FS Personal Information Manager

Technical Guide

Introduction – The FS Personal Information Manager application is used to manage your own list of TV Shows, Books, and Movies. You can use this application to easily keep track of what you have watched in one spot. You can also use this app to record future releases so you can always be on top of your favorite TV Shows, Books, and Movies.

This application was developed as a reference implementation by Frank Stadler to demonstrate the creation of a small MVC / Entity Framework application.

Audience – The technical guide is targeted towards a technical audience, primarily those with a background in .NET development interested in an in depth review of the reference application.

Tools and Packages - This application was developed in Visual Studio 2015 as an MVC 5 / Entity Framework 6 Code First application. SQL Server Express was used as the database platform. SQL Server Express is suitable for use in a development or test environment, but if you were deploying this application as a real production environment the full version of SQL Server should be utilized. (or Azure SQL Database if deploying the application to the Azure environment)

Wherever possible additional software incorporated in this application was added using the NuGet package manager. NuGet is a widely adopted package manager for within the .NET development space.

Package	Description / Notes
Entity Framework 6	Entity framework (EF) is a feature rich framework for interacting with relational databases. The use of EF can automate the 'plumbing' code that your application needs to read and write data from a relational database. EF is a type of ORM (Object relational mapping) **Install-Package EntityFramework**
PagedList.Mvc	PagedList.Mvc is an upgrade list that can be used in MVC views that has features for formatting a list and for incorporating paging behavior.

	Install-Package PagedList.Mvc
Moq	Moq is a fairly popular mocking framework used with .NET programming. The use of a mocking framework is often a much better option that trying to write your own custom build test mocking process. **Install-Package Moq**
FakeDbSet	FakeDbSet is an additional mocking component used specifically to simulate an in-memory DbSet object. (used for testing entity framework related behavior**) Install-Package FakeDbSet**
Castle.core.3.3.3	Dependency of Moq
jQuery DataTable	A JavaScript library used to format a client side table. (as opposed to the PagedList.Mvc package which utilizes a server side pattern)

Installing and Running the Application – When installing and running this application you should copy the 'PersonalInformationManager' folder and its contents to c:\projects\PersonalInformationManager. If you prefer to run the application from a different location, you can do so after modifying two entries in the web.config file. Update the 'SeedDataFolder' entry, which identifies the physical path where the migration seed data is stored and the 'ErrorLogPath' entry, which identifies the output location of the application error log.

These two path values are utilized by the application and stored in the web.config file. Storing application configuration or setup in the web.config file is a better approach than hard coding the values within the source code of the application. In ASP.NET MVC we can easily obtain the path of the running application, but that is typically the IIS Express runtime location, not the location of the source files. Therefore we need to add a reference to these folders in the web.config file.

Solution overview – The application has 4 core entities, each of which has a corresponding control class, model class, and views.

Name	Description
Source	A source represents the origin of a Book, Movie, or TV show. This can be a physical or online store or an online subscription service. The same source can be used across all of these categories, as many physical or online stores can offer all three options.
Movie	This represents a Movie. A movie can be viewed in a physical movie theater, from a disc, or from a digital download.

Show	This represents a TV show. A show can be viewed from over the air or cable service, from a disc, or from a digital download.
Book	This represents a book. A book can be viewed from a print copy or from a digital download.

A closer look at the Model classes – You will notice that each of the Movie, Show, and Book classes contains a navigation property back to the Source entity from which it originated. Each of these classes also contain a byte array to store the associated image file in the sql server tables.

The Models folder also contains IPersonalInformationManagerContext which defines an interface for the database context class and the PersonalInformationManagerContext class.

The PersonalInformationManagerContext class contains the DbSet objects which are used to load data from and write data to the entity framework. A DbSet usually corresponds to a single table in the underlying database.

The IPersonalInformationManagerContext interface is utilized because we are actually working with two different contexts. The real entity framework context will be used to read and write data to the underlying SQL Server database. A second mock context is used during unit tests, which allows us to test our controller code without actually reading or writing from the database. This makes the code much more testable.

A closer look at the Controller classes – The controller classes follow a fairly typical organization. I would highlight the following features of the controller classes which I would regard as a best practice.

Each of the controller classes contains two constructors. The default constructor loads the standard entity framework context. The second constructor accepts a context object as a parameter. This second constructor is used in the unit tests so that we can test the controller class without actually modifying data in the SQL server database. This approach with the constructor is a technique that is also called ‘dependency injection’. Without this approach the controller classes would be much harder to unit test.

