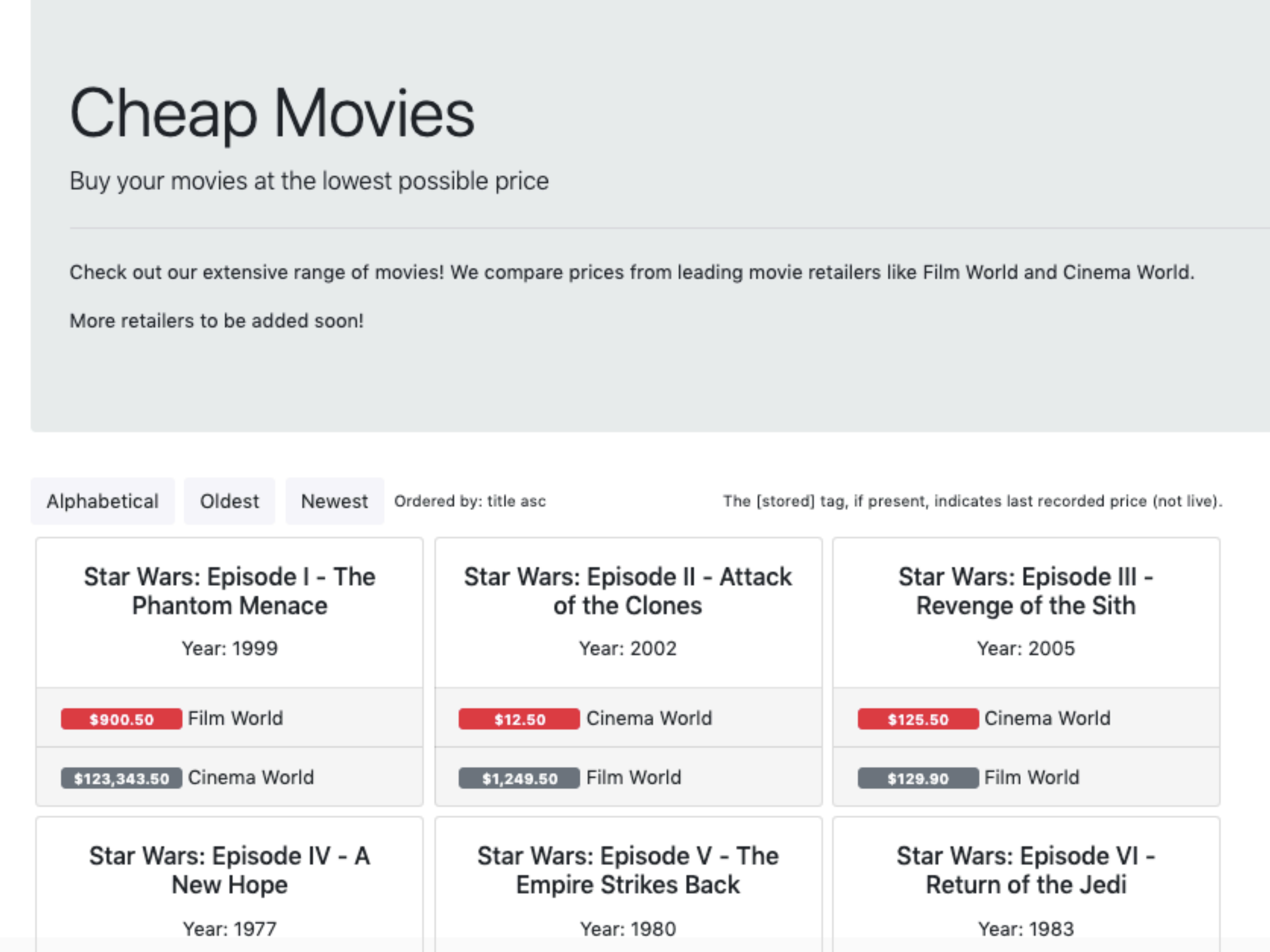


Cheap Movies

Buy your movies at the lowest possible price

Check out our extensive range of movies! We compare prices from leading movie retailers like Film World and Cinema World.

More retailers to be added soon!

Ordered by: title asc

The [stored] tag, if present, indicates last recorded price (not live).

Star Wars: Episode I - The Phantom Menace

Year: 1999

$900.50 Film World

$123,343.50 Cinema World

Star Wars: Episode II - Attack of the Clones

Year: 2002

$12.50 Cinema World

$1,249.50 Film World

Star Wars: Episode III - Revenge of the Sith

Year: 2005

$125.50 Cinema World

$129.90 Film World

Star Wars: Episode IV - A New Hope

Year: 1977

Star Wars: Episode V - The Empire Strikes Back

Year: 1980

Star Wars: Episode VI - Return of the Jedi

Year: 1983

Initial Thoughts

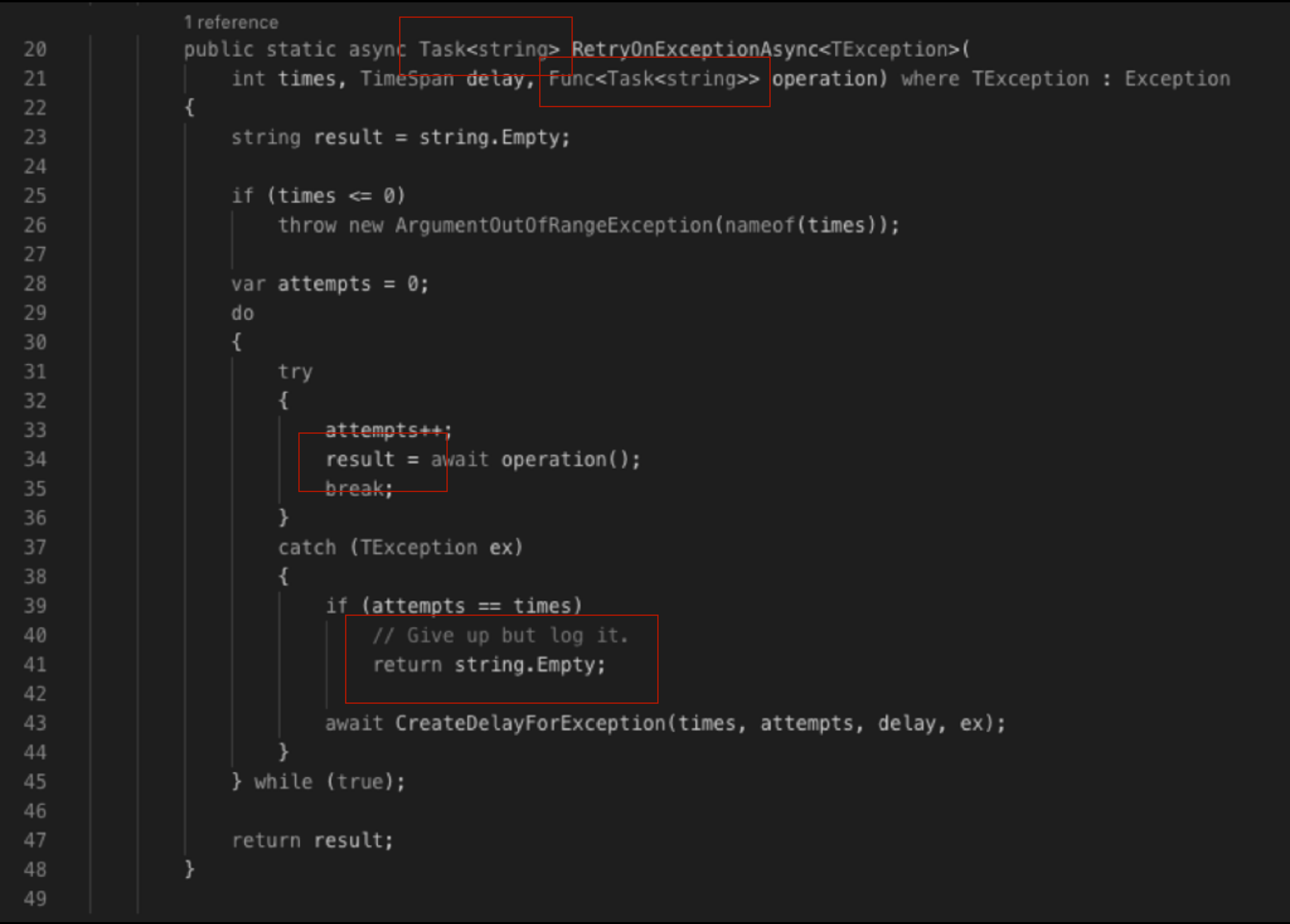
- Preference for Live Data (rely less on cache / local copy)
- Open for additional providers (ie. do not hard-code the 2 given providers)
- UI similar to Booko (booko.com.au). No need for a service to get the cheapest price. Just dynamically sort prices (as they come) from lowest to highest at the front-end.
- Best Practices (separation of concerns, leverage on existing solutions, testability)

Preference for Live Data

- For unreliable services, use a Retry utility.
- Thought about using Polly. (Polly is a .Net resilience and transient-fault handling library.)
- Polly has retry, circuit breaker, timeout, bulkhead isolation, and fallback features.
- BUT, can’t afford to overcome the learning curve for the given timeframe

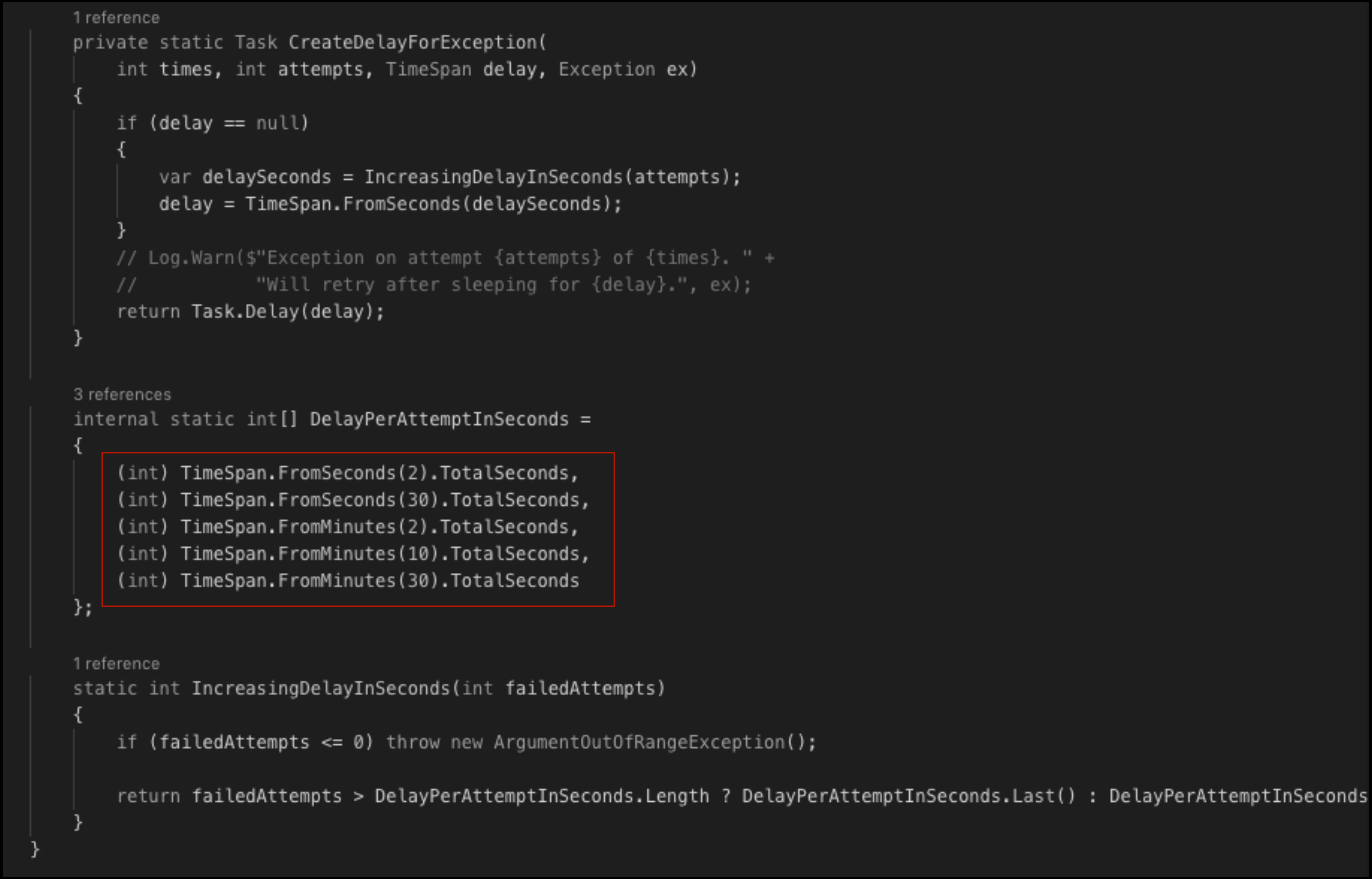
Retry Helper

- RetryHelper code was taken from a blog article that mentions Polly.
- Modify the code so to use methods that return a string. (Original only accepted void methods.)



```
1 reference
public static async Task<string> RetryOnExceptionAsync<TException>(
    int times, TimeSpan delay, Func<Task<string>> operation) where TException : Exception
{
    string result = string.Empty;

    if (times <= 0)
        throw new ArgumentOutOfRangeException(nameof(times));

    var attempts = 0;
    do
    {
        try
        {
            attempts++;
            result = await operation();
            break;
        }
        catch (TException ex)
        {
            if (attempts == times)
                // Give up but log it.
                return string.Empty;

            await CreateDelayForException(times, attempts, delay, ex);
        }
    } while (true);

    return result;
}

```

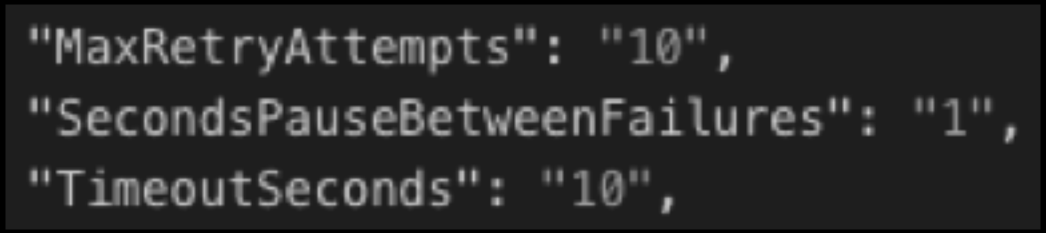
- It also has exponential / incremental backoff functionality, but it is not used in this demo.



```
1 reference
private static Task CreateDelayForException(
    int times, int attempts, TimeSpan delay, Exception ex)
{
    if (delay == null)
    {
        var delaySeconds = IncreasingDelayInSeconds(attempts);
        delay = TimeSpan.FromSeconds(delaySeconds);
    }
    // Log.Warn($"Exception on attempt {attempts} of {times}. " +
    //          "Will retry after sleeping for {delay}.", ex);
    return Task.Delay(delay);
}

3 references
internal static int[] DelayPerAttemptInSeconds =
{
    (int) TimeSpan.FromSeconds(2).TotalSeconds,
    (int) TimeSpan.FromSeconds(30).TotalSeconds,
    (int) TimeSpan.FromMinutes(2).TotalSeconds,
    (int) TimeSpan.FromMinutes(10).TotalSeconds,
    (int) TimeSpan.FromMinutes(30).TotalSeconds
};

1 reference
static int IncreasingDelayInSeconds(int failedAttempts)
{
    if (failedAttempts <= 0) throw new ArgumentOutOfRangeException();

    return failedAttempts > DelayPerAttemptInSeconds.Length ? DelayPerAttemptInSeconds.Last() : DelayPerAttemptInSeconds
}
}
```

Retry Configuration

- Instead of using incremental backoff, use fixed values that work best.

