**Quinlan v1**

Written By Jeff Campbell

The Quinlan application is an effort to manage the inventory of a sports card collection that my client has inherited. It will also provide tools to bundle parts of the inventory together as sellable products, if my client decides to do so.

Overview

The application is written using .NET Core v3 and utilizes the following:

|  |  |  |
| --- | --- | --- |
| Server Side | Client Side | Testing |
| - C#  - ASP.Net Core  - ASP.Net Identity  - Kestrel Web Server  - Logging  - Entity Framework Core  - Dependency Injection Framework  - MVC  - Web API  - LINQ  - T-SQL | - HTML5  - CSS3  - Razor w/ Tag Helpers  - Partial Views  - JavaScript  - Angular  - TypeScript  - Bootstrap | - XUnit  - InMemory E/F |

Projects

The solution contains the following projects for the initial release. :

|  |  |
| --- | --- |
| Quinlan.Data | Data models  Entity Framework  Code value, Data access, and Query services |
| Quinlan.Domain | Domain models  CRUD, Domain Search, and Helper services |
| Quinlan.Initialize | Import Files  Import SQL Script |
| Quinlan.Web.MVC | MVC website  View Models  Page, Form, and Helper services |
| Quinlan.Web.SPA | Angular website  Web API  API services |
| Quinlan.Test | Unit tests for Data, Domain, MVC, and SPA projects  Seed services |

**Quinlan.Data**

The DATA project contains the Entity Framework context, the data model, and the data access services used to query and update the database. The data model has been written to achieve a relational database with as much shared information as possible.

QdbContext

Entity Framework context that defines the following DbSets and generates the corresponding database tables. Code table values are seeded when the database is created.

|  |  |  |
| --- | --- | --- |
| Data Tables | Code Tables | Import Tables |
| Collectibles  Colleges  Grades  Graders  Organizations  Manufacturers  People  PersonSports  Products  Sets  Teams | CardTypes  CollectibleStatuses  CollectibleTypes  League  ProductStatuses  ProductTypes  Sports | ImportCollectibles  ImportColleges  ImportPeople  ImportProducts  ImportTeams |

Exceptions

* InvalidCodeValueChangeException
* InvalidDeleteException
* InvalidIdException

Interfaces

* ICodeService
* IDataService
* IDatabaseService
* IQueryService
* ICollectibleQueryService

Data Models

These are POCO classes used to define the DbSet<T> lists in the context and corresponding database tables. Note that all data models include Id property which becomes the primary key in the database by default.

* CardType
* Collectible
* CollectibleStatus
* CollectibleType
* College
* Grade
* Grader
* ImportCollectible
* ImportCollege
* ImportPerson
* ImportProduct
* ImportTeam
* League
* Manufacturer
* Organization
* Person
* PersonSport
* Product
* ProjectStatus
* ProductType
* Set
* Sport
* Team

Filter Models

These are classes that define filter options for query services.

* CollectibleFilterOptions
* PersonFilterOptions
* TeamFilterOptions

Code Services

These are static classes that define and expose code values that cannot be modified.

* CardTypeCodeService
* CollectibleStatusCodeService
* CollectibleTypeCodeService
* LeagueCodeService
* ProductStatusCodeService
* ProductTypeCodeService
* SportCodeService

Data Services

These are classes that inherit from IDataService and provide the basic Select, Insert, Update, and Delete functionality for a specific data point.

* CollectibleDataService
* CollegeDataService
* GradeDataService
* GraderDataService
* PersonDataService
* ProductDataService
* TeamDataService

Query Services

These are classes that inherit from IQueryService and query a specific data source using the filter options.

* CollectibleQueryService
* PersonQueryService
* TeamQueryService

**Quinlan.Initialize**

The INITIALIZE project contains functionality to seed the database for development purposes. Once the application is launched, this project must be retired.

InitFiles

These are CSV files containing initialization data. The data was extracted from an MS Access database and has been augmented and pruned since then.

