Building Reusable Angular Services: Configuration Management

CREATE A CLASS TO HOLD GLOBAL SETTINGS

Paul D. Sheriff

BUSINESS SOLUTIONS ARCHITECT, FAIRWAY TECHNOLOGIES, INC.

www.fairwaytech.com psheriff@fairwaytech.com

Goals

Create service for global settings

Retrieve and store settings in...

- Class
- JSON file
- Local storage
- SQL Server via Web API

Why configuration management?

- Avoid hard-coding values
- Developer creates defaults
- User can change default values
- Use these defaults in your application

I assume you...

- Are an Angular developer
- Are familiar with
  - Angular
  - TypeScript
  - Web API

Related Pluralsight Courses

Angular: Getting Started

Angular: Forms

Play by Play: Angular RxJS

Angular: Fundamentals

The Modules in This Course

Modules

Create a Class to Hold Global Settings

- Class to hold settings
- Return instance of class from Config service

Read Settings from a JSON File

- Create JSON file
- Settings match global settings class
- Read file and return from Config service

Modules

Store Settings in Local Storage

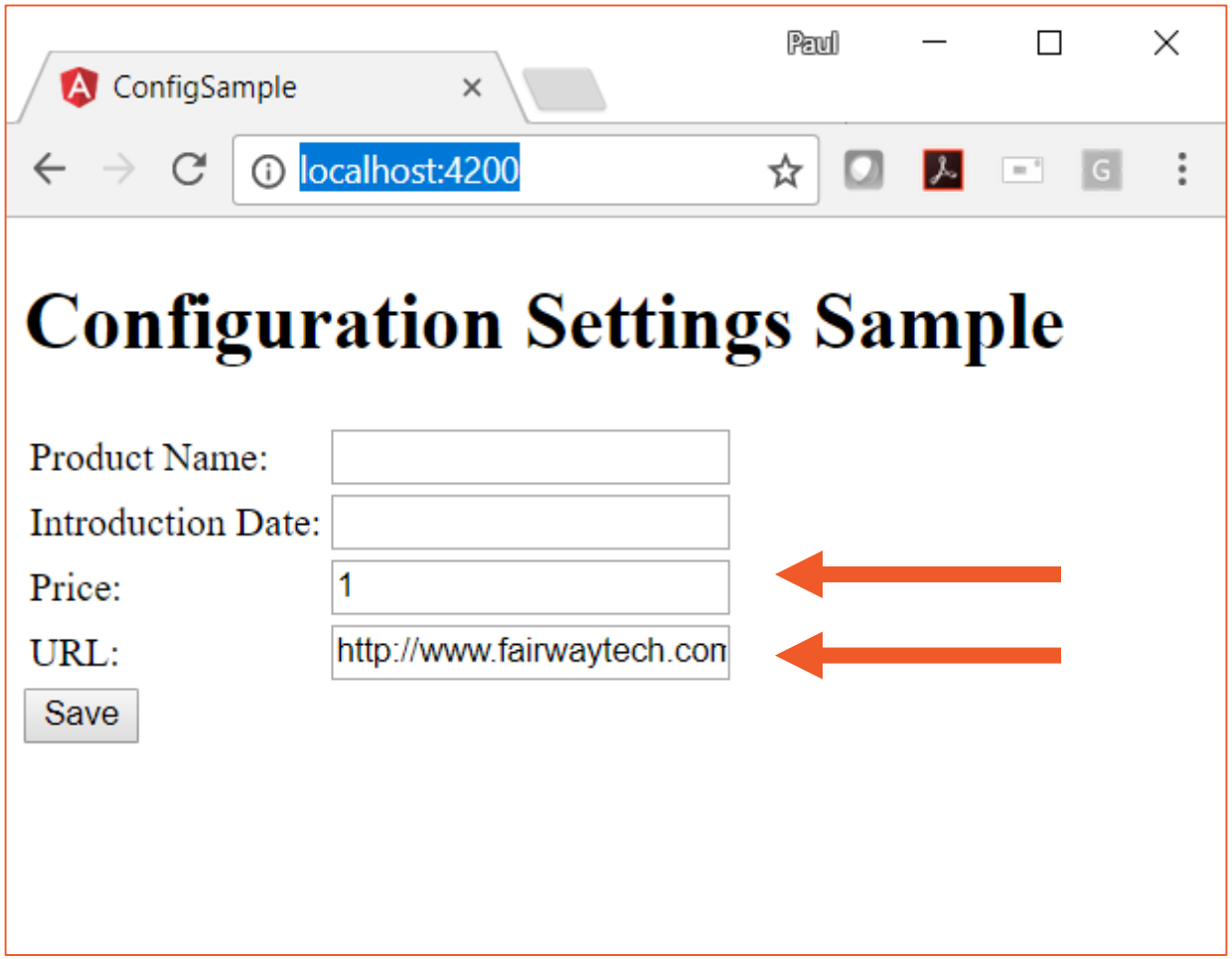
- Store settings into local storage
- Allow user to modify these settings
- Delete settings to return to defaults

Retrieve Settings from SQL Server via a Web API Call

- Build SQL Server table
- Create Web API
- Get settings from SQL Server via Web API

Configuration Management Architecture

Sample Page


Paul
ConfigSample
localhost:4200
Configuration Settings Sample
Product Name:
Introduction Date:
Price:
1
URL:
http://www.fairwaytech.com
Save

