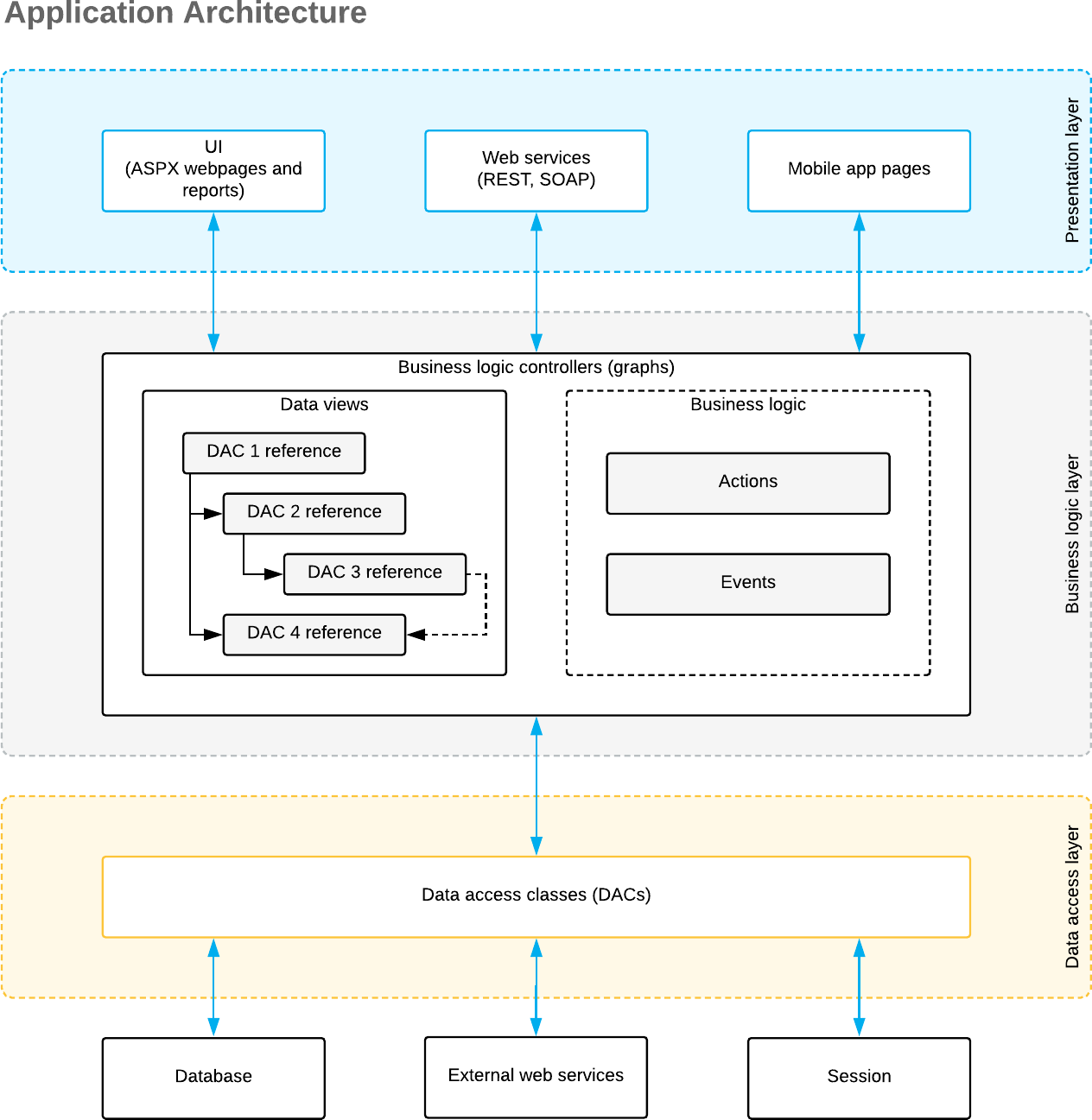
# T200 Maintenance Forms 2022 R1

## Application Programming Overview

Acumatica Framework provides the platform and tools for developing cloud business applications. This topic explains the runtime structure of Acumatica Framework, introduces the main components of this platform, and illustrates the relationships between these components by using simple examples.

### Runtime Structure and Components

An application written with Acumatica Framework has *n*-tier architecture with a clear separation of the presentation, business, and data access layers, as shown in the following diagram. You can find details about each layer in the sections below.



### Data Access Layer

The data access layer of an application written using Acumatica Framework is implemented as a set of data access classes (DACs) that wrap data from database tables or data received through other external sources (such as Amazon Web Services).

The instances of data access classes are maintained by the business logic layer. Between requests, these instances are stored in the session. On a standalone Acumatica ERP server, session data is stored in the server memory. In a cluster of application servers, session data is serialized and stored in a high-performance remote server through a custom optimized serialization mechanism.

For details about data storage in a session, see [*Session*](https://help-2022r1.acumatica.com/Help?ScreenId=ShowWiki&pageid=4009fc41-b1ec-4e2d-9940-c92a32d5ca9b). For details on working with the data access layer, see

[*Accessing Data*](https://help-2022r1.acumatica.com/Help?ScreenId=ShowWiki&pageid=cc602399-bac4-46bf-b23c-79e86251c521).

