A diagram of a service

Description automatically generatedA diagram of a software application

Description automatically generated

|  |
| --- |
| **Cloud-Native Concepts –** Microservices, Containers, Continuous Delivery, DevOps  **Benefits –** Scalability, Resilience, Flexibility, Cost Efficiency  **Key Azure Services for cloud Native** -AKS,app services, function, CosmosDB, Service Bus / Event Grid, Monitor, application insight, Azure AD, Azure SQL Database  **Building and Deploying Cloud-Native Applications with Azure and .NET –** Containerization, CI/CD with Azure DevOps |

|  |  |
| --- | --- |
| dotnet add package **IdentityServer4**  add package **Microsoft.AspNetCore.Authentication.JwtBearer**  public void **ConfigureServices**(IServiceCollection services)  {  services.**AddIdentityServer**()  .**AddInMemoryClients**(Config.GetClients())  .**AddInMemoryApiScopes**(Config.GetApiScopes())  .**AddInMemoryIdentityResources**(Config.GetIdentityResources())  .AddTestUsers(Config.GetUsers())  .AddDeveloperSigningCredential(); // Developer signing credential for demo purposes    services.**AddControllers**();  }  **Configure**  app.UseIdentityServer();  **Create API**  dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer  dotnet add package Swashbuckle.AspNetCore  public void ConfigureServices(IServiceCollection services)  {  services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)  .AddJwtBearer(options =>  {  options.Authority = "https://localhost:5001"; // IdentityServer URL  options.Audience = "api1"; // The API scope from IdentityServer  });  services.AddSwaggerGen();  services.AddControllers();  } | public static class Config  {  public static IEnumerable<Client> GetClients()  {  return new List<Client>  {  new Client  {  ClientId = "client",  AllowedGrantTypes = GrantTypes.ClientCredentials,  ClientSecrets = { new Secret("secret".Sha256()) },  AllowedScopes = { "api1" }  }  };  }  public **static IEnumerable<ApiScope> GetApiScopes**()  {  return new List<ApiScope>  {  **new ApiScope("api1", "My API")**  };  }  public static IEnumerable<IdentityResource> GetIdentityResources()  {  return new List<IdentityResource>  {  new IdentityResources.OpenId(),  new IdentityResources.Profile()  };  }  public static List<TestUser> **GetUsers**()  {  return new List<TestUser>  {  new TestUser  {  SubjectId = "1",  **Username = "admin",**  **Password = "password"**  }  };  }  } |
|  |  |

|  |
| --- |
| public void ConfigureServices(IServiceCollection services)  {  // Configure JWT Bearer Authentication  services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)  .AddJwtBearer(options =>  {  options.Authority = "https://your-identity-server.com"; // Identity server URL  options.Audience = "your-api1"; // Audience from IdentityServer  options.TokenValidationParameters = new TokenValidationParameters  {  ValidateIssuer = true,  ValidateAudience = true,  ValidateLifetime = true,  ValidateIssuerSigningKey = true  };  });  services.AddAuthorization(options =>  {  options.AddPolicy("Admin", policy => policy.RequireRole("Admin"));  });  services.AddControllers();  }  public void Configure(IApplicationBuilder app)  {  app.UseRouting();  // Enable Authentication Middleware  app.UseAuthentication();  app.UseAuthorization();  app.UseEndpoints(endpoints =>  {  endpoints.MapControllers();  });  } |