Create Configuration Service

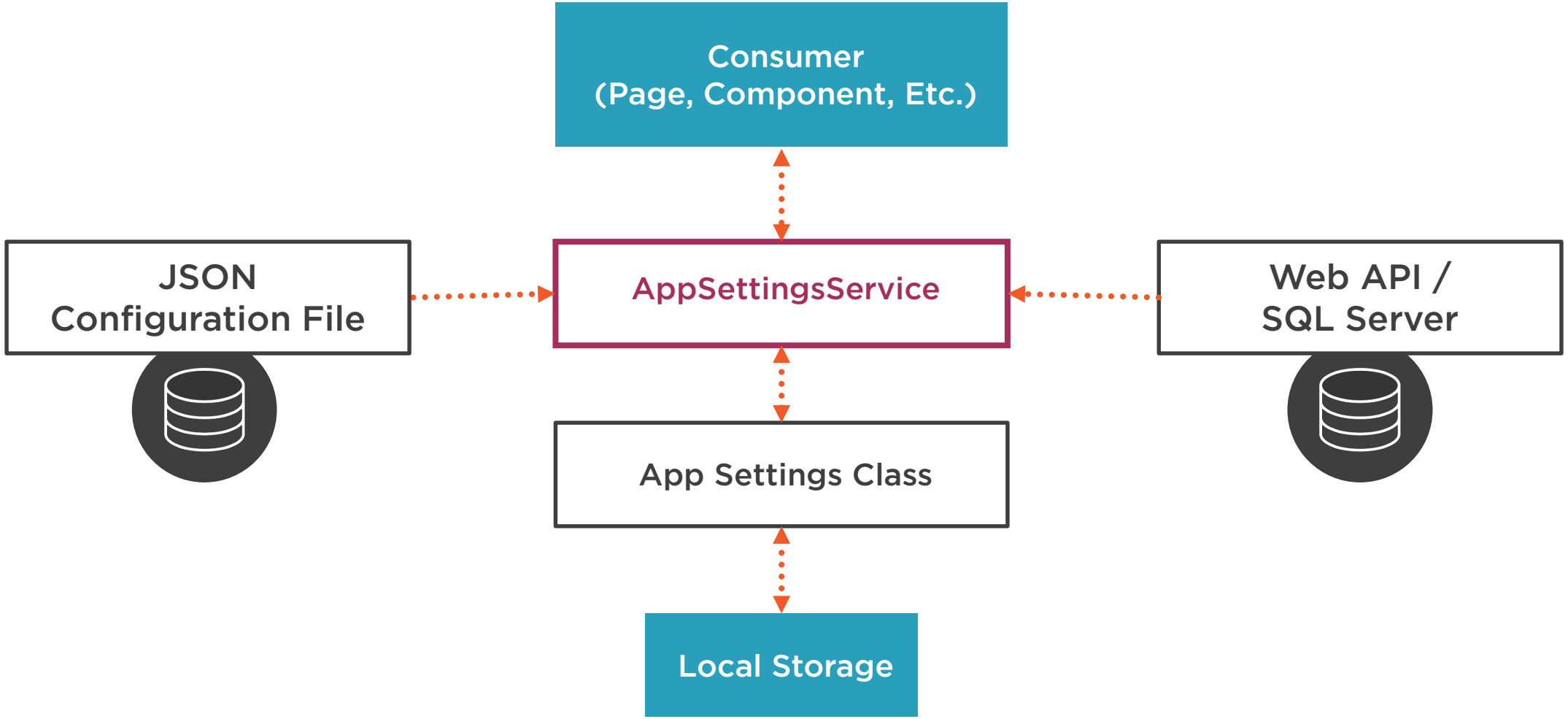
Angular is all about services

Create AppSettingsService class

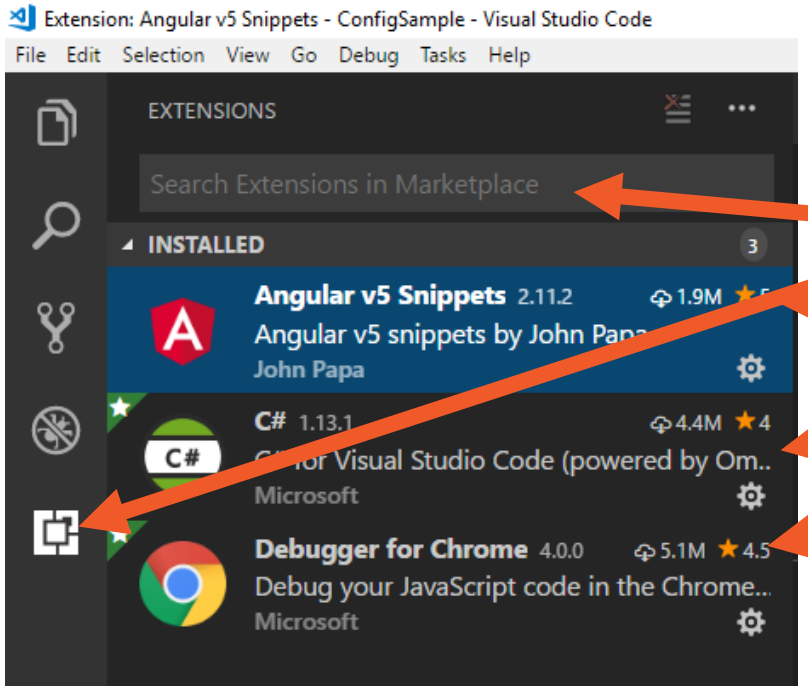
Injectable into any component

Offers great flexibility

Configuration Management Architecture



Tools for This Course



Visual Studio Code

Install Extensions

- Angular v5 Snippets
- C#
- Debugger for Chrome

SQL Server

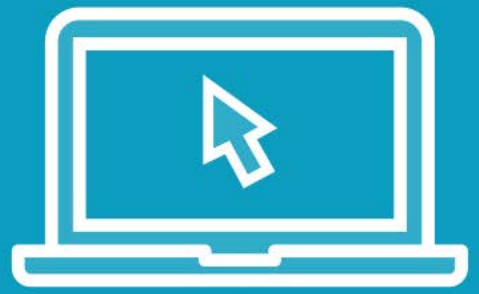
AppSettings Class

```
export class AppSettings {
  defaultPrice: number = 1;
  defaultUrl: string = "http://www.fairwaytech.com";
}
```

App Settings Service

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import { of } from 'rxjs/observable/of';

@Injectable()
export class AppSettingsService {
  getSettings(): Observable<AppSettings> {
    let settings = new AppSettings();
    return of(settings);
  }
}
```

Demo



Create application settings class

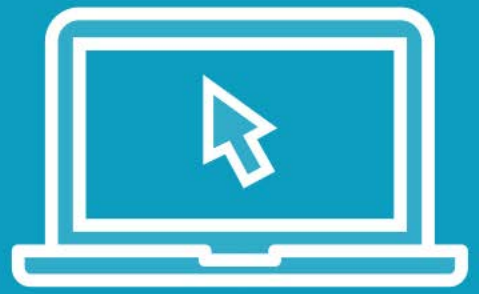
Create application settings service

A Product Class and Page

Product Class

```
export class Product {
    productId: number;
    productName: string;
    introductionDate: Date;
    price: number;
    url: string;
}
```

Demo

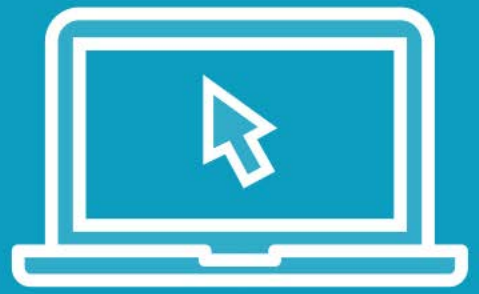


Create product class

Product Page

```
<table>
  // OTHER FIELDS HERE
  <tr>
    <td>Price:</td>
    <td><input [(ngModel)]="product.price" /></td>
  </tr>
  <tr>
    <td>URL:</td>
    <td><input [(ngModel)]="product.url" /></td>
  </tr>
</table>
```

Demo



Create product html page

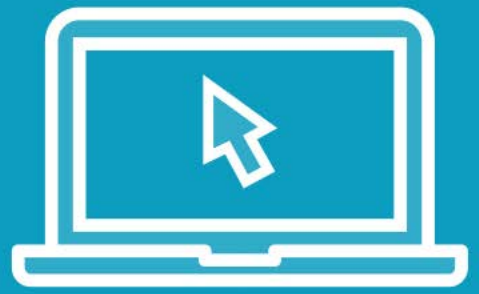
Product Page Component

```
@Component({})
export class ProductDetailComponent implements OnInit {
  constructor(private appSettingsService:
                      AppSettingsService) { }
  product: Product;
  settings: AppSettings;
  ngOnInit(): void { }
}
```

Retrieve Settings

