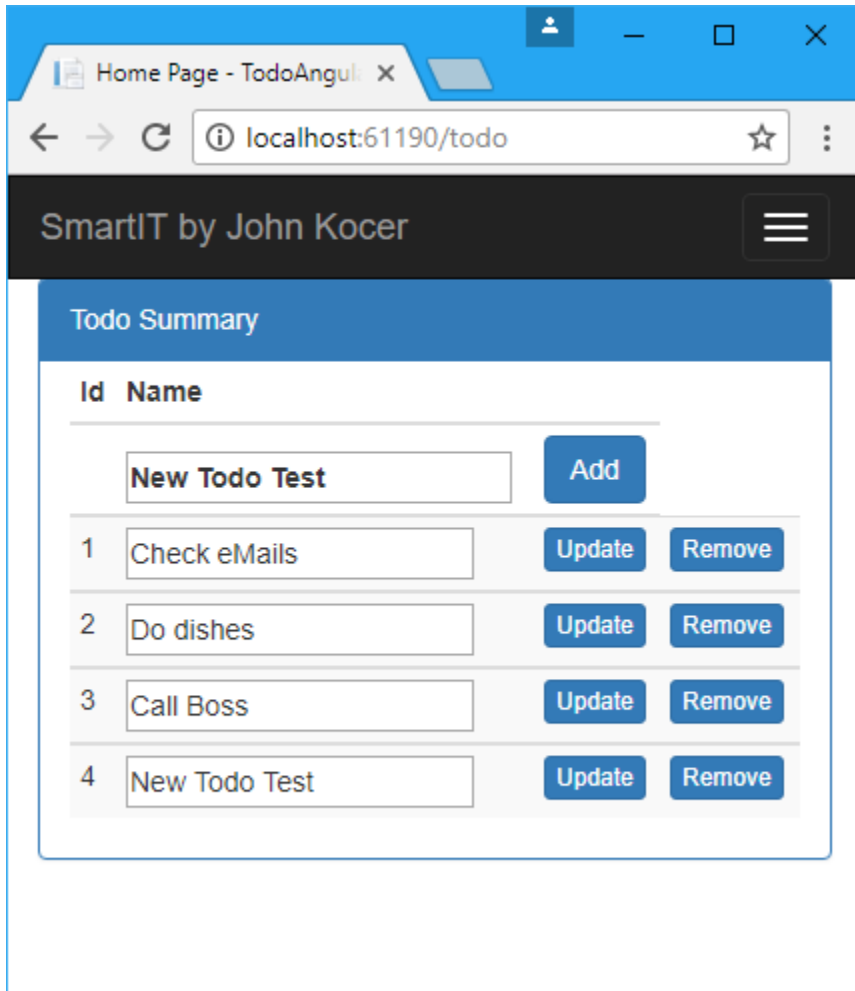


How to Create To-do CRUD operation with ASP.NET MVC Core, Angular 4.0

Visual Studio 2017, ASP.NET Core 2.0

By John kocer- SmartIT



Introduction

In this session, we will learn

- How to consume Todo in-memory database using TodoRepository.
- How to create custom ASP.NET custom API controller with CRUD operations.
- List of Objects
- Create a new insert item
- Update an Item
- Delete an Item
- Write HttpPost Create API method.
- Write HttpPut Edit API method.
- Write HttpPost Delete API method.
- Loose Coupling
- Dependency Injection

- Singleton
- Route
- Responsive Web Design

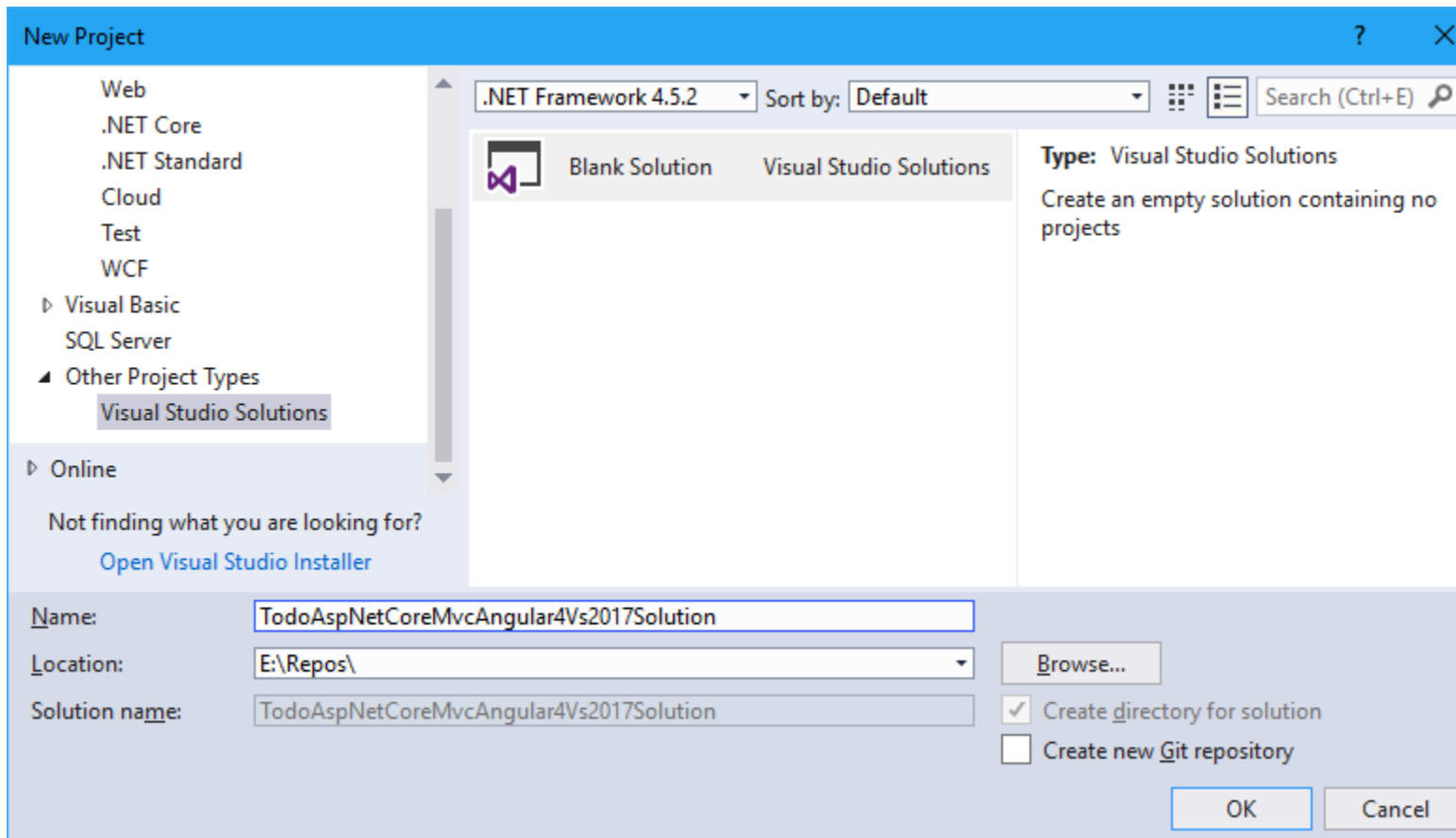
- A lab exercise for you to demonstrate what have you learned from this training material to create your own Employee CRUD operation using EmployeeRepository included in this training material.