|  |
| --- |
| public void ConfigureServices(IServiceCollection services)  {  services.AddIdentityServer()  .AddInMemoryClients(Config.GetClients()) // Clients for OAuth  .AddInMemoryApiScopes(Config.GetApiScopes()) // API scopes  .AddInMemoryIdentityResources(Config.GetIdentityResources()) // Identity resources like OpenID, Profile  .AddTestUsers(Config.GetUsers()) // Test users  .AddDeveloperSigningCredential(); // For development only, use a real key in production  services.AddControllers();  }  public static class Config  {  public static IEnumerable<Client> GetClients()  {  return new List<Client>  {  new Client  {  ClientId = "client",  AllowedGrantTypes = GrantTypes.ResourceOwnerPassword,  ClientSecrets = { new Secret("secret".Sha256()) },  **AllowedScopes = { "api1", "openid", "profile" }**  }  };  }  public static IEnumerable<ApiScope> GetApiScopes()  {  return new List<ApiScope>  {  new ApiScope("api1", "My API")  };  }  public static IEnumerable<IdentityResource> GetIdentityResources()  {  return new List<IdentityResource>  {  new IdentityResources.OpenId(),  new IdentityResources.Profile()  };  }  public static List<TestUser> GetUsers()  {  return new List<TestUser>  {  new TestUser  {  SubjectId = "1",  Username = "admin",  Password = "password" // Test user with username "admin" and password "password"  }  };  }  } |
| **Client (e.g., Postman) to Get Access Token Using Username/Password**  **Method**: POST  **URL**: <https://your-identity-server.com/connect/token>  grant\_type=password  client\_id=client  client\_secret=secret  username=admin  password=password  scope=api1 openid profile  **Response**:  {  "access\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  "expires\_in": 3600,  "token\_type": "bearer"  } |
| I see! You want to understand how to authenticate users with **username** and **password** directly using **API Management (APIM)** while still securing your backend API with **role-based authorization** in your .NET Core API.  Let’s break this down step-by-step. The scenario here is:   1. **API Management (APIM)** is used to expose your API. 2. **User authentication** happens via **username** and **password** directly through **IdentityServer (or another identity provider)**. 3. **API Management** will handle the token issuance (after username and password are validated), but you don’t want **APIM** itself to authenticate the user — rather, it should **pass the user credentials to your Identity Provider** (e.g., IdentityServer) and get an **access token** that your .NET Core API will validate.   **Overview of the Flow**   1. **User logs in with username/password** via **IdentityServer** or another Identity Provider. 2. **IdentityServer** authenticates the user and issues an **access token**. 3. The **client** (Postman, etc.) uses this token to make **authenticated API calls** to your backend **secured API**. 4. **API Management (APIM)** validates the token (using OAuth2/JWT) and forwards the request to your backend **.NET Core API**. 5. The **.NET Core API** validates the token and processes the request with role-based authorization.   **1. IdentityServer Setup for Username and Password Authentication**  Assume you’re using **IdentityServer** as your Identity Provider.  First, let's ensure IdentityServer is set up to handle **username and password** login via the **Resource Owner Password Credentials Grant**.  **IdentityServer Configuration (in IdentityServer project):**  In your IdentityServer setup (in Startup.cs), you will configure it to use **Password Grant** (where the user submits their **username** and **password** directly):  csharp  Copy code  public void ConfigureServices(IServiceCollection services)  {  services.AddIdentityServer()  .AddInMemoryClients(Config.GetClients()) // Clients for OAuth  .AddInMemoryApiScopes(Config.GetApiScopes()) // API scopes  .AddInMemoryIdentityResources(Config.GetIdentityResources()) // Identity resources like OpenID, Profile  .AddTestUsers(Config.GetUsers()) // Test users  .AddDeveloperSigningCredential(); // For development only, use a real key in production  services.AddControllers();  }  public static class Config  {  public static IEnumerable<Client> GetClients()  {  return new List<Client>  {  new Client  {  ClientId = "client",  AllowedGrantTypes = GrantTypes.ResourceOwnerPassword,  ClientSecrets = { new Secret("secret".Sha256()) },  AllowedScopes = { "api1", "openid", "profile" }  }  };  }  public static IEnumerable<ApiScope> GetApiScopes()  {  return new List<ApiScope>  {  new ApiScope("api1", "My API")  };  }  public static IEnumerable<IdentityResource> GetIdentityResources()  {  return new List<IdentityResource>  {  new IdentityResources.OpenId(),  new IdentityResources.Profile()  };  }  public static List<TestUser> GetUsers()  {  return new List<TestUser>  {  new TestUser  {  SubjectId = "1",  Username = "admin",  Password = "password" // Test user with username "admin" and password "password"  }  };  }  }  This setup allows users to authenticate using **username** and **password** directly with **IdentityServer**.  **2. Client (e.g., Postman) to Get Access Token Using Username/Password**  Once IdentityServer is set up, the client (Postman, etc.) will request an access token by providing **username** and **password** to the /connect/token endpoint of IdentityServer.  In **Postman**, follow these steps to get the **access token**:   * **Method**: POST * **URL**: https://your-identity-server.com/connect/token * **Body** (x-www-form-urlencoded):   plaintext  Copy code  grant\_type=password  client\_id=client  client\_secret=secret  username=admin  password=password  scope=api1 openid profile  In the response, you will get an **access token**:  json  Copy code  {  "access\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  "expires\_in": 3600,  "token\_type": "bearer"  }  The **access token** is what the client will use to access your backend API securely.  **3. API Management (APIM) Configuration for Token Validation** |

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Secure azure function (Managed identity) call with Azure AD and calling from azure function