### Business Logic Layer

The business logic is implemented though the business logic controller (also called *graph*). Graphs are classes that you derive from the special API class (PXGraph) and that are tied to one or more data access classes.

Each graph conceptually consists of two parts:

* Data views, which include the references to the required data access classes, their relationships, and other meta information
* Business logic, which consists of actions and events associated with the modified data.

Each graph can be accessed from the presentation layer or from the application code that is implemented within another graph. When the graph receives an execution request, it extracts the data required for request execution from the data access classes included in the data views, triggers business logic execution, returns the result of the execution to the requesting party, and updates the data access classes instances with the modified data.

For details on working with the business logic layer, see [*Implementing Business Logic*](https://help-2022r1.acumatica.com/Help?ScreenId=ShowWiki&pageid=6aea8e6f-3cc6-4f61-88a3-d7f49cd6c4b7).

### Presentation Layer

The presentation layer provides access to the application business logic through the UI, web services, and Acumatica mobile application. The presentation layer is completely declarative and contains no business logic.

The UI consists of ASPX webpages (which are based on the ASP.NET Web Forms technology) and reports created with Acumatica Report Designer. The ASPX webpages are bound to particular graphs.

When the user requests a new webpage, the presentation layer is responsible for processing this request. Webpages are used for generating static HTML page content and providing additional service information required for the dynamic configuration of the web controls. When the user receives the requested page and starts browsing or entering data, the presentation layer is responsible for handling asynchronous HTTP requests. During processing, the presentation layer submits a request to the business logic layer for execution. Once execution is completed, the business logic layer analyzes any changes in the graph state and generates the response that is sent back to the browser as an XML document.

For details on the configuration of ASPX webpages, see [*Configuring ASPX Pages and Reports*](https://help-2022r1.acumatica.com/Help?ScreenId=ShowWiki&pageid=ca6d5149-db7a-42b9-bce1-d6f26a5c2ca7).

## Querying of the Data

Acumatica Framework provides a custom language called *BQL (business query language)* that developers can use for writing database queries. BQL is written in C# and based on generic class syntax, but is still very similar to SQL syntax.

Acumatica Framework provides two dialects of BQL: traditional BQL and fluent BQL. We recommend that you use fluent BQL because statements written in fluent BQL are simpler and shorter than the ones written with traditional BQL. Further in this topic, the examples are written in fluent BQL.

You can also use LINQ to select records from the database or to apply additional filtering to the data of a BQL query. For details on which approach to use, see [*Comparison of Fluent BQL, Traditional BQL,*](https://help-2022r1.acumatica.com/Help?ScreenId=ShowWiki&pageid=b5126e1e-70d9-408d-bcf9-e625a38a052e)[*and LINQ*](https://help-2022r1.acumatica.com/Help?ScreenId=ShowWiki&pageid=b5126e1e-70d9-408d-bcf9-e625a38a052e).

BQL has almost the same keywords as SQL does, and they are placed in the same order as they are in SQL, as shown in the following example of BQL.

SelectFrom<Product>.Where<Product.availQty.IsNotNull.

And<Product.availQty.IsGreater<Product.bookedQty>>>

If the database provider is Microso SQL Server, the framework translates this expression into the following SQL query.

SELECT \* FROM Product

WHERE Product.AvailQty IS NOT NULL

AND Product.AvailQty > Product.BookedQty

BQL extends several benefits to the application developer. It does not depend on the specifics of the database provider, and it is object-oriented and extendable. Another important benefit of BQL is compile-time syntax validation, which helps to prevent SQL syntax errors.

Because BQL is implemented on top of generic classes, you need data types that represent database tables. In the context of Acumatica Framework, these types are called *data access classes (DACs)*. As an example of a DAC, you would define the Product data access class as shown in the following code fragment to execute the SQL query from the previous code example.

using System; using PX.Data;

[PXCacheName("Product")]

public class Product : PX.Data.IBqlTable

{

// The property holding the ProductID value in a record [PXDBIdentity(IsKey = true)]

public virtual int? ProductID { get; set; }

// The type used in BQL statements to reference the ProductID column public abstract class productID : PX.Data.BQL.BqlInt.Field<productID> { }

// The property holding the AvailQty value in a record [PXDBDecimal(2)]

public virtual decimal? AvailQty { get; set; }

// The type used in BQL statements to reference the AvailQty column

public abstract class availQty : PX.Data.BQL.BqlDecimal.Field<availQty> { }

// The property holding the BookedQty value in a record [PXDBDecimal(2)]

public virtual decimal? BookedQty { get; set; }

// The type used in BQL statements to reference the BookedQty column

public abstract class bookedQty : PX.Data.BQL.BqlDecimal.Field<bookedQty> { }

}

Each table field is declared in a data access class in two diﬀerent ways, each for a diﬀerent purpose:

* As a public virtual property (which is also referred to as a *property field*) to hold the table field data
* As a public abstract class (which is also referred to as a *class field* or *BQL field*) to reference a field in the BQL command

You will learn more about data access classes later in this course.

#### Related Links

* [*Querying Data in Acumatica Framework*](https://help-2022r1.acumatica.com/Help?ScreenId=ShowWiki&pageid=5978bf0a-e65c-47ca-8fec-40aa733ae2b8)