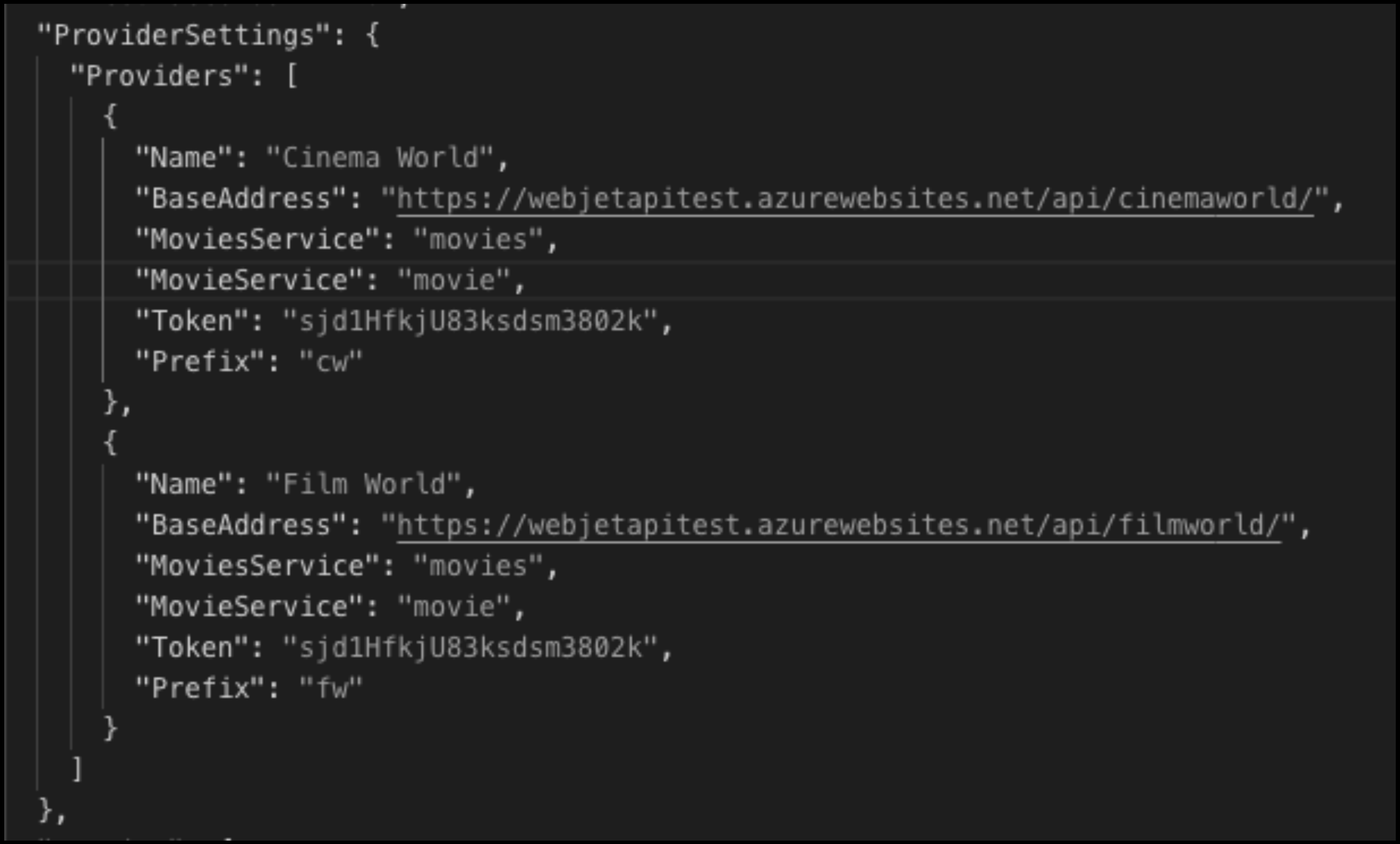


```
"MaxRetryAttempts": "10",
"SecondsPauseBetweenFailures": "1",
"TimeoutSeconds": "10",
```

- If it fails, use a stored value (key-value store).

Configurable Providers

- Have a configurable list of providers to get values from, instead of hard-coding them.



```
"ProviderSettings": {
  "Providers": [
    {
      "Name": "Cinema World",
      "BaseAddress": "https://webjetapitest.azurewebsites.net/api/cinemaworld/",
      "MoviesService": "movies",
      "MovieService": "movie",
      "Token": "sjd1HfkjU83ksdsm3802k",
      "Prefix": "cw"
    },
    {
      "Name": "Film World",
      "BaseAddress": "https://webjetapitest.azurewebsites.net/api/filmworld/",
      "MoviesService": "movies",
      "MovieService": "movie",
      "Token": "sjd1HfkjU83ksdsm3802k",
      "Prefix": "fw"
    }
  ]
},
```

Configurable Providers

- Let the SPA have a list of providers retrieved from the server (GetProviders), and retrieve each data from the server asynchronously (GetMovies / GetMovie).
- Services:
  - GetProviders
  - GetMovies?serviceId=XX
  - GetMovie?serviceId=XX&movieId=YY

Separation of Concerns

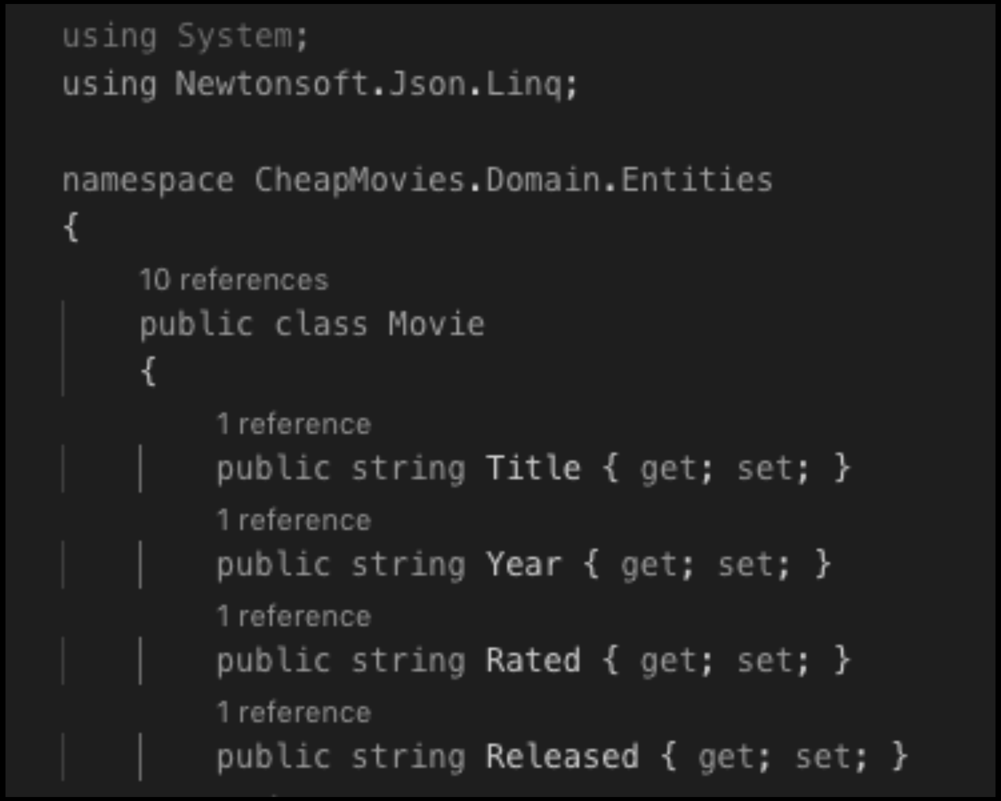
- CheapMovies.Domain: For the Movie entity
- CheapMovies.Services: MovieDataService (communicating with web services) and MovieService (for returning Movie data to API)
- CheapMovies.Store: key-value storage
- CheapMovies.Common: RetryHelper and constants (config keys)

Separation of Concerns

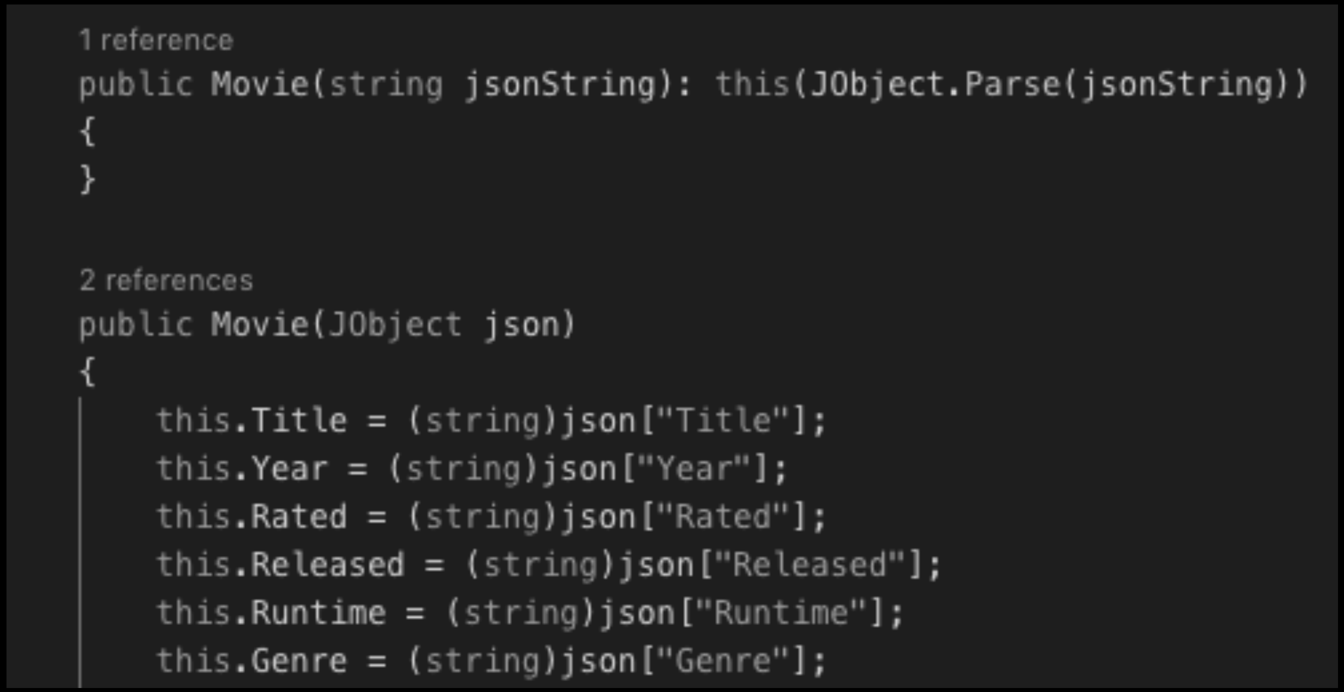
- CheapMovies.Api: controllers that call services
- CheapMovies.Web: Startup, Angular SPA, Configuration (appsettings.json)
- CheapMovies.Tests: xUnit tests

CheapMovies.Domain

- Movie entity, with constructor that accepts JSON string and JSON object



```
using System;
using Newtonsoft.Json.Linq;

namespace CheapMovies.Domain.Entities
{
    10 references
    public class Movie
    {
        1 reference
        public string Title { get; set; }
        1 reference
        public string Year { get; set; }
        1 reference
        public string Rated { get; set; }
        1 reference
        public string Released { get; set; }
```

