Architecture Details

int.prd.002.03 | 07.11.25

UPM 25.x

ZINFI Confidential & Proprietary

Shared Under NDA

Contents

Introduction

ZINFI UPM Architecture facilitates a separation of development of the graphical user interface – GUI code – from development of the business logic or back-end logic (the data model). The view model is a value converter, meaning the view model is responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented. The Architecture was invented by ZINFI specifically to simplify event-driven programming of user interfaces. The pattern was incorporated into UPM embedding Angular JS, .NET Core and Microsoft SQL Server.

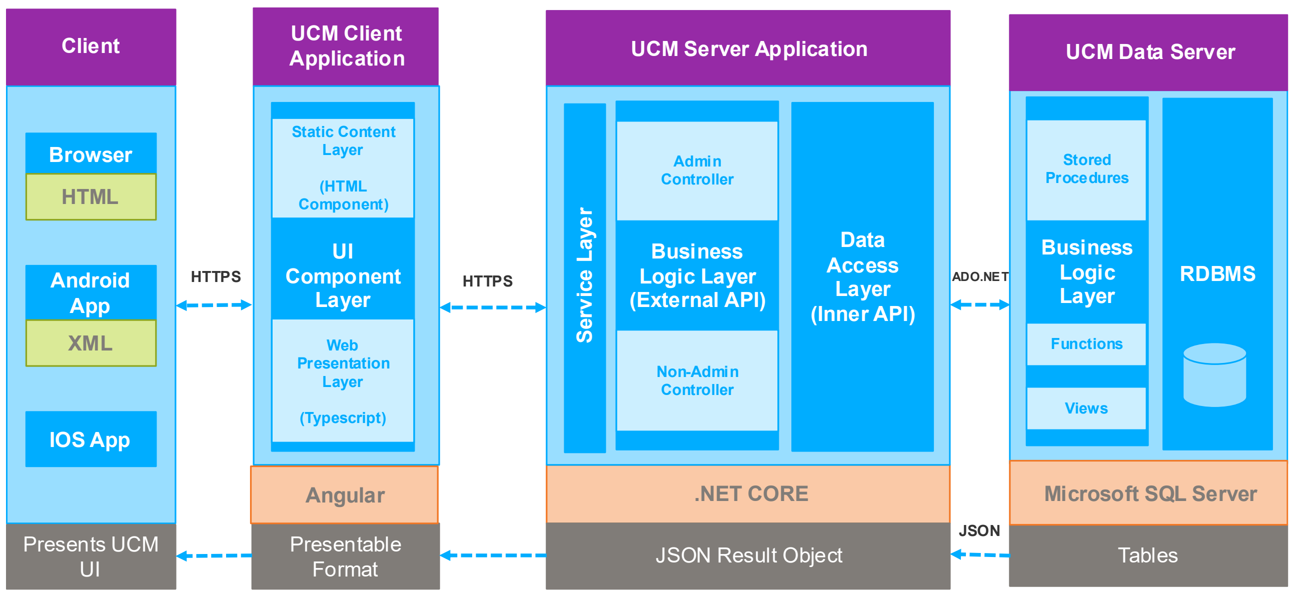
The ZINFI UPM's Layered security architecture refers to security systems that use multiple components to protect operations on multiple levels, or layers. The central idea behind layered security or defense is that in order to protect systems from a broad range of attacks, using multiple strategies will be more effective. Layered security involves security protocols at the system or network levels, at the application level, or at the transmission level.

ZINFI UPM's aSaaS is a ground-breaking technology in use, assisting you to tailor the application, hosted on our infrastructure or yours.

ZINFI aSaaS architecture is hard to believe until one has seen it in action:

- Develop full enterprise class applications in a couple of weeks, without coding.
- Deploy as a SaaS Service or move behind the firewall in just 10 mouse clicks.
- Scale to address multiple partners, without changing a line of code.
- Upgrade painlessly to receive program enhancements with no impact on your customizations.

Architectural Insights



System Components Summary

Client

- The Client Interface of the UPM Application, which communicates with the UPM Client Application and utilized to display the UPM UI Components. Communication between the Client Application and Web Client is processed via HTTPS.
- The UPM Application UI is available in two formats:
    - **Desktop** – The application runs through OS based Browser.
    - **Mobile** – (a)Android – The application runs through a native app exclusively developed for Android. (b) iOS – The application runs through a native app made developed for iOS.