Pre-Requirements

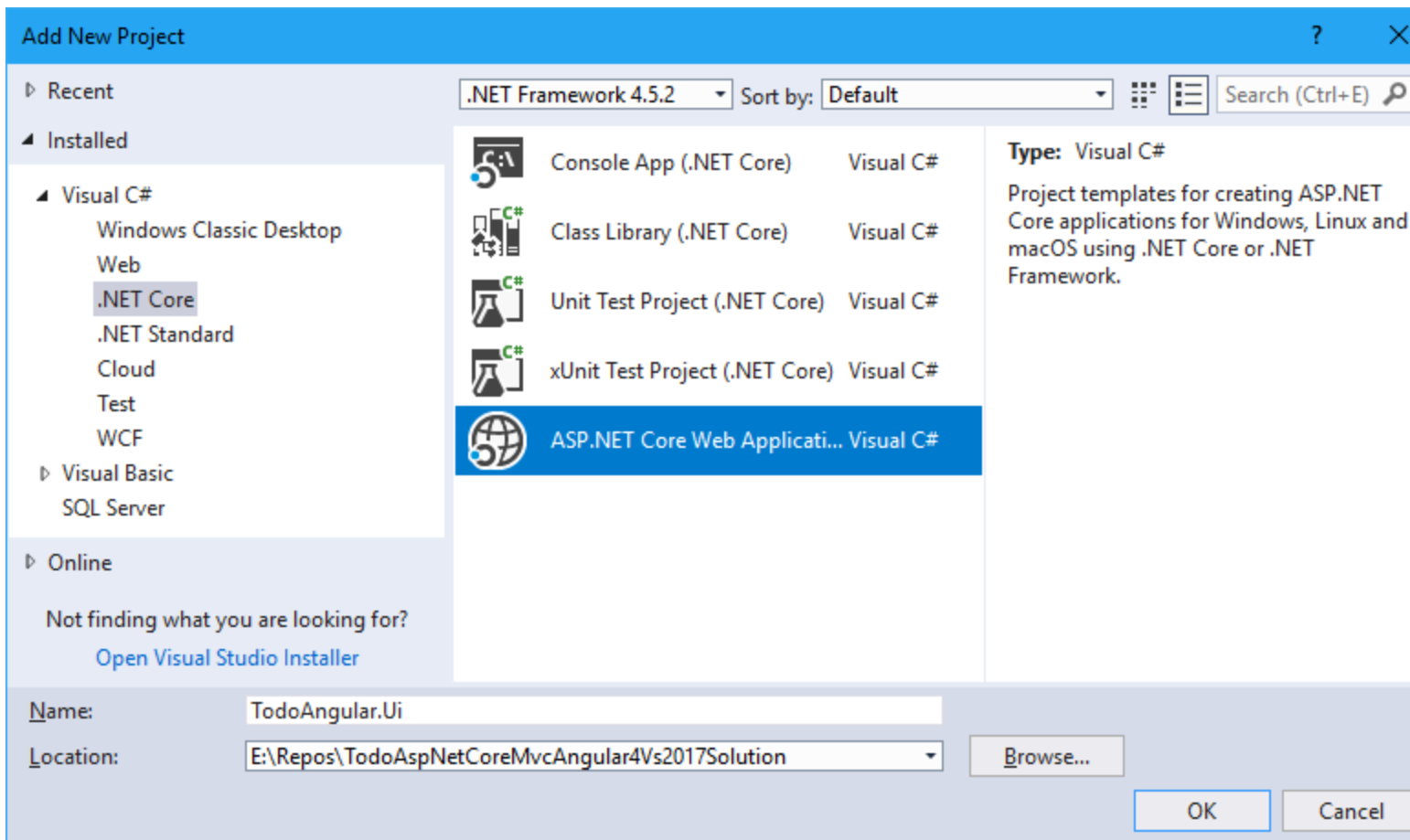
In order to be able to run the example from the download or build it from scratch, you need to have the following tools:

- Visual Studio 2017 latest version
- .NET Core 2.0 or above
- TypeScript 1.8 or above
- Node.js
- Angular 4.0 or above

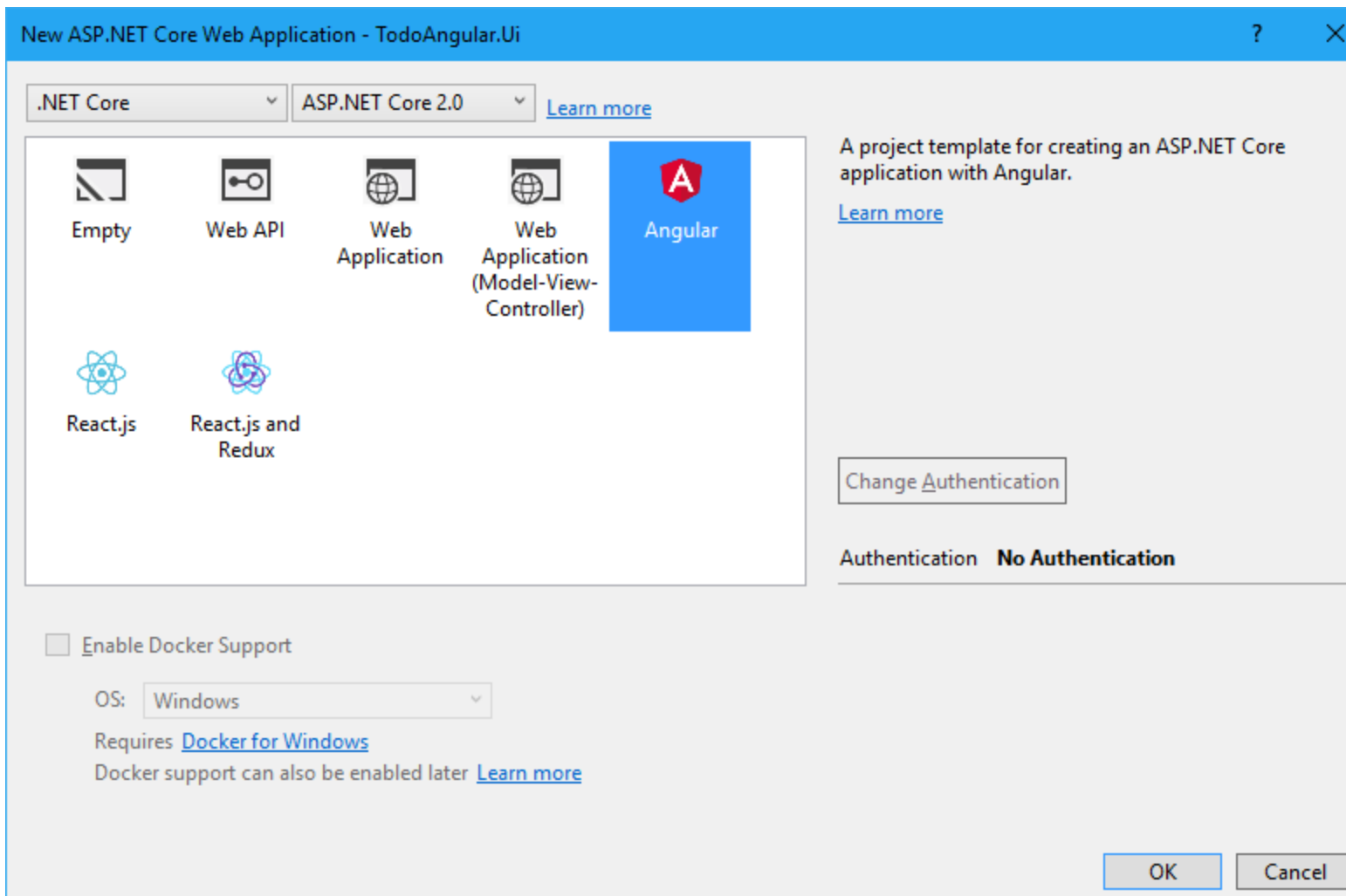
-Create a new solution and name it `TodoAspNetCoreMvcAngular4Vs2017Solution`