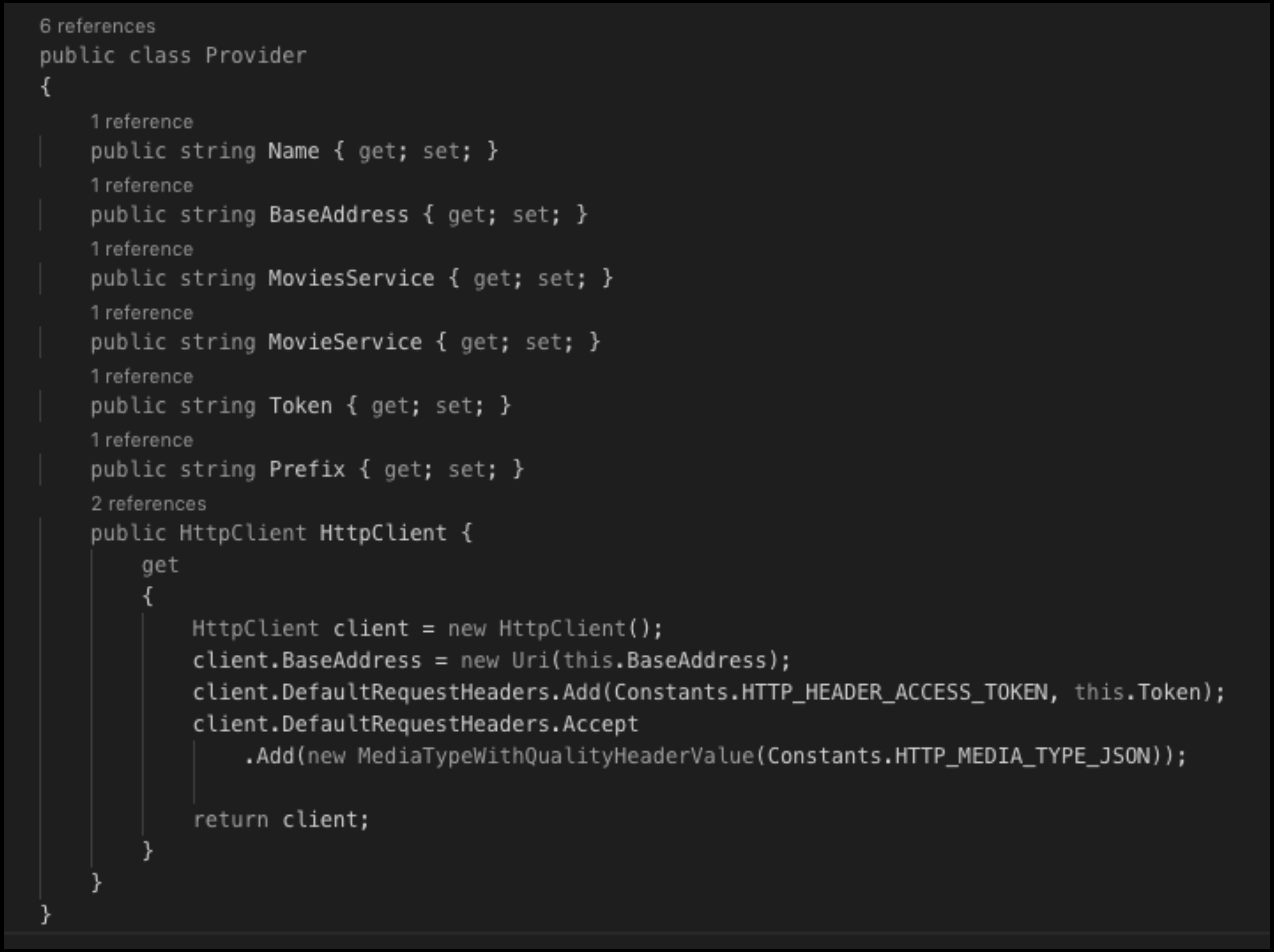


```
1 reference
public Movie(string jsonString): this(JObject.Parse(jsonString))
{
}

2 references
public Movie(JObject json)
{
    this.Title = (string)json["Title"];
    this.Year = (string)json["Year"];
    this.Rated = (string)json["Rated"];
    this.Released = (string)json["Released"];
    this.Runtime = (string)json["Runtime"];
    this.Genre = (string)json["Genre"];
```

CheapMovies.Services

- MovieDataService
  - Provider collection
  - GetMoviesOperationAsync / GetMovieOperationAsync
  - GetMoviesAsync / GetMovieAsync

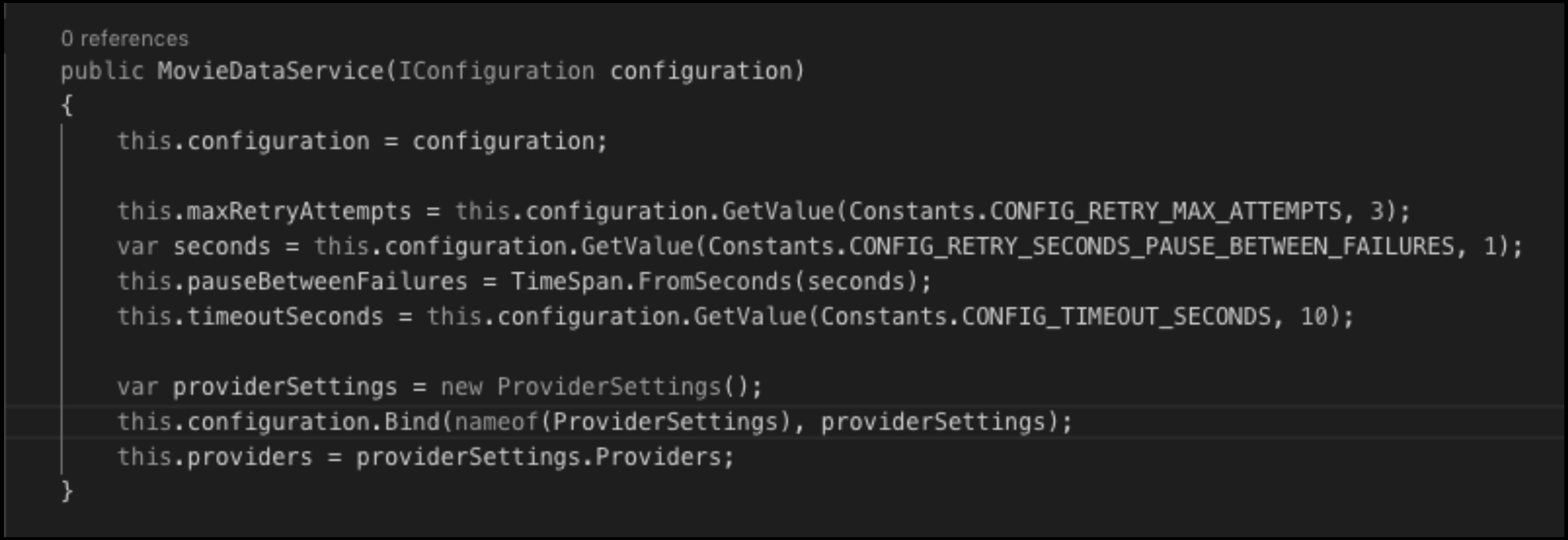
CheapMovies.Services



```
6 references
public class Provider
{
    1 reference
    public string Name { get; set; }
    1 reference
    public string BaseAddress { get; set; }
    1 reference
    public string MoviesService { get; set; }
    1 reference
    public string MovieService { get; set; }
    1 reference
    public string Token { get; set; }
    1 reference
    public string Prefix { get; set; }
    2 references
    public HttpClient HttpClient {
        get
        {
            HttpClient client = new HttpClient();
            client.BaseAddress = new Uri(this.BaseAddress);
            client.DefaultRequestHeaders.Add(Constants.HTTP_HEADER_ACCESS_TOKEN, this.Token);
            client.DefaultRequestHeaders.Accept
                .Add(new MediaTypeWithQualityHeaderValue(Constants.HTTP_MEDIA_TYPE_JSON));

            return client;
        }
    }
}
```

CheapMovies.Services

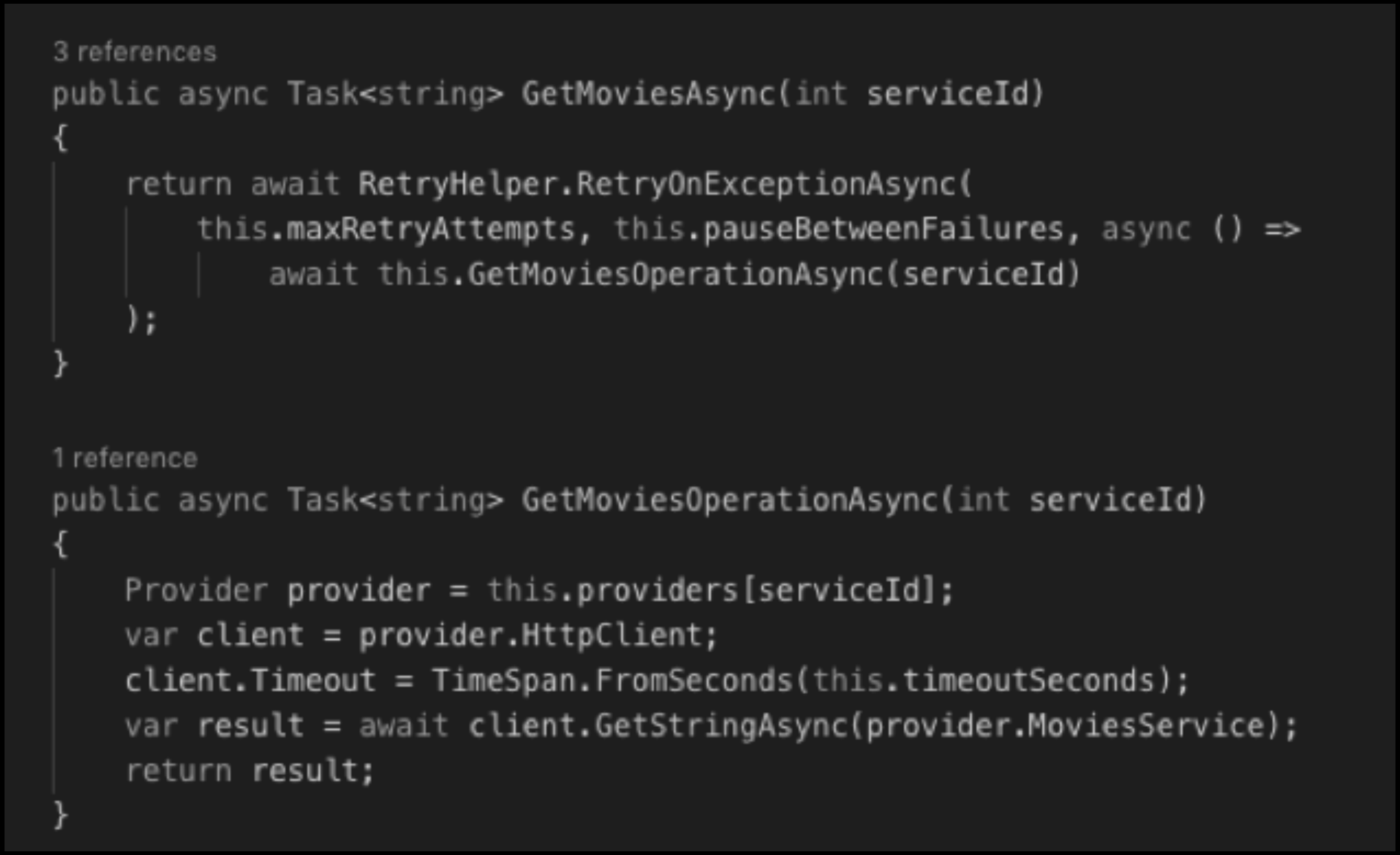
- Constructor gets settings and providers from configuration.



```
0 references
public MovieDataService(IConfiguration configuration)
{
    this.configuration = configuration;

    this.maxRetryAttempts = this.configuration.GetValue(Constants.CONFIG_RETRY_MAX_ATTEMPTS, 3);
    var seconds = this.configuration.GetValue(Constants.CONFIG_RETRY_SECONDS_PAUSE_BETWEEN_FAILURES, 1);
    this.pauseBetweenFailures = TimeSpan.FromSeconds(seconds);
    this.timeoutSeconds = this.configuration.GetValue(Constants.CONFIG_TIMEOUT_SECONDS, 10);

    var providerSettings = new ProviderSettings();
    this.configuration.Bind(nameof(ProviderSettings), providerSettings);
    this.providers = providerSettings.Providers;
}
```

CheapMovies.Services

- GetMoviesAsync: Retries one or more calls
- GetMoviesOperationAsync: A single web service call