A blue and black shield with a check mark

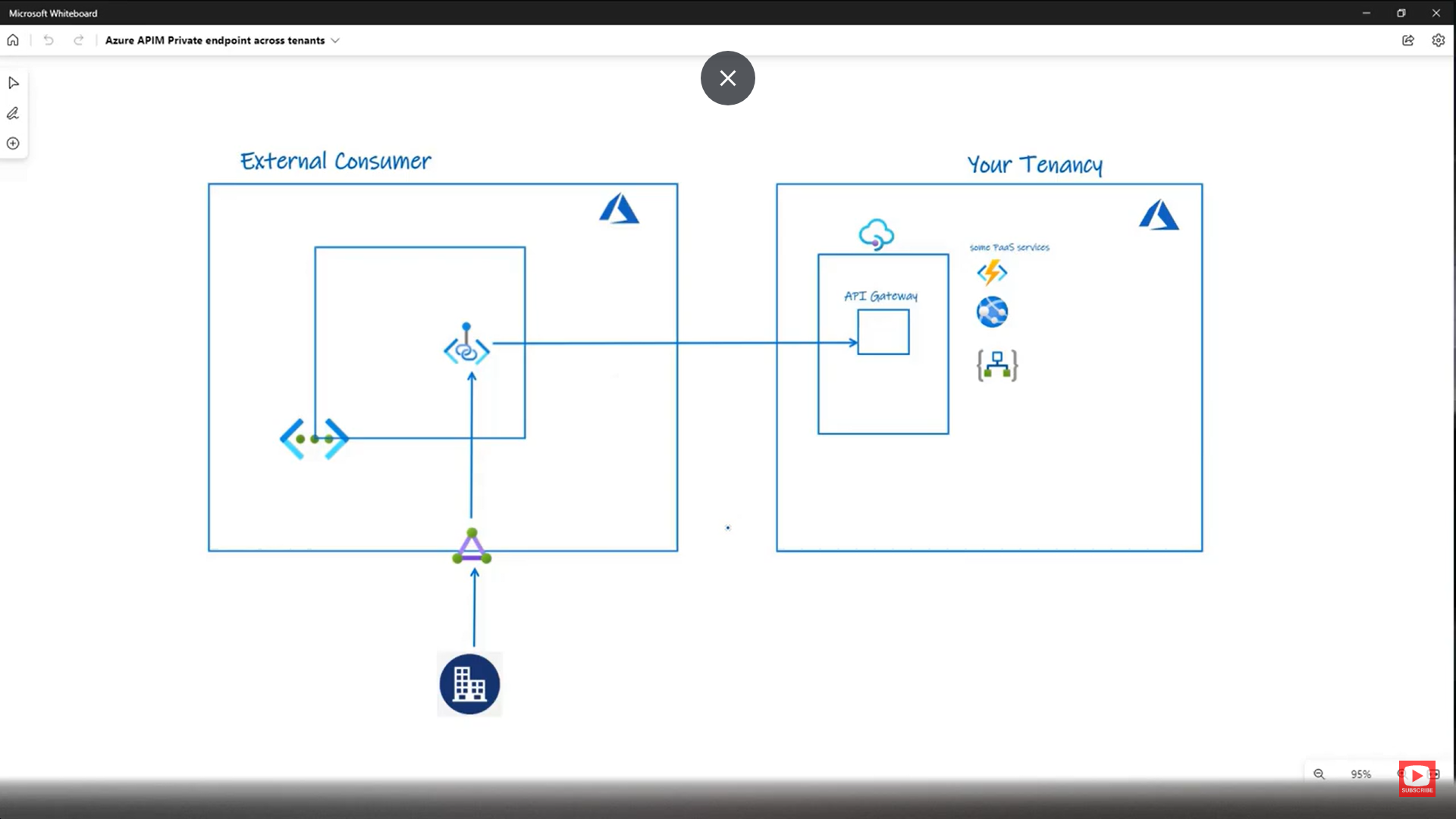
Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



API management -> Network 🡪 Private end point

A diagram of a group of text

Description automatically generated**How to store value objects in the database?**

Let’s say we have two classes in our domain model: Person entity and Address value object:

