Table of Contents

[2 Setup 2](#_Toc431825036)

[3 Project Layout 2](#_Toc431825037)

[3.1 Blue.Data 2](#_Toc431825038)

[3.2 Blue.General 3](#_Toc431825039)

[3.3 Blue.Library 3](#_Toc431825040)

[3.4 Blue.TestSuite 4](#_Toc431825041)

[3.5 Blue.WebAPI 4](#_Toc431825042)

[3.6 HELIX Development Guidelines 4](#_Toc431825043)

Blueprint

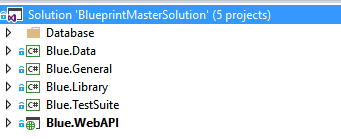
# Setup

Blueprint requires the following installed on a developer’s workstation:

* Visual Studio 2015 / .NET 4.5 or 4.6
* IIS Express v9 or newer
* GIT (github.com)
* Github @username with access to the repository
* SQL Server 2012 or newer
* Recommended: npm, bower, nuget, and curl (or Fiddler)

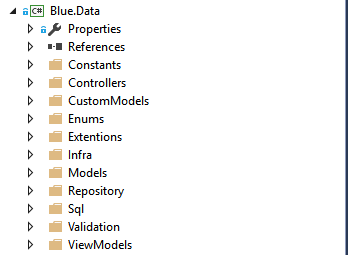
# Project Layout

The Blueprint project follows the standard HELIX project layout:

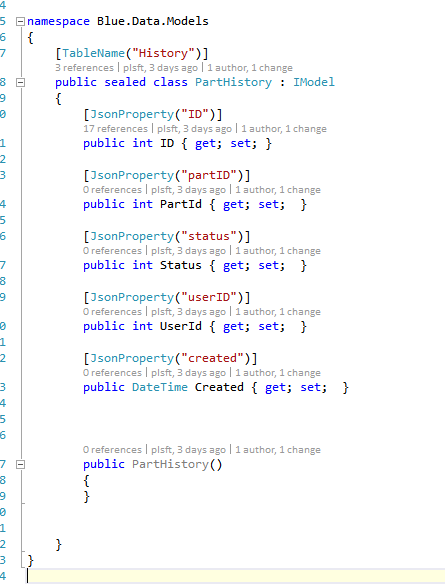


## Blue.Data

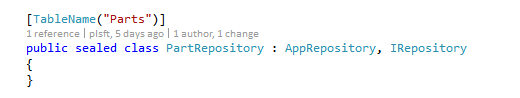
The DATA project contains all database access layer code, including enumerations, generics and controllers for every “model.” For the purposes of HELIX, a model has a 1:1 relationship to a SQL TABLE.



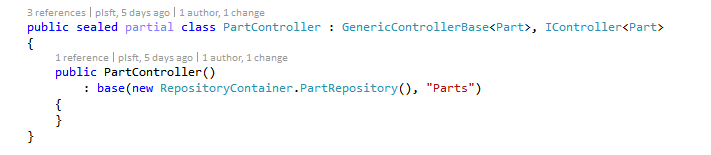
Every table must have the following elements to be accessible by the HELIX framework: A repository class (\repository), a model class (\models), and a generic controller (\controller). For example:



A sample “PartHistory” model



A sample “Parts” repository



A sample “Parts” controller generic class

## Blue.General

The GENERAL project contains all general utility classes and methods like configuration, error handling, mailing and application general settings. General project should only be used as a reference for the rest of the project and should not need any internal solution references.

## Blue.Library

The LIBRARY project contains all code application business logic for API and business/domain services. All business-layer exceptions, services, responses and results should be placed here.

A **service** encapsulates one or more controller from DAL adding in any business layer or data layer rules.

A **response** is a response from a service call containing success or failure, message(s) and the underlying result.

A **result** is any generic result from an internal operation from the LIBRARY or can be referenced by a response.

An **exception** encapsulates any business layer exception and should perform logging.

## Blue.TestSuite

The TEST SUITE project contains all code test suite code using Nunit, Moq etc. Tests are broken down into several types:

* Performance Tests
* Functional Tests
* Moq Tests
* Nnit Debug Tests

Included in the Blueprint solution are performance and debug tests to illustrate basic library and project functionality.

## Blue.WebAPI

The WebAPI project exposes all the services available via a RESTful API structure and wraps the “SERVICES” section of the Blue.Library project. The WebAPI project also contains token/authentication/authorization framework samples implemented.