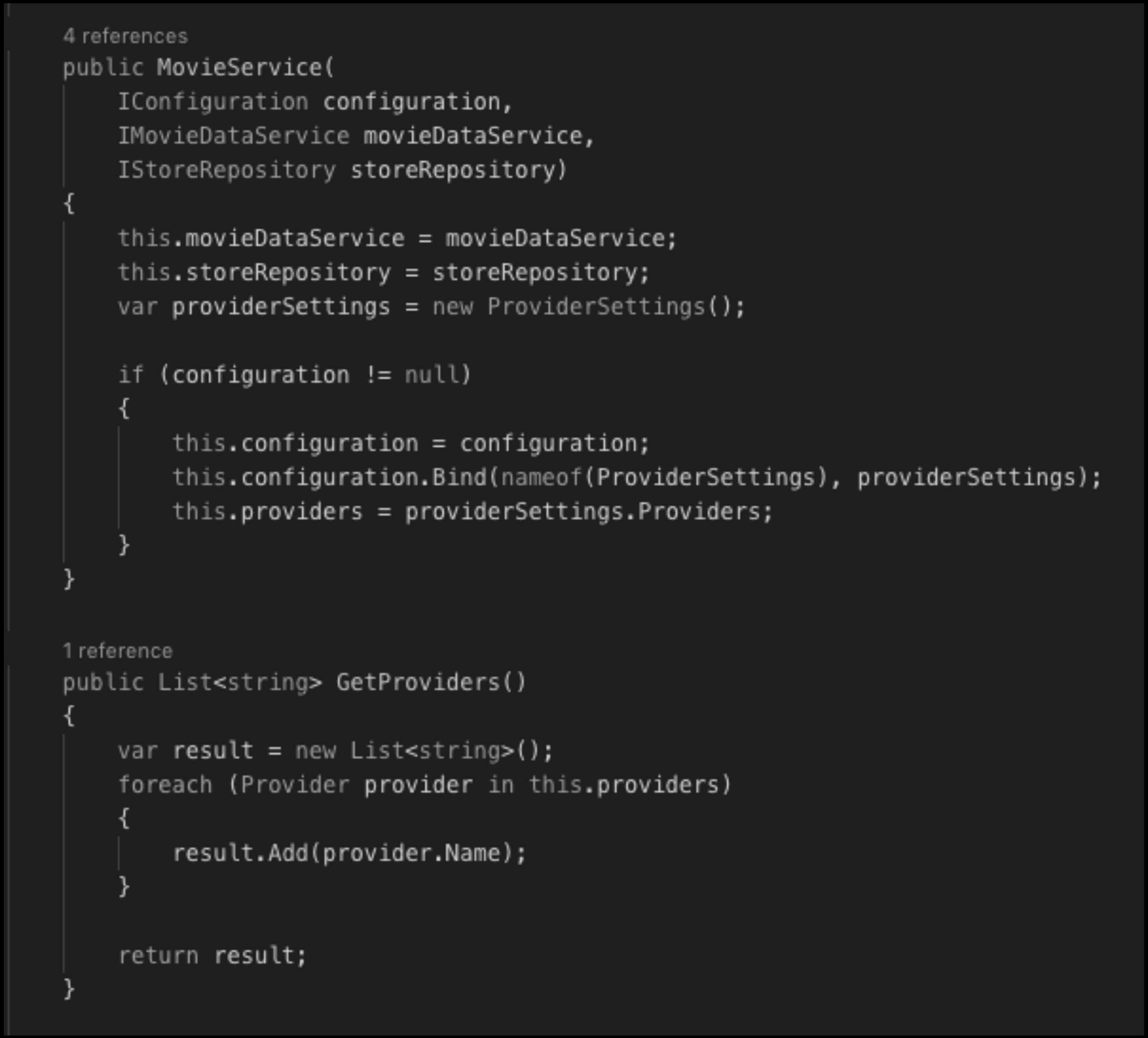


```
3 references
public async Task<string> GetMoviesAsync(int serviceId)
{
    return await RetryHelper.RetryOnExceptionAsync(
        this.maxRetryAttempts, this.pauseBetweenFailures, async () =>
            await this.GetMoviesOperationAsync(serviceId)
    );
}

1 reference
public async Task<string> GetMoviesOperationAsync(int serviceId)
{
    Provider provider = this.providers[serviceId];
    var client = provider.HttpClient;
    client.Timeout = TimeSpan.FromSeconds(this.timeoutSeconds);
    var result = await client.GetStringAsync(provider.MoviesService);
    return result;
}
```

CheapMovies.Services

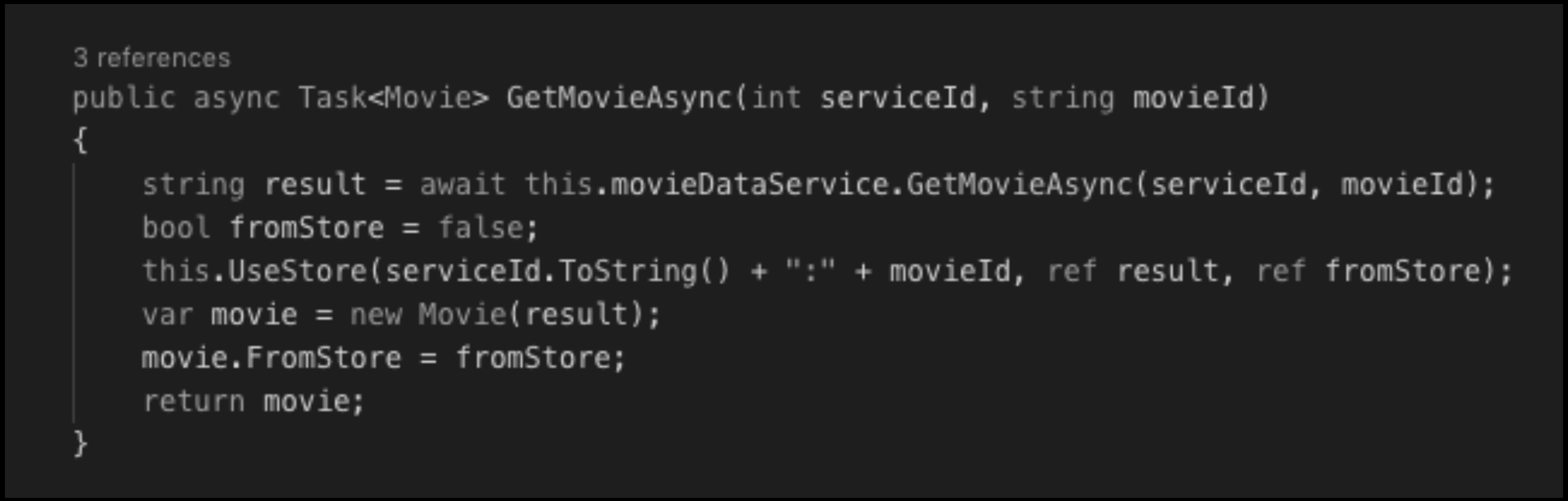
- MovieService
  - GetProviders (returns providers in config settings)
  - GetMoviesAsync / GetMovieAsync
  - UseStore
  - ParseMovies

CheapMovies.Services

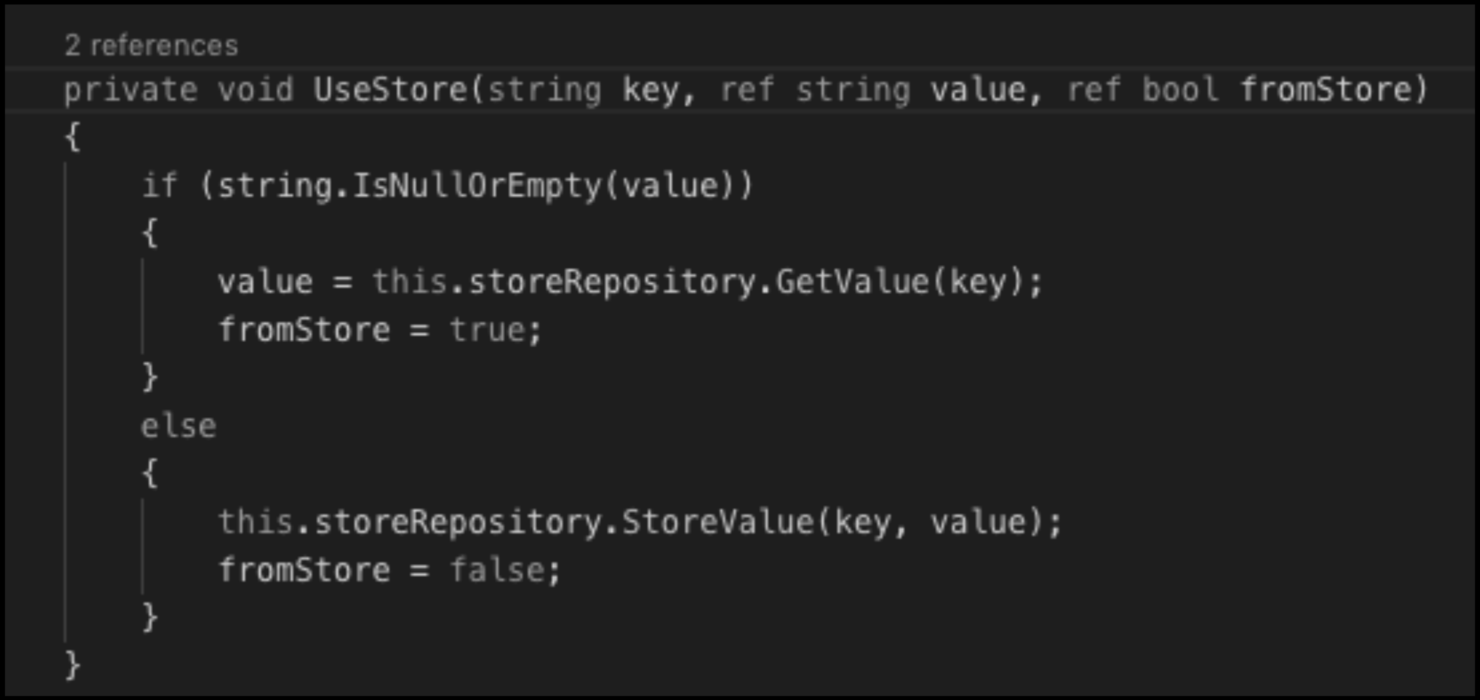


```
4 references
public MovieService(
    IConfiguration configuration,
    IMovieDataService movieDataService,
    IStoreRepository storeRepository)
{
    this.movieDataService = movieDataService;
    this.storeRepository = storeRepository;
    var providerSettings = new ProviderSettings();

    if (configuration != null)
    {
        this.configuration = configuration;
        this.configuration.Bind(nameof(ProviderSettings), providerSettings);
        this.providers = providerSettings.Providers;
    }
}

1 reference
public List<string> GetProviders()
{
    var result = new List<string>();
    foreach (Provider provider in this.providers)
    {
        result.Add(provider.Name);
    }

    return result;
}
```

CheapMovies.Services



```
3 references
public async Task<Movie> GetMovieAsync(int serviceId, string movieId)
{
    string result = await this.movieDataService.GetMovieAsync(serviceId, movieId);
    bool fromStore = false;
    this.UseStore(serviceId.ToString() + ":" + movieId, ref result, ref fromStore);
    var movie = new Movie(result);
    movie.FromStore = fromStore;
    return movie;
}
```

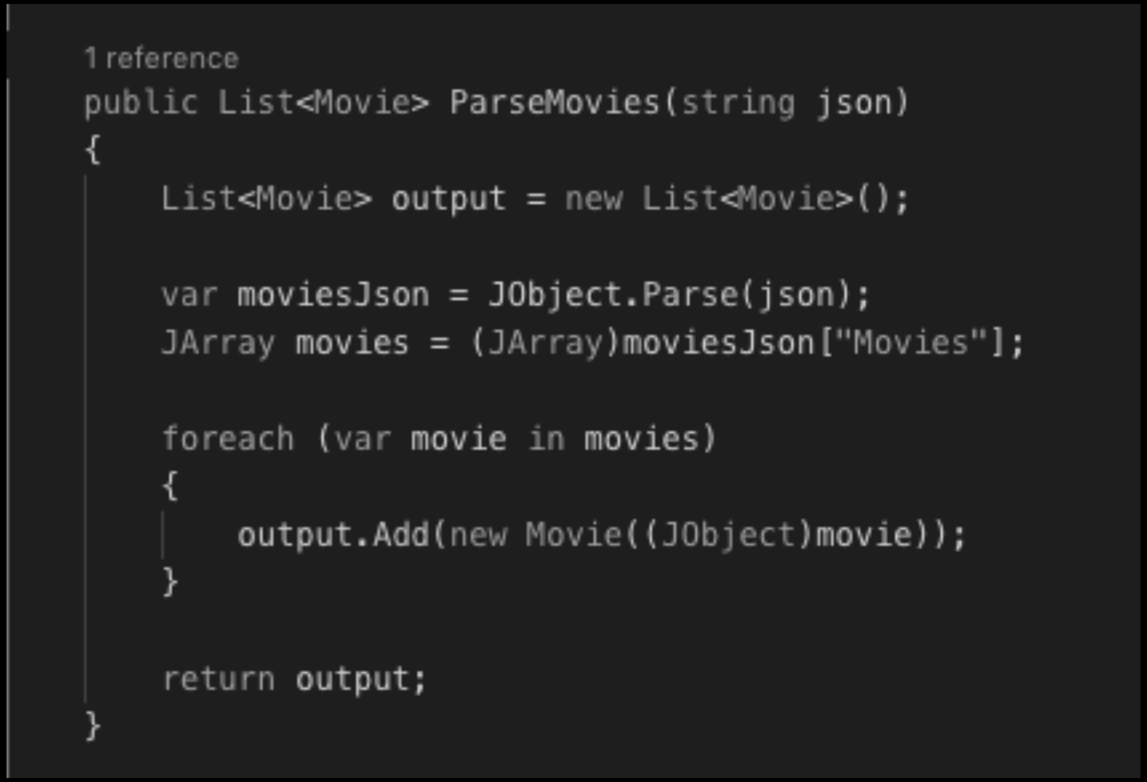


```
2 references
private void UseStore(string key, ref string value, ref bool fromStore)
{
    if (string.IsNullOrEmpty(value))
    {
        value = this.storeRepository.GetValue(key);
        fromStore = true;
    }
    else
    {
        this.storeRepository.StoreValue(key, value);
        fromStore = false;
    }
}
```

CheapMovies.Services

- Why use a custom local store?
  - I wanted Redis but it it is not free in Azure.
  - It can also be a local store solution that uses both in-memory cache and a key-value database like Redis.