* Cards.csv
* Colleges.csv
* People.csv
* Products.csv
* Teams.csv

SQL

Import script that pulls data from InitFiles into an empty database

* Add2DB.sql

**Quinlan.Domain**

The DOMAIN project contains the domain model as well as services that are used

to hydrate those classes. The domain model is intended to include flatter and more descriptive classes. For example, the Card, Figurine, and Magazine domain models are all constructed using Collectible data.

Exceptions

Exception classes that are specific to the domain

* DeleteNotSupportedException
* GetNotSupportedException
* ItemNotFoundException
* PostNotSupportedException
* PutNotSupportedException

Interfaces

* ICollectibleSearchService
* ICrudService
* ISearchService
* ISummaryService

Domain Models

Note that all domain models must support ToString() override.

* Card
* CardType
* College
* DataSummary
* Figurine
* Grade
* Grader
* ImportCard
* League
* Magazine
* Person
* Product
* Sport
* SubTotal
* Team

Filter Options Models

* CardSearchFilterOptions
* FigurineSearchFilterOptions
* MagazineSearchFilterOptions
* PersonSearchFilterOptions
* TeamSearchFilterOptions

Search Models

* CardSearch
* FigurineSearch
* MagazineSearch
* PersonSearch
* TeamSearch

CRUD Services

* CardService
* CollegeService
* FigurineService
* GradeService
* GraderService
* LeagueService
* MagazineService
* ManufacturerService
* PersonService
* ProductService
* SportService
* TeamService

Helper Services

* QueryFilterService

Database Services

* CollectibleSummaryService

Search Services

* CardSearchService
* FigurineSearchService
* MagazineSearchService
* PersonSearchService
* TeamSearchService

**Quinlan.MVC**

The MVC project contains an ASP.Net Core website using the built-in MVC functionality. Note the following:

1. Controllers have very little functionality. They call a Page service method to generate a View Model which is passed to the corresponding View. Try/Catch surrounds all logic with exceptions written to the logger.
2. Models have no functionality. All functionality to hydrate models is handled by MVC services.
3. Views utilize Razor and Tag Helpers to render pages. Partial views utilized for shared search criteria section. More are likely to be written.
4. No JavaScript has been written. All functionality is server-side

Controllers

CardsController

CollegesController

FigurinesController

HomeController

LeaguesController

MagazinesController

PeopleController

ProductsController

SportsController

TeamsController

Exceptions

None – expect some to be identified.

Interfaces

Interfaces are defined for each type of page

* IDetailsService
* IEditService
* IFormService
* IHomeService
* IIndexService
* IViewService

Binding View Models

Used to receive data from a form in a submitted from an HTML Form

* CardFilterOptionsViewModel
* CardViewModel
* CollegeViewModel
* FigurineFilterOptionsViewModel
* FigurineViewModel
* LeagueViewModel
* MagazineFilterOptionsViewModel
* MagazineViewModel
* PersonFilterOptionsViewModel
* PersonViewModel
* ProductFilterOptionsViewModel
* SportViewModel
* TeamFilteroptionsViewModel
* TeamViewModel

ListItem View Models

Models for data to be listed on a page.

* CardListItemViewModel
* FigurineListItemViewModel
* MagazineListItemViewModel
* PersonListItemViewModel
* ProductListItemViewModel
* TeamListItemViewModel

Page View Models

View models for page level data bindings.

* CardEdit
* CardView
* CollegeDetails
* CollegeEdit
* FigurineIndex
* Home
* LeagueDetails
* MagazineIndex
* PersonDetails
* PersonEdit
* PersonIndex
* ProductIndex
* SportDetails
* Summary
* TeamDetails
* TeamEdit
* TeamIndex

Shared Models

View models that are shared across pages

* CardSearchViewModel
* SearchTotalsViewModel
* SubTotalViewmodel

Form Services

Services that validate and save data submitted from an HTML form.

* CardFormService
* CollegeFormService
* PersonFormService
* TeamFormService

Helper Services

Services that are used by other services.

* FormatService
* MvcService
* SearchFilterService

Page Services

Services that generate the data bindings required to render a page.