```
ngOnInit(): void {
  this.appSettingsService.getSettings()
    .subscribe(settings => this.settings = settings,
     () => null,
     () => {
       this.product = new Product();
       this.product.price = this.settings.defaultPrice;
       this.product.url = this.settings.defaultUrl; });
}
```

Demo



Create product page component class

Summary

Why build a configuration system?

Configuration architecture

The modules in this course

Configuration class and service

Coming up in the next module...

Store settings in JSON file

Read file to create configuration class

Read Settings from a JSON File

Paul D. Sheriff

BUSINESS SOLUTIONS ARCHITECT, FAIRWAY TECHNOLOGIES, INC.

www.fairwaytech.com psheriff@fairwaytech.com

Goals

- Add bootstrap styles
- Create settings input screen
- Put settings into JSON file
- Read settings using HttpClient
- Add exception handling

Add Bootstrap

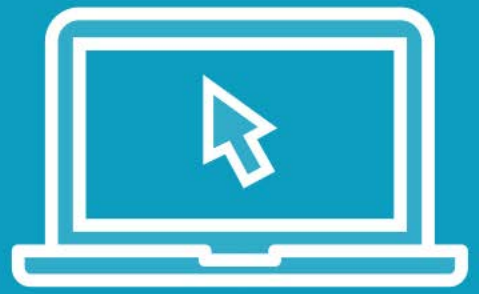
Add Bootstrap Styles

Use npm to install

Add styles and scripts to .angular-cli.json

Fix product detail page

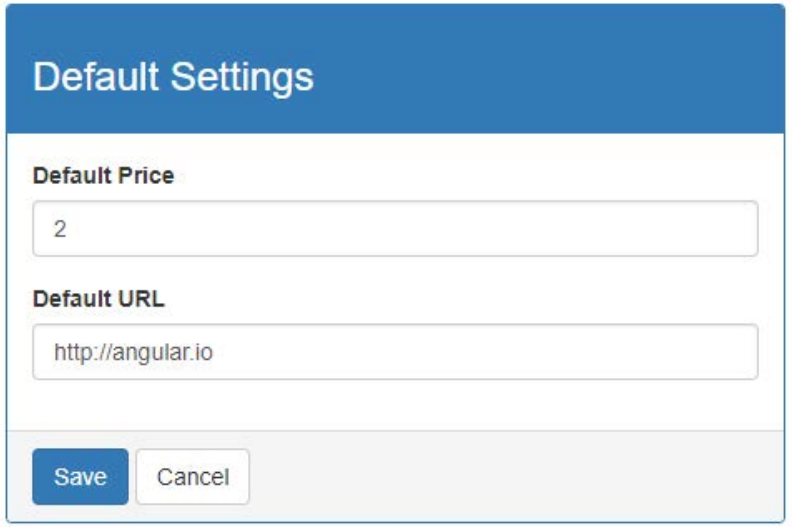
Demo



Add bootstrap

Fix up product detail page

Create Default Settings Page



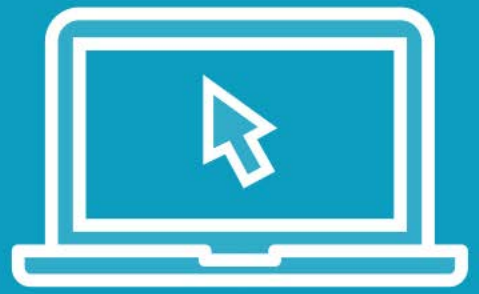
Create page for user to see all defaults

Create \src\app\settings folder

Build settings.component.html

Build settings.component.ts

Demo



Create settings page

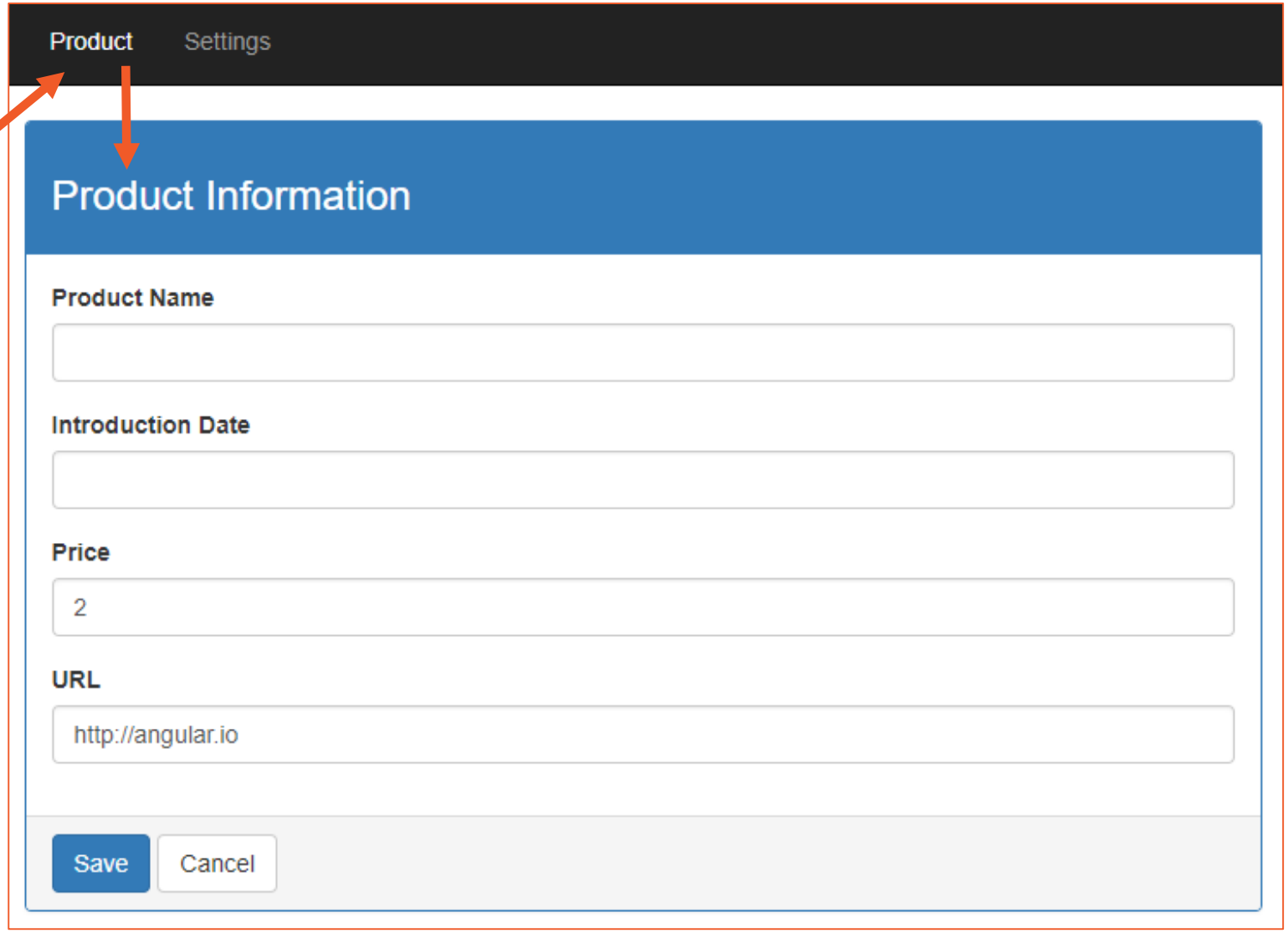
Add Routing and Menu

Add Routing

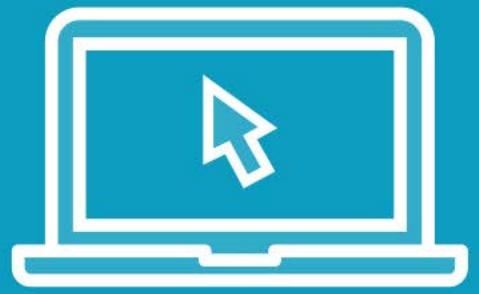
```
const routes: Routes = [
  { path: 'product', component: ProductDetailComponent },
  { path: 'settings', component: SettingsComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Add bootstrap navigation