CheapMovies.Services



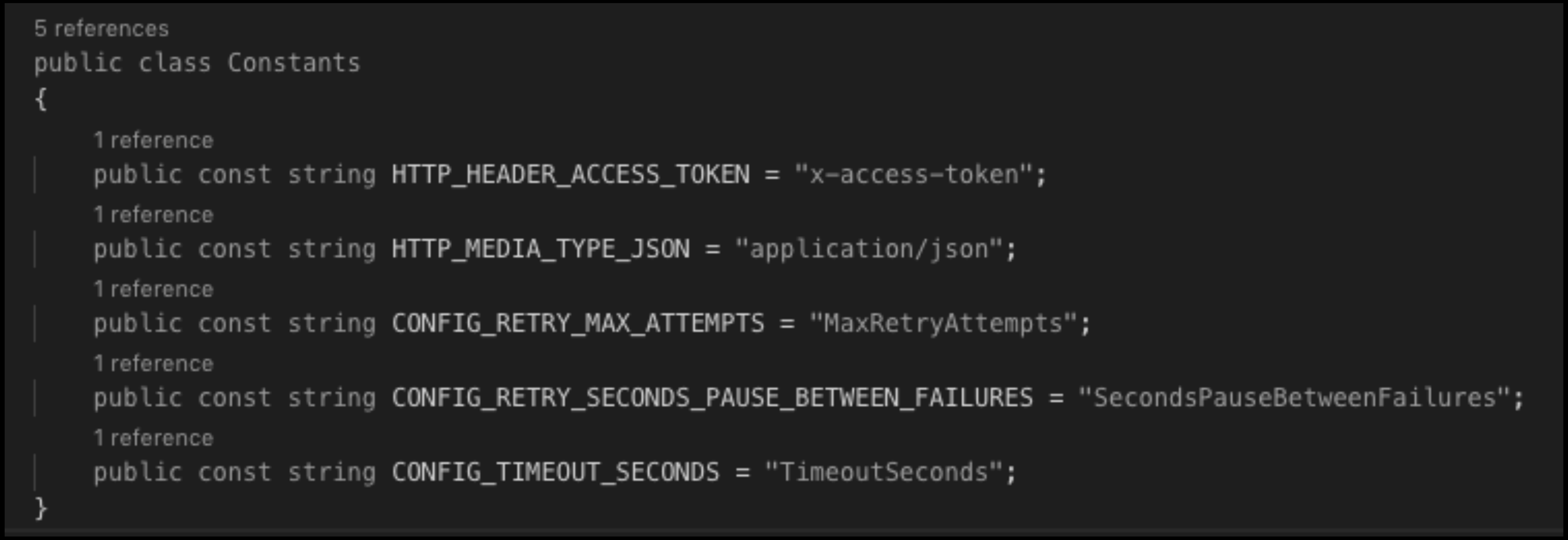
```
1 reference
public List<Movie> ParseMovies(string json)
{
    List<Movie> output = new List<Movie>();

    var moviesJson = JObject.Parse(json);
    JArray movies = (JArray)moviesJson["Movies"];

    foreach (var movie in movies)
    {
        output.Add(new Movie((JObject)movie));
    }

    return output;
}
```

CheapMovies.Store

- StoreRepository
  - Uses Entity Framework DbContext (StoreDbContext)
  - StoreValue (key, value) - adds or updates
  - GetValue (key) - returns value

CheapMovies.Common

- RetryHelper
- Constants

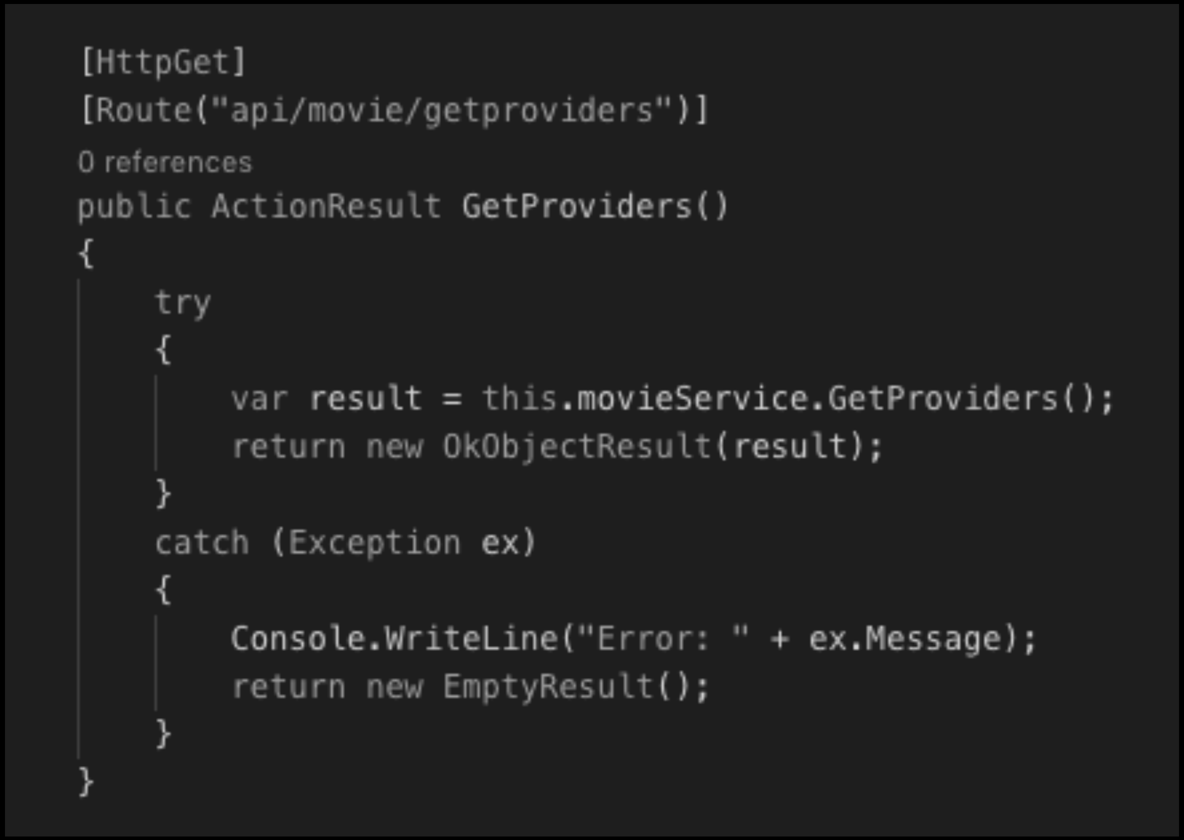


```
5 references
public class Constants
{
    1 reference
    public const string HTTP_HEADER_ACCESS_TOKEN = "x-access-token";
    1 reference
    public const string HTTP_MEDIA_TYPE_JSON = "application/json";
    1 reference
    public const string CONFIG_RETRY_MAX_ATTEMPTS = "MaxRetryAttempts";
    1 reference
    public const string CONFIG_RETRY_SECONDS_PAUSE_BETWEEN_FAILURES = "SecondsPauseBetweenFailures";
    1 reference
    public const string CONFIG_TIMEOUT_SECONDS = "TimeoutSeconds";
}
```

CheapMovies.Api

- MovieController - thin controller that calls services
  - GetProviders - api/movies/getproviders
  - GetMoviesAsync - api/movies/getmovies?serviceId=XX
  - GetMovieAsync - apy/movies/getmovie?serviceId=XX &movieId=YY

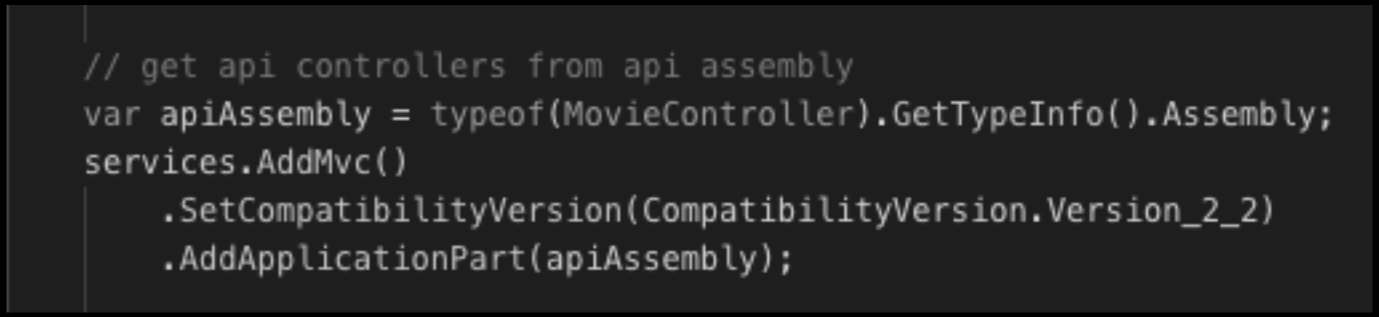
CheapMovies.Api



```
[HttpGet]
[Route("api/movie/getproviders")]
0 references
public ActionResult GetProviders()
{
    try
    {
        var result = this.movieService.GetProviders();
        return new OkObjectResult(result);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error: " + ex.Message);
        return new EmptyResult();
    }
}
```

CheapMovies.Api

- Controllers are loaded from CheapMovies.Web through application parts.

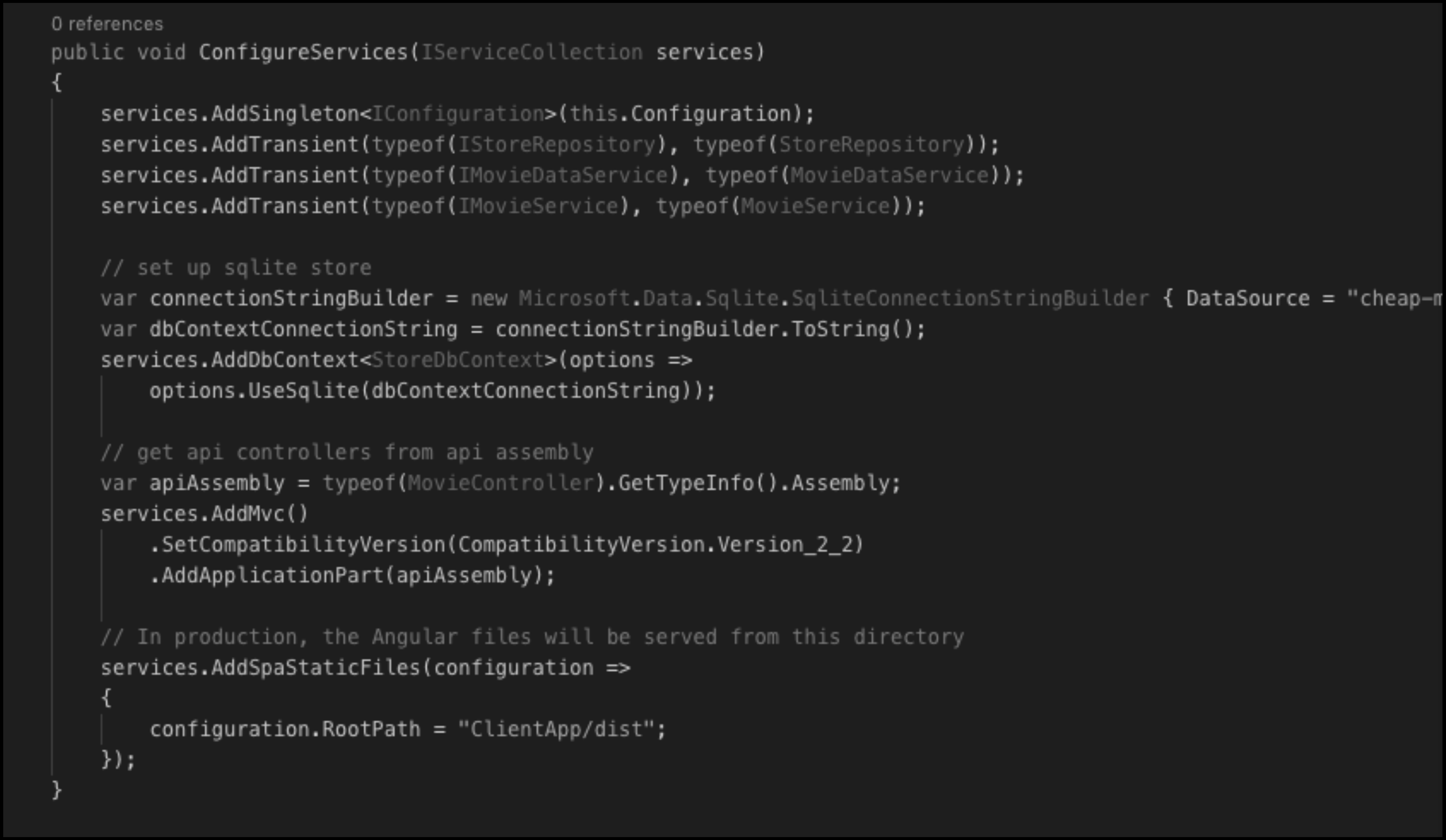


```
// get api controllers from api assembly
var apiAssembly = typeof(MovieController).GetTypeInfo().Assembly;
services.AddMvc()
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
    .AddApplicationPart(apiAssembly);
```