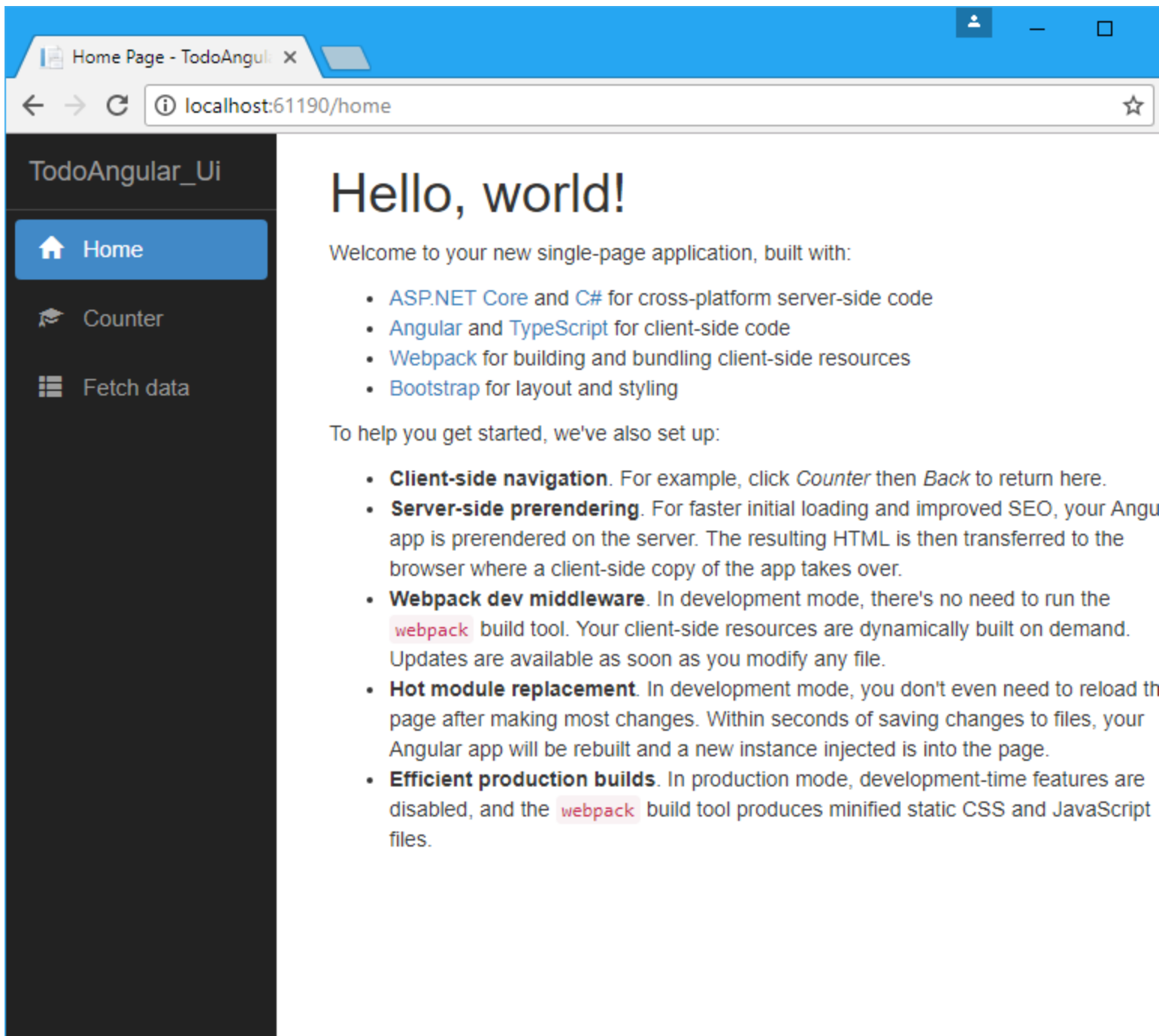
-Add a new ASP.NET Core Web Application project and name it **TodoAngular.Ui**



- Visual Studio 2017 ASP.NET Core 2.0 comes with an Angular project template.
- Next screen select Angular Web Application project template.

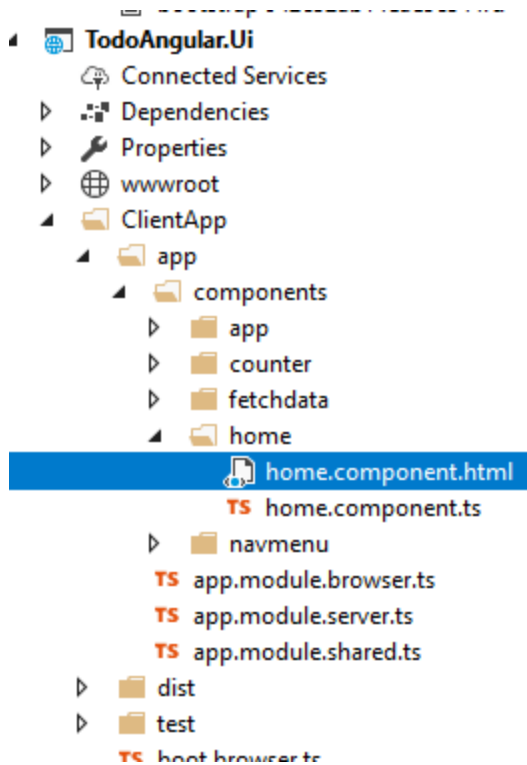


-Compile and run the application and we'll see the home page.



-Lets change to home page content

-Go to the [home.component.html](#) in the ClientApp directory.



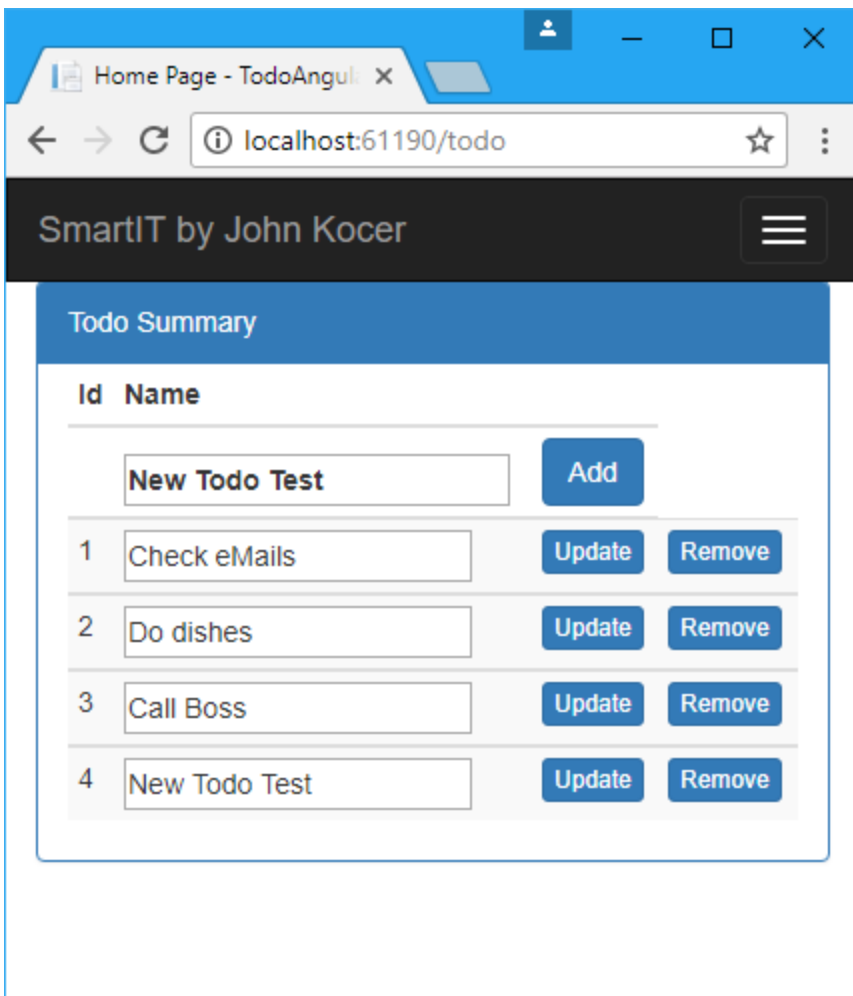
-Replace the home content with below lines when the project is still running. Angular has change detection functionally and reflects your changes on the running application.

-Home content will change on the run time.



```
<h1>To-do Angular 4.0 .NETCore CRUD Application </h1>  
<p>Provided by -SmartIT, John Kocer!</p>
```

We want to create below single page application with CRUD operation, Add, Update and Remove functionality with in-memory TodoRepository Database.