* CardEditService
* CardViewService
* CollegeDetailsService
* CollegeEditService
* FigurineIndexService
* HomeDetailsService
* HomeIndexService
* LeagueDetailsService
* MagazineIndexService
* PersonDetailsService
* PersonEditService
* PersonIndexService
* ProductIndexService
* SportDetailsService
* TeamDetailsService
* TeamEditService
* TeamIndexService

Views

Views which contain the markup and razor code required to render a page

* Card Edit
* Card View
* College Details
* College Edit
* Figurines Index
* Home
* Summary
* Magazines Index
* Person Details
* Person Edit
* People Index
* Product Index
* Sport Details
* Team Details
* Team Edit
* Teams Index

Shared Views

These are partial views which are shared across pages.

* \_FilterOptions
* \_Layout
* \_SearchTotals

**Quinlan.SPA**

Single Page Application (SPA) written using ASP.Net Core for the back-end and Angular/TypeScript for the front-end.

ClientApp

Interfaces

* card.interface
* college.interface
* person.interface
* sport.interface
* team.interface

Components

* home.component
* navMenu.component
* college.component
* person.component
* sport.component
* team.component

Services

* colleges.service
* people.service
* sports.service
* teams.service

Controllers

* CollegesController
* PeopleController
* SportsController
* TeamsController

Interfaces

* IApiService

Models

* CardDTO
* CollegeDTO
* PersonDTO
* SportDTO
* TeamDTO

Services

* CollegesApiService
* PeopleApiService
* SportsApiService
* TeamsApiService

**Quinlan.Test**

All functionality has been developed in service classes which can be individually tested. The corresponding unit tests utilize an InMemory instance of the database which are initialized using Seed services which interact directly with QdbContext. Tests are performed on a separate instance of the InMemory database context which is injected into the service being tested.

Note that InMemory databases are persisted for the life of the unit test run. It is unclear whether maintaining the tests in a single project is the right way to go from a DevOps perspective.

Interfaces

* ICodeServiceTests
* IDataServiceTests
* IQueryServiceTests
* ICrudServiceTests
* ISearchServiceTests
* IDetailsServiceTests
* IEditServiceTests
* IIndexServiceTests

Services

* SeedService

Code Service Tests

* CardTypeCodeServiceTests
* CollectibleStatusCodeServiceTests
* CollectibleTypeCodeServiceTests
* LeagueCodeServiceTests
* ProductStatusCodeServiceTests
* SportCodeServiceTests

Data Service Tests

* CollectibleDataServiceTests
* CollegeDataServiceTests
* PersonDataServiceTests
* ProductDataServiceTests
* TeamDataServiceTests

Query Service Tests

* CollectibleQueryServiceTests
* PersonQueryServiceTests
* TeamQueryServiceTests

CRUD Service Tests

* CardServiceTests
* FigurineServiceTests
* LeagueServiceTests
* MagazineServiceTests
* PersonServiceTests
* SportServiceTests
* TeamServiceTests

Domain Database Service Tests

* CollectibleSummaryServiceTests

Domain Helper Service Tests

* QueryFilterServiceTests

Domain Search Service Tests

* CardSearchServiceTests
* FigurineSearchServiceTests
* MagazineSearchServiceTests
* PersonSearchServiceTests
* TeamSearchServiceTests

MVC Page Service Tests

* CardEditServiceTests
* CardViewServiceTests
* CollegeDetailsServiceTests
* CollegeEditServiceTests
* FigurineIndexServiceTests
* HomePageServiceTests
* LeagueDetailsServiceTests
* MagazineIndexServiceTests
* PersonDetailsServiceTests
* PersonEditServiceTests
* PersonIndexServiceTests
* SportDetailsServiceTests
* TeamDetailsServiceTests
* TeamEditServiceTests
* TeamIndexServiceTest

MVC Helper Service Tests

* FormatServiceTests
* MvcServiceTests
* SearchFilterServiceTests

Form Service Tests

* CardFormServiceTests
* CollegeFormServiceTests
* PersonFormServiceTests
* TeamFormServiceTests

API Tests

* SportsApiServiceTests