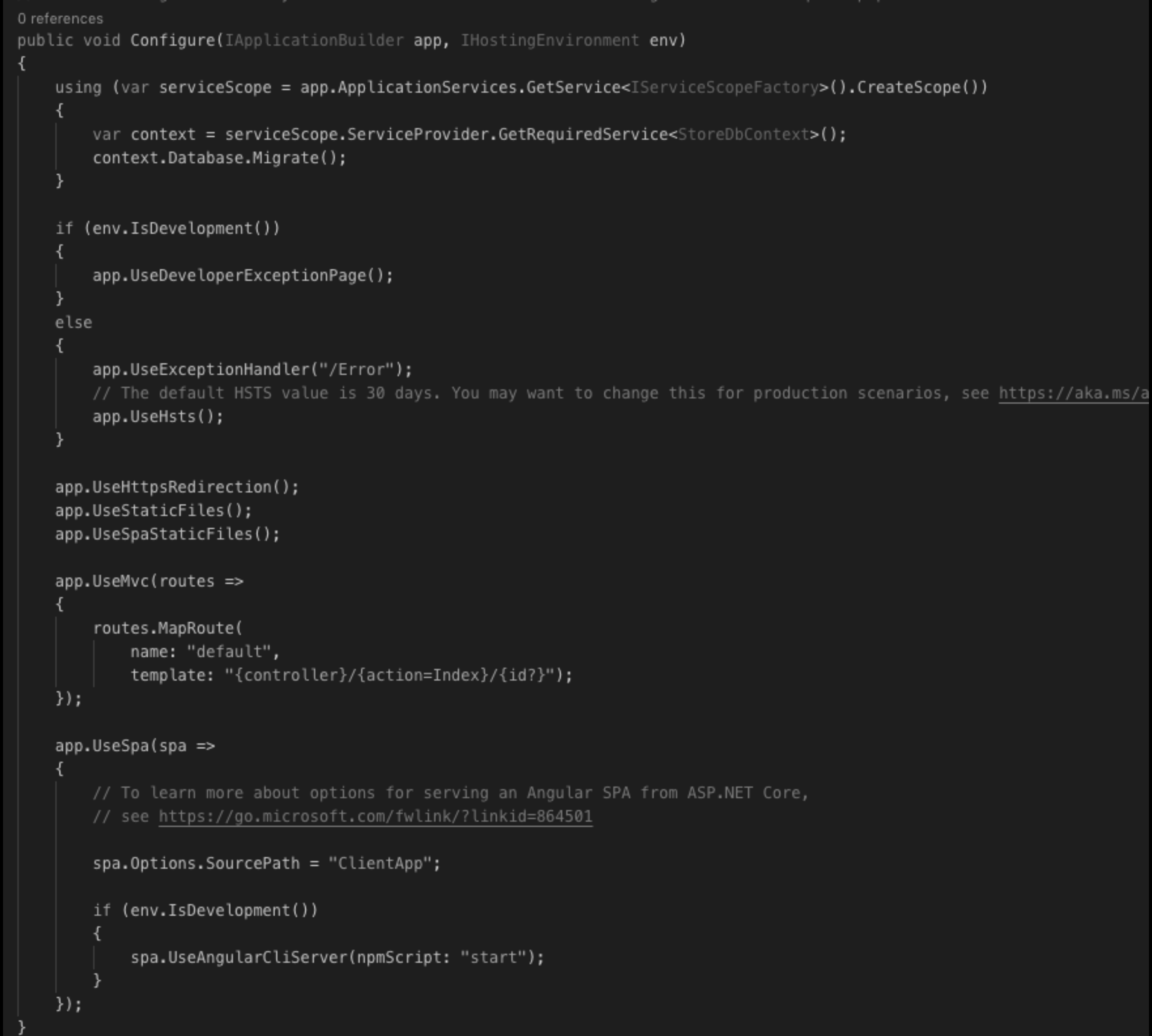
- If converting to a microservices architecture, the Movies API (and other APIs) can be loaded within its own host.

CheapMovies.Web

- Startup



```
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IConfiguration>(this.Configuration);
    services.AddTransient(typeof(IStoreRepository), typeof(StoreRepository));
    services.AddTransient(typeof(IMovieDataService), typeof(MovieDataService));
    services.AddTransient(typeof(IMovieService), typeof(MovieService));

    // set up sqlite store
    var connectionStringBuilder = new Microsoft.Data.Sqlite.SqliteConnectionStringBuilder { DataSource = "cheap-m
    var dbContextConnectionString = connectionStringBuilder.ToString();
    services.AddDbContext<StoreDbContext>(options =>
        options.UseSqlite(dbContextConnectionString));

    // get api controllers from api assembly
    var apiAssembly = typeof(MovieController).GetTypeInfo().Assembly;
    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
        .AddApplicationPart(apiAssembly);

    // In production, the Angular files will be served from this directory
    services.AddSpaStaticFiles(configuration =>
    {
        configuration.RootPath = "ClientApp/dist";
    });
}
```



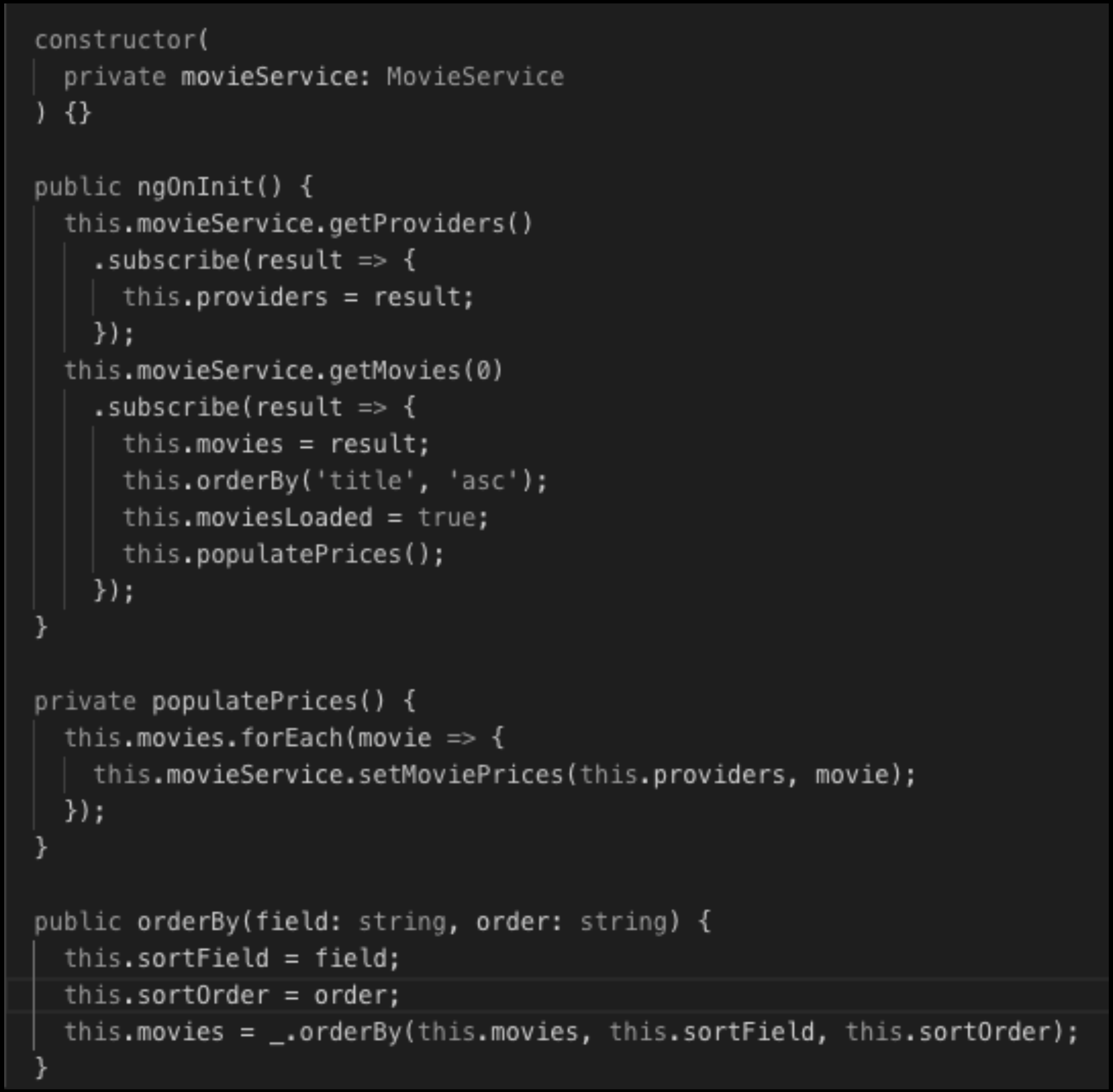
```
0 references
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    using (var serviceScope = app.ApplicationServices.GetService<IServiceScopeFactory>().CreateScope())
    {
        var context = serviceScope.ServiceProvider.GetRequiredService<StoreDbContext>();
        context.Database.Migrate();
    }

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/a
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseSpaStaticFiles();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller}/{action=Index}/{id?}");
    });

    app.UseSpa(spa =>
    {
        // To learn more about options for serving an Angular SPA from ASP.NET Core,
        // see https://go.microsoft.com/fwlink/?linkid=864501

        spa.Options.SourcePath = "ClientApp";

        if (env.IsDevelopment())
        {
            spa.UseAngularCliServer(npmScript: "start");
        }
    });
}
```

CheapMovies.Web

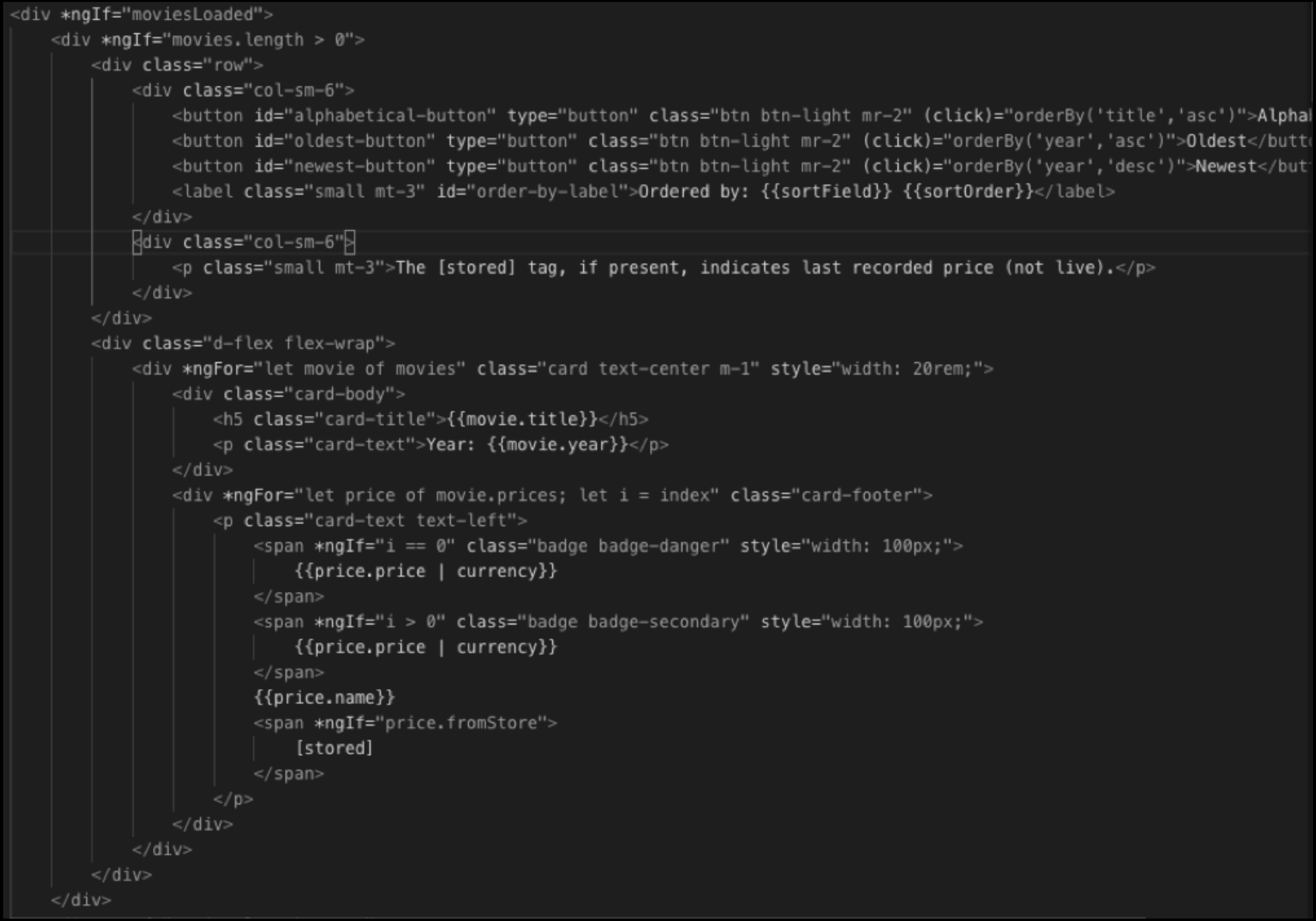
- Angular SPA
  - app.module
  - movies.component
  - movies.service
  - movie entity
  - price entity