-We need to do below tasks

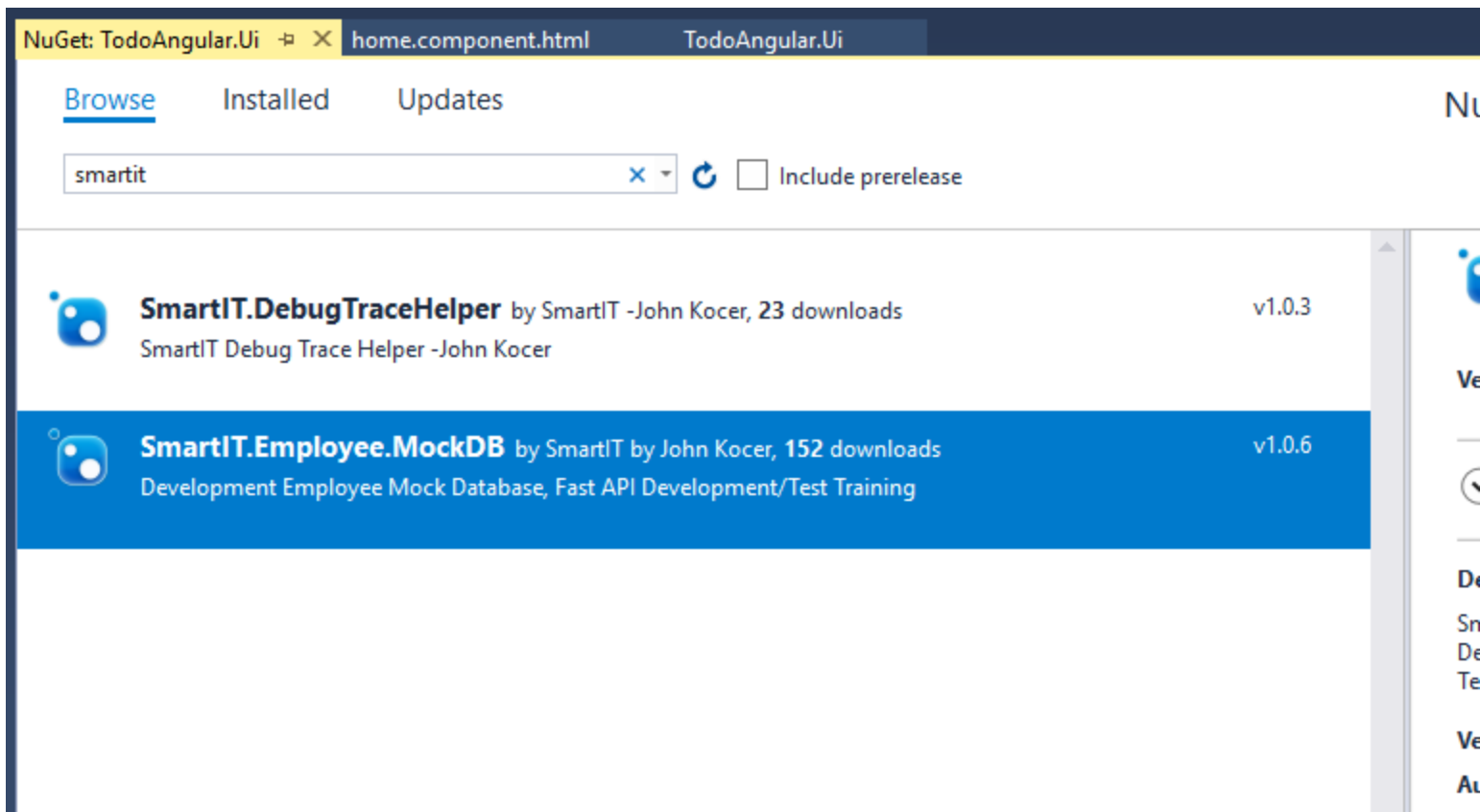
- 1- Set TodoRepository
- 2- Create TodoController
- 3-Add Dependency Injection
- 4- Create todoService.ts
- 5-Create todo.component.html
- 6-Create todo.component.ts
- 7-Add new created component into AppModule
- 8-Update Angular routing to go Todo page.

1- Set TodoRepository

-Use SmartIT.Employee.MockDB which has TodoRepository in it with in-memory MOCK database which has dependency injection interfaces implemented . You may go to below web site and read the usage examples.

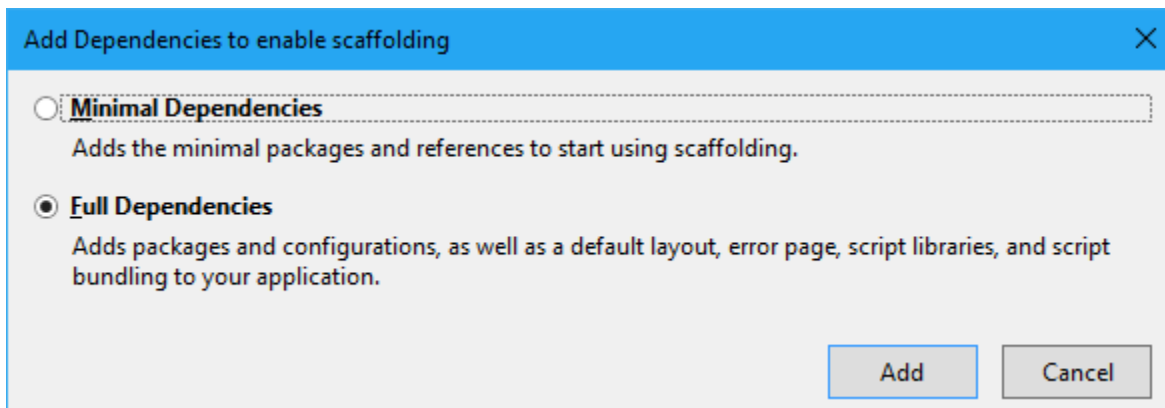
<https://www.nuget.org/packages/SmartIT.Employee.MockDB/>

-Use Nuget Package manager to Install SmartIT.Employee.MockDB from nugget.org.



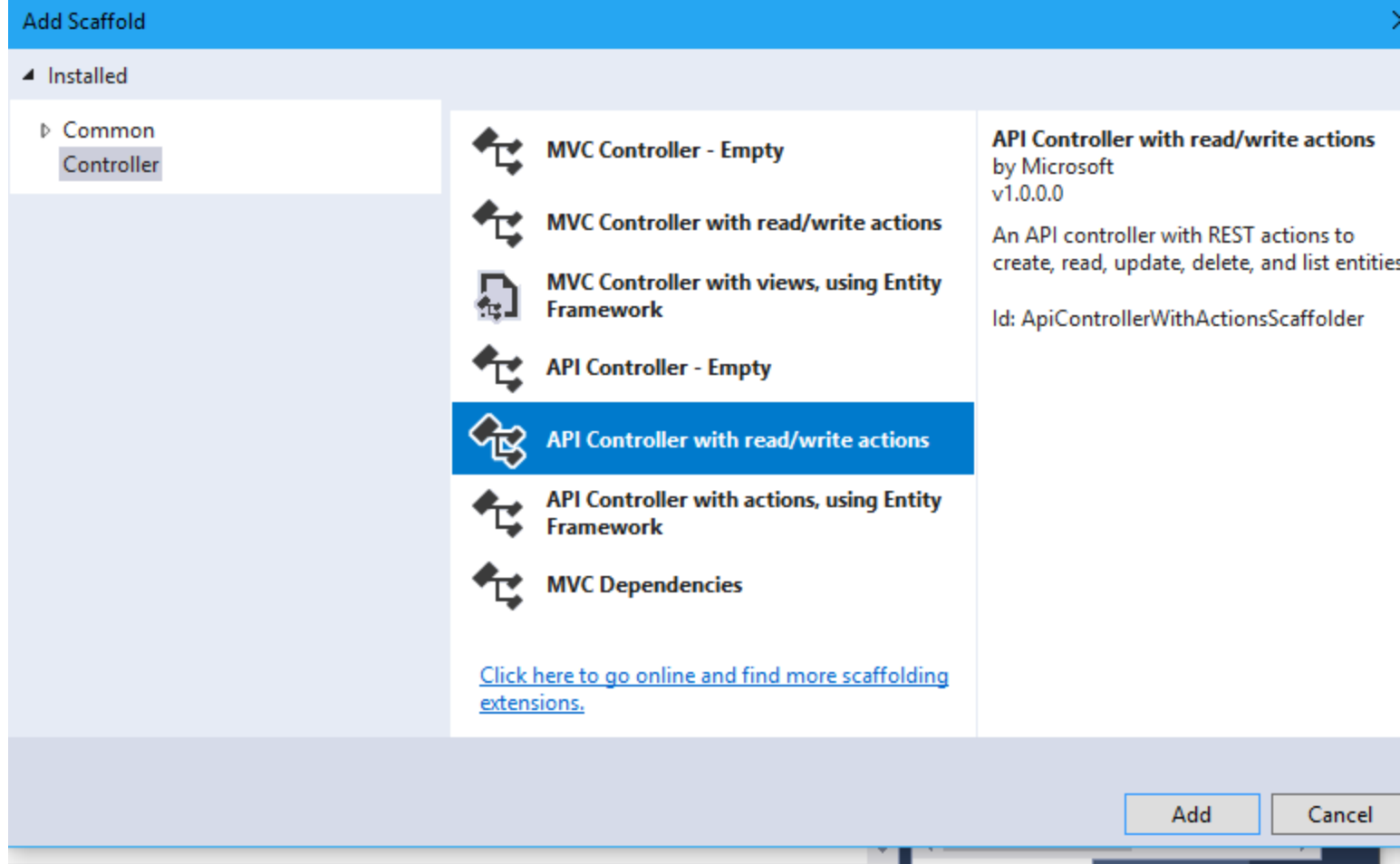
2- Create TodoController

-On the controllers directory add a new controller select Full Dependencies and Add.