// Entity

public class Person

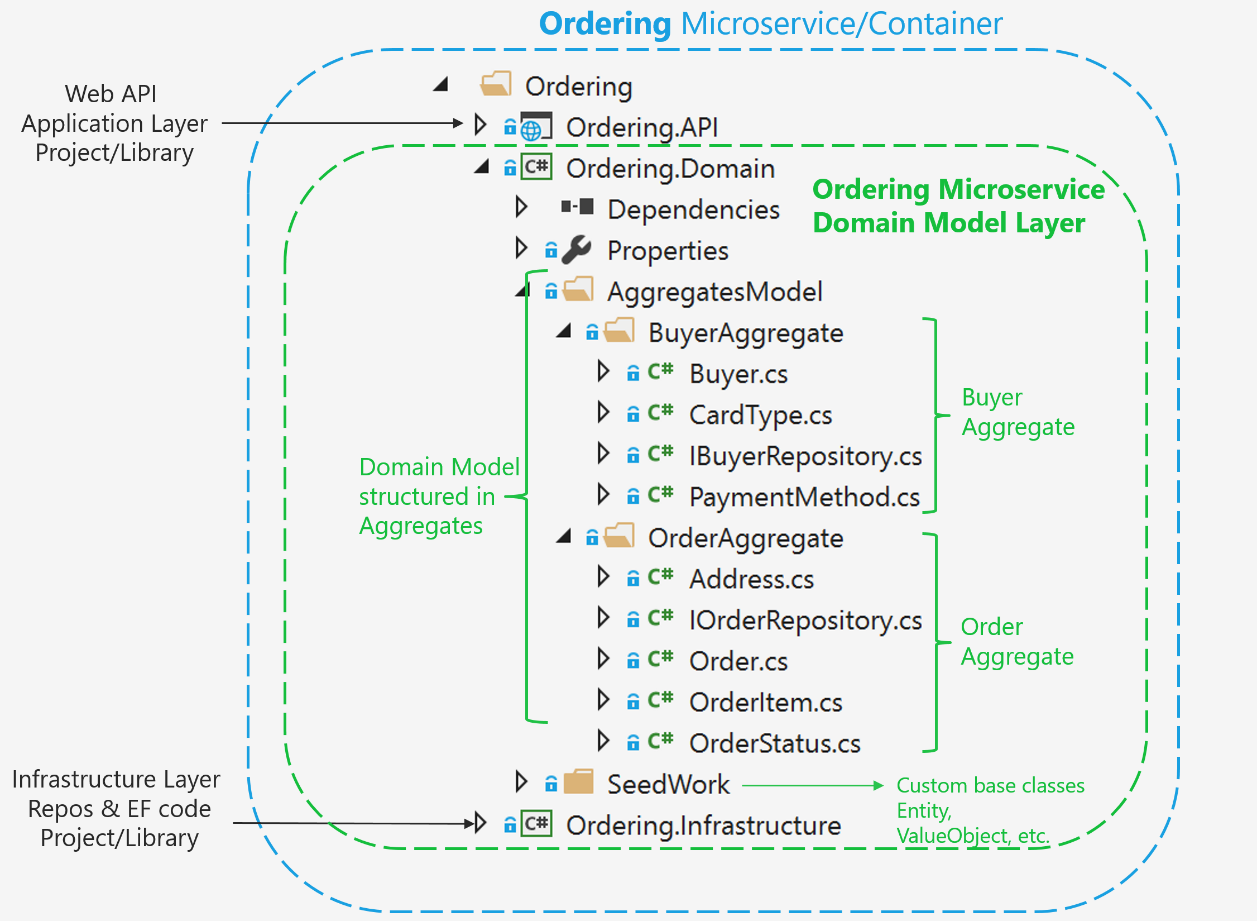
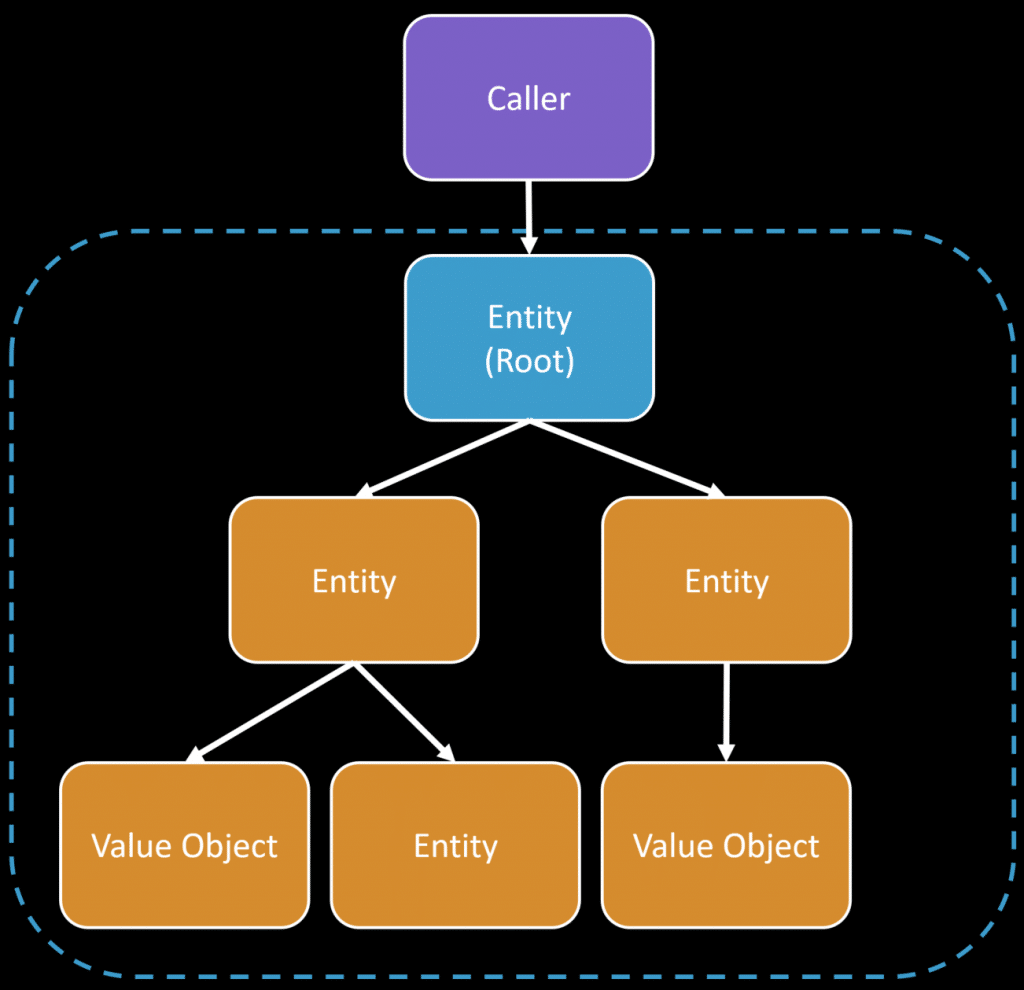
{