CheapMovies.Web

- Component



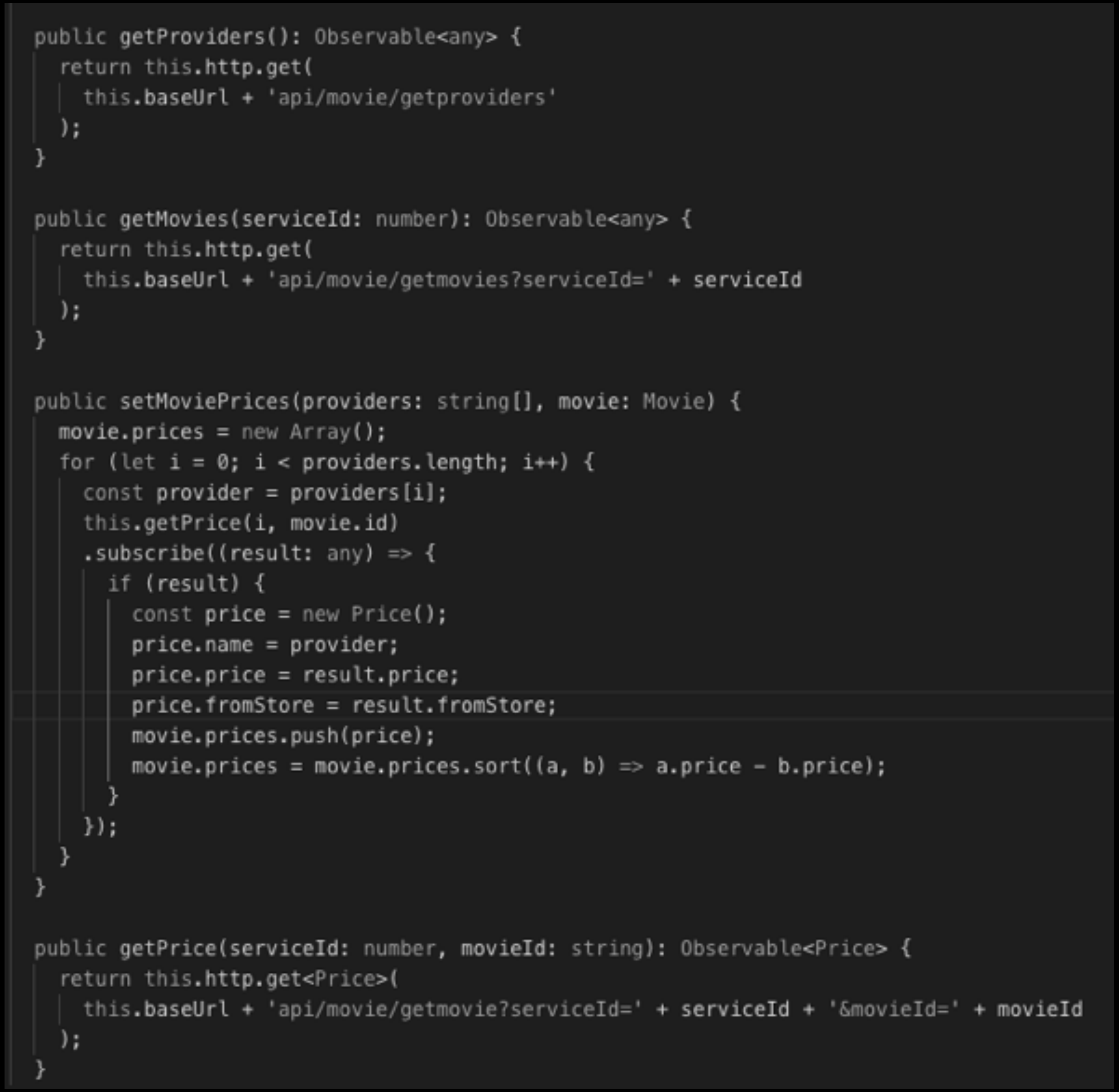
```
constructor(
  private movieService: MovieService
) {}

public ngOnInit() {
  this.movieService.getProviders()
    .subscribe(result => {
      this.providers = result;
    });
  this.movieService.getMovies(0)
    .subscribe(result => {
      this.movies = result;
      this.orderBy('title', 'asc');
      this.moviesLoaded = true;
      this.populatePrices();
    });
}

private populatePrices() {
  this.movies.forEach(movie => {
    this.movieService.setMoviePrices(this.providers, movie);
  });
}

public orderBy(field: string, order: string) {
  this.sortField = field;
  this.sortOrder = order;
  this.movies = _.orderBy(this.movies, this.sortField, this.sortOrder);
}
```



```
<div *ngIf="moviesLoaded">
    <div *ngIf="movies.length > 0">
        <div class="row">
            <div class="col-sm-6">
                <button id="alphabetical-button" type="button" class="btn btn-light mr-2" (click)="orderBy('title','asc')">Alpha
                <button id="oldest-button" type="button" class="btn btn-light mr-2" (click)="orderBy('year','asc')">Oldest</butt
                <button id="newest-button" type="button" class="btn btn-light mr-2" (click)="orderBy('year','desc')">Newest</but
                <label class="small mt-3" id="order-by-label">Ordered by: {{sortField}} {{sortOrder}}</label>
            </div>
            <div class="col-sm-6">
                <p class="small mt-3">The [stored] tag, if present, indicates last recorded price (not live).</p>
            </div>
        </div>
        <div class="d-flex flex-wrap">
            <div *ngFor="let movie of movies" class="card text-center m-1" style="width: 20rem;">
                <div class="card-body">
                    <h5 class="card-title">{{movie.title}}</h5>
                    <p class="card-text">Year: {{movie.year}}</p>
                </div>
                <div *ngFor="let price of movie.prices; let i = index" class="card-footer">
                    <p class="card-text text-left">
                        <span *ngIf="i == 0" class="badge badge-danger" style="width: 100px;">
                            {{price.price | currency}}
                        </span>
                        <span *ngIf="i > 0" class="badge badge-secondary" style="width: 100px;">
                            {{price.price | currency}}
                        </span>
                        {{price.name}}
                        <span *ngIf="price.fromStore">
                            [stored]
                        </span>
                    </p>
                </div>
            </div>
        </div>
    </div>
```

CheapMovies.Web

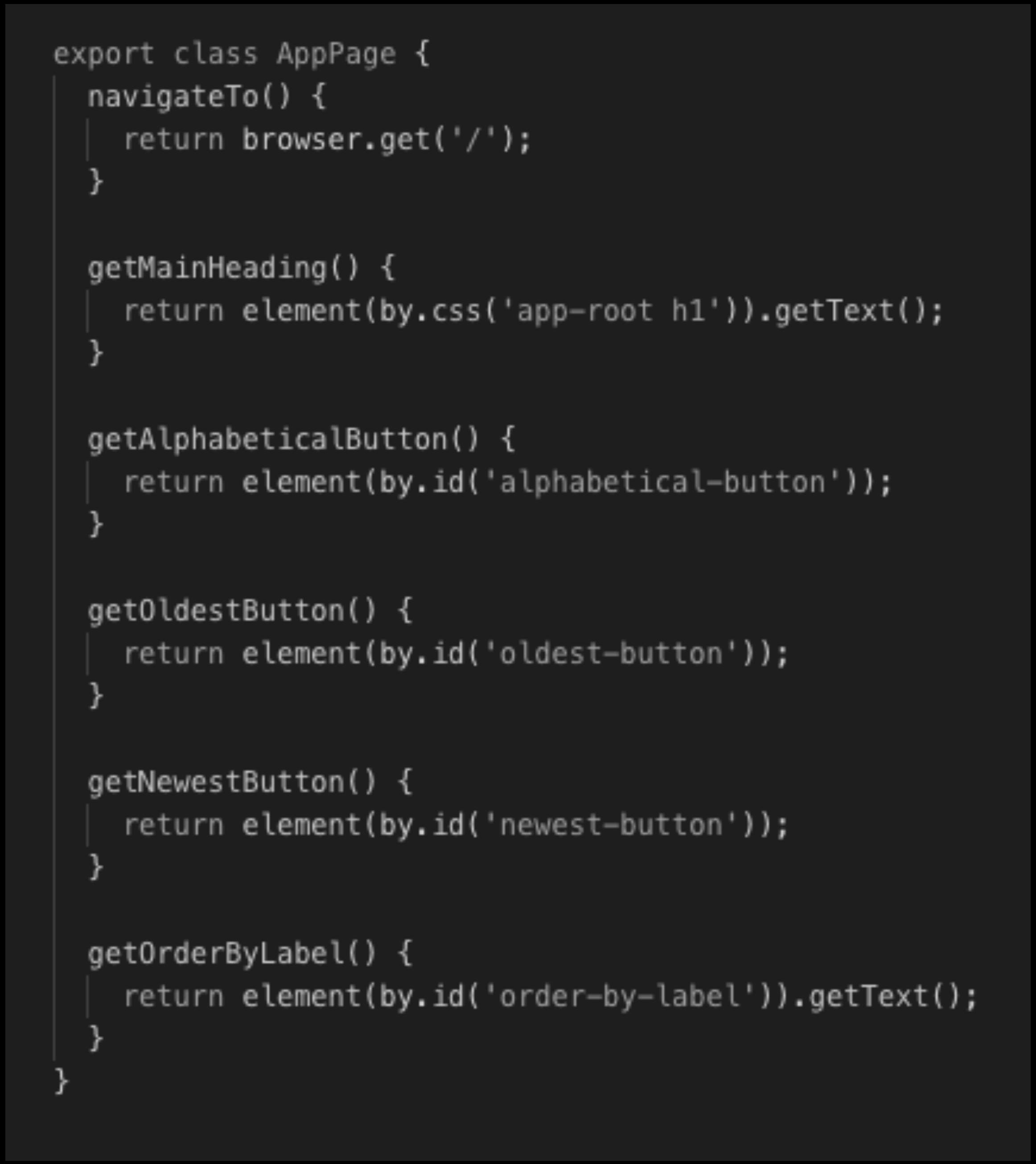
- Service



```
public getProviders(): Observable<any> {
  return this.http.get(
    this.baseUrl + 'api/movie/getproviders'
  );
}

public getMovies(serviceId: number): Observable<any> {
  return this.http.get(
    this.baseUrl + 'api/movie/getmovies?serviceId=' + serviceId
  );
}

public setMoviePrices(providers: string[], movie: Movie) {
  movie.prices = new Array();
  for (let i = 0; i < providers.length; i++) {
    const provider = providers[i];
    this.getPrice(i, movie.id)
    .subscribe((result: any) => {
      if (result) {
        const price = new Price();
        price.name = provider;
        price.price = result.price;
        price.fromStore = result.fromStore;
        movie.prices.push(price);
        movie.prices = movie.prices.sort((a, b) => a.price - b.price);
      }
    });
  }
}

public getPrice(serviceId: number, movieId: string): Observable<Price> {
  return this.http.get<Price>(
    this.baseUrl + 'api/movie/getmovie?serviceId=' + serviceId + '&movieId=' + movieId
  );
}
```

CheapMovies.Web

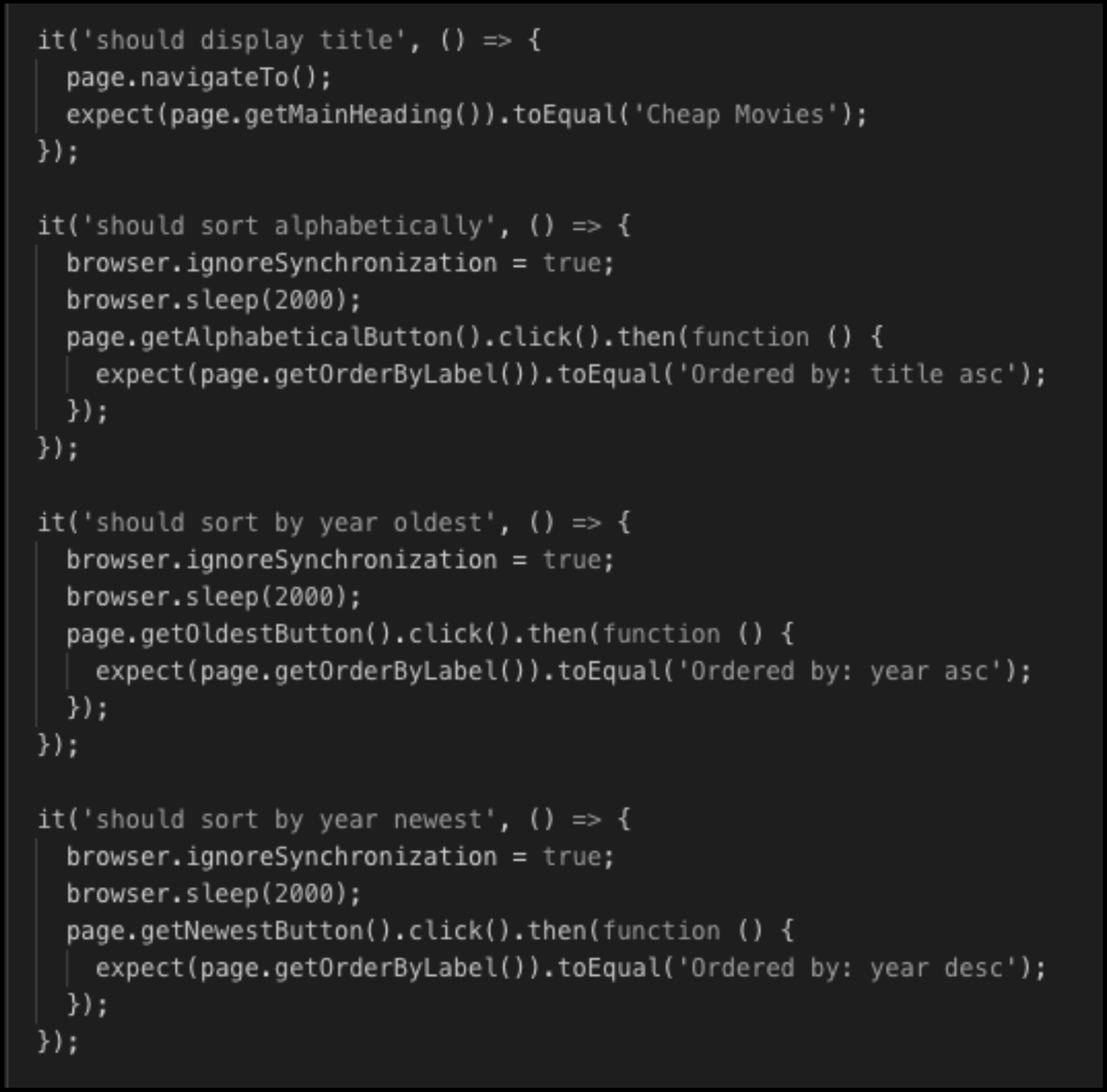
- E2E testing page object



```
export class AppPage {
  navigateTo() {
    return browser.get('/');
  }

  getMainHeading() {
    return element(by.css('app-root h1')).getText();
  }

  getAlphabeticalButton() {
    return element(by.id('alphabetical-button'));
  }

  getOldestButton() {
    return element(by.id('oldest-button'));
  }

  getNewestButton() {
    return element(by.id('newest-button'));
  }

  getOrderByLabel() {
    return element(by.id('order-by-label')).getText();
  }
}
```