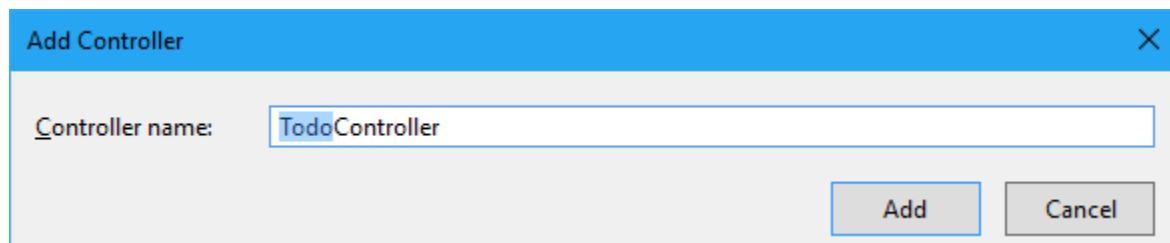


- Select API Controller with read/write actions from the list and click Add

nv)



-Name it **TodoController** and Click Add.



-TodoController will be created.

```

namespace TodoAngular.Ui.Controllers
{
    [Produces("application/json")]
    [Route("api/ToDo")]
    public class TodoController : Controller
    {
        // GET: api/ToDo
        [HttpGet]
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET: api/ToDo/5
        [HttpGet("{id}", Name = "Get")]
        public string Get(int id)
        {
            return "value";
        }

        // POST: api/ToDo
        [HttpPost]
        public void Post([FromBody]string value)
        {
        }

        // PUT: api/ToDo/5
        [HttpPut("{id}")]
        public void Put(int id, [FromBody]string value)
        {
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public void Delete(int id)
        {
        }
    }
}

```

3-Add Dependency Injection

-Go to the Startup page ConfigureServices method

```

public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
    }
}

```

-Add the SmartIT.Employee.MockDB using statement top of the Startup page.

```
using SmartIT.Employee.MockDB;
```

-Add the AddSingleton service. Why we chose as Singleton? Because our service is an In-memory repository data services. We want only a single instance will be created.

```
services.AddSingleton<ITodoRepository, TodoRepository>(); //Add the services here
```

```
using SmartIT.Employee.MockDB;

namespace TodoAngular_Ui
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc();
            services.AddSingleton<ITodoRepository, TodoRepository>(); //Add the services here
        }
    }
}
```

-Add top of the TodoController page

```
using SmartIT.Employee.MockDB;
```

-Add `private readonly ITodoRepository _todoRepository;` readonly ItodoRepositor Interface member.

-Add a Todo Constructor

```
using SmartIT.Employee.MockDB;

namespace TodoAngular.Ui.Controllers
{
    [Produces("application/json")]
    [Route("api/Todo")]
    public class TodoController : Controller
    {
        private readonly ITodoRepository _todoRepository;
        public TodoController(ITodoRepository todoRepository)
        {
            _todoRepository = todoRepository;
        }
    }
}
```

-Add GetAllTodos Httpget mehot with `[Route("~/api/GetAllTodos")]`

-Here is the final changes in below on the TodoController

```

using SmartIT.Employee.MockDB;

namespace TodoAngular.Ui.Controllers
{
    [Produces("application/json")]
    [Route("api/ToDo")]
    public class TodoController : Controller
    {
        private readonly ITodoRepository _todoRepository;
        public TodoController(ITodoRepository todoRepository)
        {
            _todoRepository = todoRepository;
        }
        // GET: api/ToDo
        [Route("~/api/GetAllTodos")]
        [HttpGet]
        public IEnumerable<ToDo> GetAllTodos()
        {
            return _todoRepository.GetAll();
        }
    }
}

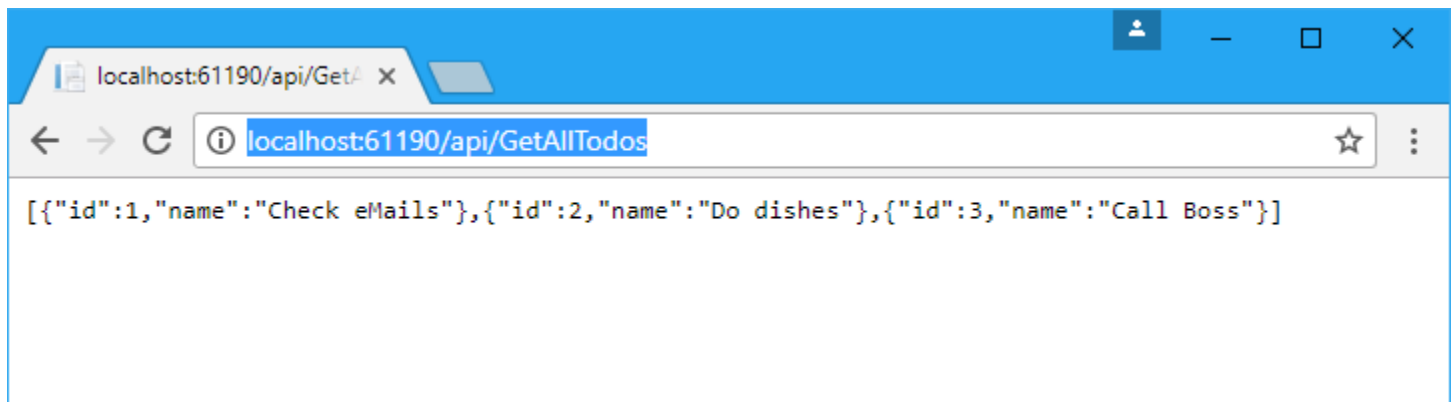
```

-Compile and run the application.

-Test get all todos with calling to API

<http://localhost:61190/api/GetAllTodos>

Note: Your port number may be different than ours, use your local port number.