Use routing links to call each page



Demo

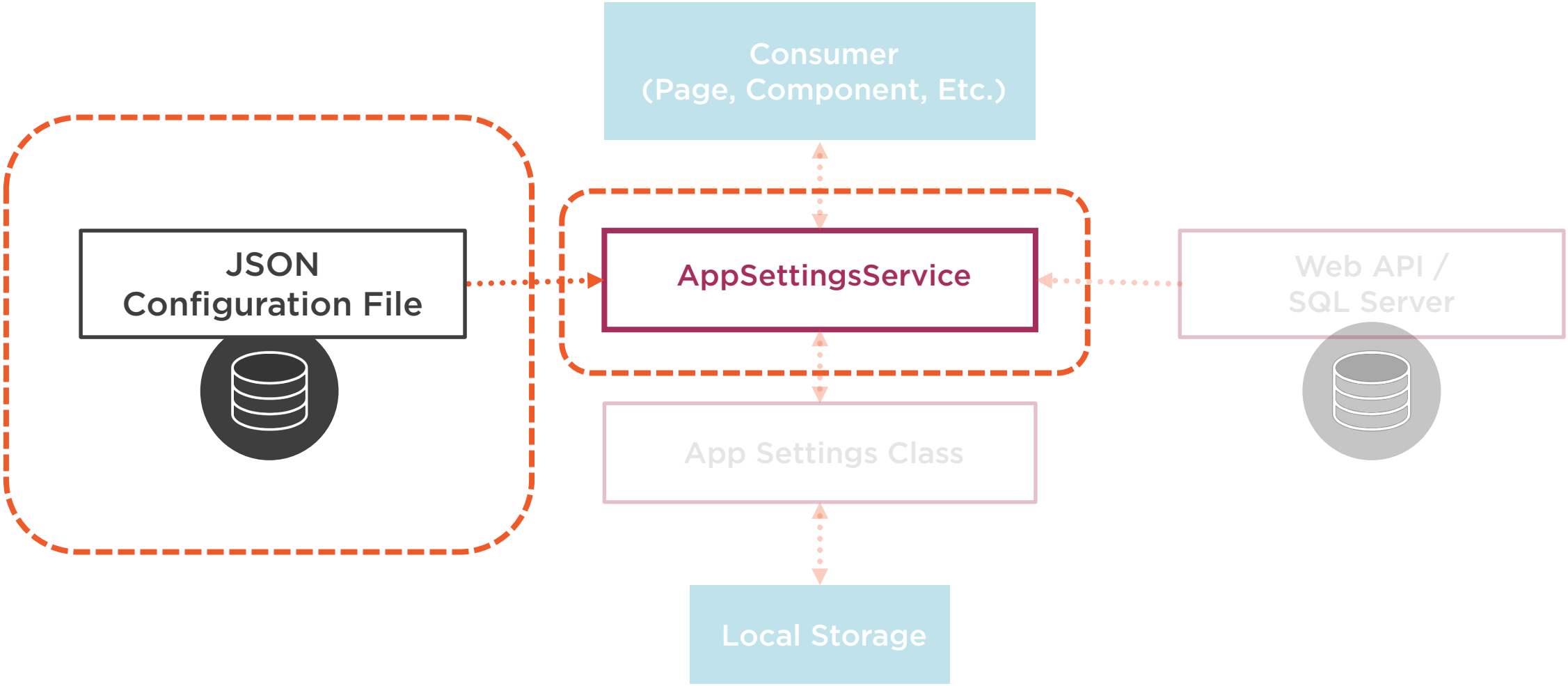


Add routing

Add bootstrap navigation

Create JSON File

Configuration Management Architecture



\assets\appsettings.json

```
{
  "defaultPrice": 2,
  "defaultUrl":
      "http://angular.io"
}
```

- **Add appsettings.json file**
  Add into \assets folder

- **Add two properties to match AppSettings class**
  defaultPrice
  defaultUrl

Modify AppSettingsService Class

Import HttpClient

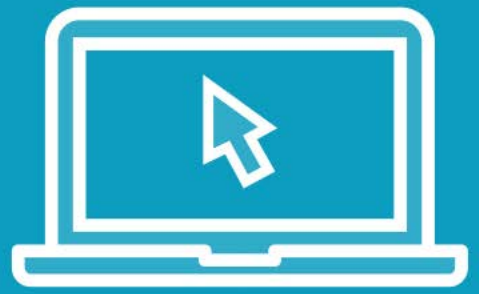
Import catchError

Add constant for JSON file location

Inject HttpClient

Modify getSettings() to read JSON

Demo



Add JSON file

Modify AppSettingsService class

Add Exception Handling

Exception Handling

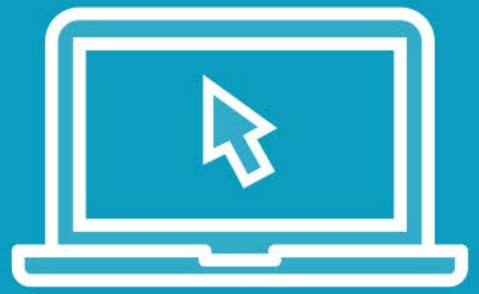
Add exception handling when reading JSON file

Don’t throw exception

Return instance of AppSettings class

Allows application to continue

Demo



Add exception handling

Summary

- Added bootstrap styles
- Created page to display settings
- Added routing and menu system
- Read from JSON file
- Added exception handling

Coming up in the next module...