CheapMovies.Web

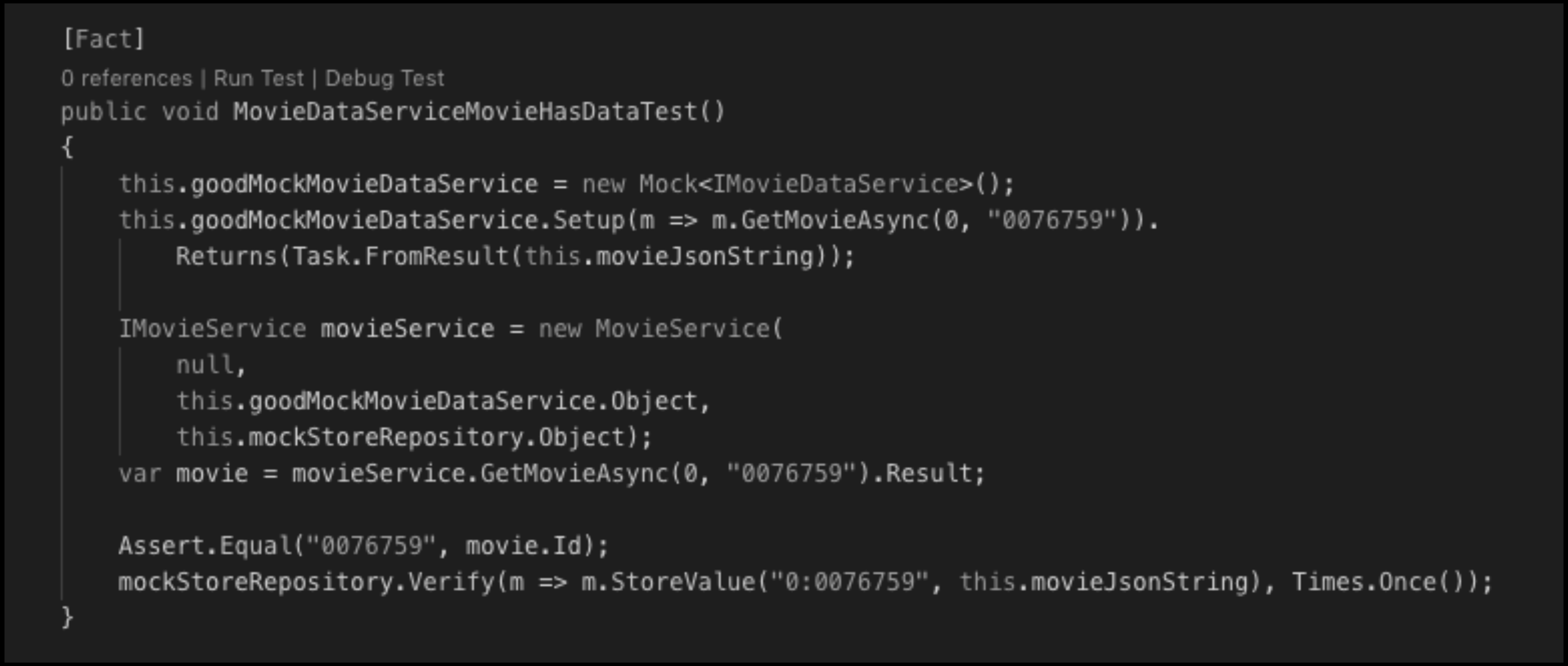
- E2E testing Protractor tests



```
it('should display title', () => {
  page.navigateTo();
  expect(page.getMainHeading()).toEqual('Cheap Movies');
});

it('should sort alphabetically', () => {
  browser.ignoreSynchronization = true;
  browser.sleep(2000);
  page.getAlphabeticalButton().click().then(function () {
    expect(page.getOrderByLabel()).toEqual('Ordered by: title asc');
  });
});

it('should sort by year oldest', () => {
  browser.ignoreSynchronization = true;
  browser.sleep(2000);
  page.getOldestButton().click().then(function () {
    expect(page.getOrderByLabel()).toEqual('Ordered by: year asc');
  });
});

it('should sort by year newest', () => {
  browser.ignoreSynchronization = true;
  browser.sleep(2000);
  page.getNewestButton().click().then(function () {
    expect(page.getOrderByLabel()).toEqual('Ordered by: year desc');
  });
});
```

CheapMovies.Tests

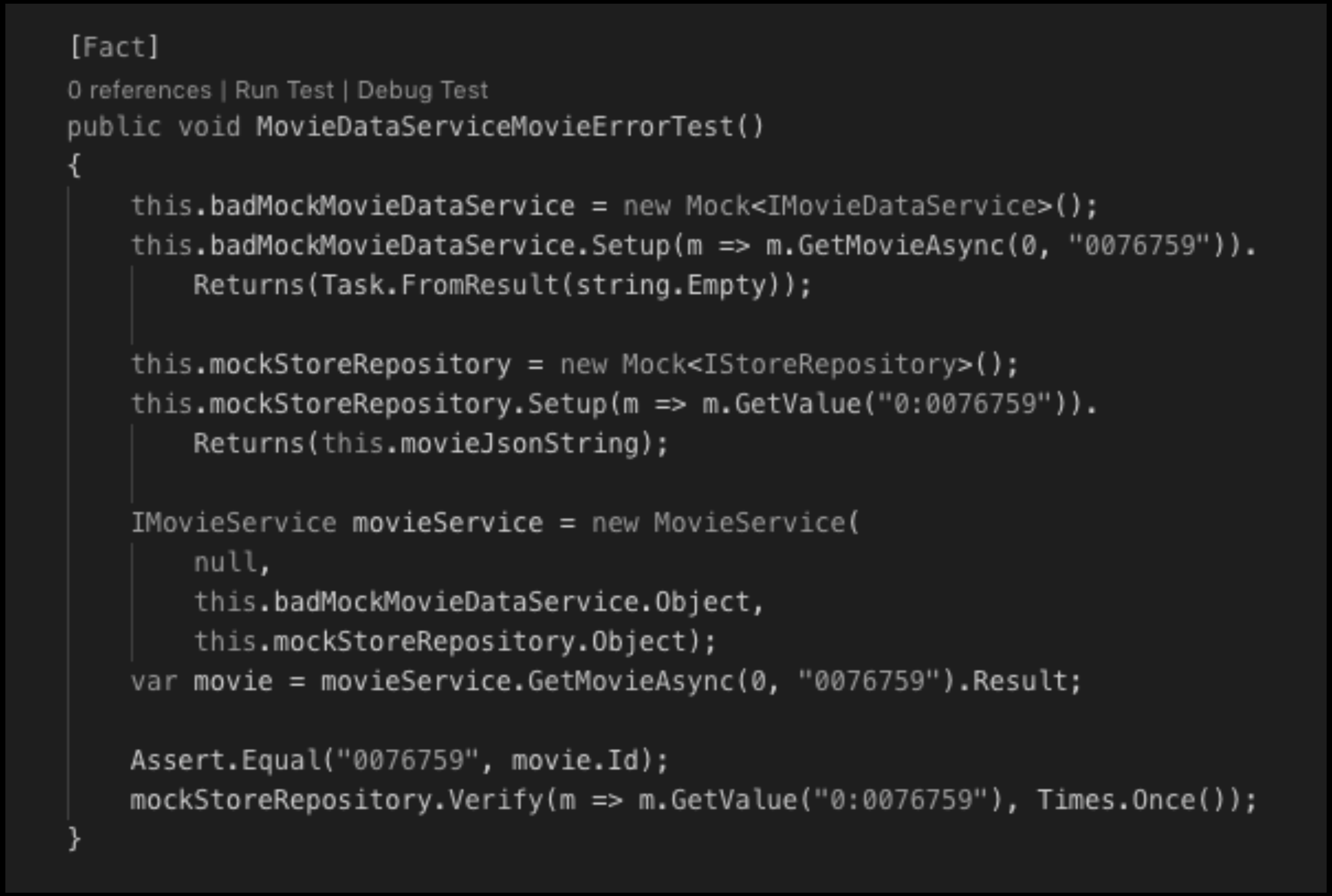
- Movie Data Service - “Movie” Has Data Test



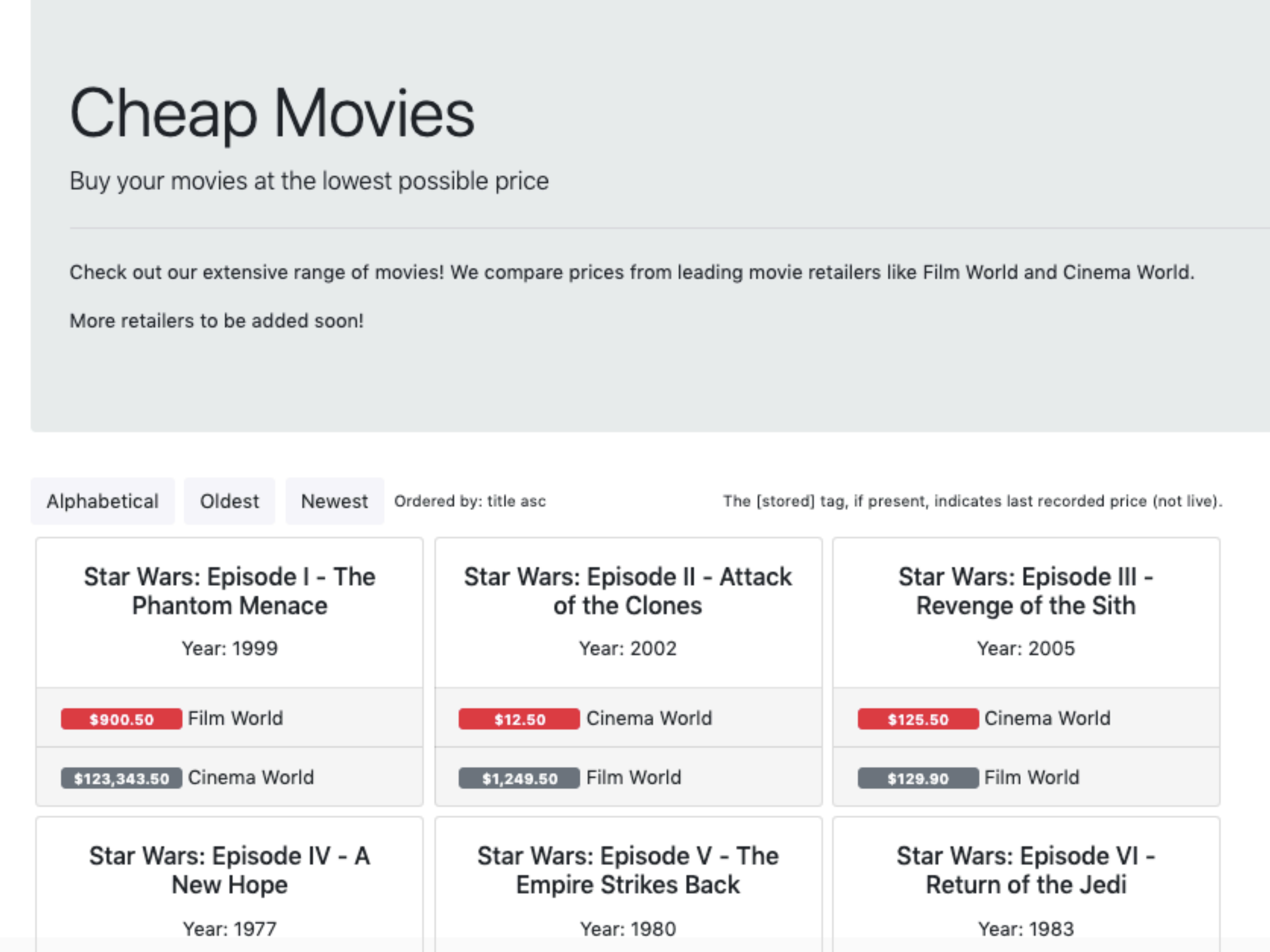
```
[Fact]
0 references | Run Test | Debug Test
public void MovieDataServiceMovieHasDataTest()
{
    this.goodMockMovieDataService = new Mock<IMovieDataService>();
    this.goodMockMovieDataService.Setup(m => m.GetMovieAsync(0, "0076759")).
        Returns(Task.FromResult(this.movieJsonString));

    IMovieService movieService = new MovieService(
        null,
        this.goodMockMovieDataService.Object,
        this.mockStoreRepository.Object);
    var movie = movieService.GetMovieAsync(0, "0076759").Result;

    Assert.Equal("0076759", movie.Id);
    mockStoreRepository.Verify(m => m.StoreValue("0:0076759", this.movieJsonString), Times.Once());
}
```

CheapMovies.Tests

- Movie Data Service - “Movie” Error Test



```
[Fact]
0 references | Run Test | Debug Test
public void MovieDataServiceMovieErrorTest()
{
    this.badMockMovieDataService = new Mock<IMovieDataService>();
    this.badMockMovieDataService.Setup(m => m.GetMovieAsync(0, "0076759")).
        Returns(Task.FromResult(string.Empty));

    this.mockStoreRepository = new Mock<IStoreRepository>();
    this.mockStoreRepository.Setup(m => m.GetValue("0:0076759")).
        Returns(this.movieJsonString);

    IMovieService movieService = new MovieService(
        null,
        this.badMockMovieDataService.Object,
        this.mockStoreRepository.Object);
    var movie = movieService.GetMovieAsync(0, "0076759").Result;

    Assert.Equal("0076759", movie.Id);
    mockStoreRepository.Verify(m => m.GetValue("0:0076759"), Times.Once());
}
```



Cheap Movies

Buy your movies at the lowest possible price

Check out our extensive range of movies! We compare prices from leading movie retailers like Film World and Cinema World.

More retailers to be added soon!

Alphabetical Oldest Newest Ordered by: title asc The [stored] tag, if present, indicates last recorded price (not live).

Star Wars: Episode I - The Phantom Menace

Year: 1999

$900.50 Film World

$123,343.50 Cinema World

Star Wars: Episode II - Attack of the Clones

Year: 2002

$12.50 Cinema World

$1,249.50 Film World

Star Wars: Episode III - Revenge of the Sith

Year: 2005

$125.50 Cinema World

$129.90 Film World

Star Wars: Episode IV - A New Hope

Year: 1977

Star Wars: Episode V - The Empire Strikes Back

Year: 1980

Star Wars: Episode VI - Return of the Jedi

Year: 1983