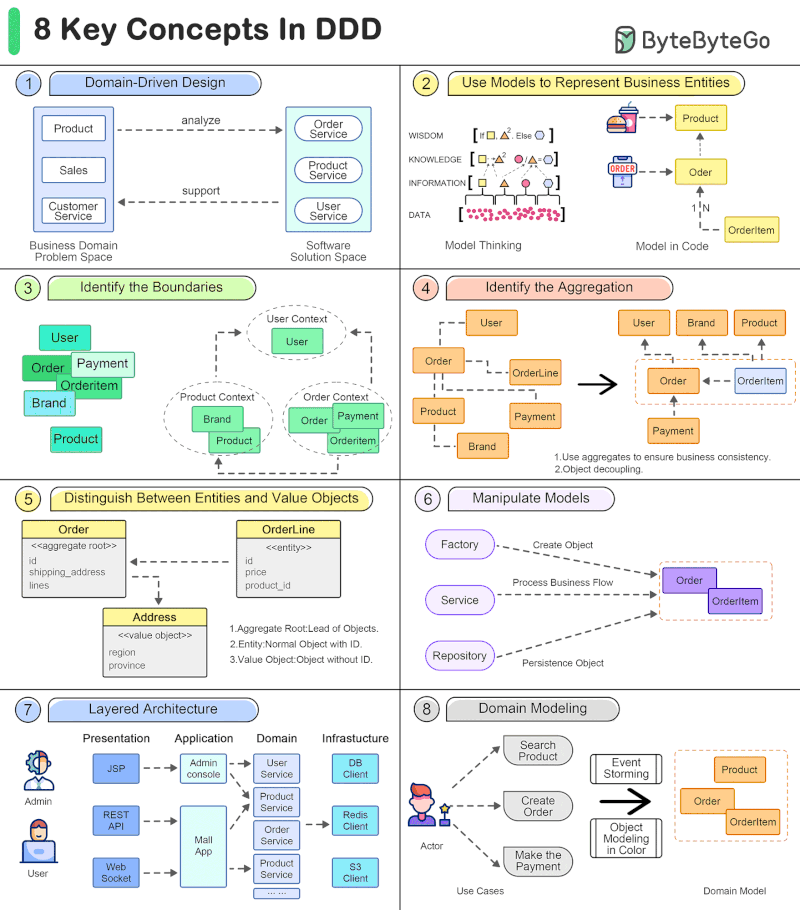
    public int Id { get; set; }

    public string Name { get; set; }

    public Address Address { get; set; }

}

A white background with green text

Description automatically generated

// Value Object

public class Address

{

    public string City { get; set; }

    public string ZipCode { get; set; }

}