The controller classes also override the OnException method for the purpose of logging the error before displaying an error page to the end user.

A closer look at the views – The views for the application follow the general MVC approach. The index views use the PagedList.mvc package to control the display and formatting of the list. The index views also incorporate functionality to allow for sorting and searching.

The views also incorporate a bit of style / design for the application including the use of a consistent look and feel of having a top row of buttons / links depending on the particular view we are working with.

HomeDashboard ViewModel – This solution contains an additional folder, 'ViewModel'. While not part of the out of the box folder structure of MVC, this is a fairly common convention to separate ViewModels from standard Model classes.

A ViewModel is a class that contains a unique combination of other model classes and primitive data types, typically with a particular view in mind. In this case we are displaying a dashboard on the application's Home/Index view. This dashboard displays the most recently viewed Movie, Book, and TV Show. Rather than taking a less professional approach such as loading the data in the ViewBag, I created a ViewModel class to represent this combination of data to be displayed on the Home/Index view.

Bootstrap - This application utilizes Bootstrap. Bootstrap is a fairly popular HTML / CSS framework for the display of web pages. Bootstrap is a responsive framework, meaning that the page content is optimized for different device form factors ranging from desktop devices all the way down to a phone form factor.

The use of an existing responsive framework like bootstrap is generally a better choice than trying to develop a responsive solution from scratch using css media queries.

jQuery DataTable View – An alternate approach to displaying data is illustrated with the Shows/IndexDataTable view. The jQuery DataTable library is an open source JavaScript library focused on displaying a set of data in an interactive client side table. The DataTable library operates on top of the jQuery library. As opposed to the PagedList.Mvc package used for the main application, which triggers an HTTP post when moving from page to page or changing the sort order, the DataTable view demonstrates the approach of loading the data once and sending it client side. This allows the client's browser to sort, filter, and page through the data without triggering an HTTP post back to the server. This reduces the number of calls to the server and can improve the responsiveness of the client.

I have highlighted a few of the advantages of client side display of data, but there are also some limitations. This approach should not be taken for secure data, as this client side approach allows the user to capture all of the data. And any application that uses this technique should also have server side controls to verify the authenticity and integrity of the data that is included with server side requests to rule out tampering or manipulation of the data. (the ASP.NET anti-forgery token is a good approach) There are also limitations in terms of performance, loading a large set of data may cause the user to perceive the application as slow. This can be overcome with more sophisticated patterns such as loading the first chuck of data and displaying it immediately, followed by asynchronous calls that load other chunks of data, creating a better user experience.

Migrations and Seed Data – This application was developed in an EF code first style. The application uses the Migrations feature which helps to manage changes in the database's structure. In the package manager console you enter 'Enable Migrations' to enable this feature. After you make one or more changes to the model classes you would return to the package manager console and enter 'Add Migration X' (where X is a name you give for that particular change). This generates a .cs file which contains the code to apply the changes to your database. Examples would include creating tables, adding columns, or defining a foreign key. You apply the migration to your code by entering the 'Update Database' command in the package manager console. This command runs the migration .cs file.

In the Migrations folder in the solution you will find a configuration.cs file. This configuration class contains a Seed method which runs as part of the 'Update Database' command. This seed method written for the application generates a complete set of test data for the application. This enables the developer to start from either a blank or existing database and have a consistent set of test data covering all the major features of the application.

Error Logging – Each controller has an OnException method override which logs the details of the error to a text file before forwarding the user to an error page. In this instance we are logging the errors to a text file specified in the web.config file under the 'ErrorLogLocation' element. Logging errors in this fashion is typically adequate for small applications or applications under development.

In configuring error logging for your application you should also investigate if there are logging standards for your organization. Some shops utilize log parsing software which aggregate the log files, others may utilize a specific format or logging framework, and others may favor logging to a sql database or a nosql datastore or even an error microservice. Standards and preferences can vary widely so it is often best to identify the in-house standards before incorporating them into your application

Unit testing - The application includes a test project that includes 35 unit tests. The use of automated unit testing can provide many benefits during the development of an application and on an ongoing basis as the application is revised, enhanced, and maintained.

Unit tests provide an automated red light / green light verification to your development effort. Because the tests are automated, the cost of testing goes down and you can run your unit tests frequently. More rapid testing can allow a developer to spot issues sooner in the development process.

Systems without automated unit tests can become 'brittle' over time especially as systems are handed off from one individual or team to another. Automated unit tests are often more valuable to developers than written documentation because requirements and documentation can change over time and the developer often has to

do additional research to find the ‘source of truth’ as to how the system should function and be tested.