-Update the remainder of the TodoController like below.

```

using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using SmartIT.Employee.MockDB;

namespace TodoAngular.Ui.Controllers
{
    [Produces("application/json")]
    [Route("api/ToDo")]
    public class TodoController : Controller
    {
        private readonly ITodoRepository _todoRepository;
        public TodoController(ITodoRepository todoRepository)
        {
            _todoRepository = todoRepository;
        }

        [Route("~/api/GetAllTodos")]
        [HttpGet]
        public IEnumerable<Todo> GetAllTodos()
        {
            return _todoRepository.GetAll();
        }

        [Route("~/api/AddToDo")]
        [HttpPost]
        public Todo AddToDo([FromBody]Todo item)
        {
            return _todoRepository.Add(item);
        }

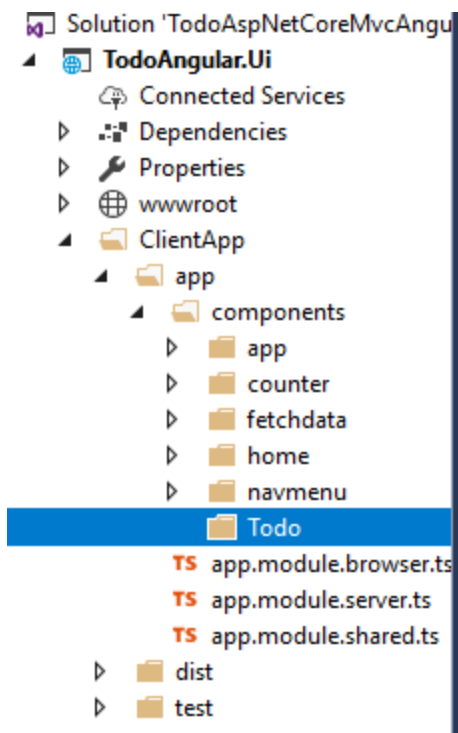
        [Route("~/api/UpdateToDo")]
        [HttpPut]
        public Todo UpdateToDo([FromBody]Todo item)
        {
            return _todoRepository.Update(item);
        }

        [Route("~/api/DeleteToDo/{id}")]
        [HttpDelete]
        public void Delete(int id)
        {
            var findTodo = _todoRepository.FindById(id);
            if (findTodo != null)
                _todoRepository.Delete(findTodo);
        }
    }
}

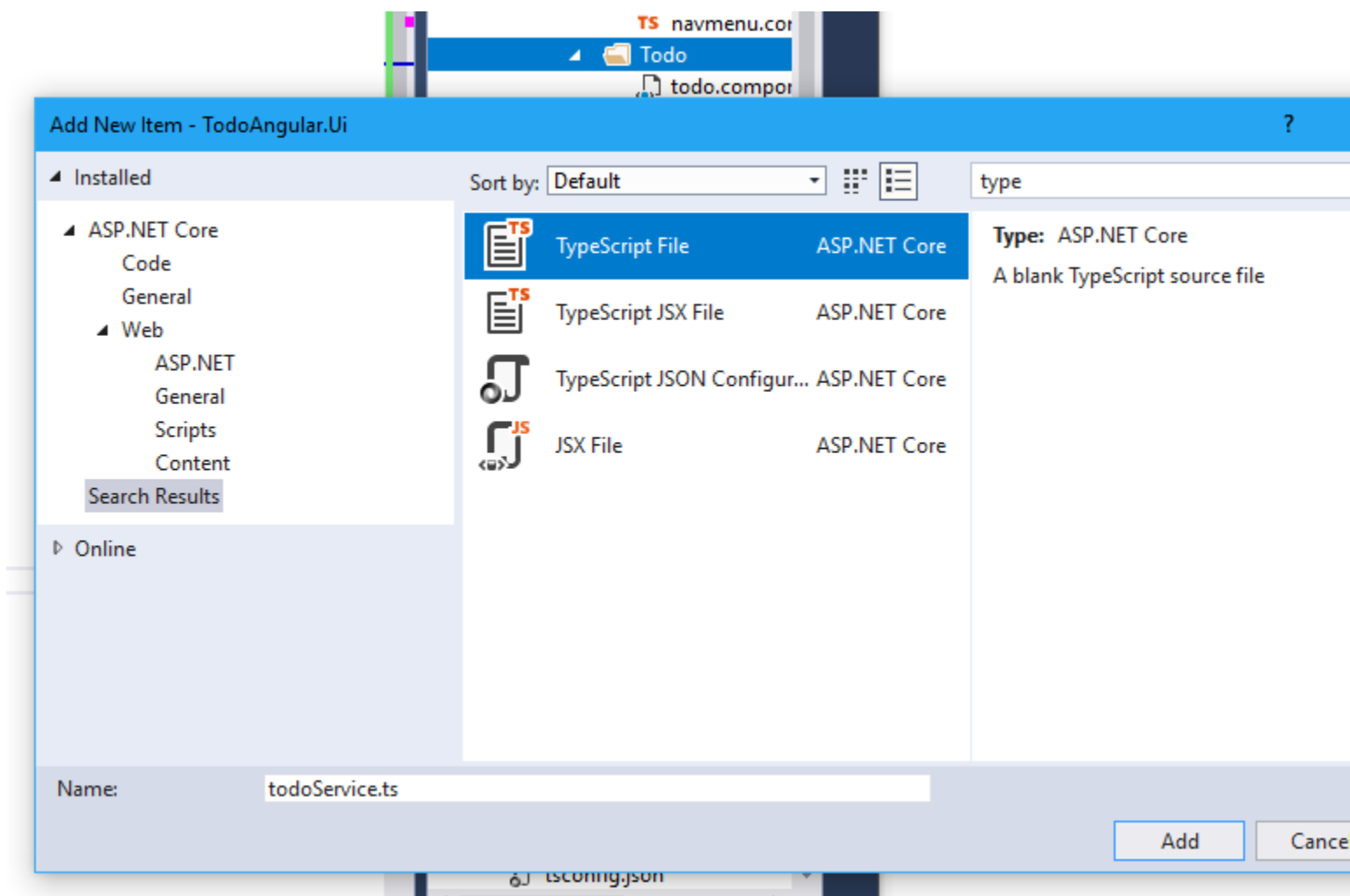
```

4- Create todoService.ts

-Add a Todo folder under components directory.



-Add a TypeScript file todoService.ts into Todo directory.



-Paste the below code into your `todoService.ts`


```

import { Injectable } from "@angular/core";
import { Http, Response, Headers } from "@angular/http";
import "rxjs/add/operator/map";
import 'rxjs/add/operator/do'; // debug
import { Observable } from "rxjs/Observable";
import { BehaviorSubject } from 'rxjs/BehaviorSubject';

@Injectable()
export class TodoService {

  public todoList: Observable<Todo[]>;
  private _todoList: BehaviorSubject<Todo[]>;
  private baseUrl: string;
  private dataStore: {
    todoList: Todo[];
  };

  constructor(private http: Http) {
    this.baseUrl = '/api/';
    this.dataStore = { todoList: [] };
    this._todoList = <BehaviorSubject<Todo[]>>new BehaviorSubject([]);
    this.todoList = this._todoList.asObservable();
  }

  getAll() {
    this.http.get(`${this.baseUrl}GetAllTodos`)
      .map(response => response.json())
      .subscribe(data => {
        this.dataStore.todoList = data;
        this._todoList.next(Object.assign({}, this.dataStore).todoList);
      }, error => console.log('Could not load todo.'));
  }

  public addTodo(newTodo: Todo) {
    console.log("addTodo");
    var headers = new Headers();
    headers.append('Content-Type', 'application/json; charset=utf-8');
    console.log('add todo : ' + JSON.stringify(newTodo));

    this.http.post(`${this.baseUrl}AddTodo/`, JSON.stringify(newTodo), { headers: headers })
      .map(response => response.json()).subscribe(data => {
        this.dataStore.todoList.push(data);
        this._todoList.next(Object.assign({}, this.dataStore).todoList);
      }, error => console.log('Could not create todo.'));
  };

  public updateTodo(newTodo: Todo) {
    console.log("updateTodo");
    var headers = new Headers();
    headers.append('Content-Type', 'application/json; charset=utf-8');
    console.log('add todo : ' + JSON.stringify(newTodo));

    this.http.put(`${this.baseUrl}UpdateTodo/`, JSON.stringify(newTodo), { headers: headers })
      .map(response => response.json()).subscribe(data => {
        this.dataStore.todoList.forEach((t, i) => {
          if (t.id === data.id) { this.dataStore.todoList[i] = data; }
        });
      }, error => console.log('Could not update todo.'));
  };

  removeItem(todoId: number) {
    var headers = new Headers();
    headers.append('Content-Type', 'application/json; charset=utf-8');
  }
}

```

```

    console.log("removeItem:" + todoId);
    this.http.delete(`${this.baseUrl}DeleteTodo/${todoId}`, { headers: headers }).subscribe(response
=> {
    this.dataStore.todoList.forEach((t, i) => {
        if (t.id === todoId) { this.dataStore.todoList.splice(i, 1); }
    });

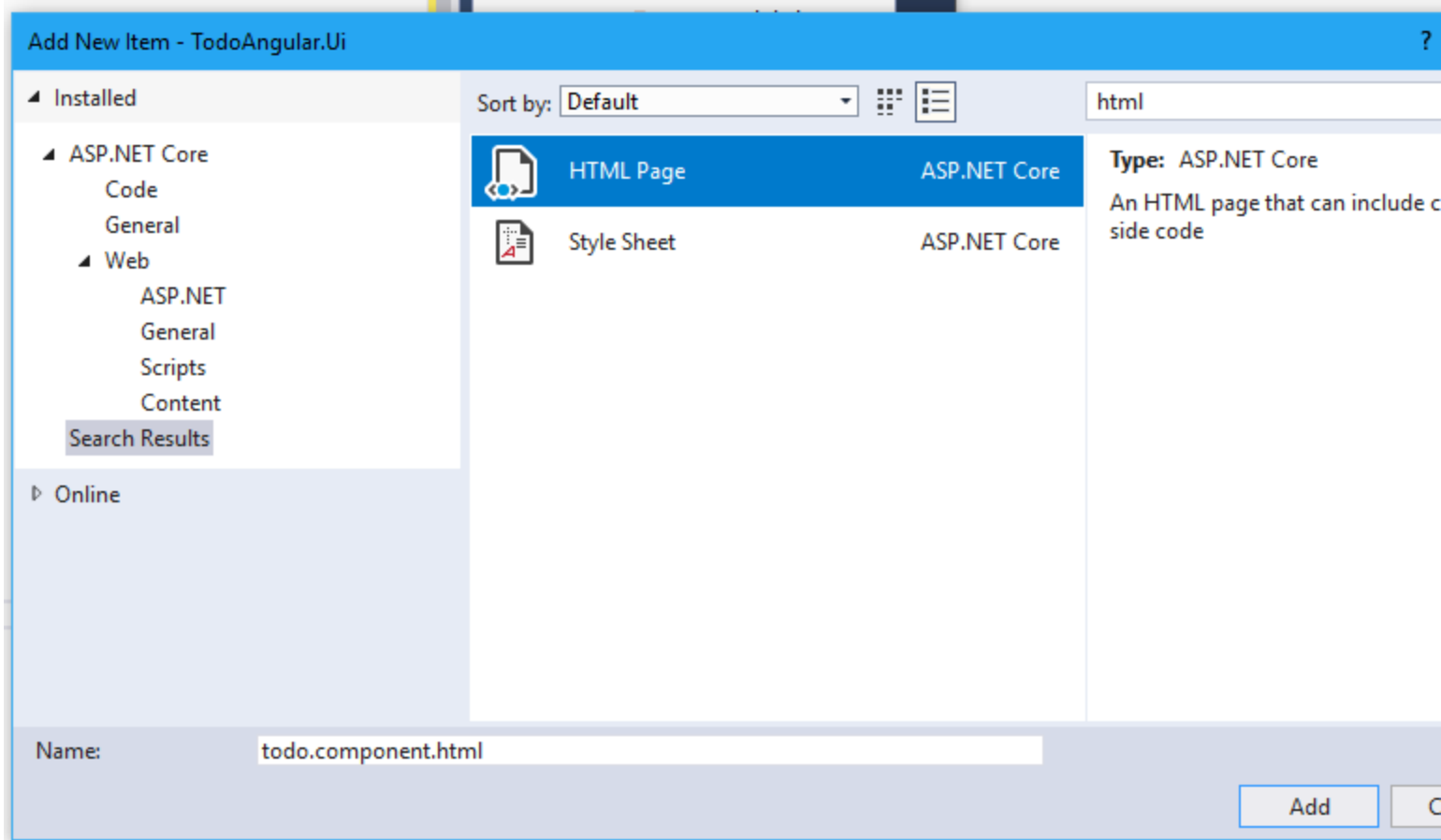
    this._todoList.next(Object.assign({}, this.dataStore).todoList);
}, error => console.log('Could not delete todo.));
}
private _serverError(err: any) {
    console.log('sever errorOK:', err); // debug
    if (err instanceof Response) {
        return Observable.throw(err.json().error || 'backend server error');
        // if you're using lite-server, use the following line
        // instead of the line above:
        //return Observable.throw(err.text() || 'backend server error');
    }
    return Observable.throw(err || 'backend server error');
}
}

export class Todo {
    public id: number;
    public name: string;
}