## HELIX Development Guidelines

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Name** | **Description** | **References** | **Detail** |
| **Blueprint.Data** | Data access infrastructure code, Models, Compound Models, Sql, Generic Repository and View Models | helixfx; Blueprint.General | |  |  |  | | --- | --- | --- | | **Directory Name** | **What's inside?** | **Code?** | | **Custom Models** | Class for custom data-binding via SQL or SPROC | Yes; POCO Model only | | **Infra** | IBlueprintRepository ( : IHelixPetaRepository) | No | |  | abstract partial BlueprintPetaRepository ( : HelixPetaRepo, IBPRepo) | No | |  | abstract partial BlueprintPetaRepository.Events | Yes; custom database event handlers; e.g. OnAfterDelete, OnBeforeInsert, etc. | | **Models** | Domain model. | Yes; POCO Model only; Add "TableName" attributes and "Ignore" or "Result" attributes where required. | | **Repository** | IRepository ( : IBlueprintRepository) | No | |  | RepositoryContainer class | Yes; | | **Sql** | sealed class for SqlCommands | Yes. Static properties for raw SQL with @ params used in ViewModels | | **ViewModels** | ViewModels are views of the data model for specific workflows, e.g. billing, or loading FK-based data in one view for complex ops. | Yes. If Custom, abstract "CustomView" base class for direct data access for collections, otherwise use repo classes to load relevant data in view. | | **Controllers  and/or**  **Services** | ControllersContainer wraps Repository classes; Exposed methods are Save() [insert or update], Update() [custom update e.g. field(s)], Destroy() [deletes by object ID] | Yes; | |
| **Blueprint.Database** | SqlConnect Project used to maintain SQL DDL | None. | This project isn't used by any other project. Should not be referenced anywhere. |
| **Blueprint.General** | General access code like settings, database connection name, strings, and app-wide enums. This project is referenced by several other projects. Should have NO inbound references other than Helixfx. | helixfx; | |  |  |  | | --- | --- | --- | | **Directory/File** | **What's inside?** | **Code**? | | Config.cs | Exposes name of connection string and connection string from app.config or web.config only. | Y; getter only. | | BlueprintSettings.cs | Exposes Settings.Get() method in helixfx in static properties | Y; getter only. | |
| **Blueprint.Library** | The majority of application functionality lives here including: Contexts, Controllers, Exceptions, Services, UnitsOfWork, Helpers, Providers and Utility methods | helixfx; BP.Data, BP.General, BP.ThirdParty | |  |  |  | | --- | --- | --- | | **Directory** | **What's inside?** | **Code?** | | Context | Contextual object containers for specific application transactions modeled after HttpContext, e.g. ChargingContext used to bill customers. | Yes. Inject ViewModels, expose any custom or relevant objects here. | |  |  |  | | Exceptions | Used to store collection of business exceptions; publishes errors to log | Yes; should publish exceptions | | Extensions | Used to store class extensions methods e(name); | Yes; Pluralize() method is first method in this class. Ref: System.Data.Entity.Design.Pluralization | | Helpers | Simpleton classes to wrap complex or compound objects. | Yes; "new" up objects from Providers | | Providers | Wraps 3rd party clients via DI; naming convention XXXClientProvider.cs, e.g. EmailClientProvider.cs | Yes; ref 3rd party public clients via DI. | | Responses | Processes and returns strongly typed responses from Providers or helpers. | Yes; Pass in response (usually string) to parse in constructor. | | Services | Compound classes/objects to perform app work-flow, complex functions | Yes; all based on requirements | | Units | Units of Work wrappers for complex controller-based transactions. Expose compound Save(), etc., methods should obfuscate complex logic. | Yes; DI the relevant controllers; | | Utility | Utility classes; Used for business auditors, other utility classes used throughout the library. | Yes; | |
| **Blueprint.ThirdParty** | Directly references third-party components, e.g. payment providers, Pdf libraries, faxing, emailing, etc. | helixfx; any third party libs; no other project references should be used. | This exposes "client" classes to wrap functionality of third party controls. only "simple" native objects or tuples, dynamics should be returned, not library-based response or status objects. |
| **Blueprint.Tests** | Test Suite library; Each class in every project should have it's own test fixture (class). | JustMock; helixfx; all other BP project under test. | Each method should have expected input/output, exception case, and any edge cases that appear as development progresses. |
| **Blueprint.WebAPI** | WebApi referencing WebAPI, AttributeRouting.  etc. Directly connects to either controllers or services classes to expose functions. Often used to expose reporting for AJAX-based jQuery calls for charting. | WebAPI; helixfx; | Core API for BP application. Exposes underlying domain model's controllers and services. Optionally add meters to track response times for API calls via DI. |

## Blue Release notes

There are still a few bugs in the search and type ahead autocomplete in the ang client. Work will continue to fix the bugs and add more documentation and use cases for the backend and the angular client.