Client Application

- UPM Client App consists of the UI Component Layer which interacts with the Server App to call services and fetch information to display via the Client Interface. It consists of the following layers:
- **UI Component Layer** – Designed in Angular using TypeScript.
    - **Static Content Layer** – This layer contains the HTML components.

- **Web Presentation Layer** – This layer contains the TypeScript code files, which gets embedded with HTML components to produce the desired design of the page.

Server Application

The Server Application Core holds the UPM business logic layer, which includes interfaces and controllers. These Controllers include abstractions for operations that will be performed using Infrastructure, such as data access.

- **Service Layer** – Architected in .NET Core.
    - This layer consists of the handler and service component classes written in TypeScript. These classes are then invoked in Presentation Layer to deliver the UI components.
    - A service layer is an additional layer that mediates communication between the controller and repository layer. The service layer contains business logic along with validation logic.
- **Business Logic Layer [External API]** – This Layer is designed in .NET Core. When the control enters from Angular to .NET Core via the Service Layer, it first hits the Logic Layer.
    - **Admin Controller** – This layer contains the API classes and method calls for Administrative module(s) of ZINFI UPM.
    - **Non-Admin Controller** – This layer contains the API classes and method calls for rest of the modules of ZINFI UPM, except the administrative module(s).
- **Data Access Layer [Inner API]** – Designed in .NET Core, this layer contains the method definitions that are called from the Logic Layer. With the help of ADO.NET Technology, UPM Server App communicates with Data Server for retrieval of data (as required).

Data Server

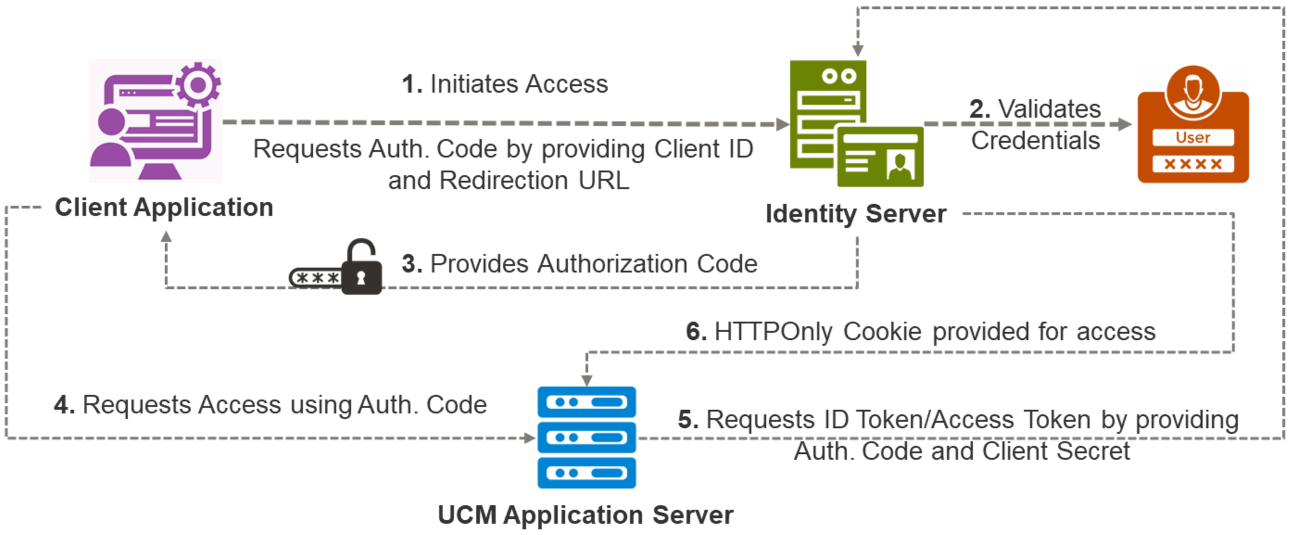
This section contains the business logic of ZINFI UPM in **Business Logic Layer** and **Relational Database** to store application data. Microsoft SQL Server is used for this purpose.