```

5-Create todo.component.html

-Add an HTML page into Todo directory and name it **todo.component.html**



-Paste the below code in your **todo.component.html**

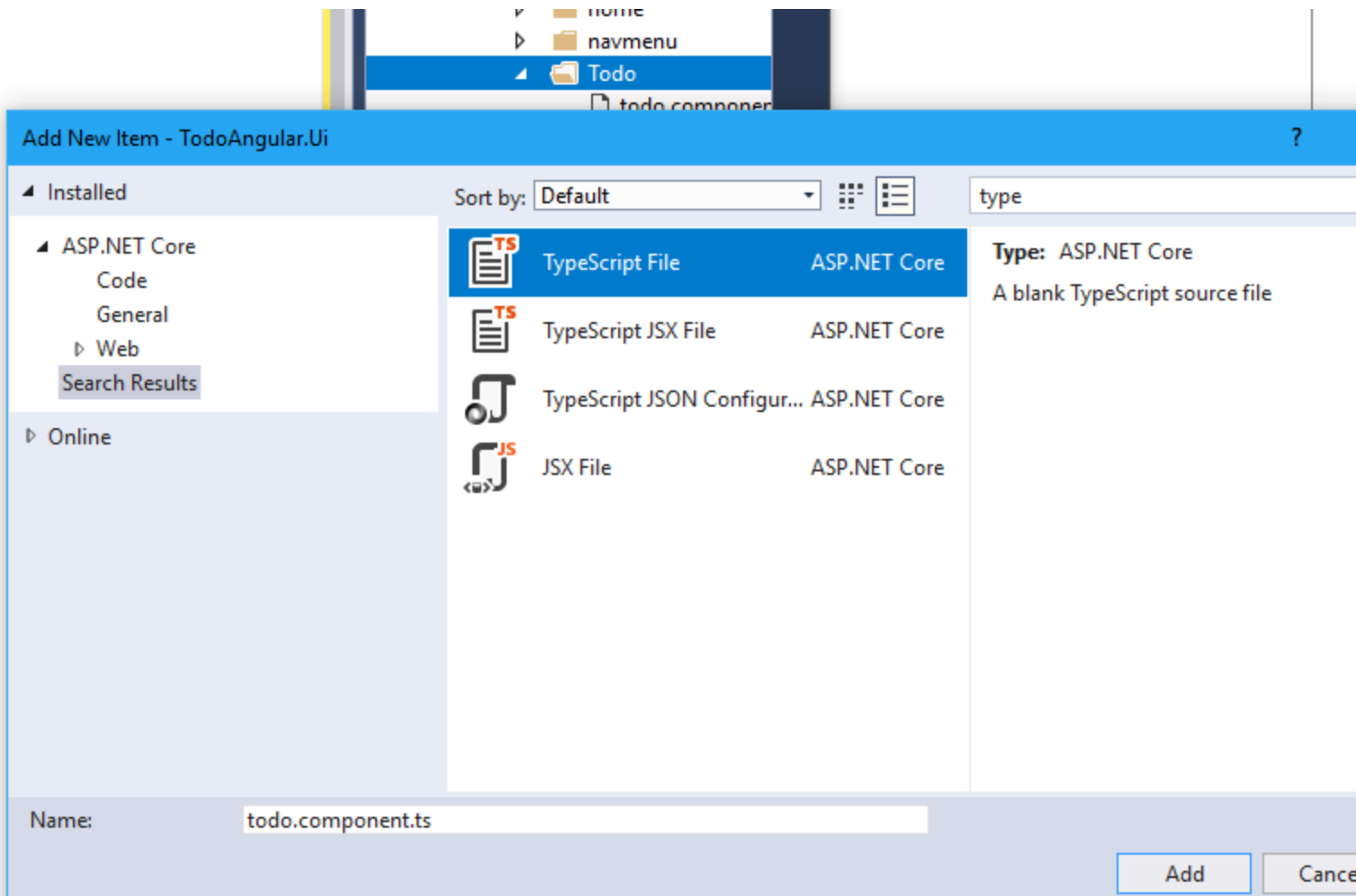
```

<div id="summary" class="section panel panel-primary">
  <div class="panel-heading"> Todo Summary</div>
  <div class="container">
    <!--{{todoList | json}}-->
    <table class="table table-striped table-condensed" >
      <thead>
        <tr>
          <th>Id</th>
          <th>Name</th>
          <th></th>
        </tr>
        <tr>
          <th> </th>
          <th>
            <input id="newName" [(ngModel)]='newTodo.name' type="text" placeholder="newName" />
          </th>
          <th>
            <button class="btn btn-primary" (click)="addTodo(newTodo)">Add </button>
          </th>
        </tr>
      </thead>
      <tbody *ngFor="let item of todoList | async">
        <tr>
          <td>{{item.id}} </td>
          <td><input type="text" [(ngModel)]='item.name' /></td>
          <td>
            <button class="btn btn-xs btn-primary" (click)="updateTodo(item)">Update</button>
          </td>
          <td>
            <button class="btn btn-xs btn-primary" (click)="deleteTodo(item.id)">Remove</button>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</div>

```

6-Create todo.component.ts

-Add TypeScript file called **to.component.ts** into Todo directory.



-Paste below code into your **to.component.ts** .

```

import { Component, OnInit } from '@angular/core';
import 'rxjs/add/operator/catch';
import { TodoService } from "../todoService";
import { Observable } from "rxjs/Observable";
import { Todo } from "../todoService"

@Component({
  selector: 'todo',
  providers: [TodoService],
  templateUrl: './todo.component.html'
})
export class TodoComponent implements OnInit {
  public todoList: Observable<Todo[]>;

  showEditor = true;
  myName: string;
  newTodo: Todo;
  constructor(private dataService: TodoService) {
    this.newTodo = new Todo();
  }
  // if you want to debug info just uncomment the console.log lines.
  ngOnInit() {
    // console.log("in ngOnInit");
    this.todoList = this.dataService.todoList;
    this.dataService.getAll();
  }
  public addTodo(item: Todo) {
    //console.dir(item);
    let todoId = this.dataService.addTodo(this.newTodo);
  }
  public updateTodo(item: Todo) {
    // console.dir(item);
    //console.log("In updateTodo: " + item);
    this.dataService.updateTodo(item);
    // console.log("in updateTodo:" );
  }
  public deleteTodo(todoId: number) {
    this.dataService.removeItem(todoId);
  }
}