Save settings into local storage

Retrieve settings from local storage

Delete settings from local storage

Read Settings from a JSON File

Paul D. Sheriff

BUSINESS SOLUTIONS ARCHITECT, FAIRWAY TECHNOLOGIES, INC.

www.fairwaytech.com psheriff@fairwaytech.com

Goals

- Add bootstrap styles
- Create settings input screen
- Put settings into JSON file
- Read settings using HttpClient
- Add exception handling

Add Bootstrap

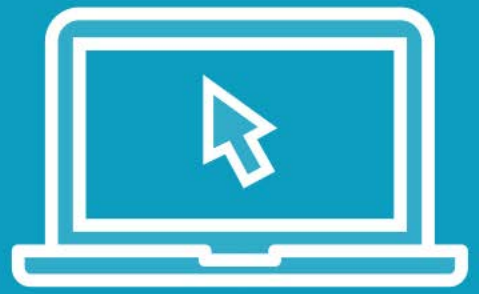
Add Bootstrap Styles

Use npm to install

Add styles and scripts to .angular-cli.json

Fix product detail page

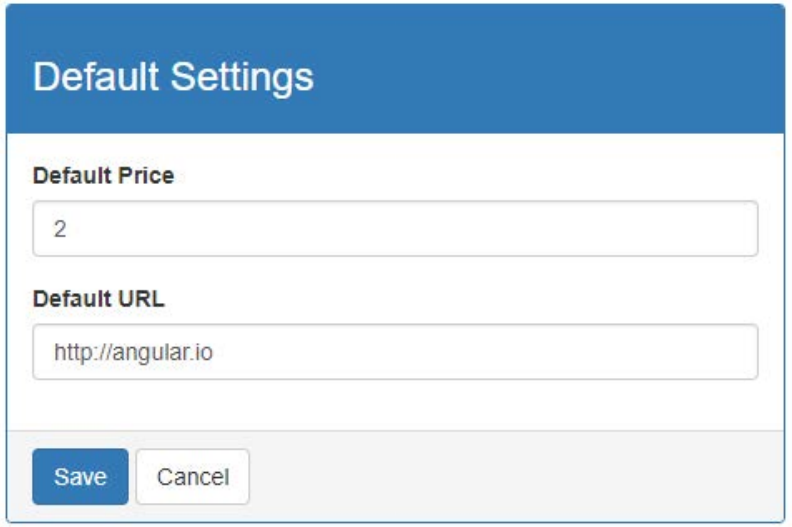
Demo



Add bootstrap

Fix up product detail page

Create Default Settings Page



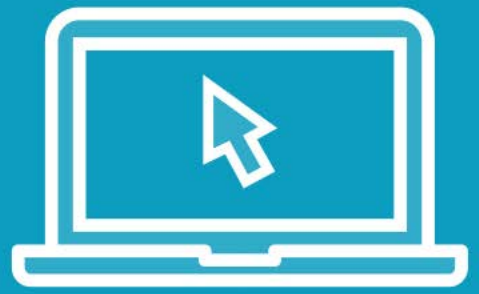
Create page for user to see all defaults