- **Business Logic Layer** – Contains Stored Procedures, Functions and Views which constitute the business logic of ZINFI UPM.
- **Relational Database** – Contains data along with their relational constraints.

Authentication Server

Used for authenticating the client log-in request, in any one of the two ways:

- by retrieving the application token from **Authentication Data Server**, when logged-in directly using ZINFI UPM.
- by retrieving the application token from **Authentication Data Server** via third-party API, when logged-in using CRM login credentials utilizing Single Sign-On (SSO) functionality, for example, Salesforce, MS Dynamics etc.


1. Initiates Access
Requests Auth. Code by providing Client ID and Redirection URL
2. Validates Credentials
User
xxxx
Client Application
Identity Server
3. Provides Authorization Code
6. HTTPOnly Cookie provided for access
4. Requests Access using Auth. Code
5. Requests ID Token/Access Token by providing Auth. Code and Client Secret
UCM Application Server

Authentication Components Summary

Technology used

- Identity Server, ASP.NET Core, OAuth
- Authentication server is used for enhanced security. Identity Server provides the OpenID and OAuth services. ASP.NET Core is used for creating the UI interfaces for login/logout.

Identity Server

- An authentication server that implements OpenID Connect (OIDC) and OAuth standards for ASP.NET Core.
- Designed to provide a common way to authenticate requests to the application.
- Typically involves token issuance, verification, and renewal without any user interface.

Advantages of using separate authentication server

- De-couples the authentication logic from the system.
- Client-specific resource access grants.
- Overall, provides better-secured SSO with other services.

Control Flows

OpenID-OAuth Login Process

Authentication Server Validation Checks

The architecture of UPM follows the **OpenID Implicit Flow** methodology. In this new process, when a user logs into the application, the authentication process follows Implicit Flow steps:

1. Client Application further referred to as Client (JavaScript / C#) prepares an Authentication Request containing the desired request parameters.
2. Client sends the request to the Authentication Server.
3. Authentication Server authenticates the End-User.
4. Authentication Server obtains End-User Consent/Authorization.
5. Authentication Server sends the End-User back to the Client with an ID Token and, if requested, an Access Token.
6. Client validates the tokens and retrieves the End-User's Subject Identifier.

Application Database Validation Checks

After authenticating the user from the Authentication Server, checks are done that whether the same exists in Application Database (in Authentication Data Server). If the user does not exist there, then it is created, so that a user logging into the Application Database can be tracked.

User Roles

When the application logs-in with a user role after performing required validation checks, it queries table **Roles_Users** – which keeps track of all users and the available role(s) that has been assigned to them. Only roles assigned to that user are viewable. Example of roles are CMM, CP, Admin etc.

Assigning Groups to Roles

After a role has been selected by logged-in user, the application checks availability of groups that has been assigned to that specific user role. For this purpose, the application queries table **ROLE_GROUPS_ASSOC** – which keeps track of the group(s) that has been assigned to that role.

Flow of Control across Modules

After the user logs-in to the application, the flow of control across various modules are described below in the order it occurs.

1. The Client Application sends HTTPS request to Server Application to fetch data required to display in its UI.
2. The Server Application receives request from its predecessor, and then acts in following way:
    a. The UI Component Layer receives request. The Static Content Layer passes request to Web Presentation Layer. The Web Presentation Layer receives request and passes the same for further processing to Service Layer.
    b. The Internal / MVVM Controller of Service Layer receives request and passes it to Logic Layer [External API].
    c. The Logic Layer receives request and checks, if the request is for administrative section of application then it passes the same to System Controller, or else the request is passed to Values Controller.
    d. After processing request in Logic Layer, it is passed to Data Access Layer [Inner API]. For further processing, through ADO.NET, the request is passed to Data Server.
3. In Data Server, the Business Logic Layer receives that request, and then calls the respective Stored Procedure, Function or View as required based on logic written for processing. The raw data with relational constraints are retrieved based on queries performed.
4. The resultant dataset tables from Data Server is transferred to Server Application in JSON format.
5. The JSON Result Object received from Data Server is moved through Data Access Layer [Inner API] to Service Layer.
6. The UI Component Layer transforms JSON Result Object to Presentable Format like Tables, Views etc.
7. The Client Application receives that Presentable Format from UI Component Layer of Server Application, and then displays the same in the UI to user.