```

7-Add new created component into AppModule

```

import { CounterComponent } from './components/counter/counter.component';
import { TodoComponent } from './components/todo/todo.component';

@NgModule({
  declarations: [
    AppComponent,
    NavMenuComponent,
    CounterComponent,
    FetchDataComponent,
    HomeComponent,
    TodoComponent
  ],
  imports: [
    CommonModule,
    HttpClientModule,
    FormsModule,
    RouterModule.forRoot([
      { path: '', redirectTo: 'home', pathMatch: 'full' },
      { path: 'todo', component: TodoComponent },
      { path: 'home', component: HomeComponent },
      { path: 'counter', component: CounterComponent },
      { path: 'fetch-data', component: FetchDataComponent },
      { path: '**', redirectTo: 'home' }
    ])
  ]
})
export class AppModuleShared {

```

-Lets update the menu

-Change this line from `TodoAngular_Ui`

To `SmartIT by John Kocer`

-Add new menu Item Todo

```

<li [routerLinkActive]="['link-active']">
  <a [routerLink]="['/todo']">
    <span class='glyphicon glyphicon-user'></span> Todo
  </a>
</li>

```

-Here is updated menu

```

<div class='main-nav'>
  <div class='navbar navbar-inverse'>
    <div class='navbar-header'>
      <button type='button' class='navbar-toggle' data-toggle='collapse' data-target='.navbar-collapse'>
        <span class='sr-only'>Toggle navigation</span>
        <span class='icon-bar'></span>
        <span class='icon-bar'></span>
        <span class='icon-bar'></span>
      </button>
      <a class='navbar-brand' [routerLink]="['/home']">SmartIT by John Kocer</a>
    </div>
    <div class='clearfix'></div>
    <div class='navbar-collapse collapse'>
      <ul class='nav navbar-nav'>
        <li [routerLinkActive]="['link-active']">
          <a [routerLink]="['/todo']">
            <span class='glyphicon glyphicon-user'></span> Todo
          </a>
        </li>

        <li [routerLinkActive]="['link-active']">
          <a [routerLink]="['/home']">
            <span class='glyphicon glyphicon-home'></span> Home
          </a>
        </li>
        <li [routerLinkActive]="['link-active']">
          <a [routerLink]="['/counter']">
            <span class='glyphicon glyphicon-education'></span> Counter
          </a>
        </li>
        <li [routerLinkActive]="['link-active']">
          <a [routerLink]="['/fetch-data']">
            <span class='glyphicon glyphicon-th-list'></span> Fetch data
          </a>
        </li>
      </ul>
    </div>
  </div>
</div>

```

-Update the RouteModule

```

RouterModule.forRoot([
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'todo', component: TodoComponent },
  { path: 'home', component: HomeComponent },
  { path: 'counter', component: CounterComponent },
  { path: 'fetch-data', component: FetchDataComponent },
  { path: '**', redirectTo: 'home' }
])

```

-Here is the updated RouteModule


```

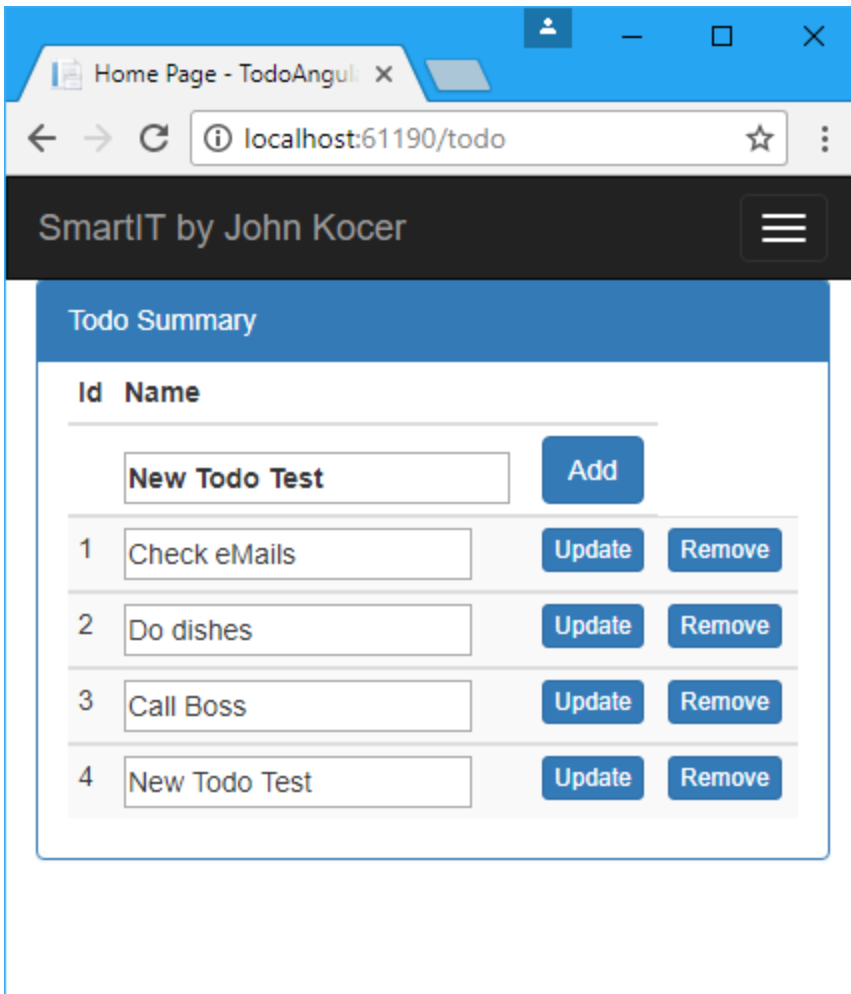
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';
import { RouterModule } from '@angular/router';

import { AppComponent } from './components/app/app.component';
import { NavMenuComponent } from './components/navmenu/navmenu.component';
import { HomeComponent } from './components/home/home.component';
import { FetchDataComponent } from './components/fetchdata/fetchdata.component';
import { CounterComponent } from './components/counter/counter.component';
import { TodoComponent } from './components/todo/todo.component';

@NgModule({
  declarations: [
    AppComponent,
    NavMenuComponent,
    CounterComponent,
    FetchDataComponent,
    HomeComponent,
    TodoComponent
  ],
  imports: [
    CommonModule,
    HttpClientModule,
    FormsModule,
    RouterModule.forRoot([
      { path: '', redirectTo: 'todo', pathMatch: 'full' },
      { path: 'todo', component: TodoComponent },
      { path: 'home', component: HomeComponent },
      { path: 'counter', component: CounterComponent },
      { path: 'fetch-data', component: FetchDataComponent },
      { path: '**', redirectTo: 'todo' }
    ])
  ]
})
export class AppModuleShared {
}

```

-Here is the final result of responsive web design result.



Summary

In this article, we will learned

- How to consume Todo in-memory database using TodoRepository.
- How to create custom ASP.NET MVC custom controller with CRUD operations.
- List of Objects
- Create a new insert object
- Update an item
- Delete an Item
- Write HttpPost Create API method.
- Write HttpPut Edit API method.
- Write HttpPost Delete API method.
- Loose Coupling
- Dependency Injection
- Singleton
- Route
- Responsive Web Design

Lab Exercise

A lab exercise for you to demonstrate what have you learned from this training material to create your own Employee Angular CRUD Responsive Design SPA application. The EmployeeRepository included in this training material.

-You can follow above steps to create your own Employee CRUD ASP.NET MVC Angular 4.0 application.

```
//Use below Employee repository to created your CRUD operation
EmployeeRepository _employeeRepository= new EmployeeRepository();
_employeeRepository.Add(new Employee() { Name = "Mat Stone", Gender = "Male", DepartmentId = 2,
Salary = 8000 });
_employeeRepository.CDump("Employees");
// DebugHelper.cs(29): Employees
//{
"Items":[{"Id":1,"Name":"Mike","Gender":"Male","Salary":8000,"DepartmentId":1,"Department":null},
//{"Id":2,"Name":"Adam","Gender":"Male","Salary":5000,"DepartmentId":1,"Department":null},
//{"Id":3,"Name":"Jacky","Gender":"Female","Salary":9000,"DepartmentId":1,"Department":null},
//{"Id":4,"Name":"Mat
Stone","Gender":"Male","Salary":8000,"DepartmentId":2,"Department":null}], "Count":4}
```

For ASP.NET MVC Core Angular 4 CRUD Application

Download color pdf version of this article with pictures.

<https://github.com/SmartITAz/ToDoAspNetCoreMvcAngular4Vs2017Solution/blob/master/ToDoAspNetCoreMvcAngular4Vs2017.pdf>

NOTE: If you need to copy and paste the code download the pdf file locally.

Download source code from GitHub: <https://github.com/SmartITAz/ToDoAspNetCoreMvcAngular4Vs2017Solution>