Create \src\app\settings folder

Build settings.component.html

Build settings.component.ts

Demo



Create settings page

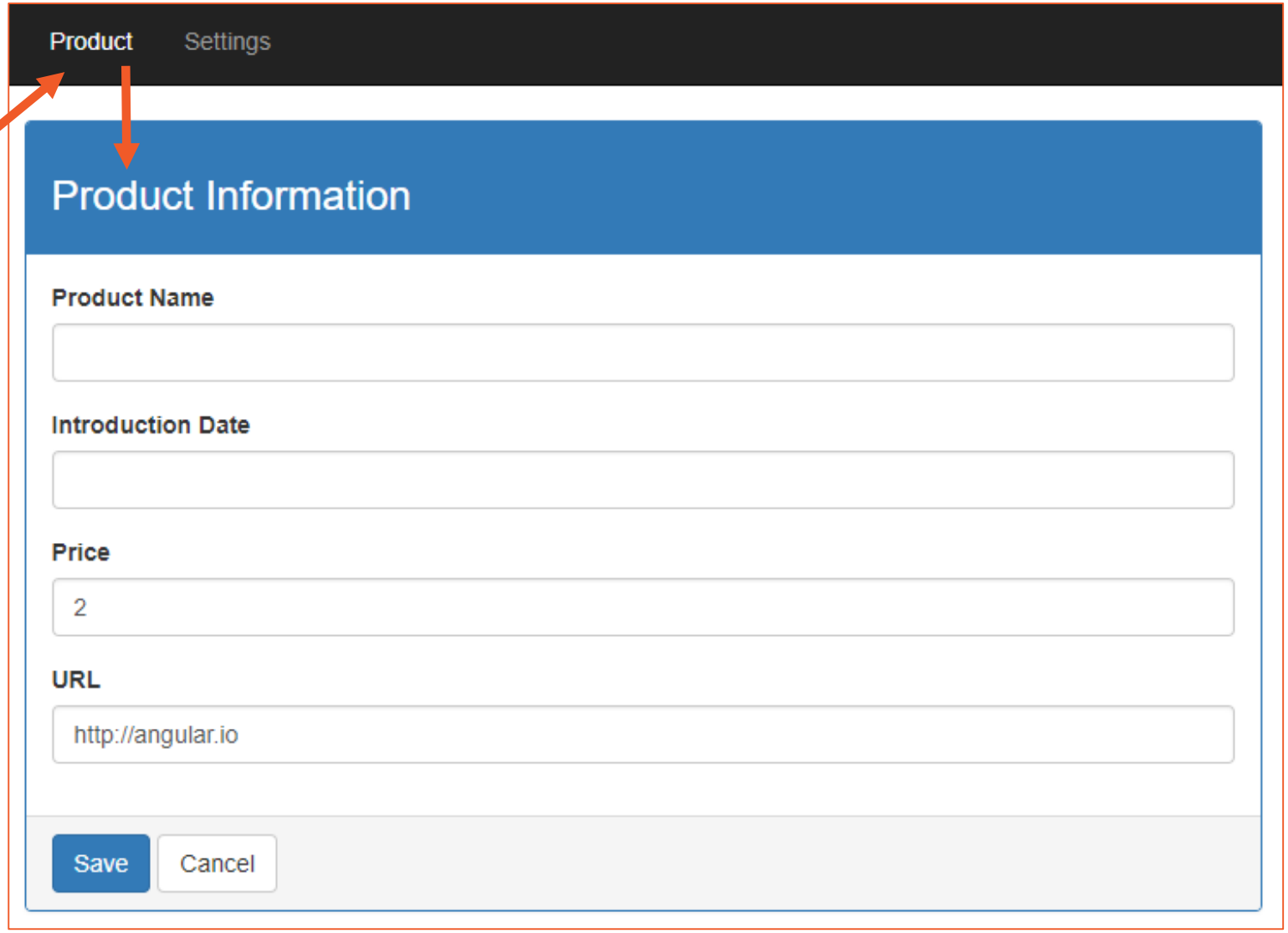
Add Routing and Menu

Add Routing

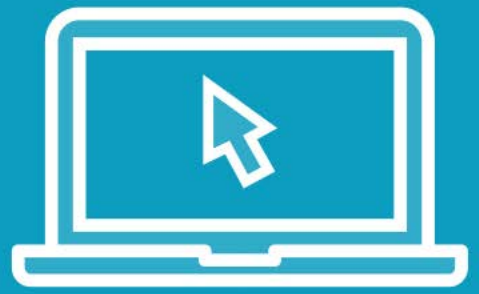
```
const routes: Routes = [
  { path: 'product', component: ProductDetailComponent },
  { path: 'settings', component: SettingsComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Add bootstrap navigation

Use routing links to call each page



Demo

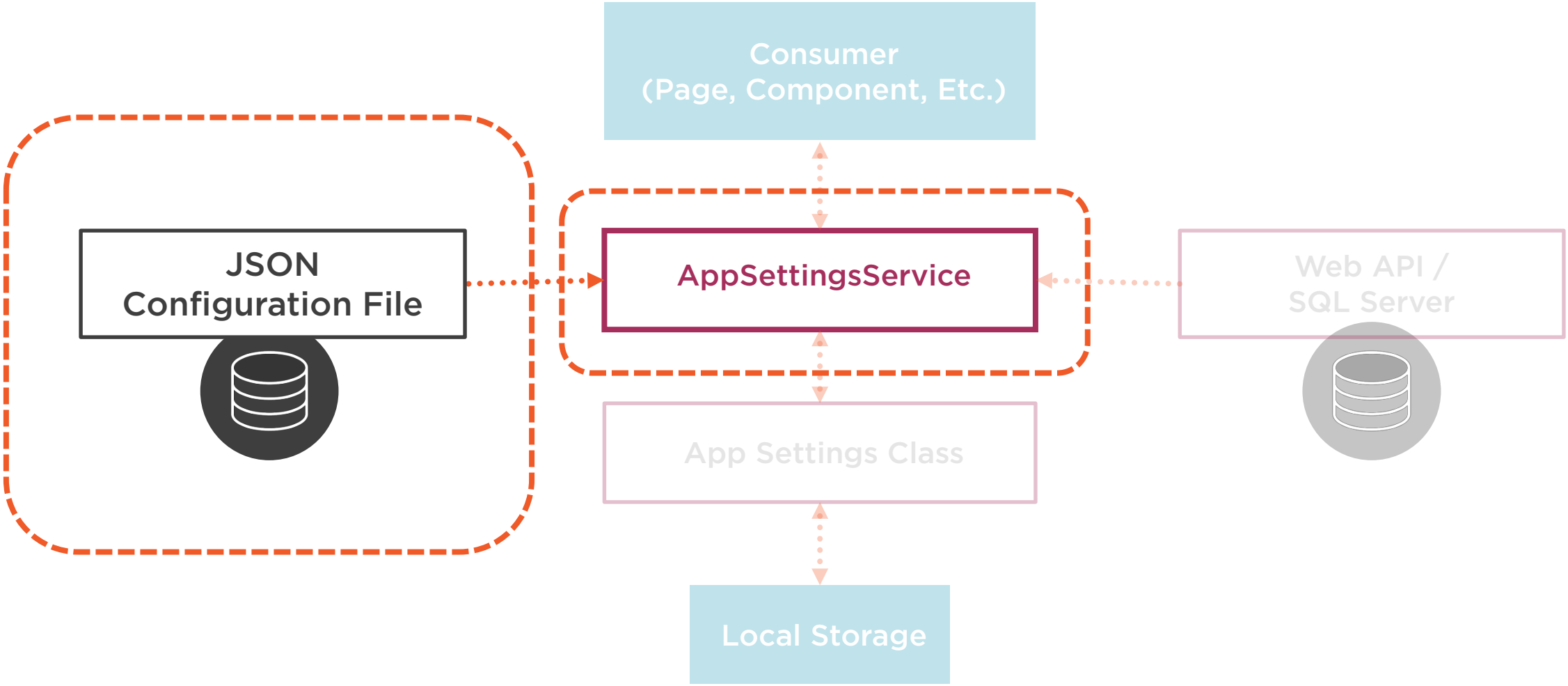


Add routing

Add bootstrap navigation

Create JSON File

Configuration Management Architecture



\assets\appsettings.json

```
{
  "defaultPrice": 2,
  "defaultUrl":
      "http://angular.io"
}
```

- **Add appsettings.json file**
  Add into \assets folder

- **Add two properties to match AppSettings class**
  defaultPrice
  defaultUrl

Modify AppSettingsService Class

Import HttpClient

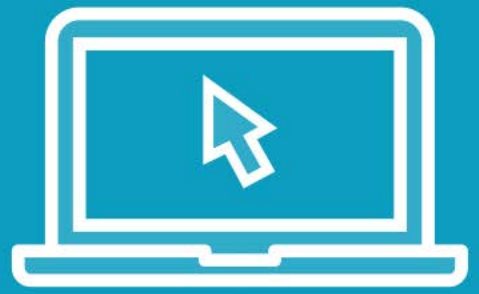
Import catchError

Add constant for JSON file location

Inject HttpClient

Modify getSettings() to read JSON

Demo



Add JSON file

Modify AppSettingsService class

Add Exception Handling

Exception Handling

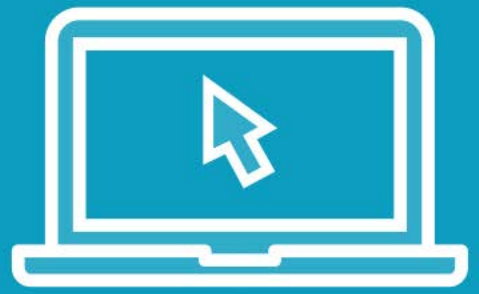
Add exception handling when reading JSON file

Don’t throw exception

Return instance of AppSettings class

Allows application to continue

Demo



Add exception handling

Summary

- Added bootstrap styles
- Created page to display settings
- Added routing and menu system
- Read from JSON file
- Added exception handling

Coming up in the next module...

Save settings into local storage

Retrieve settings from local storage

Delete settings from local storage

Store Settings in Local Storage

Paul D. Sheriff

BUSINESS SOLUTIONS ARCHITECT, FAIRWAY TECHNOLOGIES, INC.

www.fairwaytech.com psheriff@fairwaytech.com

Goals

- Work with local storage
- Save settings
- Retrieve settings
- Delete settings

Saving Data

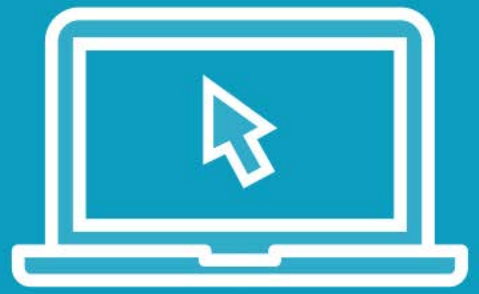
Modify AppSettingsService

```
const SETTINGS_KEY = "configuration";

saveSettings(settings: AppSettings) {
  localStorage.setItem(SETTINGS_KEY,
                       JSON.stringify(settings));
}
```

Modify SettingsComponent

```
<button class="btn btn-primary"
        (click)="saveSettings()">
  Save
</button>


saveSettings(settings: AppSettings) {
  this.appSettingsService.saveSettings(this.settings);
}
```

Demo



Save Settings to Local Storage

Retrieve Settings

Working with Local Storage

Attempt to get data from local storage

If not found, return data from JSON file, save to local storage

If found, return data

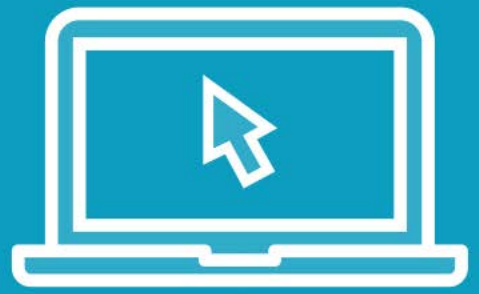
Modify AppSettingsService

```
getSettings(): Observable<AppSettings> {
  let settings = localStorage.getItem(SETTINGS_KEY);
   if (settings) {
    return of(JSON.parse(settings));
  }
  else {
    // Get data from JSON file
  }
}
```

Modify AppSettingsService

```
else {
  return this.http.get<AppSettings>(SETTINGS_LOCATION)
  .pipe(tap(settings => {
    if (settings) this.saveSettings(settings);
  }),
  catchError(this.handleError<AppSettings>('getSettings',
                new AppSettings()))
  );
 }
```

Demo



Retrieve settings

Delete Settings

Modify AppSettingsService

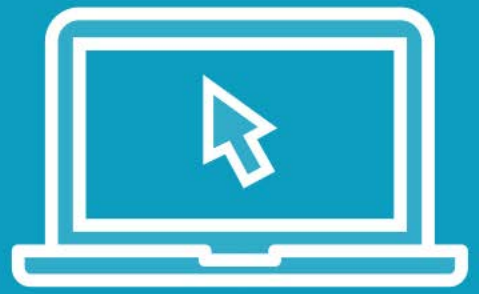
```
deleteSettings(): void {
  localStorage.removeItem(SETTINGS_KEY);
}
```

Modify the Settings Component

```
<button class="btn btn-primary"
        (click)="deleteSettings()">
  Delete Settings
</button>
```

```
deleteSettings(): void {
  this.appSettingsService.deleteSettings();
}
```

Demo



Delete settings

Summary

Save into local storage

Retrieve from local storage or JSON file

Delete local storage values

Coming up in the next module...

Build Web API using ASP.NET Core

Create table to hold settings

Return settings from Web API

Store Web API settings into local storage

Retrieve Settings from SQL Server via a Web API Call

Paul D. Sheriff

BUSINESS SOLUTIONS ARCHITECT, FAIRWAY TECHNOLOGIES, INC.

www.fairwaytech.com psheriff@fairwaytech.com

Goals

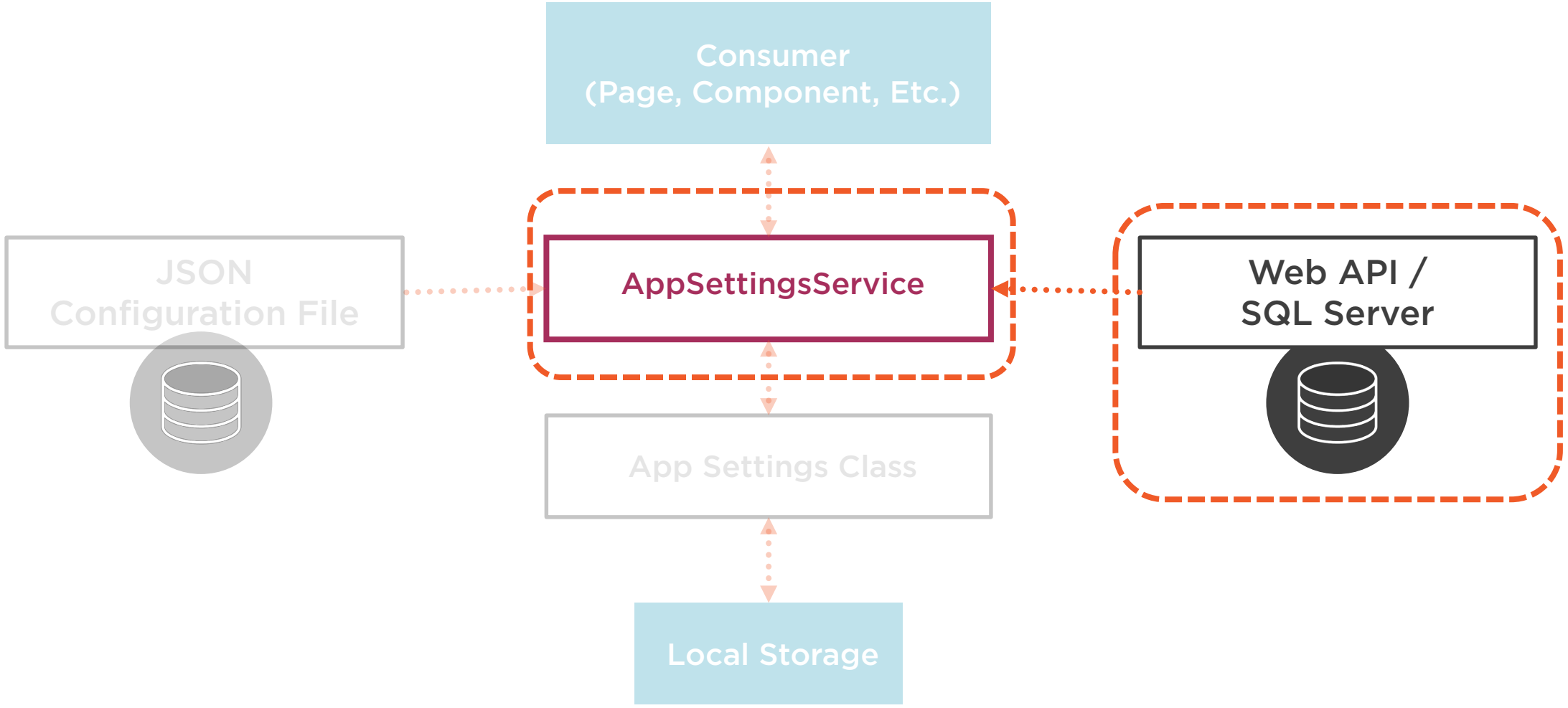
Create AppSettings table

Create Web API using ASP.NET Core

Modify Angular service to call Web API

Overview

Configuration Management Architecture



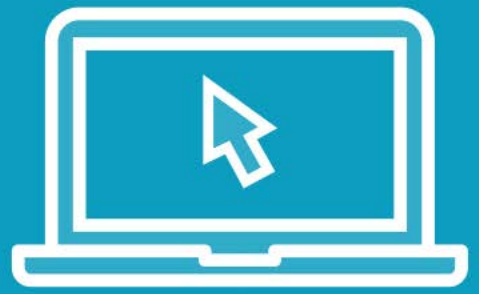
Steps

- Create AppSettings table
- Create Web API using ASP.NET Core
- Add Entity Framework
- Add Get() method to ConfigController class
- Configure Web API for JSON and CORS
- Modify Angular service to call Web API

Create AppSettings Table

```
CREATE TABLE AppSettings
(
  AppSettingsId int NOT NULL IDENTITY(1,1) PRIMARY KEY NONCLUSTERED,
  DefaultPrice money NOT NULL DEFAULT(1),
  DefaultUrl nvarchar(255) NOT NULL,
  IsFromLocalStorage bit NOT NULL DEFAULT(0)
);
INSERT INTO AppSettings(DefaultPrice, DefaultUrl)
VALUES(42, 'http://www.google.com');
```

Demo



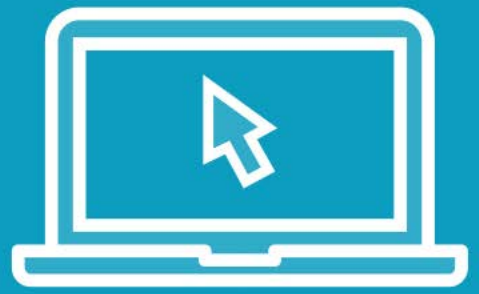
Create AppSettings table

Create Web API

Create ASP.NET Core Web API

```
// Drop to command prompt

D:
cd Samples
md ConfigWebApi
cd ConfigWebApi

dotnet new webapi
code .
```

Demo



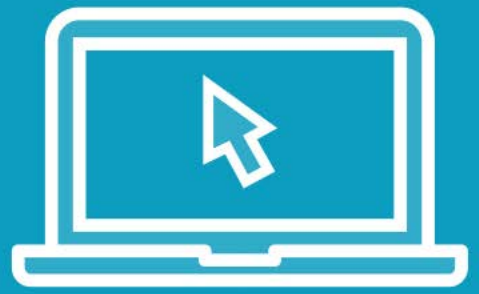
Create ASP.NET Core Web API project

Add Entity Framework

Add Entity Framework

```
// Open integrated terminal

dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

Demo

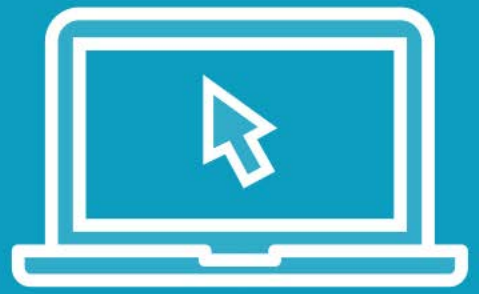


Add Entity Framework

Add AppSettings Class

```
public class AppSettings {
  [Required]
  public int AppSettingsId {get; set;}
  [Required]
  public decimal DefaultPrice { get; set; }
  [Required]
  public string DefaultUrl { get; set; }
  [Required]
  public bool IsFromLocalStorage { get; set; }
}
```

Demo

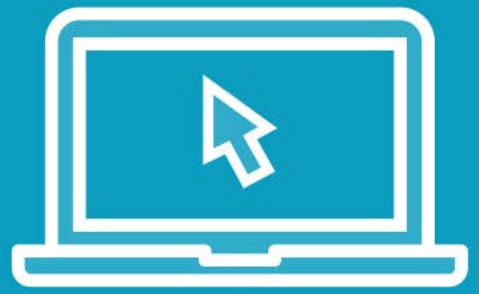


Add AppSettings entity class

Add DbContext Class

```
public class ConfigDB : DbContext {
  public DbSet<AppSettings> AppSettings { get; set; }
  protected override void OnConfiguring(
    DbContextOptionsBuilder optionsBuilder) {
      optionsBuilder.UseSqlServer(@"Server=Localhost;
        Database=Sandbox;Trusted_Connection=True;
        MultipleActiveResultSets=true");
    }
}
```

Demo



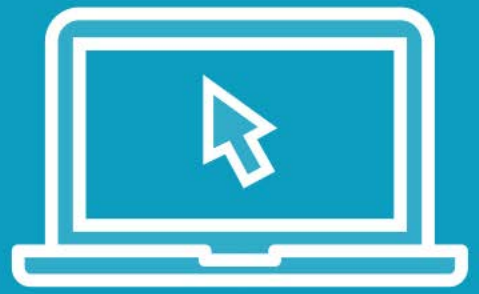
Add DbContext class

Get() Method in Controller

Modify Config Controller Get() Method

```
public IActionResult Get() {
  ConfigDB db = new ConfigDB();

  return StatusCode(StatusCodes.Status200OK,
                db.AppSettings.FirstOrDefault());
}
```

Demo



Create Get() method in controller

Configure Web API for JSON and CORS

Steps

Create AppSettings table

Create Web API using ASP.NET Core

Add Entity Framework

Add Get() method to ConfigController class

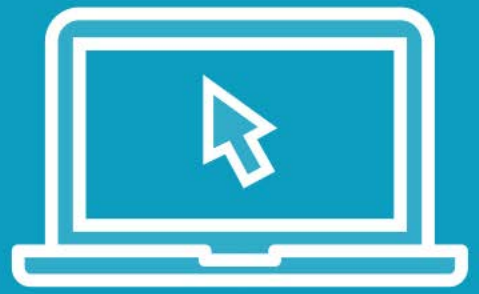
Configure Web API for JSON and CORS

Modify Angular service to call Web API

Modify ConfigureServices Method

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
    .AddJsonOptions(options =>
        options.SerializerSettings.ContractResolver =
            new CamelCasePropertyNamesContractResolver());
}
```

Demo



Convert camel case to pascal case

Add CORS

```
// Open integrated terminal

dotnet add package Microsoft.AspNetCore.Cors
```

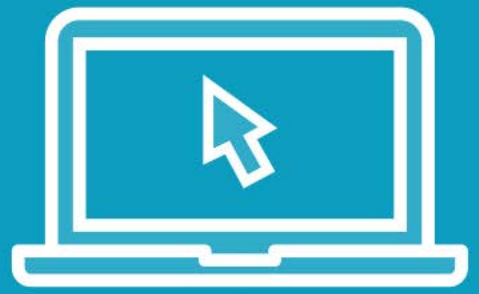
Modify ConfigureServices Method

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors();
    services.AddMvc();
    .AddJsonOptions(options =>
        options.SerializerSettings.ContractResolver =
            new CamelCasePropertyNamesContractResolver());
}
```

Modify Configure Method

```
public void Configure(...)
{
  // Other code here
  app.UseCors(
    options => options.WithOrigins("http://localhost:4200")
      .AllowAnyMethod()
  );
}
```

Demo



Add CORS

Modify Angular Code

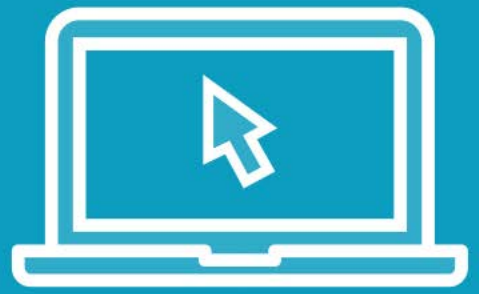
Modify AppSettingsService

```
// Modify constant

const SETTINGS_LOCATION =
    "http://localhost:5000/api/config";
```

Modify AppSettings Class

```
export class AppSettings {
  appSettingsId: number = null;
  defaultPrice: number = 1;
  defaultUrl: string = "http://www.fairwaytech.com";
  isFromLocalStorage: boolean = false;
}
```

Demo



Modify Angular Code

Summary

Added SQL Server table

Added Web API

Called Web API from Angular

Course Summary

- Created service for default settings
- Stored settings into JSON file
- Saved settings into local storage
- Created Web API for default settings

I hope you enjoyed
this course!